Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería de Tecnologías Industriales

Desarrollo de una librería para compartir variables entre procesadores ESP32

Autor:

Martínez Ortega, Javier Antonio

Tutor:

De Pablo Gómez, Santiago
Tecnología Electrónica

Valladolid, Julio, 2025.

Resumen

Este Trabajo de Fin de Grado tiene como objetivo crear una librería en C++ desarrollada en el entorno de programación Arduino IDE que permita la intercomunicación entre microprocesadores mediante el protocolo ESP-NOW. La librería facilitará el envío, lectura y escritura de variables compartidas mediante la simulación de una memoria común (memoria compartida) entre dispositivos.

Dado que la implementación de la tecnología ESP-NOW puede resultar compleja, esta librería simplifica el proceso, ofreciendo una interfaz accesible para gestionar la intercomunicación entre nodos de forma eficiente sin necesidad de estructura Wi-Fi adicional. Además, se consigue facilitar el proceso de creación de herramientas software reduciendo los conocimientos técnicos necesarios para ello.

Palabras Clave

ESP32, ESP-NOW, Memoria Compartida, Computación Distribuida, IoT

Abstract

This Final Degree Project aims to create a C++ library developed in Arduino IDE programing environment, design to enable intercommunication between microprocessors through the ESP-NOW protocol. This library will make sending, reading and writing of shared variables simple by simulating a common memory (shared memory) between devices.

Since the implementation of ESP-NOW technology could be complex, this library summarizes the process, offering an accessible interface for efficiently managing intercommunication between nodes without needing an extra WiFi infrastructure. Furthermore, this facilitates the development of software tools, reducing technical knowledge required to do so.

Keywords

ESP32, ESP-NOW, Shared Memory, Distributed Computing, IoT

Índice

Contenido

Introducción y objetivo

En el ámbito del internet de las cosas (IoT), la comunicación eficiente entre dispositivos representa un pilar fundamental para el desarrollo de soluciones autónomas. Los microcontroladores ESP32 permiten implementar redes de comunicación inalámbricas, versátiles y de bajo coste entre dispositivos gracias a su conectividad Wi-Fi y Bluetooth integradas.

ESP-NOW es una tecnología de comunicación inalámbrica desarrollada por Espressif Systems que ofrece una alternativa eficiente a las redes Wi-Fi tradicionales ya que permite la comunicación directa de dispositivos sin necesidad de una infraestructura de red centralizada. Sin embargo, poder dominar esta tecnología para el manejo de información a través de redes Wi-Fi sigue siendo un problema para muchos desarrolladores.

En este proyecto se propone el diseño y desarrollo de una librería software para que, a nivel de programación, el intercambio de información entre dispositivos sea más simple y práctico, permitiendo que la creación de redes sea más eficiente y sencilla.

Justificación

Relevancia

En la actualidad, el crecimiento exponencial de dispositivos conectados ha generado una necesidad creciente de desarrollar soluciones IoT eficientes y de bajo coste donde, además, las soluciones simples y flexibles cada vez ganan mayor protagonismo. Con tecnologías como ESP-NOW se consigue una comunicación directa entre dispositivos en entornos donde no es viable establecer infraestructuras centralizadas de red como las tradicionales, además, se elimina la necesidad de requerir puntos de acceso Wi-Fi o servidores intermedios para ello.

A pesar de las claras ventajas de la tecnología ESP-NOW, esta ofrece también ciertos desafíos técnicos que dificultan su implementación en aplicaciones complejas. Por ello, la creación de una librería que facilite la compartición de datos en memoria entre dispositivos ESP32 utilizando ESP-NOW supone una mejora práctica y necesaria para ecosistemas de desarrollo con microcontroladores. Esta herramienta permitiría:

1. Reducir la complejidad a la hora de crear soluciones software.
2. Mejorar la eficiencia a la hora de transmitir datos entre dispositivos.
3. Permitir el desarrollo de soluciones autónomas en entornos sin conectividad.

En definitiva, este proyecto proporciona una solución que facilita el diseño de redes colaborativas en el entorno IoT para los desarrolladores con distintos niveles de experiencia técnica.

Fundamentación Teórica

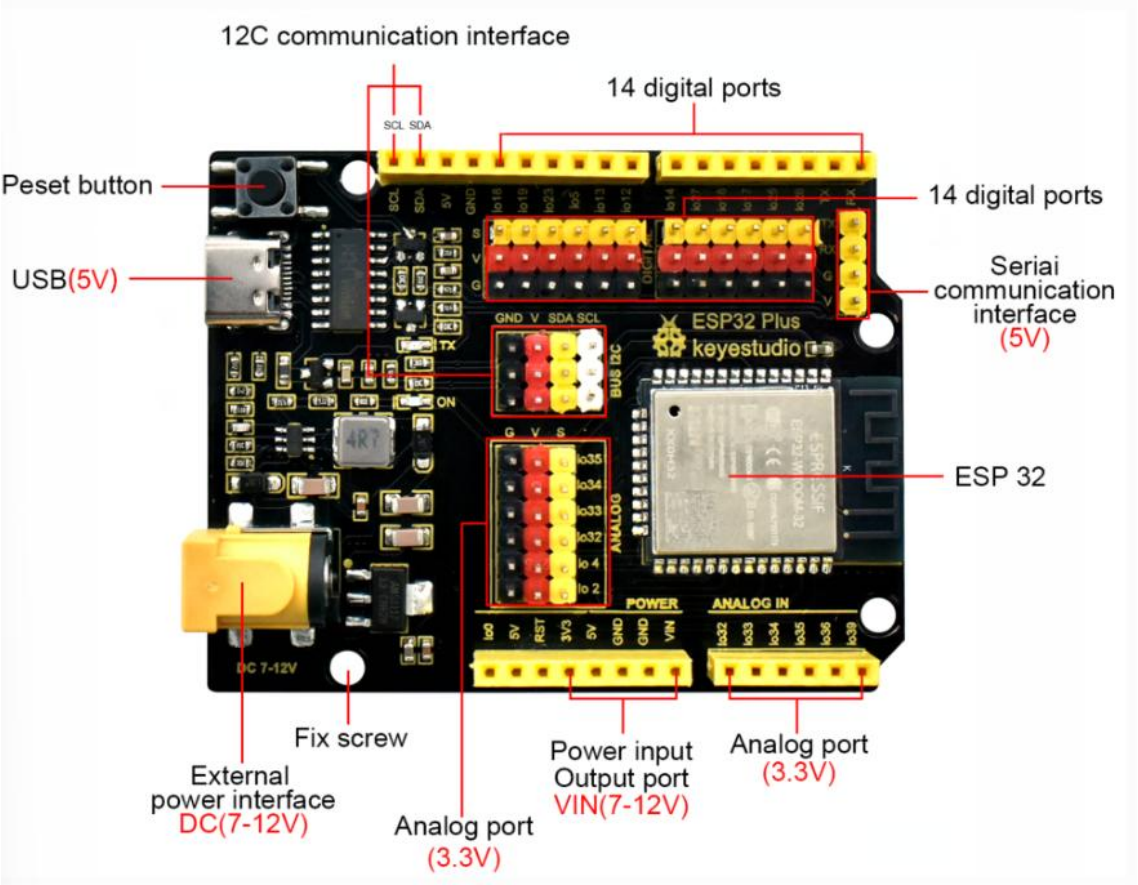
La computación distribuida trata de ejecutar tareas coordinadas empleando múltiples dispositivos con el objetivo de resolver un problema. Para ello, uno de los modelos de comunicación más empleado es el de la memoria compartida donde diferentes nodos tienen la capacidad de acceder a la memoria de los demás compartiendo datos con el resto de nodos dentro de la misma red.

Esta compartición se hace de forma inalámbrica, por ello, para la aplicación creada se usarán placas ESP32 de KeyStudio KS5016. Se trata de unas placas integradas con un módulo ESP32-WROOM-32: con conectividad Wi-Fi y bluetooth. Son compatibles con el entorno de Arduino IDE, y con las especificaciones técnicas siguientes:

- Voltaje: 3,3V-5V
- Intensidad máxima de salida: 1,2A
- Potencia máxima de salida: 10W
- Temperatura de trabajo: -10ºC-50ºC
- Dimensiones: 69*54*15,5mm
- Peso: 25,5g

Además, integra gran cantidad de puertos de entrada y salida para poder conectar multitud de dispositivos.

Partes:



1. Módulo ESP32-WROOM-32: Se trata de un microprocesador de alto rendimiento y bajo consumo desarrollado por Espressif Systems. Permite conectividad Wi-Fi IEEE 802.11 b/g/n (2.4 GHz) para comunicaciones de medio alcance y capacidad de conectividad a una red LAN y bluetooth v4.2 BR/EDR y BLE. Contiene una memoria SRAM de 520KB para instrucciones y datos.
2. Conector USB: para la alimentación y carga de programas.
3. Botón reset: para resetear el microprocesador.
4. Entradas y salidas

La aplicación se ha desarrollado en el entorno de programación de Arduino: "Arduino IDE" debido a su simplicidad y gran ecosistema. "Arduino IDE" se trata de un entorno de programación de código abierto, centrado en desarrollo de código para microcontroladores. Permite facilitar el desarrollo de software debido a su accesibilidad e interfaz intuitiva, consiguiendo disminuir la necesidad de conocimientos técnicos para crear herramientas software. Este entorno permite programación en lenguaje C y C++, contiene un compilador integrado, capacidad para trabajar con gran variedad de placas y bibliotecas, ya sean desarrolladas por Arduino o por la comunidad global de desarrolladores con la que el entorno de programación cuenta. Además, no requiere de hardware de altas prestaciones para funcionar y la carga de programas se hace realmente sencilla debido a la capacidad del IDE de detección de puertos, compilación y carga de firmware.

Debido a la accesibilidad, sencillez y capacidad de programación en diferentes tipos de placas, además de la gran cantidad de librerías de las que dispone, se ha elegido "Arduino IDE" como entorno de programación para el desarrollo de esta aplicación.

Antecedentes

La compartición de datos entre dispositivos ha sido y continúa siendo un pilar fundamental en el desarrollo de proyectos de computación distribuida. Entre las distintas tecnologías desarrolladas para ello, la memoria compartida ha demostrado ser una de las soluciones más útiles y eficientes, especialmente en sistemas donde se requiere un intercambio de información rápido y directo.

Este intercambio de información requiere de un protocolo determinado que pueda soportar la intercomunicación entre dispositivos. Entre los protocolos más empleados usados por empresas importantes se encuentran:

- **WiFi tradicional:** Requiere infraestructura de red, presenta alto consumo energético, alcance medio y latencias relativamente altas.
  Ejm.: MindSphere de Siemens. Es un sistema operativo capaz de conectar dispositivos, obtener datos y utilizarlos para su análisis.
- **LoRa:** Proporciona bajo consumo y alto alcance, pero requiere infraestructura externa (Gateway) además de presentar mayor complejidad.
  Ejm.: Territorio Rural Inteligente (TRI): proyecto de la junta de Castilla y León en coordinación con SATEC para la monitorización de parámetros de servicios públicos en municipios pequeños a lo largo de la comunidad. En este proyecto, debido a las grandes distancias que se han de recorrer se emplea tecnología LoRa para el envío de parámetros.

- **Bluetooth (BLE):** Ofrece bajo consumo energético, pero tiene un alcance corto y posibilidad de conexión con un número limitado de dispositivos. Ejm.: Pulseras de actividad. Utilizan Bluetooth para el envío de parámetros biométricos al móvil.
- **MQTT:** Requiere de un broker, e infraestructura externa, lo que le añade complejidad.
  Ejem.: Matternet, empresa dedicada a la intercomunicación entre drones.
- **ZigBee:** Protocolo de bajo consumo y alcance medio, pero con alto grado de complejidad.
  Ejm.: Bombillas inteligentes como Philips Hue. Utilizan tecnología ZigBee para la comunicación en malla entre las bombillas y entre las bombillas y el dispositivo móvil.
- **ESP-NOW**: No es un protocolo empleado generalmente por las empresas, sin embargo, presenta características interesantes que deben tenerse en cuenta: Bajo consumo, alcance medio, ausencia de necesidad de infraestructura de red.

Se priorizarán protocolos que no requieran de infraestructura externa, y que ofrezcan un alcance medio-alto. Los protocolos que cumplen estas condiciones son ESP-NOW y ZigBee, ambos cumplen características similares. Por ello la elección del protocolo dependerá principalmente de las características y accesibilidad de las placas que implementen estos protocolos. A continuación, se comparan los dispositivos más representativos que emplean ESP-NOW y ZigBee:

- **ESP-32** (Espressif): Soporta protocolos ESP-NOW, Bluetooth y WiFi. Presenta mayor consumo energético que los dispositivos EF32 y CC2530. Los entornos de programación compatibles incluyen Arduino, ESP-IDF y MicroPython, ofreciendo gran flexibilidad y facilidad de desarrollo.
- **EF32** (Silicon Labs): Compatible con ZigBee y BLE, tiene menor consumo energético que el ESP32 pero mayor que el CC2530. Su entorno de programación es Simplicity Studio, específico para esta plataforma.
- **CC2530** (Texas Instruments): Emplea exclusivamente ZigBee y presenta menor consumo energético que ESP32 y EF32. Se programa mediante Code Composer studio.

A pesar de requerir mayor consumo energético, los dispositivos ESP32 destacan por su versatilidad en protocolos de comunicación ya que, además de poder emplear el protocolo ESP-NOW, pueden utilizar Bluetooth y WiFi, cosa que el EF32, está limitado a ZigBee y BLE, mientras que el CC2530 solo soporta ZigBee.

El ecosistema de desarrollo para los dispositivos ESP32 también es más amplio y accesible, facilitando la programación y acelerando el proceso de desarrollo, a diferencia de los dispositivos EF32 y CC2530, que requieren de plataformas más específicas, necesitando conocimientos y herramientas particulares.

Por estas razones, se elige ESP32 como dispositivo donde desarrollar la librería y por lo tanto, se empleará ESP-NOW como protocolo de comunicación.

Cabe destacar que, a pesar de sus ventajas, ESP-NOW presenta una alta complejidad en su implementación, dificultando su adopción por desarrolladores como solución en proyectos de programación. Por ello, surge la necesidad de crear una librería que facilite la creación de una memoria compartida para la compartición de datos en memoria entre dispositivos ESP32.

Tradicionalmente, el concepto de memoria compartida implica una región común de memoria física accesible desde múltiples nodos. Sin embargo, cuando se trabaja con dispositivos físicamente separados, como son los ESP-32, la memoria compartida debe ser simulada a través de comunicación inalámbrica entre los nodos.

Actualmente, existen múltiples soluciones que permiten este intercambio de información, cada uno con sus ventajas e inconvenientes. Se conocen numerosos proyectos famosos de empresas conocidas que usan diferentes protocolos de comunicación para el desarrollo de productos:

- Google Docs: Permite crear, editar y borrar documentos simultáneamente entre diferentes usuarios desde la nube. Se trata de una memoria compartida simulada, ya que Google mantiene una memoria compartida sincronizada de cada documento.
- Dropbox: Permite sincronizar archivos entre diferentes dispositivos guardándolos en la nube. Funciona como una memoria compartida asíncrona.
- Apache Ignite: Plataforma de computación distribuida que permite compartir memoria física entre diferentes nodos. Se trata de una memoria compartida real.

A pesar de estas soluciones, la mayoría de las herramientas existentes están diseñadas para entornos con infraestructura de red como ordenadores y servidores. En el ámbito de los microcontroladores y en arquitecturas de bajo consumo como los ESP32, no se disponen de librerías estandarizadas, de fácil implementación que permitan la compartición directa de variables entre nodos sin requerir de infraestructura externa.

En este contexto, la comunidad de desarrolladores ha creado diferentes proyectos que permiten la intercomunicación y sincronización de datos entre

microcontroladores empleando protocolos como MQTT o LoRa. Sin embargo, no es común encontrar soluciones genéricas y reutilizables en forma de librerías que permitan simular una memoria compartida empleando ESP-NOW.

Por ello, aparece una oportunidad para el desarrollo de herramientas que faciliten la programación distribuida entre nodos ESP-32, reduciendo la complejidad de su manejo y permitiendo aprovechar las ventajas del protocolo ESP-NOW.

Vinculación con competencias del Título

Este proyecto se enmarca en el ámbito de la informática industrial, la programación de sistemas embebidos, la electrónica industrial, y las tecnologías de la comunicación.

Diseño

La librería pretende que los dispositivos puedan actuar como emisores y receptores de datos a la vez para conseguir una comunicación entre dispositivos fluida. Por ello, todos los dispositivos son capaces de leer y escribir datos en la memoria compartida, crear nuevas memorias compartidas definiéndolas a través de un nombre y un tamaño, además de ser capaces de registrar y borrar dispositivos de su lista de “peer”.

Especificaciones

1. La librería debe permitir el registro de dispositivos para asegurar la intercomunicación entre ambos. Esto se conseguirá mediante las direcciones MAC de los dispositivos ESP-32.

2. La librería debe permitir borrar el registro de un dispositivo empleando nuevamente la dirección MAC.

3. La librería debe permitir informar al usuario de los dispositivos que están registrados.

4. La librería debe permitir crear segmentos de memoria (Bloques de memoria) con un nombre y tamaño concretos.

5. La librería debe permitir vincular los bloques de memoria creados con los creados en otros dispositivos si coinciden en nombre y tamaño.

6. La librería debe permitir informar al usuario del contenido de cualquier posición de memoria dentro de un bloque de memoria creado.

7. La librería debe permitir escribir en las direcciones de memoria dentro de los bloques de memoria creados.

8. La librería debe permitir dar acceso a ciertas posiciones de memoria dentro de un bloque de memoria creado y vinculado con otro dispositivo. De forma que, para un bloque de memoria vinculado, cada dispositivo solo podrá escribir en las posiciones de memoria a las que se tenga acceso.

9. La librería debe permitir asociar las direcciones MAC del dispositivo local y de los registrados a un nombre concreto definido por el usuario para usarlo en aquellas funciones donde la dirección MAC sea necesaria con el fin de simplificar el uso de estas funciones (Ejm: función Borrar).

Código: Archivo MemoriaCompartida.h

La definición de todas las funciones y variables globales se muestran en el siguiente archivo: MemoriaCompartida.h:

```
#ifndef MEMORIACOMPARTIDA_H
#define MEMORIACOMPARTIDA_H

#include <esp_now.h>
#include <WiFi.h>
#include <vector>
#include <string>
#include <stdlib.h>
#include <stdint.h>

typedef struct MensajeEstructurado{
  char text[240];
} Mensaje_Estructurado;

class MemoriaCompartida{
public:
  MemoriaCompartida();
  void init();
  void Registrar(const String& mac);
  void Borrar(String& Mac);
  void Dispositivos();
```

```
  static void EscribirMemoria(String NombreBloque, int Indice,
String Valor);
  static void EditarValorDeMemoria(String MacReceptor, String
NombreBloque, int Indice, int Valor);
  static void enRecepcionDeDatos(const esp_now_recv_info* info,
const uint8_t* DatosRecibidos, int tamaño);
  static void enEnvioDeDatos (const uint8_t* MAC,
esp_now_send_status_t status);
  static bool ConvertirMacStringABytes(const String& macString,
uint8_t* macBytes);
  static String NormalizarMac(const String& Mac);
  static uint32_t MostrarMemoria(String NombreBloque, int Indice);
  static std::vector<uint32_t*> MemoriaDispositivos;
  void Alias ();
  static bool esAlias(String Alias);
  static String ConvertirAliasAMacString(String Alias);
  void AsignarVectorMemoriaCompartida(String Mac, String
NombreBloque);
  static void CrearBloqueMemoria(String NombreBloque, uint32_t
TamanoBloque);
  static void VincularVector();
  static void EnviarValorVinculado(String NombreBloque, int Indice,
String Valor);
  static uint32_t InfoMostrarMemoria(String NombreBloque, int
Indice);
  static void CrearVectorValoresResponsabilizados(String
NombreBloque, int IndiceMin, int IndiceMax);

private:
  static MensajeEstructurado MensajeEnviado;
  static MensajeEstructurado MensajeRecibido;
  static uint8_t peerMAC[6];
  static bool RespuestaRecibida;
  static bool EnvioCompletado;
  static std::vector<String> DispositivosConectados;
  static bool esDireccionMac(const String& mac);
  static std::vector<String> DispositivosAlias;
  static std::vector<String> Bloques;
  static std::vector<String> VectoresVinculados;
  static std::vector<String> ValoresResponsabilizados;
  static std::vector<String> ListaRespuestasPendientes;
  static std::vector<int> Respuestas;
  static bool SePuedeEscribir;
};

#endif
```

Código: Archivo MemoriaCompartida.cpp

El archivo .cpp contiene variables globales y funciones que siguen dos tipos de tecnología distinta: funciones “blocking” y funciones “non-blocking”. Las funciones “blocking” son aquellas que detienen la ejecución del programa hasta que se completa la operación solicitada. Este tipo de funciones son más simples de implementar, pero pueden ocasionar problemas de rendimiento ya que el programa debe esperar a ejecutar por completo una tarea para comenzar con la siguiente.

Por el contrario, las funciones “non-blocking” permiten ejecutar más de una tarea a la vez al no detener la ejecución del programa, consiguiendo una ejecución continua del mismo sin interrupciones.

Funciones Blocking:

Entre las funciones “blocking” creadas, se encuentran funciones de registro y borrado de dispositivos, funciones de lectura y escritura de datos, funciones de creación y vinculación de bloques de memoria y otras funciones de carácter secundario.

1. MemoriaCompartida::esDirecciónMac()

```
1. bool MemoriaCompartida::esDireccionMac(const String& mac) {
2.   if (mac.length() != 17) {
3.     return false;
4.   }
5.   for (int i = 0; i < mac.length(); i++) {
6.     if (i % 3 == 2) {
7.       if (mac[i] != ':') {
8.         return false;
9.       }
10.    } else if (!isxdigit(mac[i])) {
11.      return false;
12.    }
13.  }
14.  return true;
15.}
```

La función esDireccionMac permite validar si una cadena de texto (String mac) tiene el formato adecuado para ser una dirección MAC y poder emplear esa cadena de texto en otras funciones.

La función primeramente comprueba si la longitud de la cadena de texto introducida es de 17 caracteres: 6 caracteres hexadecimales (12 caracteres en total) y 5 dos puntos que sirven para separar los pares de caracteres. Si la longitud no es de 17, la función devuelve “false”.

```
2.    if (mac.length() != 17) {
3.      return false;
4.    }
```

A continuación, se recorre la cadena de texto usando un bucle “for”,

Por un lado, se comprueba que cada tercer carácter es un signo de puntuación “:” a través de dos condiciones “if”. Si no se cumple, se devuelve un “false”.

Por otro lado, se comprueba que cada primer y segundo carácter es un hexadecimal usando la función “isxdigit()”. Y si no lo es, se devuelve un “false”.

```
4.  for (int i = 0; i < mac.length(); i++) {
5.      if (i % 3 == 2) {
6.        if (mac[i] != ':') {
7.          return false;
8.        }
9.      } else if (!isxdigit(mac[i])) {
10.       return false;
11.     }
12.   }
```

Por último, si ninguna de las condiciones anteriores devuelve un “false”, la función devolverá un “true”: la cadena de texto tiene el formato de dirección MAC correcto, y la variable se podrá usar en otras funciones de la librería.

```
14.   return true;
```

Esta función es una implementación simple pero efectiva y fundamental a la hora de validar las direcciones MAC, asegura una correcta identificación de los dispositivos para la utilización en funciones y operaciones que requieran de direcciones MAC.

2. MemoriaCompartida::ConvertirMacStringABytes()

```
bool MemoriaCompartida::ConvertirMacStringABytes(const String&
macString, uint8_t* macBytes) {
```

```
  int elementos[6];
  if (sscanf(macString.c_str(), "%x:%x:%x:%x:%x:%x",
  &elementos[0], &elementos[1], &elementos[2], &elementos[3],
  &elementos[4], &elementos[5]) == 6) {
    for (int i = 0; i < 6; i++) {
      macBytes[i] = (uint8_t)elementos[i];
    }
    return true;
  }
  return false;
}
```

La función ConvertirMacStringABytes permite convertir una cadena de texto (String macString) en un vector de bytes (uint8_t macBytes) para su utilización en comunicaciones entre dispositivos a través de protocolos de red.

Primero, se utiliza la función “sscanf()” para extraer los seis caracteres hexadecimales contenidos en “macString” e introducirlos en el vector de enteros “elementos” de tamaño seis previamente creado. Cada carácter hexadecimal se introduce en cada posición del vector “elementos”. A través de la condición “if”, si se devuelve un vector con seis elementos, se convertirá cada elemento del vector en un “uint8_t” y se almacenará en el vector “macBytes” a través de un “for” y se devuelve un “true”. Si la cadena no tiene seis elementos se devuelve un “false”.

La función “ConvertirMacStringABytes” es simple y directa. Gracias a la función “sscanf”, que permite simplificar el código mucho. Además, es imprescindible en funciones donde se pretenda comunicar dispositivos entre sí.

3. MemoriaCompartida::Registrar()

```
void MemoriaCompartida::Registrar(const String& mac) {
  if (esDireccionMac(mac)) {
    if (!ConvertirMacStringABytes(mac, peerMAC)) {
      Serial.println("Error al convertir la Mac.");
      return;
    }
    uint8_t MacBytesLocal[6];
    WiFi.macAddress(MacBytesLocal);
    bool MismaMac = true;
    for (int i = 0; i < 6; i++) {
      if (peerMAC[i] != MacBytesLocal[i]) {
        MismaMac = false;
```

```
13.         break;
14.       }
15.     }
16.     if (MismaMac) {
17.       Serial.println("Error: La MAC introducida es la de este
    dispositivo.");
18.       return;
19.     }
20.     esp_now_peer_info_t InfoPeer = {};
21.     memcpy(InfoPeer.peer_addr, peerMAC, 6);
22.     InfoPeer.channel = 0;
23.     InfoPeer.encrypt = false;
24.     if (esp_now_is_peer_exist(peerMAC)) {
25.       Serial.println("Error: Este dispositivo ya está
    registrado.");
26.       return;
27.     } else {
28.       esp_err_t Resultado = esp_now_add_peer(&InfoPeer);
29.       if (Resultado != ESP_OK) {
30.         Serial.println("Error al agregar peer temporalmente");
31.         return;
32.       }
33.     }
34.     strcpy(MensajeEnviado.text, "Prueba de conexión");
35.     Serial.print("Iniciando conexión con: ");
36.     Serial.println(mac);
37.     esp_err_t ResultadoDeEnvio = esp_now_send(peerMAC,
    (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
38.     if (ResultadoDeEnvio == ESP_OK) {
39.       Serial.println("Conectando...");
40.       RespuestaRecibida = false;
41.       unsigned long Contador = millis();
42.       while (millis() - Contador < 20000) {
43.         if (RespuestaRecibida == true) {
44.           esp_now_add_peer(&InfoPeer);
45.           break;
46.         }
47.       }
48.       if (!RespuestaRecibida) {
49.         Serial.println("No se pudo conectar. Eliminando
    dispositivo.");
50.         esp_now_del_peer(peerMAC);
51.       }
52.     } else {
53.       Serial.print("Error al conectar. Código de error: ");
54.       Serial.println(ResultadoDeEnvio);
55.       esp_now_del_peer(peerMAC);
```

```
    }
  } else {
    Serial.println("Dirección MAC no válida.");
  }
}
```

La función “Registrar()” permite agregar dispositivos a la red utilizando el protocolo ESP-NOW y asegura que el dispositivo con la dirección MAC que se pretende registrar está en alcance suficiente para el intercambio de información.

Esta función realizará diferentes procesos de verificación:

1. Se verifica que la cadena de texto (String mac) contiene una dirección MAC válida empleando la función “esDirecciónMac()” a través de una condición “if”.
2. Dentro de la condición anterior se verifica si se ha podido convertir la cadena de texto (mac) a bytes guardados en la variable “peerMac” a través de la función “ConvertirMacStringABytes()”.

```
  if (esDireccionMac(mac)) {
    if (!ConvertirMacStringABytes(mac, peerMAC)) {
      Serial.println("Error al convertir la Mac.");
      return;
    }
```

3. Se realiza una tercera comprobación: Que la cadena de texto (mac) no contenga la dirección MAC del propio dispositivo y se intente registrar un dispositivo que es él mismo. Para ello se crea una variable “uint8_t MacBytesLocal[6]” donde se guarda el contenido de la dirección de memoria del propio dispositivo a través de la función de ESP-NOW “WiFi.macAddress()”. Se declara una variable booleana “MismaMac” y se inicializa en true. A través de un “for” se comprueba que cada elemento de “peerMac” sea distinto a “MacBytesLocal”. Si es así, la variable booleana se pone en “false” y se sale del bucle. En una nueva condición se comprueba que si “MismaMac” es “true” salte un error por pantalla.

```
    uint8_t MacBytesLocal[6];
    WiFi.macAddress(MacBytesLocal);
    bool MismaMac = true;
    for (int i = 0; i < 6; i++) {
      if (peerMAC[i] != MacBytesLocal[i]) {
        MismaMac = false;
        break;
      }
```

```
15.     }
16.     if (MismaMac) {
17.       Serial.println("Error: La MAC introducida es la de
   este dispositivo.");
18.       return;
19.     }
```

4. Se realiza una cuarta comprobación: La cadena de texto (mac) no debe contener la dirección MAC de un dispositivo ya registrado. Usando la librería ESP-NOW se prepara un objeto "esp_now_peer_info_t" que guarda la información del dispositivo que se quiere registrar en una variable "InfoPeer". A través de una condición "if" y la función "esp_now_is_peer_exist()" de ESP-NOW se verifica si la MAC contenida en el vector "peerMAC" ya está registrada en la red y si es así saca un mensaje de error por pantalla.

```
20.esp_now_peer_info_t InfoPeer = {};
21.     memcpy(InfoPeer.peer_addr, peerMAC, 6);
22.     InfoPeer.channel = 0;
23.     InfoPeer.encrypt = false;
24.     if (esp_now_is_peer_exist(peerMAC)) {
25.       Serial.println("Error: Este dispositivo ya está
   registrado.");
26.       return;
27.     }
```

5. Como última comprobación se verifica si el dispositivo se ha agregado correctamente. Con el objeto "esp_err_t" de ESP-NOW que contiene información sobre un posible error guardada en la variable "Resultado", si el "Resultado" no contiene "ESP_OK" significa que no se ha podido agregar el dispositivo y salta un error.

```
27.else {
28.       esp_err_t Resultado = esp_now_add_peer(&InfoPeer);
29.       if (Resultado != ESP_OK) {
30.         Serial.println("Error al agregar peer
   temporalmente");
31.         return;
32.       }
33.     }
```

Una vez hechas todas las comprobaciones, se atiende al proceso específico de registro de un dispositivo. Primero se envía el mensaje "Prueba de conexión" al dispositivo con dirección MAC almacenada en "peerMAC" utilizando "esp_now_send()" de ESP-NOW.

```
34.     strcpy(MensajeEnviado.text, "Prueba de conexión");
35.     Serial.print("Iniciando conexión con: ");
36.     Serial.println(mac);
37.     esp_err_t ResultadoDeEnvio = esp_now_send(peerMAC,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
```

Una vez enviado el mensaje se esperará veinte segundos a recibir un mensaje de vuelta para verificar si la conexión fue exitosa y el dispositivo está en alcance. Si se recibe el mensaje dentro del tiempo establecido, el dispositivo queda registrado y se mantiene en la red, pero si no se recibe mensaje, se elimina el peer.

```
38.     if (ResultadoDeEnvio == ESP_OK) {
39.       Serial.println("Conectando...");
40.       RespuestaRecibida = false;
41.       unsigned long Contador = millis();
42.       while (millis() - Contador < 20000) {
43.         if (RespuestaRecibida == true) {
44.           esp_now_add_peer(&InfoPeer);
45.           break;
46.         }
47.       }
48.       if (!RespuestaRecibida) {
49.         Serial.println("No se pudo conectar. Eliminando
   dispositivo.");
50.         esp_now_del_peer(peerMAC);
51.       }
52.     }
```

Por último, se saca por pantalla un error si no se recibe respuesta y se elimina el peer y si el contenido de la MAC (mac) introducido no es una dirección MAC válida.

```
52.     else {
53.       Serial.print("Error al conectar. Código de error: ");
54.       Serial.println(ResultadoDeEnvio);
55.       delay(10);
56.       esp_now_del_peer(peerMAC);
57.     } else {
58.     Serial.println("Dirección MAC no válida.");
59.   }
60.}
```

4. MemoriaCompartida::NormalizarMac()

```
String MemoriaCompartida::NormalizarMac(const String& Mac) {
  String MacNormalizada = Mac;
  MacNormalizada.toUpperCase();
  return MacNormalizada;
}
```

La función “NormalizarMac()” es una función muy sencilla que asegura que todos los caracteres estén en mayúsculas. Es una función muy útil a la hora de tener que comparar direcciones MAC a lo largo del código.

5. MemoriaCompartida::Dispositivos()

```
void MemoriaCompartida::Dispositivos() {
  if (DispositivosConectados.empty()) {
    Serial.println("No hay dispositivos conectados.");
    Serial.println("Dispositivo local: ");
    Serial.println(WiFi.macAddress());
    delay(100);
  } else {
    Serial.println("Dispositivo local:");
    Serial.println(WiFi.macAddress());
    Serial.println("Dispositivos conectados: ");
    for (size_t i = 0; i < DispositivosConectados.size(); ++i) {
      Serial.println(NormalizarMac(DispositivosConectados[i]));
    }
  }
  Serial.println();
}
```

La función “Dispositivos()” permite informar sobre la cantidad de dispositivos registrados en una red.

A través de una condición “if” se comprueba si hay dispositivos conectados:

- Si no es así, muestra la dirección MAC del dispositivo local.

- Si sí es así, muestra tanto la dirección MAC del dispositivo local como las de los dispositivos conectados.

6. MemoriaCompartida::Borrar()

```
void MemoriaCompartida::Borrar(String& Mac) {
  bool Registrado = false;
  if (!esDireccionMac(Mac)) {
    if (!esAlias(Mac)){
```

```
5.        Serial.println("Error: Formato de MAC inválido.");
6.        return;
7.      }
8.      if(esAlias(Mac)){
9.        ConvertirAliasAMacString(Mac);
10.       String mac = ConvertirAliasAMacString(Mac);
11.       Mac = mac;
12.     }
13.   }
14.   if (NormalizarMac(Mac) == NormalizarMac(WiFi.macAddress())) {
15.     Serial.println("Error: La MAC introducida es la de este
    dispositivo.");
16.     return;
17.   }
18.   if (!ConvertirMacStringABytes(Mac, peerMAC)) {
19.     Serial.println("Error al convertir la MAC");
20.     return;
21.   }
22.   for (auto it = DispositivosConectados.begin(); it !=
    DispositivosConectados.end(); ++it) {
23.     if (NormalizarMac(*it) == NormalizarMac(Mac)) {
24.       char Texto[30];
25.       sprintf(Texto, "Borrado %s", WiFi.macAddress().c_str());
26.       strcpy(MensajeEnviado.text, Texto);
27.       esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
    sizeof(MensajeEnviado));
28.       DispositivosConectados.erase(it);
29.       esp_now_del_peer(peerMAC);
30.       Serial.print("El dispositivo ");
31.       Serial.print(Mac);
32.       Serial.println(" ha sido elimindado.");
33.       Registrado = true;
34.       for (size_t i = 0; i < VectoresVinculados.size(); i++) {
35.         if(NormalizarMac(VectoresVinculados[i]) ==
    NormalizarMac(*it)) {
36.           VectoresVinculados.erase(VectoresVinculados.begin() + (i
    - 1));
37.           VectoresVinculados.erase(VectoresVinculados.begin() + (i
    - 1));
38.           i -= 1;
39.         }
40.       }
41.       break;
42.     }
43.   }
44.   for (int i = 0; i < DispositivosAlias.size(); i += 2) {
```

```
45.     if (NormalizarMac(DispositivosAlias[i]) ==
   NormalizarMac(Mac)) {
46.       DispositivosAlias.erase(DispositivosAlias.begin() + i);
47.       DispositivosAlias.erase(DispositivosAlias.begin() + i);
48.       Registrado = true;
49.       break;
50.     }
51.   }
52.   delay(100);
53.   esp_now_del_peer(peerMAC);
54.   Registrado = true;
55.   if (!Registrado) {
56.     Serial.println("Error: Dispositivo no registrado.");
57.   }
58.}
```

La función “Borrar()” permite eliminar un dispositivo de la lista de “peers” de la red, de la lista de dispositivos conectados y de la lista de vectores vinculados. Busca una dirección MAC determinada y la elimina si se encuentra entre los dispositivos conectados.

Primero se crea una variable booleana (Registrado) que servirá, al final del código, para avisar al usuario si la cadena de texto “Mac” no contuviese una dirección MAC registrada en el dispositivo.

```
2.    bool Registrado = false;
```

Esta función realiza una serie de comprobaciones:

1. Se verifica que la cadena de texto (String Mac) contiene una dirección MAC válida empleando la función “esDirecciónMac()” a través de una condición “if”. Si no es una dirección MAC válida, se realiza otra comprobación más: Se verifica si el contenido de la cadena de texto “Mac” es un alias a través de la función “esAlias()”.
   + Si no es así, saca por pantalla un error.
   + Si sí es así se ejecuta la función “ConvertirAliasAMacString()” para obtener la dirección MAC vinculada a ese alias.

```
3.  if (!esDireccionMac(Mac)) {
4.      if (!esAlias(Mac)){
5.        Serial.println("Error: Formato de MAC inválido.");
6.        return;
7.      }
8.      if(esAlias(Mac)){
9.        ConvertirAliasAMacString(Mac);
10.       String mac = ConvertirAliasAMacString(Mac);
```

```
11.       Mac = mac;
12.     }
13.   }
```

2. Si el contenido de la cadena de texto “Mac” sí es una dirección Mac se realiza la siguiente verificación: Con ayuda de la función “NormalizarMac()” se comprueba si la cadena de texto contiene la dirección del propio dispositivo, si es así, devuelve un mensaje de error.

```
14.if (NormalizarMac(Mac) == NormalizarMac(WiFi.macAddress()))
   {
15.     Serial.println("Error: La MAC introducida es la de este
   dispositivo.");
16.     return;
17.   }
```

3. Si no es la MAC del propio dispositivo, realiza la siguiente comprobación: Con “ConvertirMacStringABytes()” se verifica si se ha podido convertir la cadena de texto (Mac) a bytes guardados en la variable “peerMac”.

```
18.if (!ConvertirMacStringABytes(Mac, peerMAC)) {
19.     Serial.println("Error al convertir la MAC");
20.     return;
21.   }
```

Una vez completadas todas las comprobaciones para asegurar el correcto funcionamiento del flujo se procede con la eliminación del dispositivo. Mediante un “for” se recorren todos los dispositivos almacenados en la lista “DispositivosConectados”, cuando un elemento del vector coincide con la dirección MAC almacenada en la cadena de texto, se prepara el mensaje “Borrado” y se envía al dispositivo con dicha MAC, se elimina el dispositivo del vector de “DispositivosConectados()” y se imprime un mensaje informando al usuario. Mediante otro “for” se recorre el vector “VectoresVinculados” y cuando el contenido de ese vector sea igual a la dirección MAC que se quiere borrar, se elimina tanto esa dirección MAC como el vector vinculado a esa MAC de vector “VectoresVinculados”.

```
22.   for (auto it = DispositivosConectados.begin(); it !=
   DispositivosConectados.end(); ++it) {
23.   if (NormalizarMac(*it) == NormalizarMac(Mac)) {
24.     char Texto[30];
25.     sprintf(Texto, "Borrado %s", WiFi.macAddress().c_str());
26.     strcpy(MensajeEnviado.text, Texto);
27.     esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
28.     DispositivosConectados.erase(it);
29.     esp_now_del_peer(peerMAC);
```

```
30.     Serial.print("El dispositivo ");
31.     Serial.print(Mac);
32.     Serial.println(" ha sido elimindado.");
33.     Registrado = true;
34.       for (size_t i = 0; i < VectoresVinculados.size(); i++) {
35.         if(NormalizarMac(VectoresVinculados[i]) ==
   NormalizarMac(*it)) {
36.         VectoresVinculados.erase(VectoresVinculados.begin() + (i
   - 1));
37.         VectoresVinculados.erase(VectoresVinculados.begin() + (i
   - 1));
38.           i -= 1;
39.         }
40.       }
41.     break;
42.     }
43.   }
```

Luego se recorre el vector “DispositivosAlias” para, una vez borrada la MAC del peer y del listado “DispositivosConectados”, borrar del vector “DispositivosAlias” tanto la dirección MAC con el alias asociado a esa MAC.

```
44.   for (int i = 0; i < DispositivosAlias.size(); i += 2) {
45.     if (NormalizarMac(DispositivosAlias[i]) ==
   NormalizarMac(Mac)) {
46.       DispositivosAlias.erase(DispositivosAlias.begin() + i);
47.       DispositivosAlias.erase(DispositivosAlias.begin() + i);
48.       Registrado = true;
49.       break;
50.     }
51.   }
```

Por último, con un “delay” de 0,1 segundos, se ayuda al sistema a estabilizarse y elimina la MAC introducida si no ha sido eliminada a través del flujo anterior, es decir, si se hubiese introducido una dirección MAC de un dispositivo no registrado. Además, si la variable booleana creada anteriormente devuelve un “false” se saca por pantalla un mensaje de error.

```
46.   delay(100);
47.   esp_now_del_peer(peerMAC);
48.   Registrado = true;
49.   if (!Registrado) {
50.   Serial.println("Error: Dispositivo no registrado.");
51.   }
52.}
```

La función “Borrar” sirve para validar, buscar y eliminar cualquier referencia (dirección MAC o/y Alias) de cualquier dispositivo que tenga el dispositivo que lo ejecuta. Además, es capaz de detectar errores e informar de ellos al usuario.

7. MemoriaCompartida::CrearBloqueMemoria()

```
1. void MemoriaCompartida::CrearBloqueMemoria(String NombreBloque,
   uint32_t TamanoBloque) {
2.   if (NombreBloque.length() == 0) {
3.     Serial.println("El nombre del bloque no puede estar vacío.");
4.     return;
5.   }
6.   for (size_t i = 0; i < Bloques.size(); i += 2) {
7.     if (Bloques[i] == NombreBloque) {
8.       Serial.println("Error: El nombre del bloque ya existe.");
9.       return;
10.    }
11.  }
12.  if (TamanoBloque == 0){
13.    Serial.println("Error: El tamaño deber ser un número y mayor
   a 0");
14.    return;
15.  }
16.  uint32_t EspacioGastado = 0;
17.  for (size_t i = 0; i < Bloques.size(); i += 2){
18.    EspacioGastado += (Bloques[i + 1].toInt()) * 4;
19.  }
20.  uint32_t MemoriaLibre = ESP.getFreeHeap();
21.  MemoriaLibre -= EspacioGastado;
22.  if (TamanoBloque * 4 > MemoriaLibre) {
23.    Serial.print("Error: No hay suficiente memoria disponible
   para este bloque. Espacio libre disponible: ");
24.    Serial.println(MemoriaLibre/4);
25.    return;
26.  }
27.  Bloques.push_back(NombreBloque);
28.  Bloques.push_back(String(TamanoBloque));
29.  Serial.print("Bloque de memoria '");
30.  Serial.print(NombreBloque);
31.  Serial.print("' creado con tamaño de ");
32.  Serial.print(TamanoBloque);
33.  Serial.println(" direcciones de memoria.");
34.  uint32_t* DireccionMemoria = (uint32_t*)malloc(TamanoBloque *
   sizeof(uint32_t));
```

```
35.  if (DireccionMemoria != NULL) {
36.    MemoriaDispositivos.push_back(DireccionMemoria);
37.    for (size_t i = 0; i < TamanoBloque; i++) {
38.      DireccionMemoria[i] = 0;
39.    }
40.    VincularVector();
41.  } else {
42.    Serial.println("Error al reservar memoria para el bloque.");
43.  }
44.}
```

La función “CrearBloqueMemoria()” permite crear bloques de memoria con nombre y tamaño variable para su posterior asignación a dispositivos determinados.
Mediante una condición “if” se asegura de que el nombre introducido no esté vacío:

```
2.   if (NombreBloque.length() == 0) {
3.     Serial.println("El nombre del bloque no puede estar vacío.");
4.     return;
5.   }
```

Luego se asegura de que el nombre introducido no esté repetido:

```
6. for (size_t i = 0; i < Bloques.size(); i += 2) {
7.     if (Bloques[i] == NombreBloque) {
8.       Serial.println("Error: El nombre del bloque ya existe.");
9.       return;
10.    }
11.  }
```

Y también asegura que el tamaño introducido sea válido: un número mayor a 0.

```
12.if (TamanoBloque == 0){
13.    Serial.println("Error: El tamaño deber ser un número y mayor
   a 0");
14.    return;
15.  }
```

A continuación, se calcula el espacio disponible para avisar al usuario de si hay espacio suficiente para crear dicho bloque:

```
16.uint32_t EspacioGastado = 0;
17.  for (size_t i = 0; i < Bloques.size(); i += 2){
```

```
    EspacioGastado += (Bloques[i + 1].toInt()) * 4;
}
    uint32_t MemoriaLibre = ESP.getFreeHeap();
    MemoriaLibre -= EspacioGastado;
    if (TamanoBloque * 4 > MemoriaLibre) {
      Serial.print("Error: No hay suficiente memoria disponible para este bloque. Espacio libre disponible: ");
      Serial.println(MemoriaLibre/4);
      return;
    }
```

Una vez creado el bloque, se estructura el vector “Bloques” a pares, de forma que la información de cada bloque se almacena cada dos elementos: en los impares (primera posición, tercera…) se almacena el nombre del bloque. En las posiciones pares (segunda, cuarta…) se almacena el tamaño del bloque, de forma que el vector estará formado por lo nombres de los bloques creados seguidos de su respectivo tamaño.

```
    Bloques.push_back(NombreBloque);
    Bloques.push_back(String(TamanoBloque));
    Serial.print("Bloque de memoria '");
    Serial.print(NombreBloque);
    Serial.print("' creado con tamaño de ");
    Serial.print(TamanoBloque);
    Serial.println(" direcciones de memoria.");
```

Por último, con la función “malloc” se reserva ese espacio de memoria a ese bloque concreto.

```
uint32_t* DireccionMemoria = (uint32_t*)malloc(TamanoBloque * sizeof(uint32_t));
    if (DireccionMemoria != NULL) {
      MemoriaDispositivos.push_back(DireccionMemoria);
      for (size_t i = 0; i < TamanoBloque; i++) {
        DireccionMemoria[i] = 0;
      }
      VincularVector();
    } else {
      Serial.println("Error al reservar memoria para el bloque.");
    }
  }
```

8. MemoriaCompartida::VincularVector()

```
void MemoriaCompartida::VincularVector() {
  if (DispositivosConectados.size() == 0) {
```

```
3.      Serial.println("No hay dispositivos conectados para
   vincular.");
4.      return;
5.    }
6.    if (Bloques.size() == 0) {
7.      Serial.println("No hay bloques creados para vincular.");
8.      return;
9.    }
10.   for (size_t i = 1; i < Bloques.size(); i += 2) {
11.     std::vector<int> IndiceDispositivo;
12.     for (size_t j = 0; j < VectoresVinculados.size(); j++) {
13.       if (Bloques[i - 1] == VectoresVinculados[j]) {
14.         IndiceDispositivo.push_back(j + 1);
15.       }
16.     }
17.     for (size_t j = 0; j < DispositivosConectados.size(); j++) {
18.       bool yaVinculado = false;
19.       for (size_t k = 0; k < IndiceDispositivo.size(); k++) {
20.         if (DispositivosConectados[j] ==
   VectoresVinculados[IndiceDispositivo[k]]) {
21.           yaVinculado = true;
22.           Serial.print("El dispositivo ");
23.           Serial.print(NormalizarMac(DispositivosConectados[j]));
24.           Serial.print(" ya tiene vinculado el bloque ");
25.           Serial.println(Bloques[i - 1]);
26.           break;
27.         }
28.       }
29.       if (!yaVinculado) {
30.         char Texto[100];
31.         sprintf(Texto, "Vincular %s %lu", Bloques[i - 1].c_str(),
   Bloques[i].toInt());
32.         strcpy(MensajeEnviado.text, Texto);
33.         ConvertirMacStringABytes(DispositivosConectados[j],
   peerMAC);
34.         esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));

35.         Serial.print("El vector: ");
36.         Serial.print(Bloques[i - 1]);
37.         Serial.print(" se está intentando vincular con ");
38.         Serial.println(NormalizarMac(DispositivosConectados[j]));
39.       }
40.     }
41.     IndiceDispositivo.clear();
42.   }
43. }
```

La función VincularVector() permite sincronizar el contenido de un bloque de memoria creado con otros dispositivos registrados siempre que estos tengan un bloque creado con el mismo nombre y tamaño.
Primero se verifica que haya dispositivos registrados para poder vincular vectores, en caso contrario informa al usuario:

```
2. if (DispositivosConectados.size() == 0) {
3.     Serial.println("No hay dispositivos conectados para
   vincular.");
4.     return;
5.   }
```

A continuación, se verifica que haya bloques creados para poder vincular con otro dispositivo:

```
6. f (Bloques.size() == 0) {
7.     Serial.println("No hay bloques creados para vincular.");
8.     return;
9.   }
```

Una vez hechas las verificaciones, se recorre el vector “Bloques” y “VectoresVinculados” y se comprueba que el nombre del bloque sea del mismo que el bloque vinculado. En ese caso se almacena el índice del vector “VectoresVinculados” en el vector previamente creado “IndiceDispositivo” para su utilización posteriormente.

```
10.for (size_t i = 1; i < Bloques.size(); i += 2) {
11.    std::vector<int> IndiceDispositivo;
12.    for (size_t j = 0; j < VectoresVinculados.size(); j++) {
13.      if (Bloques[i - 1] == VectoresVinculados[j]) {
14.        IndiceDispositivo.push_back(j + 1);
15.      }
16.    }
```

Ahora se recorre el vector “DispositivosConectados” y el vector “IndiceDispositivo” y en caso de que un dispositivo registrado sea uno de los almacenados en el vector “VectoresVinculados”, se tomará el bloque como ya vinculado. En caso de que el bloque no haya sido vinculado con ningún dispositivo, se prepara el mensaje para iniciar el flujo de vinculación y se informa de ello al usuario.

```
17.for (size_t j = 0; j < DispositivosConectados.size(); j++) {
18.      bool yaVinculado = false;
19.      for (size_t k = 0; k < IndiceDispositivo.size(); k++) {
```

```
20.         if (DispositivosConectados[j] ==
    VectoresVinculados[IndiceDispositivo[k]]) {
21.           yaVinculado = true;
22.           Serial.print("El dispositivo ");
23.           Serial.print(NormalizarMac(DispositivosConectados[j]));
24.           Serial.print(" ya tiene vinculado el bloque ");
25.           Serial.println(Bloques[i - 1]);
26.           break;
27.         }
28.       }
29.       if (!yaVinculado) {
30.         char Texto[100];
31.         sprintf(Texto, "Vincular %s %lu", Bloques[i - 1].c_str(),
    Bloques[i].toInt());
32.         strcpy(MensajeEnviado.text, Texto);
33.         ConvertirMacStringABytes(DispositivosConectados[j],
    peerMAC);
34.         esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
    sizeof(MensajeEnviado));

35.         Serial.print("El vector: ");
36.         Serial.print(Bloques[i - 1]);
37.         Serial.print(" se está intentando vincular con ");
38.         Serial.println(NormalizarMac(DispositivosConectados[j]));
39.       }
40.     }
```

Una vez completado el mensaje de envío, se limpia el vector “IndiceDispositivo” mediante la función “clear()”, para su reutilización.

```
41.     IndiceDispositivo.clear();
42.   }
43. }
```

9. MemoriaCompartida::CrearVectorValoresResponsabilizados()

```
1.  void
    MemoriaCompartida::CrearVectorValoresResponsabilizados(String
    NombreBloque, int IndiceMin, int IndiceMax) {
2.    bool BloqueEncontrado = false;
3.    bool BloqueVinculado = false;
4.    uint32_t TamanoBloque;
5.    for (size_t i = 0; i < Bloques.size(); i += 2) {
6.      if (NombreBloque == Bloques [i]) {
7.        BloqueEncontrado = true;
8.        TamanoBloque = Bloques[i+1].toInt();
9.        break;
10.     }
```

```
  }
  if (!BloqueEncontrado) {
    Serial.print("Error: El vector ");
    Serial.print(NombreBloque);
    Serial.println(" no está creado en este dispositivo");
    return;
  } else {
    for (size_t i = 0; i < VectoresVinculados.size(); i++) {
      if (NombreBloque == VectoresVinculados[i]) {
        BloqueVinculado = true;
        break;
      }
    }
    if (!BloqueVinculado) {
      Serial.println("Error: Primero hay que vincular el vector.");
      return;
    }
  }
  if (IndiceMin > IndiceMax || IndiceMax > TamanoBloque) {
    Serial.println("Error: Indices mal definidos.");
    return;
  }
  for (size_t i = 0; i < VectoresVinculados.size(); i++) {
    if (NombreBloque == VectoresVinculados[i]) {
      ListaRespuestasPendientes.push_back(VectoresVinculados[i+1]);
      ListaRespuestasPendientes.push_back("0");
      char Texto[100];
      sprintf(Texto, "Responsable %s %d %d", NombreBloque.c_str(), IndiceMin, IndiceMax);
      strcpy(MensajeEnviado.text, Texto);
      ConvertirMacStringABytes(VectoresVinculados[i+1], peerMAC);
      esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
    }
  }
}
```

La función CrearVectorValoresResponsabilizados() permite enviar el mensaje “Responsable” para su procesamiento en otra función.

Primero se obtiene el tamaño del bloque recorriendo el vector “Bloques()” y guardando el tamaño en una variable. Con el mismo flujo se obtiene si el bloque ha sido encontrado en el vector “Bloques()” o no.

```
2.    bool BloqueEncontrado = false;
3.    bool BloqueVinculado = false;
4.    uint32_t TamanoBloque;
5.    for (size_t i = 0; i < Bloques.size(); i += 2) {
6.      if (NombreBloque == Bloques [i]) {
7.        BloqueEncontrado = true;
8.        TamanoBloque = Bloques[i+1].toInt();
9.        break;
10.     }
11.   }
```

Si el bloque no se encuentra, se imprime un mensaje de error.

```
12.if (!BloqueEncontrado) {
13.     Serial.print("Error: El vector ");
14.     Serial.print(NombreBloque);
15.     Serial.println(" no está creado en este dispositivo");
16.     return;
17.}
```

Si el bloque se encuentra, se averigua mediante un “for” si ese bloque está vinculado. Si no lo está se imprime un mensaje de error.

```
17.   else {
18.     for (size_t i = 0; i < VectoresVinculados.size(); i++) {
19.       if (NombreBloque == VectoresVinculados[i]) {
20.         BloqueVinculado = true;
21.         break;
22.       }
23.     }
24.     if (!BloqueVinculado) {
25.       Serial.println("Error: Primero hay que vincular el
    vector.");
26.       return;
27.     }
28.   }
```

Si el índice menor es mayor que el índice mayor o viceversa, se imprime otro mensaje de error.

```
29.   if (IndiceMin > IndiceMax || IndiceMax > TamanoBloque) {
30.     Serial.println("Error: Indices mal definidos.");
31.     return;
32.   }
```

Una vez hechas las comprobaciones, se recorre el vector “VectoresVinculados()” y cuando ese vector contenga el nombre del bloque, se

enviará el mensaje “Responsable” a la dirección MAC con ese nombre vinculado. El vector “VectoresVinculados” está estructurado a pares, siendo el primer elemento el bloque vinculado, y el segundo elemento la dirección MAC con la que ese bloque está vinculado. Además, se crean los vectores “ListaRespuestasPendientes()” que será de utilidad en la función que procese este mensaje: el “Callback” “EnRecepciónDeDatos()”.

```
33.  for (size_t i = 0; i < VectoresVinculados.size(); i++) {
34.    if (NombreBloque == VectoresVinculados[i]) {
35.
   ListaRespuestasPendientes.push_back(VectoresVinculados[i+1]);
36.      ListaRespuestasPendientes.push_back("0");
37.      char Texto[100];
38.      sprintf(Texto, "Responsable %s %d %d",
   NombreBloque.c_str(), IndiceMin, IndiceMax);
39.      strcpy(MensajeEnviado.text, Texto);
40.      ConvertirMacStringABytes(VectoresVinculados[i+1], peerMAC);
41.      esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
42.    }
43.  }
44.}
```

10. MemoriaCompartida::MostrarMemoria()

```
1. uint32_t MemoriaCompartida::MostrarMemoria(String NombreBloque,
   int Indice) {
2.   size_t index = -1;
3.   for (size_t i = 0; i < Bloques.size(); i += 2) {
4.     if (Bloques[i] == NombreBloque) {
5.       index = i/2;
6.     }
7.   }
8.   if (index == -1) {
9.     return -1;
10.  }
11.  uint32_t* DireccionMemoria = MemoriaDispositivos[index];
12.  if (DireccionMemoria != nullptr) {
13.    for (size_t i = 0; i < Bloques.size(); ++i) {
14.      if (Bloques[i] == NombreBloque) {
15.        uint32_t TamanoBloque = Bloques [i + 1].toInt();
16.        if (Indice >= 0 && Indice < TamanoBloque) {
17.          return DireccionMemoria[Indice];
18.        } else {
19.          return -1;
20.        }
21.      }
```

```
22.     }
23.   }
24. }
```

La función “MostrarMemoria()” permite mostrar el valor almacenado dentro de una posición específica de un bloque de memoria.

Primero se recorre el vector Bloques de dos en dos y se comprueba si el contenido de ese vector es el nombre del bloque almacenado en la variable “NombreBloque”. Si el contenido del vector es el nombre del bloque especificado, la variable “index” valdrá la posición en la que el vector contiene el nombre del bloque, dividido entre dos, esto es debido a que el “for” recorre el vector de dos en dos. En caso contrario la variable “index” devolverá un -1.

```
2.    size_t index = -1;
3.    for (size_t i = 0; i < Bloques.size(); i += 2) {
4.      if (Bloques[i] == NombreBloque) {
5.        index = i/2;
6.      }
7.    }
8.    if (index == -1) {
9.      return -1;
10.   }
```

A continuación, se obtiene un puntero a la memoria del bloque usando el índice. Se recorre el vector Bloques y se comprueba si el vector contiene el nombre del bloque concreto para obtener el tamaño del bloque y poder comprobar si el índice introducido es mayor o no al tamaño del bloque definido en la función CrearBloqueMemoria().

```
11.   uint32_t* DireccionMemoria = MemoriaDispositivos[index];
12.   if (DireccionMemoria != nullptr) {
13.     for (size_t i = 0; i < Bloques.size(); ++i) {
14.       if (Bloques[i] == NombreBloque) {
15.         uint32_t TamanoBloque = Bloques [i + 1].toInt();
16.         if (Indice >= 0 && Indice < TamanoBloque) {
17.           return DireccionMemoria[Indice];
18.         } else {
19.           return -1;
20.         }
21.       }
22.     }
23.   }
24. }
```

11. MemoriaCompartida::InfoMostrarMemoria();

```
1. uint32_t MemoriaCompartida::InfoMostrarMemoria(String
   NombreBloque, int Indice) {
2.   size_t index = -1;
3.   for (size_t i = 0; i < Bloques.size(); i += 2) {
4.     if (Bloques[i] == NombreBloque) {
5.       index = i/2;
6.     }
7.   }
8.   if (index == -1) {
9.     Serial.println ("Error: El bloque de memoria no existe");
10.    return -1;
11.  }
12.  uint32_t* DireccionMemoria = MemoriaDispositivos[index];
13.  if (DireccionMemoria != nullptr) {
14.    for (size_t i = 0; i < Bloques.size(); ++i) {
15.      if (Bloques[i] == NombreBloque) {
16.        uint32_t TamanoBloque = Bloques [i + 1].toInt();
17.        if (Indice >= 0 && Indice < TamanoBloque) {
18.          Serial.print("Valor del bloque de memoria ");
19.          Serial.print(NombreBloque);
20.          Serial.print(" en la dirección de memoria reservada ");
21.          Serial.print(Indice);
22.          Serial.print(": ");
23.          Serial.println(DireccionMemoria[Indice]);
24.          return DireccionMemoria[Indice];
25.        } else {
26.          Serial.println("Error: Índice fuera de rango.");
27.          return -1;
28.        }
29.      }
30.    }
31.  }
32.}
```

La función “InfoMostrarMemoria()” es similar a la función “MostrarMemoria”, la única diferencia es que esta permite informar al usuario del valor obtenido, o de si existe algún error, mientras que la otra no lo hace. “MostrarMemoria()”se usa para realizar comparaciones internas para el correcto del funcionamiento del sistema y si sacase mensajes cada vez que se usa, saturaría el monitor serial de mensajes.

12. MemoriaCompartida::EscribirMemoria()

```
1. void MemoriaCompartida::EscribirMemoria(String Mac, int Indice,
   String Valor) {
```

```
2.    if (!esDireccionMac(Mac)) {
3.      if (!esAlias(Mac)){
4.        Serial.println("Error: Formato de MAC de bloque de memoria
    reservado inválido.");
5.        return;
6.      }
7.      if(esAlias(Mac)){
8.        ConvertirAliasAMacString(Mac);
9.        String mac = ConvertirAliasAMacString(Mac);
10.       Mac = mac;
11.     }
12.   }
13.   std::vector<String> Dispositivos = DispositivosConectados;
14.   Dispositivos.push_back(WiFi.macAddress());
15.   Mac.toUpperCase();
16.   size_t index = -1;
17.   for (size_t i = 0; i < Dispositivos.size(); ++i) {
18.     String MacLocal = Dispositivos[i];
19.     MacLocal.toUpperCase();
20.     if (MacLocal == Mac) {
21.       index = i;
22.       break;
23.     }
24.   }
25.   if (index == -1) {
26.     Serial.println("No se ha asignado memoria para este
    dispositivo.");
27.     return;
28.   }
29.   if (index < MemoriaDispositivos.size()) {
30.     uint32_t* DireccionMemoria = MemoriaDispositivos[index];
31.     uint32_t ValorBytes = Valor.toInt();
32.     if (ValorBytes == 0 && Valor != "0") {
33.       Serial.println("Valor no válido. Asegúrate de que el valor
    sea un número.");
34.       return;
35.     }
36.     uint32_t TamanoMaximo = (CalcularMemoria() + 59) /
    Dispositivos.size();
37.     if (Indice < 0 || Indice >= TamanoMaximo) {
38.       Serial.println("Índice fuera de rango.");
39.       return;
40.     }
41.     DireccionMemoria[Indice] = ValorBytes;

42.     Serial.print("Escrito el valor ");
43.     Serial.print(ValorBytes);
```

```
44.     Serial.print(" en la dirección de memoria ");
45.     Serial.print(Indice);
46.     Serial.print(" del bloque de memoria reservado a la dirección
   MAC ");
47.     Serial.println(Mac);
48.   }
49.}
```

La función “EscribirMemoria()” permite escribir un valor determinado dentro de una posición específica de un bloque de memoria asignado a un dispositivo.

Primero se verifica que la cadena de texto (String Mac) contiene una dirección MAC válida empleando la función “esDirecciónMac()” a través de una condición “if”. Si no es una dirección MAC válida, se realiza otra comprobación: Se verifica si el contenido de la cadena de texto “Mac” es un alias a través de la función “esAlias()”.

+ Si no es así, saca por pantalla un error.

+ Si sí es así se ejecuta la función “ConvertirAliasAMacString()” para obtener la dirección MAC vinculada a ese alias.

```
2. if (!esDireccionMac(Mac)) {
3.     if (!esAlias(Mac)){
4.       Serial.println("Error: Formato de MAC de bloque de memoria
   reservado inválido.");
5.       return;
6.     }
7.     if(esAlias(Mac)){
8.       ConvertirAliasAMacString(Mac);
9.       String mac = ConvertirAliasAMacString(Mac);
10.      Mac = mac;
11.    }
12.  }
```

A continuación, se crea el vector “Dispositivos”, donde se almacena la dirección MAC de los dispositivos conectados y del dispositivo local. Además, se crea la variable “index”, que sirve para avisar al usuario cuando la cadena de texto introducida (Mac) no es la dirección MAC de ningún dispositivo registrado. Se recorren todos los dispositivos almacenados en el vector “Dispositivos”, cuando la dirección MAC de uno de estos dispositivos registrados coincide con la dirección MAC de la variable “Mac”, la variable índice pasa de valer “-1” a valer la posición del vector “Dispositivos” donde está almacenada la MAC. Si la dirección MAC de “Mac” no se encuentra en el vector “Dispositivos” se informa

al usuario del error, pues se está intentando acceder al bloque de memoria asignado a un dispositivo que no tiene ningún bloque de memoria asignado.

```
13.std::vector<String> Dispositivos = DispositivosConectados;
14.  Dispositivos.push_back(WiFi.macAddress());
15.  Mac.toUpperCase();
16.  size_t index = -1;
17.  for (size_t i = 0; i < Dispositivos.size(); ++i) {
18.    String MacLocal = Dispositivos[i];
19.    MacLocal.toUpperCase();
20.    if (MacLocal == Mac) {
21.      index = i;
22.      break;
23.    }
24.  }
25.  if (index == -1) {
26.    Serial.println("No se ha asignado memoria para este
   dispositivo.");
27.    return;
28.  }
```

A través de la condición “if” se asegura de que el dispositivo al cual se intenta escribir en memoria tenga un bloque de memoria reservado. Si es así, el puntero “DireccionMemoria” apunta al bloque de memoria reservado a ese dispositivo. Además, se almacena el valor a escribir en la dirección de memoria (String Valor) a entero y se saca por pantalla un error si este no es un valor numérico.

```
29.  if (index < MemoriaDispositivos.size()) {
30.    uint32_t* DireccionMemoria = MemoriaDispositivos[index];
31.    uint32_t ValorBytes = Valor.toInt();
32.    if (ValorBytes == 0 && Valor != "0") {
33.      Serial.println("Valor no válido. Asegúrate de que el valor
   sea un número.");
34.      return;
35.    }
```

Por último, se comprueba que no se está intentando escribir fuera de los límites del bloque reservado, si es así, se avisa al usuario, y si el índice es correcto y no se pretende escribir fuera de los límites, se actualiza el valor almacenado en la posición definida por la variable “Indice” con el valor “ValorBytes”.

```
36.    uint32_t TamanoMaximo = (CalcularMemoria() + 59) /
   Dispositivos.size();
37.    if (Indice < 0 || Indice >= TamanoMaximo) {
38.      Serial.println("Índice fuera de rango.");
39.      return;
```

```
    }
    DireccionMemoria[Indice] = ValorBytes;

    Serial.print("Escrito el valor ");
    Serial.print(ValorBytes);
    Serial.print(" en la dirección de memoria ");
    Serial.print(Indice);
    Serial.print(" del bloque de memoria reservado a la dirección MAC ");
    Serial.println(Mac);
  }
}
```

13. MemoriaCompartida::EnviarValorDeMemoria()

```
void MemoriaCompartida::EnviarValorVinculado(String NombreBloque, int Indice, String Valor){
  uint32_t ValorBytes = Valor.toInt();
  bool BloqueVinculado = false;
  bool BloqueResponsabilizado = false;
  for (size_t i = 0; i < VectoresVinculados.size(); i++) {
    if(NombreBloque == VectoresVinculados[i]) {
      BloqueVinculado = true;
    }
  }
  if(BloqueVinculado) {
    if(ValoresResponsabilizados.size() == 0) {
      Serial.println("Error: Antes debes responsabilizarte de unos índices.");
    } else{
      for(size_t i = 0; i < ValoresResponsabilizados.size(); i++) {
        if (NombreBloque == ValoresResponsabilizados[i]) {
          BloqueResponsabilizado =  true;
          if (Indice < ValoresResponsabilizados[i+1].toInt() || Indice > ValoresResponsabilizados[i+2].toInt()) {
            Serial.print("Error: Para el bloque ");
            Serial.print(NombreBloque);
            Serial.print(" solo puedes escribir entre ");
            Serial.print(ValoresResponsabilizados[i+1]);
            Serial.print(" y ");
            Serial.println(ValoresResponsabilizados[i+2]);
          } else {
            EscribirMemoria(NombreBloque, Indice, Valor);
            if (VectoresVinculados.size() > 0) {
```

```
27.              for (size_t i = 0; i < VectoresVinculados.size();
   i++) {
28.                if (NombreBloque == VectoresVinculados[i]) {
29.                  EditarValorDeMemoria(VectoresVinculados[i+1],
   NombreBloque, Indice, ValorBytes);
30.                  delayMicroseconds(200);
31.                }
32.              }
33.            }
34.          }
35.        }
36.      }
37.    }
38.  } else {
39.    EscribirMemoria(NombreBloque, Indice, Valor);
40.  }
41.}
```

La función “EnviarValorVinculado()” permite escribir en un bloque de memoria e índice definido un valor determinado. Además, envía un mensaje a todos los dispositivos que tengan vinculados ese mismo bloque para gestionar el mensaje y poder escribir en el bloque de memoria de los otros dispositivos.

Primero se crean todas las variables necesarias para la correcta utilización de la función. Se recorre el vector “VectoresVinculados” y si el bloque en el que se desea escribir pertenece al vector: el bloque estará vinculado a otros dispositivos. realizan las comprobaciones necesarias para un correcto funcionamiento de la función:

```
2.   uint32_t ValorBytes = Valor.toInt();
3.   bool BloqueVinculado = false;
4.   bool BloqueResponsabilizado = false;
5.   for (size_t i = 0; i < VectoresVinculados.size(); i++) {
6.     if(NombreBloque == VectoresVinculados[i]) {
7.       BloqueVinculado = true;
8.     }
9.   }
```

Si el bloque está vinculado, solo se podrá escribir en él si tiene unos índices responsabilizados, esto es, un dispositivo con un vector vinculado con otro dispositivo solo podrá escribir valores en los índices que tenga permiso. Este permiso viene definido por el vector “ValoresResponsabilizados()”. En los índices que no sea así, solo se podrá mostrar el valor del índice. Si se intenta escribir en unos índices a los que el dispositivo no tiene permiso para ello, se

mostrará un mensaje de error. Si se intenta escribir en unos índices con permiso, se escribirá dicho dato y se llamará a la función "EditarValorDeMemoria" encargada de preparar el mensaje para que el resto de dispositivos también lo hagan.

```
10.  if(BloqueVinculado) {
11.    if(ValoresResponsabilizados.size() == 0) {
12.      Serial.println("Error: Antes debes responsabilizarte de
   unos índices.");
13.    } else {
14.      for(size_t i = 0; i < ValoresResponsabilizados.size(); i++)
   {
15.        if (NombreBloque == ValoresResponsabilizados[i]) {
16.          BloqueResponsabilizado =  true;
17.          if (Indice < ValoresResponsabilizados[i+1].toInt() ||
   Indice > ValoresResponsabilizados[i+2].toInt()) {
18.            Serial.print("Error: Para el bloque ");
19.            Serial.print(NombreBloque);
20.            Serial.print(" solo puedes escribir entre ");
21.            Serial.print(ValoresResponsabilizados[i+1]);
22.            Serial.print(" y ");
23.            Serial.println(ValoresResponsabilizados[i+2]);
24.          } else {
25.            EscribirMemoria(NombreBloque, Indice, Valor);
26.            if (VectoresVinculados.size() > 0) {
27.              for (size_t i = 0; i < VectoresVinculados.size();
   i++) {
28.                if (NombreBloque == VectoresVinculados[i]) {
29.                  EditarValorDeMemoria(VectoresVinculados[i+1],
   NombreBloque, Indice, ValorBytes);
30.                  delayMicroseconds(200);
31.                }
32.              }
33.            }
34.          }
35.        }
36.      }
37.    }
38.  }
```

Si el bloque no está vinculado, simplemente se escribirá en dicho bloque de forma local.

```
38.  else {
39.  EscribirMemoria(NombreBloque, Indice, Valor);
40.  }
```

```
41.}
```

14. MemoriaCompartida::EditarValorDeMemoria()

```
1. void MemoriaCompartida::EditarValorDeMemoria(String MacReceptor,
   String NombreBloque, int Indice, int Valor) {
2.   if (!esDireccionMac(MacReceptor)) {
3.     if (!esAlias(MacReceptor)){
4.       Serial.println("Error: Formato de MAC inválido.");
5.       return;
6.     }
7.     if(esAlias(MacReceptor)){
8.       ConvertirAliasAMacString(MacReceptor);
9.       String mac = ConvertirAliasAMacString(MacReceptor);
10.      MacReceptor = mac;
11.    }
12.  }
13.  if (NormalizarMac(MacReceptor) ==
   NormalizarMac(WiFi.macAddress())) {
14.    Serial.println("Error: La MAC introducida es la de este
   dispositivo.");
15.    return;
16.  }
17.  if (!ConvertirMacStringABytes(MacReceptor, peerMAC)) {
18.    Serial.println("Error al convertir la MAC");
19.    return;
20.  }
21.  char Texto[100];
22.  sprintf(Texto, "Editar %s %d %d", NombreBloque.c_str(), Indice,
   Valor);
23.  strcpy(MensajeEnviado.text, Texto);
24.  ConvertirMacStringABytes(MacReceptor, peerMAC);
25.  esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
26.}
```

La función “EditarValorDeMemoria()” permite enviar un mensaje para que los dispositivos que lo reciban lo procesen y escriban de forma local en el bloque de memoria reservado por “NombreBloque” en el índice definido por “Indice” un valor definido por “Valor”.

Primero se realizan las comprobaciones necesarias para un correcto funcionamiento de la función:

4. Se verifica que la cadena de texto (String MacReceptor) contiene una dirección MAC válida empleando la función “esDirecciónMac()” a través

de una condición “if”. Si no es una dirección MAC válida, se realiza otra comprobación: Se verifica si el contenido de la cadena de texto “MacReceptorInterna” es un alias a través de la función “esAlias()”.

+ Si no es así, saca por pantalla un error.

+ Si sí es así se ejecuta la función “ConvertirAliasAMacString()” para obtener la dirección MAC vinculada a ese alias.

```
2.  if (!esDireccionMac(MacReceptorInterna)) {
3.      if (!esAlias(MacReceptorInterna)){
4.        Serial.println("Error: Formato de MAC de bloque de
    memoria reservado inválido.");
5.        return;
6.      }
7.      if(esAlias(MacReceptorInterna)){
8.        ConvertirAliasAMacString(MacReceptorInterna);
9.        String Mac =
    ConvertirAliasAMacString(MacReceptorInterna);
10.       MacReceptorInterna = Mac;
11.     }
12.   }
```

5. Si el contenido de la cadena de texto “MacReceptor” sí es una dirección Mac se realiza la siguiente verificación: Con ayuda de la función “NormalizarMac()” se comprueba si la cadena de texto contiene la dirección del propio dispositivo, si es así, devuelve un mensaje de error.

```
13.   if (NormalizarMac(MacReceptor) ==
    NormalizarMac(WiFi.macAddress())) {
14.     Serial.println("Error: La mac introducida es la del
    propio dispositivo.");
15.     return;
16.   }
```

6. Si no es la MAC del propio dispositivo, realiza la siguiente comprobación: Con “ConvertirMacStringABytes()” se verifica si se ha podido convertir la cadena de texto (Mac) a bytes guardados en la variable “peerMac”:

```
17.   if (!ConvertirMacStringABytes(MacReceptor, peerMAC)) {
18.     Serial.println("Error al convertir la dirección MAC.");
19.     return;
20.   }
```

Una vez hechas las comprobaciones se prepara el mensaje “Editar” seguido de la dirección MAC del bloque de memoria donde se contiene la dirección de memoria a editar definida por “Indice” y el valor a editar: “Valor”.

```
21.char Texto[100];
22.   sprintf(Texto, "Editar %s %d %d", NombreBloque.c_str(),
   Indice, Valor);
23.   strcpy(MensajeEnviado.text, Texto);
24.   ConvertirMacStringABytes(MacReceptor, peerMAC);
25.   esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
26.}
```

15. MemoriaCompartida::Alias()

```
1.  void MemoriaCompartida::Alias() {
2.    Serial.print("Introduce alias para ");
3.    Serial.print(WiFi.macAddress());
4.    Serial.print(" (dispositivo local): ");
5.    while (Serial.available() == 0) {}
6.    String AliasLocalIntroducido = Serial.readStringUntil('\n');
7.    AliasLocalIntroducido.trim();
8.    Serial.println(AliasLocalIntroducido);
9.    if (AliasLocalIntroducido.length() > 0) {
10.     bool esAliasLocalConMacVinculada = false;
11.     for (int i = 0; i < DispositivosAlias.size(); i++) {
12.       if (DispositivosAlias[i] == WiFi.macAddress()) {
13.         DispositivosAlias[i + 1] = AliasLocalIntroducido;
14.         esAliasLocalConMacVinculada = true;
15.         break;
16.       }
17.     }
18.     if (!esAliasLocalConMacVinculada) {
19.       DispositivosAlias.push_back(WiFi.macAddress());
20.       DispositivosAlias.push_back(AliasLocalIntroducido);
21.     }
22.   }

23.   for (int i = 0; i < DispositivosConectados.size(); i++) {
24.     Serial.print("Introduce alias para ");
25.     Serial.print(NormalizarMac(DispositivosConectados[i]));
26.     Serial.print(": ");
27.     while (Serial.available() == 0) {}
28.     String AliasConectadoIntroducido =
    Serial.readStringUntil('\n');
29.     AliasConectadoIntroducido.trim();
30.     Serial.println(AliasConectadoIntroducido);
31.
32.     if (AliasConectadoIntroducido.length() > 0) {
33.       bool esAliasConectadoRepetido = false;
```

```
      for (int j = 0; j < DispositivosAlias.size(); j++) {
        if (DispositivosAlias[j + 1] ==
   AliasConectadoIntroducido) {
          esAliasConectadoRepetido = true;
          break;
        }
      }
      if (esAliasConectadoRepetido) {
        Serial.println("Error: El alias ya está en uso.
   Introduzca otro alias.");
        i--;
        continue;
      }
      bool esAliasConectadoConMacVinculada = false;
      for (int j = 0; j < DispositivosAlias.size(); j++) {
        if (DispositivosAlias[j] == DispositivosConectados[i]) {
          DispositivosAlias[j + 1] = AliasConectadoIntroducido;
          esAliasConectadoConMacVinculada = true;
          break;
        }
      }
      if (!esAliasConectadoConMacVinculada) {
        DispositivosAlias.push_back(DispositivosConectados[i]);
        DispositivosAlias.push_back(AliasConectadoIntroducido);
      }
    }
  }
  Serial.println("Alias asignado a los dispositivos:");
  for (int i = 0; i < DispositivosAlias.size(); i++) {
    Serial.println(NormalizarMac(DispositivosAlias[i]));
  }
}
```

La función “Alias()” permite asignar y vincular un alias al dispositivo local y a cada dispositivo registrado facilitando la identificación de cada dispositivo mediante el uso de nombres legibles. El flujo se divide en dos partes similares: por un lado, la asignación de un alias al dispositivo local y por otro, la asignación de un alias a cada dispositivo conectado.

Primero se pide al usuario introducir el alias para el dispositivo local:

```
  Serial.print("Introduce alias para ");
  Serial.print(WiFi.macAddress());
  Serial.print(" (dispositivo local): ");
  while (Serial.available() == 0) {}
  String AliasLocalIntroducido = Serial.readStringUntil('\n');
```

```
7.    AliasLocalIntroducido.trim();
8.    Serial.println(AliasLocalIntroducido);
```

Y se comprueba si el alias introducido es válido, si es así se entra en el "for" para recorrer el vector "DispositivosAlias", si el vector no contiene la dirección MAC del dispositivo local, se la añade al igual que el alias introducido anteriormente. En caso de que el vector sí contenga la dirección MAC del dispositivo (esto ocurre cuando se esté haciendo una actualización del alias, de forma que la dirección MAC del dispositivo local ya ha sido añadido en el vector mediante el flujo definido anteriormente), actualiza el alias vinculado al dispositivo local. Este siempre se encuentra en la posición siguiente a la de la dirección MAC del dispositivo local:

```
9.  if (AliasLocalIntroducido.length() > 0) {
10.     bool esAliasLocalConMacVinculada = false;
11.     for (int i = 0; i < DispositivosAlias.size(); i++) {
12.       if (DispositivosAlias[i] == WiFi.macAddress()) {
13.         DispositivosAlias[i + 1] = AliasLocalIntroducido;
14.         esAliasLocalConMacVinculada = true;
15.         break;
16.       }
17.     }
18.     if (!esAliasLocalConMacVinculada) {
19.       DispositivosAlias.push_back(WiFi.macAddress());
20.       DispositivosAlias.push_back(AliasLocalIntroducido);
21.     }
22.   }
```

Una vez definido el flujo para la asignación del alias del dispositivo local, se define el flujo para la asignación de los dispositivos conectados. Se recorre el vector "DispositivosConectados", y para cada posición (que contiene una dirección MAC) se pide al usuario introducir un alias:

```
23.for (int i = 0; i < DispositivosConectados.size(); i++) {
24.     Serial.print("Introduce alias para ");
25.     Serial.print(NormalizarMac(DispositivosConectados[i]));
26.     Serial.print(": ");
27.     while (Serial.available() == 0) {}
28.     String AliasConectadoIntroducido =
   Serial.readStringUntil('\n');
29.     AliasConectadoIntroducido.trim();
30.     Serial.println(AliasConectadoIntroducido);
31.
```

Y se comprueba que es un alias válido, si es así, se recorre el vector "DispositivosAlias", si el vector contiene el mismo alias que se intenta vincular a un dispositivo (se está intentado vincular un mismo alias a dos direcciones MAC distintas), salta un error avisando al usuario y se pide que vuelva a introducir un alias, reduciendo el contador del "for" en uno:

```
32.     if (AliasConectadoIntroducido.length() > 0) {
33.       bool esAliasConectadoRepetido = false;
34.       for (int j = 0; j < DispositivosAlias.size(); j++) {
35.         if (DispositivosAlias[j + 1] ==
    AliasConectadoIntroducido) {
36.           esAliasConectadoRepetido = true;
37.           break;
38.         }
39.       }
40.       if (esAliasConectadoRepetido) {
41.         Serial.println("Error: El alias ya está en uso.
    Introduzca otro alias.");
42.         i--;
43.         continue;
44.       }
```

A continuación, se recorre el vector "DispositivosAlias" nuevamente. Si el vector contiene la dirección MAC del dispositivo al que se intenta vincular un alias (se está intentado cambiar el alias) se actualiza dicho alias, y si el vector no contiene la dirección MAC del dispositivo al que se intenta vincular un alias, se añade tanto la dirección MAC como el alias al vector.

```
45.       bool esAliasConectadoConMacVinculada = false;
46.       for (int j = 0; j < DispositivosAlias.size(); j++) {
47.         if (DispositivosAlias[j] == DispositivosConectados[i]) {
48.           DispositivosAlias[j + 1] = AliasConectadoIntroducido;
49.           esAliasConectadoConMacVinculada = true;
50.           break;
51.         }
52.       }
53.       if (!esAliasConectadoConMacVinculada) {
54.         DispositivosAlias.push_back(DispositivosConectados[i]);
55.         DispositivosAlias.push_back(AliasConectadoIntroducido);
56.       }
57.     }
58.   }
```

Por último se saca por pantalla los alias introducidos a cada dispositivo:

```
59.Serial.println("Alias asignado a los dispositivos:");
60.  for (int i = 0; i < DispositivosAlias.size(); i++) {
61.    Serial.println(NormalizarMac(DispositivosAlias[i]));
62.  }
63.}
```

16. MemoriaCompartida::esAlias()

```
1. bool MemoriaCompartida::esAlias(String Alias) {
2.   for (int i = 1; i < DispositivosAlias.size(); i += 2) {
3.     if (DispositivosAlias[i] == Alias) {
4.       return true;
5.     }
6.   }
7.   return false;
8. }
```

La función “esAlias()” permite comprobar si el contenido de una cadena de texto es un alias guardado en el vector “DispositivosAlias”.

Se recorre dicho vector estructurado de la siguiente forma:

1. En primera posición: la dirección MAC local.
2. En segunda posición: el alias vinculado a la dirección MAC local (si lo hay).
3. En tercera posición: la dirección MAC de un dispositivo conectado.
4. En cuarta posición: el alias vinculado a la dirección MAC del dispositivo anterior (si lo hay)
5. La estructura continúa si hubiese más dispositivos conectados.

Si el contenido de la cadena de texto (String Alias) contiene un alias perteneciente al vector “DispositivosAlias” se devuelve un “true”. En caso contrario se devuelve un “false”.

17. MemoriaCompartida::ConvertirAliasAMacString()

```
1. String MemoriaCompartida::ConvertirAliasAMacString(String Alias)
   {
2.   for (int i = 1; i < DispositivosAlias.size(); i += 2) {
3.     if (DispositivosAlias[i] == Alias) {
4.       return DispositivosAlias[i - 1];
5.     }
6.   }
7.   return "";
8. }
```

La función “ConvertirAliasAMacString” sirve para obtener la dirección MAC vinculada a un alias a partir del contenido de una cadena de texto. Como la estructura del vector “DispositivosAlias” es fija y definida, se puede obtener la MAC vinculada a un alias a partir del propio alias: Se recorre el vector “DispositivosAlias” mediante un “for”, si el contenido de la cadena de texto (String Alias) contiene un alias perteneciente al vector “DispositivosAlias”, se devuelve el contenido del vector en la posición anterior a dicho alias.

18. MemoriaCompartida::init()

```
1. void MemoriaCompartida::init() {
2.   WiFi.mode(WIFI_STA);
3.   delay(2000);
4.   Serial.print("MAC del dispositivo: ");
5.   Serial.println(WiFi.macAddress());
6.   if (esp_now_init() != ESP_OK) {
7.     Serial.println("Error al inicializar ESP-NOW.
   Reiniciando...");
8.     ESP.restart();
9.   }
10.  Serial.println("ESP-NOW iniciado correctamente.");
11.  esp_now_register_recv_cb(enRecepcionDeDatos);
12.  esp_now_register_send_cb(enEnvioDeDatos);
13.  delay(2000);
14.  Serial.println("ESP-32 listo.");
15.}
```

La función “init()” se encarga de inicializar el dispositivo, y configurar la funcionalidad de ESP_NOW.

Mediante la función “WiFi.mode(WIFI_STA)” se configura el dispositivo en modo estación. A continuación, se añade un “delay” de 2 segundos para asegurarse de la correcta configuración en modo estación. Se informa al usuario de cuál es la MAC del dispositivo local, se le avisa si inicializa correctamente o si ocurre algún error al inicializar y se reinicia en ese caso. Se utilizan las funciones de ESP-NOW “esp_now_register_recv_cb” y “esp_now_register_send_cb” para registrar los “callbacks” de las funciones “enRecepcionDeDatos” y “enEnvioDeDatos” respectivamente. Se ejecuta un “delay” de 2 segundos nuevamente para asegurar la correcta configuración del dispositivo y se saca por pantalla un mensaje avisando al usuario de que el dispositivo está listo.

Esta función será necesaria siempre que se pretenda crear una herramienta empleando esta librería. Por ejemplo, en el archivo main:

```
#include <MemoriaCompartida.h>
#include "esp_system.h"

MemoriaCompartida MemCom;

void setup() {
  Serial.begin(115200);
  delay(2000);
  MemCom.init();
}
```

Si no se incluye esta función, el resto de funciones pueden no funcionar correctamente.

Funciones Non-Blocking

Las funciones “non-blocking” creadas son las de manejo y procesamiento de mensajes mediante un flujo definido. En esta librería hay creadas dos funciones que siguen esta tecnología, estas funciones son ambas “Callback” de envío y de recepción de datos. Los “Callback” o funciones de retrollamada son funciones que se pasan como argumento a otras funciones con el objetivo de que esta se ejecute en algún momento.

Los dos “Callback” definidos en esta librería son:

1. MemoriaCompartida::enEnvioDeDatos()

```
1. void MemoriaCompartida::enEnvioDeDatos(const uint8_t* MAC,
   esp_now_send_status_t status) {
2. if (status == ESP_NOW_SEND_SUCCESS) {
3.
4. } else {
5.   Serial.println("Error al enviar el mensaje");
6.  }
7.  EnvioCompletado = true;
8. }
```

La función “enEnvioDeDatos()” permite conocer cuál es el estatus de un mensaje enviado a través de ESP-NOW. Esta función se ejecuta de forma automática cada vez que se envía un mensaje. A través de una condición “if” se informa de si ha habido un error en el envío de datos.

2. MemoriaCompartida::enRecepcionDeDatos()

```
1. void MemoriaCompartida::enRecepcionDeDatos(const
   esp_now_recv_info* info, const uint8_t* DatosRecibidos, int
   tamaño) {
2.   memcpy(&MensajeRecibido, DatosRecibidos,
   sizeof(MensajeRecibido));
3.   if (strcmp(MensajeRecibido.text, "Prueba de conexión") == 0) {
4.     String MacString = "";
5.     for (int i = 0; i < 6; i++) {
6.       MacString += String(info->src_addr[i], HEX);
7.       if (i < 5) {
8.         MacString += ":";
9.       }
10.    }
11.    if (find(DispositivosConectados.begin(),
   DispositivosConectados.end(), MacString) ==
   DispositivosConectados.end()) {
12.      DispositivosConectados.push_back(MacString);
13.    }
14.    if (!esp_now_is_peer_exist(info->src_addr)) {
15.      esp_now_peer_info_t InfoPeer = {};
16.      memcpy(InfoPeer.peer_addr, info->src_addr, 6);
17.      InfoPeer.channel = 0;
18.      InfoPeer.encrypt = false;
19.      if (esp_now_add_peer(&InfoPeer) != ESP_OK) {
20.        Serial.println("Error al agregar el peer
   temporalmente.");
21.        return;
22.      }
23.    }
24.    strcpy(MensajeEnviado.text, "Recibido");
25.    esp_err_t ResultadoDeEnvio = esp_now_send(info->src_addr,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
26.    Serial.print(MacString);
27.    Serial.println(" está intentando conectarse.");
28.    if (ResultadoDeEnvio == ESP_OK) {
29.      Serial.println("Iniciando conexión...");
30.    } else {
31.      Serial.print("Error al iniciar conexión. Código de error:
   ");
32.      Serial.println(ResultadoDeEnvio);
33.    }
34.  } else if (strcmp(MensajeRecibido.text, "Recibido") == 0) {
35.    RespuestaRecibida = true;
36.    String MacString = "";
37.    for (int i = 0; i < 6; i++) {
38.      MacString += String(info->src_addr[i], HEX);
39.      if (i < 5) {
```

```
40.         MacString += ":";
41.       }
42.     }
43.     if (find(DispositivosConectados.begin(),
   DispositivosConectados.end(), MacString) ==
   DispositivosConectados.end()) {
44.       DispositivosConectados.push_back(MacString);
45.     }
46.   } else if (strncmp(MensajeRecibido.text, "Borrado", 7) == 0) {
47.     String mensajeRecibido = MensajeRecibido.text;
48.     int espacioPos = mensajeRecibido.indexOf(' ');
49.     if (espacioPos != -1) {
50.       String MacString = mensajeRecibido.substring(espacioPos +
   1);
51.       Serial.print("El dispositivo ");
52.       Serial.print(MacString);
53.       Serial.println(" ha solicitado borrar su registro.");
54.       if (DispositivosConectados.empty()) {
55.         Serial.println("No hay dispositivos para borrar.");
56.       } else {
57.         for (auto it = DispositivosConectados.begin(); it !=
   DispositivosConectados.end(); ++it) {
58.           if (NormalizarMac(*it) == NormalizarMac(MacString)) {
59.             ConvertirMacStringABytes(MacString, peerMAC);
60.             DispositivosConectados.erase(it);
61.             esp_now_del_peer(peerMAC);
62.             Serial.print("El dispositivo ");
63.             Serial.print(MacString);
64.             Serial.println(" ha sido eliminado con éxito.");
65.             for (size_t i = 0; i < VectoresVinculados.size();
   i++) {
66.               if(NormalizarMac(VectoresVinculados[i]) ==
   NormalizarMac(*it)) {
67.                 VectoresVinculados.erase(VectoresVinculados.begin
   () + (i - 1));
68.                 VectoresVinculados.erase(VectoresVinculados.begin
   () + (i - 1));
69.                 i -= 1;
70.                 }
71.               }
72.             break;
73.           }
74.         }
75.       }
76.     }
77.   } else if (strncmp(MensajeRecibido.text, "Editar", 6) == 0) {
78.     Serial.println("Mensaje recibido");
```

```
79.     Serial.println(MensajeRecibido.text);
80.
81.     String Mensaje = String(MensajeRecibido.text);
82.
83.     int PrimerEspacio = Mensaje.indexOf(' ');
84.     int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio + 1);
85.     int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio + 1);
86.
87.     String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
   SegundoEspacio);
88.     String IndiceStr = Mensaje.substring(SegundoEspacio + 1,
   TercerEspacio);
89.     String ValorStr = Mensaje.substring(TercerEspacio + 1);
90.
91.     int IndiceBytes = IndiceStr.toInt();
92.
93.     String MacStr = "";
94.     for (int i = 0; i < 6; i++) {
95.       MacStr += String(info->src_addr[i], HEX);
96.       if (i < 5) MacStr += ":";
97.     }
98.     for (size_t i = 0; i < Bloques.size(); ++i) {
99.       for (size_t j = 0; j < VectoresVinculados.size(); ++j) {
100.           if (Bloques [i] == VectoresVinculados [j] &&
   Bloques[i] == NombreBloque) {
101.             uint32_t TamanoBloque = Bloques [i + 1].toInt();
102.             if (IndiceBytes >= 0 && IndiceBytes < TamanoBloque)
   {
103.         } else {
104.           Serial.println("Enviando error a emisor...");
105.           uint8_t MacBytes[6];
106.           ConvertirMacStringABytes(MacStr, MacBytes);
107.           strcpy(MensajeEnviado.text, "Error 3: Índice fuera de
   rango");
108.           esp_err_t Result = esp_now_send(info->src_addr,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
109.           return;
110.         }
111.       }
112.     }
113.   }
114.   EscribirMemoria(NombreBloque, IndiceBytes, ValorStr);
115.   delayMicroseconds(200);
116.   } else if (strncmp(MensajeRecibido.text, "Error 3", 7) == 0)
   {
117.     Serial.print("Mensaje recibido: ");
118.     Serial.println(MensajeRecibido.text);
```

```
119.    } else if (strncmp(MensajeRecibido.text, "Vincular", 8) ==
   0){
120.      String Mensaje = String(MensajeRecibido.text);
121.      int PrimerEspacio = Mensaje.indexOf(' ');
122.      int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio +
   1);
123.      String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
   SegundoEspacio);
124.      String TamanoBloque = Mensaje.substring(SegundoEspacio +
   1);
125.      int TamanoBloqueBytes = TamanoBloque.toInt();
126.      for (size_t i = 0; i < Bloques.size(); i++){
127.        if(Bloques[i] == NombreBloque){
128.          if (Bloques[i+1] == TamanoBloque){
129.            String MacStr = "";
130.            for (int i = 0; i < 6; i++) {
131.              MacStr += String(info->src_addr[i], HEX);
132.              if (i < 5) MacStr += ":";
133.            }
134.            ConvertirMacStringABytes(MacStr,peerMAC);
135.            NormalizarMac(MacStr);
136.            VectoresVinculados.push_back(NombreBloque);
137.            VectoresVinculados.push_back(MacStr);
138.
139.            char Texto[50];
140.            sprintf(Texto, "Vinculando %s ",
   NombreBloque.c_str());
141.            strcpy(MensajeEnviado.text, Texto);
142.            esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
143.            Serial.print("El vector: ");
144.            Serial.print(NombreBloque);
145.            Serial.print(" se está intentando vincular con ");
146.            Serial.println(NormalizarMac(MacStr));
147.            Serial.println("Vinculando...");
148.            for (int i = 0; i < TamanoBloqueBytes; i++) {
149.              if(MostrarMemoria(NombreBloque, i) != 0) {
150.                EditarValorDeMemoria(MacStr, NombreBloque, i,
   MostrarMemoria(NombreBloque, i));
151.                delayMicroseconds(1000);
152.              }
153.            }
154.            Serial.println("Vinculación completada");
155.            sprintf(Texto, "Vinculado %s", NombreBloque.c_str());
156.            strcpy(MensajeEnviado.text, Texto);
157.            esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
```

```
158.           }
159.         }
160.       }
161.     } else if (strncmp(MensajeRecibido.text, "Vinculando", 10) ==
   0) {
162.       String Mensaje = String(MensajeRecibido.text);
163.       Serial.println(Mensaje);
164.       int PrimerEspacio = Mensaje.indexOf(' ');
165.       String NombreBloque = Mensaje.substring(PrimerEspacio + 1);
166.       String MacStr = "";
167.       for (int i = 0; i < 6; i++) {
168.         MacStr += String(info->src_addr[i], HEX);
169.         if (i < 5) MacStr += ":";
170.       }
171.       Serial.print("El dispositivo ");
172.       Serial.print(NormalizarMac(MacStr));
173.       Serial.print(" está intentando vincular el bloque ");
174.       Serial.println(NombreBloque);
175.       Serial.println("Vinculando...");
176.     } else if (strncmp(MensajeRecibido.text, "Vinculado", 9) ==
   0) {
177.       Serial.println("Vinculación completada.");
178.       String Mensaje = String(MensajeRecibido.text);
179.       int PrimerEspacio = Mensaje.indexOf(' ');
180.       String NombreBloque = Mensaje.substring(PrimerEspacio + 1);
181.       String MacStr = "";
182.       for (int i = 0; i < 6; i++) {
183.         MacStr += String(info->src_addr[i], HEX);
184.         if (i < 5) MacStr += ":";
185.       }
186.       VectoresVinculados.push_back(NombreBloque);
187.       VectoresVinculados.push_back(MacStr);
188.     } else if (strncmp(MensajeRecibido.text, "Responsable", 11)
   == 0) {
189.       String Mensaje = String(MensajeRecibido.text);
190.       int PrimerEspacio = Mensaje.indexOf(' ');
191.       int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio +
   1);
192.       int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio +
   1);
193.
194.       String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
   SegundoEspacio);
195.       String StrIndiceMin = Mensaje.substring(SegundoEspacio + 1,
   TercerEspacio);
196.       String StrIndiceMax = Mensaje.substring(TercerEspacio + 1);
197.       uint32_t IndiceMin = StrIndiceMin.toInt();
```

```
198.      uint32_t IndiceMax = StrIndiceMax.toInt();
199.
200.      String MacStr = "";
201.      for (int i = 0; i < 6; i++) {
202.        MacStr += String(info->src_addr[i], HEX);
203.        if (i < 5) MacStr += ":";
204.        }
205.        ConvertirMacStringABytes(MacStr,peerMAC);
206.        for (size_t i = 0; i < ValoresResponsabilizados.size();
    i++) {
207.          if (NombreBloque == ValoresResponsabilizados[i]) {
208.            if ((IndiceMin >=
    ValoresResponsabilizados[i+1].toInt() && IndiceMin <=
    ValoresResponsabilizados[i+2].toInt()) || (IndiceMax >=
    ValoresResponsabilizados[i+1].toInt() && IndiceMax <=
    ValoresResponsabilizados[i+2].toInt()) || (IndiceMin <=
    ValoresResponsabilizados[i+1].toInt() && IndiceMax >=
    ValoresResponsabilizados[i+2].toInt())) {
209.              char Texto[100];
210.              sprintf(Texto, "ResponsError %s %d %d",
    NombreBloque.c_str(), ValoresResponsabilizados[i+1].toInt(),
    ValoresResponsabilizados[i+2].toInt());
211.              strcpy(MensajeEnviado.text, Texto);
212.              esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
    sizeof(MensajeEnviado));
213.            } else {
214.              char Texto[100];
215.              sprintf(Texto, "ResponsExito %s %d %d",
    NombreBloque.c_str(), IndiceMin, IndiceMax);
216.              strcpy(MensajeEnviado.text, Texto);
217.              esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
    sizeof(MensajeEnviado));
218.            }
219.          }
220.        }
221.        if (ValoresResponsabilizados.size() == 0) {
222.        char Texto[100];
223.        sprintf(Texto, "ResponsExito %s %d %d",
    NombreBloque.c_str(), IndiceMin, IndiceMax);
224.        strcpy(MensajeEnviado.text, Texto);
225.        esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
    sizeof(MensajeEnviado));
226.       }
227.     } else if ((strncmp(MensajeRecibido.text, "ResponsError",
    12) == 0) || ((strncmp(MensajeRecibido.text, "ResponsExito", 12)
    == 0))) {
228.       String Mensaje = String(MensajeRecibido.text);
```

```
229.      int PrimerEspacio = Mensaje.indexOf(' ');
230.      int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio +
   1);
231.      int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio +
   1);

232.      String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
   SegundoEspacio);
233.      String StrIndiceMin = Mensaje.substring(SegundoEspacio +
   1, TercerEspacio);
234.      String StrIndiceMax = Mensaje.substring(TercerEspacio +
   1);
235.      uint32_t IndiceMin = StrIndiceMin.toInt();
236.      uint32_t IndiceMax = StrIndiceMax.toInt();
237.      if (strncmp(MensajeRecibido.text, "ResponsError", 12) ==
   0) {
238.        Respuestas.push_back(0); // 0 = Error
239.      }
240.      if (strncmp(MensajeRecibido.text, "ResponsExito", 12) ==
   0) {
241.        Respuestas.push_back(1); // 1 = Exito
242.      }
243.      String MacStr = "";
244.      for (int i = 0; i < 6; i++) {
245.        MacStr += String(info->src_addr[i], HEX);
246.        if (i < 5) MacStr += ":";
247.      }
248.      ConvertirMacStringABytes(MacStr,peerMAC);
249.      for (size_t i = 0; i < ListaRespuestasPendientes.size();
   i++) {
250.        if (ListaRespuestasPendientes[i] == MacStr) {
251.          ListaRespuestasPendientes[i+1] = "1";
252.          break;
253.        }
254.      }
255.      bool TodosRespondieron = true;
256.      for (size_t i = 1; i < ListaRespuestasPendientes.size(); i
   += 2) {
257.        if (ListaRespuestasPendientes[i] != "1") {
258.          TodosRespondieron = false;
259.            break;
260.        }
261.      }
262.      bool ExisteError = false;
263.      if (TodosRespondieron) {
264.        for (size_t i = 0; i < Respuestas.size(); i++) {
265.          if (Respuestas[i] == 0) {
```

```
266.             ExisteError = true;
267.               break;
268.           }
269.         }
270.         if (ExisteError) {
271.           Serial.print("Error: Para el bloque ");
272.           Serial.print(NombreBloque);
273.           Serial.print(" no puedes responsabilizarte de unos
   índices entre ");
274.           Serial.print(IndiceMin);
275.           Serial.print(" y ");
276.           Serial.println(IndiceMax);
277.         } else {
278.           Serial.print("Para el bloque ");
279.           Serial.print(NombreBloque);
280.           Serial.print(" este dispositivo solo podrá escribir
   entre las posiciones ");
281.           Serial.print(IndiceMin);
282.           Serial.print(" y ");
283.           Serial.println(IndiceMax);
284.           ValoresResponsabilizados.push_back(NombreBloque);
285.           ValoresResponsabilizados.push_back(StrIndiceMin);
286.           ValoresResponsabilizados.push_back(StrIndiceMax);
287.         }
288.         Respuestas.clear();
289.         ListaRespuestasPendientes.clear();
290.       }
291.     }
292.   }
```

La función “enRecepcionDeDatos()” es el “Cerebro” de la librería, se encarga de gestionar los datos recibidos a través de la comunicación ESP-NOW. Esta función se activa de forma automática siempre que el dispositivo reciba datos a través de ESP-NOW, interpreta y toma decisiones en función del contenido del mensaje recibido. Todas las funciones de envío de mensajes definidas anteriormente estarían incompletas sin la utilización de esta función.

Primero se guarda el mensaje recibido en la variable “MensajeRecibido”.

```
2.    memcpy(&MensajeRecibido, DatosRecibidos,
   sizeof(MensajeRecibido));
```

A continuación, se procesará el mensaje de maneras diferentes dependiendo del contenido del mismo:

- Procesamiento de mensaje enviado por la función “Registrar()”:
  - Si se recibe el mensaje “Prueba de conexión”, se guarda la dirección MAC que envió dicho mensaje en la variable “MacString” a través de “info->src_addr”, se agrega al vector “DispositivosConectados” si no está registrado ya y se agrega como peer si no lo está ya usando la función “esp_now_add_peer()”. Una vez hecho esto, se envía el mensaje “Recibido” de vuelta al dispositivo que envió “Prueba de conexión”.

```
3. if (strcmp(MensajeRecibido.text, "Prueba de conexión") ==
   0) {
4.     String MacString = "";
5.     for (int i = 0; i < 6; i++) {
6.       MacString += String(info->src_addr[i], HEX);
7.       if (i < 5) {
8.         MacString += ":";
9.       }
10.    }
11.    if (find(DispositivosConectados.begin(),
   DispositivosConectados.end(), MacString) ==
   DispositivosConectados.end()) {
12.      DispositivosConectados.push_back(MacString);
13.    }
14.    if (!esp_now_is_peer_exist(info->src_addr)) {
15.      esp_now_peer_info_t InfoPeer = {};
16.      memcpy(InfoPeer.peer_addr, info->src_addr, 6);
17.      InfoPeer.channel = 0;
18.      InfoPeer.encrypt = false;
19.      if (esp_now_add_peer(&InfoPeer) != ESP_OK) {
20.        Serial.println("Error al agregar el peer
   temporalmente.");
21.        return;
22.      }
23.    }
24.    strcpy(MensajeEnviado.text, "Recibido");
25.    esp_err_t ResultadoDeEnvio = esp_now_send(info-
   >src_addr, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
26.    Serial.print(MacString);
27.    Serial.println(" está intentando conectarse.");
28.    if (ResultadoDeEnvio == ESP_OK) {
29.      Serial.println("Iniciando conexión...");
30.    } else {
31.      Serial.print("Error al iniciar conexión. Código de
   error: ");
```

```
32.       Serial.println(ResultadoDeEnvio);
33.     }
34.   }
```

- Si el mensaje recibido es “Recibido”, se almacena la dirección MAC del dispositivo que envía dicho mensaje. Esta parte del código sirve como comprobación de dos dispositivos están en alcance.

```
34.else if (strcmp(MensajeRecibido.text, "Recibido") == 0) {
35.     RespuestaRecibida = true;
36.     String MacString = "";
37.     for (int i = 0; i < 6; i++) {
38.       MacString += String(info->src_addr[i], HEX);
39.       if (i < 5) {
40.         MacString += ":";
41.       }
42.     }
43.     if (find(DispositivosConectados.begin(),
   DispositivosConectados.end(), MacString) ==
   DispositivosConectados.end()) {
44.       DispositivosConectados.push_back(MacString);
45.     }
46.   }
```

*El dispositivo A envía “Prueba de conexión” si el dispositivo B recibe “Prueba de conexión” guarda la MAC de A y envía “Recibido”. Si el dispositivo A recibe “Recibido” guarda la MAC de B.

- Procesamiento del mensaje enviado por la función “Borrar()”
  - Si los 7 primeros caracteres del mensaje recibido son “Borrado”, se almacena el mensaje recibido en la variable “mensajeRecibido” para poder tratarla y obtener la dirección MAC de quien envió el mensaje. Se imprime un mensaje para dar a conocer al usuario del dispositivo que se ha recibido una solicitud de borrado, se revisa si hay algún dispositivo conectado y se imprime un mensaje si no es así. A continuación, mediante un “for”, se recorre todos los dispositivos registrados y se compara mediante un “if” cada dirección almacenada con la del mensaje recibido. Si ambas direcciones son iguales se elimina el dispositivo de la lista de dispositivos conectados y se elimina de la red mediante la función de ESP-NOW “esp_now_del_peer”. Por último, se imprime un mensaje por pantalla informando al usuario sobre la eliminación del dispositivo y se elimina del vector “VectoresVinculados()” el dispositivo borrado junto al bloque vinculado.

```
46.else if (strncmp(MensajeRecibido.text, "Borrado", 7) == 0)
   {
47.     String mensajeRecibido = MensajeRecibido.text;
```

```
48.     int espacioPos = mensajeRecibido.indexOf(' ');
49.     if (espacioPos != -1) {
50.       String MacString =
   mensajeRecibido.substring(espacioPos + 1);
51.       Serial.print("El dispositivo ");
52.       Serial.print(MacString);
53.       Serial.println(" ha solicitado borrar su registro.");
54.       if (DispositivosConectados.empty()) {
55.         Serial.println("No hay dispositivos para borrar.");
56.       } else {
57.         for (auto it = DispositivosConectados.begin(); it
   != DispositivosConectados.end(); ++it) {
58.         if (NormalizarMac(*it) == NormalizarMac(MacString))
   {
59.           ConvertirMacStringABytes(MacString, peerMAC);
60.           DispositivosConectados.erase(it);
61.           esp_now_del_peer(peerMAC);
62.           Serial.print("El dispositivo ");
63.           Serial.print(MacString);
64.           Serial.println(" ha sido eliminado con éxito.");
65.           for (size_t i = 0; i < VectoresVinculados.size();
   i++) {
66.             if(NormalizarMac(VectoresVinculados[i]) ==
   NormalizarMac(*it)) {
67.               VectoresVinculados.erase(VectoresVinculados.b
   egin() + (i - 1));
68.               VectoresVinculados.erase(VectoresVinculados.b
   egin() + (i - 1));
69.               i -= 1;
70.             }
71.           }
72.           break;
73.         }
74.       }
75.     }
76.   }
77. }
```

- Procesamiento del mensaje enviado por la función “EditarValorDeMemoria()”
  - Si los 6 primeros caracteres del mensaje recibido son “Editar”, se procede con la siguiente parte del código. Primero se imprime un mensaje informando al usuario del mensaje recibido y se almacena el mensaje en la variable “Mensaje” para poder manipularla. Se buscan las posiciones de los espacios para poder extraer las diferentes

variables: El nombre del bloque de memorias reservado (NombreBloque), el índice donde se almacenará el dato a editar (IndiceStr) y el valor del dato (ValorStr). Además, se convierte el índice en un entero y se obtiene la MAC del dispositivo emisor (MacStr) a través de “info->src_addr”. Mediante dos “for” se recorren los vectores “Bloques()” y “DispositivosAsignados()” y con una condición “if” se comprueba cuándo el elemento de los mismos son iguales y son iguales al contenido de “NombreBloque”, ya que, para este caso concreto, el tamaño del bloque se encontrará en la posición siguiente del vector “Bloques()”. Mediante una condición “if” se comprueba si el índice introducido es válido: que sea mayor que 0 y menor que el tamaño del bloque y se prepara un mensaje de error si no es así. Y se llama a la función de “EscribirMemoria” para escribir el valor en el índice del bloque de memoria concretos.

```
77.  else if (strncmp(MensajeRecibido.text, "Editar", 6) == 0)
   {
78.    Serial.println("Mensaje recibido");
79.    Serial.println(MensajeRecibido.text);
80.
81. String Mensaje = String(MensajeRecibido.text);
82.
83.    int PrimerEspacio = Mensaje.indexOf(' ');
84.    int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio
   + 1);
85.    int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio
   + 1);
86.
87.    String NombreBloque = Mensaje.substring(PrimerEspacio +
   1, SegundoEspacio);
88.    String IndiceStr = Mensaje.substring(SegundoEspacio +
   1, TercerEspacio);
89.    String ValorStr = Mensaje.substring(TercerEspacio + 1);
90.
91.    int IndiceBytes = IndiceStr.toInt();
92.
93.    String MacStr = "";
94.    for (int i = 0; i < 6; i++) {
95.      MacStr += String(info->src_addr[i], HEX);
96.      if (i < 5) MacStr += ":";
97.    }
98.    for (size_t i = 0; i < Bloques.size(); ++i) {
99.      for (size_t j = 0; j < VectoresVinculados.size();
   ++j) {
100.          if (Bloques [i] == VectoresVinculados [j] &&
   Bloques[i] == NombreBloque) {
```

```
            uint32_t TamanoBloque = Bloques [i + 1].toInt();
            if (IndiceBytes >= 0 && IndiceBytes < TamanoBloque) {
          } else {
            Serial.println("Enviando error a emisor...");
            uint8_t MacBytes[6];
            ConvertirMacStringABytes(MacStr, MacBytes);
            strcpy(MensajeEnviado.text, "Error 3: Índice fuera de rango");
            esp_err_t Result = esp_now_send(info->src_addr, (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
            return;
          }
        }
      }
    EscribirMemoria(NombreBloque, IndiceBytes, ValorStr);
    delayMicroseconds(200);
```

- Si los siete primeros caracteres del mensaje recibido son “Error 3”, correspondiente al intento de modificar una dirección de memoria a la que un bloque no tiene acceso (el índice es mayor al tamaño del bloque), se imprime dicho mensaje de error.

```
  else if (strncmp(MensajeRecibido.text, "Error 3", 7) == 0) {
    Serial.print("Mensaje recibido: ");
    Serial.println(MensajeRecibido.text);
  }
```

- Procesamiento del mensaje enviado por la función “VincularVector()”
  - Si los ocho primeros caracteres del mensaje recibido son “Vincular”, se sacan los parámetros recibidos del mensaje: el nombre del bloque y el tamaño del bloque. Se almacena en “VectoresVinculados()” el nombre del bloque y la dirección MAC del dispositivo del que se recibe el mensaje y se preparan dos mensajes de aviso a modo de respuesta:
    + Primero se prepara el mensaje “Vinculando”.
    + A continuación se ejecuta la función de “EditarValorDeMemoria ()” y se ejecuta el flujo para escribir en memorias de otros dispositivos solo en los valores ya sobrescritos para que el contenido del bloque de memoria de todos los dispositivos sea el mismo.
    +Por último se prepara el mensaje “Vinculado”.

```
119. else if (strncmp(MensajeRecibido.text, "Vincular", 8) == 0){
```

```
120.    String Mensaje = String(MensajeRecibido.text);
121.    int PrimerEspacio = Mensaje.indexOf(' ');
122.    int SegundoEspacio = Mensaje.indexOf(' ',
   PrimerEspacio + 1);
123.    String NombreBloque = Mensaje.substring(PrimerEspacio
   + 1, SegundoEspacio);
124.    String TamanoBloque =
   Mensaje.substring(SegundoEspacio + 1);
125.    int TamanoBloqueBytes = TamanoBloque.toInt();
126.    for (size_t i = 0; i < Bloques.size(); i++){
127.      if(Bloques[i] == NombreBloque){
128.        if (Bloques[i+1] == TamanoBloque){
129.          String MacStr = "";
130.          for (int i = 0; i < 6; i++) {
131.            MacStr += String(info->src_addr[i], HEX);
132.            if (i < 5) MacStr += ":";
133.          }
134.          ConvertirMacStringABytes(MacStr,peerMAC);
135.          NormalizarMac(MacStr);
136.          VectoresVinculados.push_back(NombreBloque);
137.          VectoresVinculados.push_back(MacStr);
138.
139.          char Texto[50];
140.          sprintf(Texto, "Vinculando %s ",
   NombreBloque.c_str());
141.          strcpy(MensajeEnviado.text, Texto);
142.          esp_now_send(peerMAC,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
143.          Serial.print("El vector: ");
144.          Serial.print(NombreBloque);
145.          Serial.print(" se está intentando vincular con
   ");
146.          Serial.println(NormalizarMac(MacStr));
147.          Serial.println("Vinculando...");
148.          for (int i = 0; i < TamanoBloqueBytes; i++) {
149.            if(MostrarMemoria(NombreBloque, i) != 0) {
150.              EditarValorDeMemoria(MacStr, NombreBloque,
   i, MostrarMemoria(NombreBloque, i));
151.              delayMicroseconds(1000);
152.            }
153.          }
154.          Serial.println("Vinculación completada");
155.          sprintf(Texto, "Vinculado %s",
   NombreBloque.c_str());
156.          strcpy(MensajeEnviado.text, Texto);
157.          esp_now_send(peerMAC,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
```

```
158.         }
159.       }
160.     }
161.   }
```

- Si los diez primeros caracteres del mensaje recibido son “Vinculando”, se obtienen los parámetros del mensaje recibido y se imprime un mensaje de aviso, dando a conocer al usuario el comienzo del proceso de vinculación.

```
161. else if (strncmp(MensajeRecibido.text, "Vinculando", 10)
   == 0) {
162.     String Mensaje = String(MensajeRecibido.text);
163.     Serial.println(Mensaje);
164.     int PrimerEspacio = Mensaje.indexOf(' ');
165.     String NombreBloque = Mensaje.substring(PrimerEspacio
   + 1);
166.     String MacStr = "";
167.     for (int i = 0; i < 6; i++) {
168.       MacStr += String(info->src_addr[i], HEX);
169.       if (i < 5) MacStr += ":";
170.     }
171.     Serial.print("El dispositivo ");
172.     Serial.print(NormalizarMac(MacStr));
173.     Serial.print(" está intentando vincular el bloque ");
174.     Serial.println(NombreBloque);
175.     Serial.println("Vinculando...");
176. }
```

- Si los siete primeros caracteres del mensaje recibido son “Vinculado”, se saca el parámetro requerido del mensaje recibido, se avisa al usuario de la finalización del proceso de vinculación y se construye el vector “VectoresVinculados”, almacenando el nombre del bloque seguido de la dirección MAC a la que está vinculada.

```
176. else if (strncmp(MensajeRecibido.text, "Vinculado", 9)
   == 0) {
177.     Serial.println("Vinculación completada.");
178.     String Mensaje = String(MensajeRecibido.text);
179.     int PrimerEspacio = Mensaje.indexOf(' ');
180.     String NombreBloque = Mensaje.substring(PrimerEspacio
   + 1);
181.     String MacStr = "";
182.     for (int i = 0; i < 6; i++) {
183.       MacStr += String(info->src_addr[i], HEX);
184.       if (i < 5) MacStr += ":";
```

```
185.     }
186.     VectoresVinculados.push_back(NombreBloque);
187.     VectoresVinculados.push_back(MacStr);
188. }
```

- Procesamiento del mensaje enviado por la función “CrearVectorValoresResponsabilizados”
  - Si los 11 primeros caracteres del mensaje recibido son “Responsable”, se obtienen los parámetros del mensaje recibido, se recorre el vector “ValoresResponsabilizados”. Empezando por el final, si el dispositivo no tiene ningún valor responsabilizado, se envía de vuelta un mensaje de “ResponsExito”, indicándole al dispositivo que tiene permiso de responsabilizarse de esos índices (este tendrá que esperar a que todos los dispositivos le digan lo mismo para poder responsabilizarse de unos ínidces). Si el dispositivo sí tiene valores responsabilizados, comprobará que los índices a los que el otro dispositivo quiere tener acceso son válidos, esto es, que ningún índice de un bloque pueda ser accedido por más de un dispositivo. Si son válidos, se enviará el mensaje “ResponsExito” y en caso contrario “ResponsError”.

```
188. else if (strncmp(MensajeRecibido.text, "Responsable",
   11) == 0) {
189.     String Mensaje = String(MensajeRecibido.text);
190.     int PrimerEspacio = Mensaje.indexOf(' ');
191.     int SegundoEspacio = Mensaje.indexOf(' ',
   PrimerEspacio + 1);
192.     int TercerEspacio = Mensaje.indexOf(' ',
   SegundoEspacio + 1);
193.
194.     String NombreBloque = Mensaje.substring(PrimerEspacio
   + 1, SegundoEspacio);
195.     String StrIndiceMin =
   Mensaje.substring(SegundoEspacio + 1, TercerEspacio);
196.     String StrIndiceMax = Mensaje.substring(TercerEspacio
   + 1);
197.     uint32_t IndiceMin = StrIndiceMin.toInt();
198.     uint32_t IndiceMax = StrIndiceMax.toInt();
199.
200.     String MacStr = "";
201.     for (int i = 0; i < 6; i++) {
202.       MacStr += String(info->src_addr[i], HEX);
203.       if (i < 5) MacStr += ":";
204.       }
205.       ConvertirMacStringABytes(MacStr,peerMAC);
206.       for (size_t i = 0; i <
   ValoresResponsabilizados.size(); i++) {
```

```
207.         if (NombreBloque == ValoresResponsabilizados[i])
   {
208.           if ((IndiceMin >=
   ValoresResponsabilizados[i+1].toInt() && IndiceMin <=
   ValoresResponsabilizados[i+2].toInt()) || (IndiceMax >=
   ValoresResponsabilizados[i+1].toInt() && IndiceMax <=
   ValoresResponsabilizados[i+2].toInt()) || (IndiceMin <=
   ValoresResponsabilizados[i+1].toInt() && IndiceMax >=
   ValoresResponsabilizados[i+2].toInt())) {
209.             char Texto[100];
210.             sprintf(Texto, "ResponsError %s %d %d",
   NombreBloque.c_str(),
   ValoresResponsabilizados[i+1].toInt(),
   ValoresResponsabilizados[i+2].toInt());
211.             strcpy(MensajeEnviado.text, Texto);
212.             esp_now_send(peerMAC,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
213.           } else {
214.             char Texto[100];
215.             sprintf(Texto, "ResponsExito %s %d %d",
   NombreBloque.c_str(), IndiceMin, IndiceMax);
216.             strcpy(MensajeEnviado.text, Texto);
217.             esp_now_send(peerMAC,
   (uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
218.           }
219.         }
220.       }
221.       if (ValoresResponsabilizados.size() == 0) {
222.       char Texto[100];
223.       sprintf(Texto, "ResponsExito %s %d %d",
   NombreBloque.c_str(), IndiceMin, IndiceMax);
224.       strcpy(MensajeEnviado.text, Texto);
225.       esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
   sizeof(MensajeEnviado));
226.     }
227. }
```

- Si el mensaje recibido es “ResponsError” o “ResponsExito” se obtiene los parámetros necesarios del mensaje recibido. Por cada “ResponsError” que se recibe se añade un valor 0 al vector “Respuestas”, y por cada “ResponsExito” recibido se le añade un 1. Se recorre el vector “ListaDeRespuestasPendientes” compuesto de las direcciones MAC de los dispositivos que faltan por responder “ResponsExito” o “ResponsError” y se añade tantos unos como dispositivos haya sin responder. Esto sirve, para posteriormente, volver a recorrer el vector. Si dicho vector contiene algún uno no se procederá

con la siguiente parte del flujo. Cuando el vector solo contenga ceros, se recorre el vector “Respuestas”, si este vector contiene algún 0 es porque algún dispositivo ha enviado “ResponsError”: el dispositivo está intentando tener acceso a unos índices de un bloque que ya son accedidos por otro dispositivo, y se avisa de ello. En caso contrario, se avisa al usuario del éxito en la responsabilización y se estructura el vector “”ValoresResponsabilizados” en tríos de datos: primero el nombre del bloque, luego el índice mínimo al que se tiene acceso y luego el máximo.

```
227. else if ((strncmp(MensajeRecibido.text, "ResponsError",
   12) == 0) || ((strncmp(MensajeRecibido.text,
   "ResponsExito", 12) == 0))) {
228.     String Mensaje = String(MensajeRecibido.text);
229.     int PrimerEspacio = Mensaje.indexOf(' ');
230.     int SegundoEspacio = Mensaje.indexOf(' ',
   PrimerEspacio + 1);
231.     int TercerEspacio = Mensaje.indexOf(' ',
   SegundoEspacio + 1);
232.
233.     String NombreBloque =
   Mensaje.substring(PrimerEspacio + 1, SegundoEspacio);
234.     String StrIndiceMin =
   Mensaje.substring(SegundoEspacio + 1, TercerEspacio);
235.     String StrIndiceMax =
   Mensaje.substring(TercerEspacio + 1);
236.     uint32_t IndiceMin = StrIndiceMin.toInt();
237.     uint32_t IndiceMax = StrIndiceMax.toInt();
238.     if (strncmp(MensajeRecibido.text, "ResponsError",
   12) == 0) {
239.       Respuestas.push_back(0); // 0 = Error
240.     }
241.     if (strncmp(MensajeRecibido.text, "ResponsExito",
   12) == 0) {
242.       Respuestas.push_back(1); // 1 = Exito
243.     }
244.     String MacStr = "";
245.     for (int i = 0; i < 6; i++) {
246.       MacStr += String(info->src_addr[i], HEX);
247.       if (i < 5) MacStr += ":";
248.     }
249.     ConvertirMacStringABytes(MacStr,peerMAC);
250.     for (size_t i = 0; i <
   ListaRespuestasPendientes.size(); i++) {
251.       if (ListaRespuestasPendientes[i] == MacStr) {
252.         ListaRespuestasPendientes[i+1] = "1";
253.         break;
```

```
254.       }
255.     }
256.     bool TodosRespondieron = true;
257.     for (size_t i = 1; i <
   ListaRespuestasPendientes.size(); i += 2) {
258.       if (ListaRespuestasPendientes[i] != "1") {
259.         TodosRespondieron = false;
260.           break;
261.       }
262.     }
263.     bool ExisteError = false;
264.     if (TodosRespondieron) {
265.       for (size_t i = 0; i < Respuestas.size(); i++) {
266.         if (Respuestas[i] == 0) {
267.           ExisteError = true;
268.             break;
269.         }
270.       }
271.       if (ExisteError) {
272.         Serial.print("Error: Para el bloque ");
273.         Serial.print(NombreBloque);
274.         Serial.print(" no puedes responsabilizarte de
   unos índices entre ");
275.         Serial.print(IndiceMin);
276.         Serial.print(" y ");
277.         Serial.println(IndiceMax);
278.       } else {
279.         Serial.print("Para el bloque ");
280.         Serial.print(NombreBloque);
281.         Serial.print(" este dispositivo solo podrá
   escribir entre las posiciones ");
282.         Serial.print(IndiceMin);
283.         Serial.print(" y ");
284.         Serial.println(IndiceMax);
285.
   ValoresResponsabilizados.push_back(NombreBloque);
286.
   ValoresResponsabilizados.push_back(StrIndiceMin);
287.
   ValoresResponsabilizados.push_back(StrIndiceMax);
288.       }
289.       Respuestas.clear();
290.       ListaRespuestasPendientes.clear();
291.     }
292.   }
293. }
```

La función “enRecepciónDeDatos()” es una función compleja que permite manejar los mensajes recibidos a través de ESP-NOW y toma decisiones dependiendo del contenido del mensaje:

- Confirma la conexión entre dispositivos y los registra.
- Elimina dispositivos.
- Vincula bloques de memoria.
- Da permisos de edición a determinados índices en cada bloque.
- Modifica valores de memoria.
- Informa sobre errores.

Variables Globales

Las variables globales son aquellas que se definen fuera de cualquier función y por ello, pueden ser accedidas desde cualquier parte del código. Estas variables se utilizan para almacenar datos fuera de la definición de las propias funciones. Las variables globales creadas son las siguientes:

1. MensajeEstructurado MemoriaCompartida::MensajeEnviado;
   La variable de tipo “MensajeEstructurado” denominada “MensajeEnviado”, permite almacenar el contenido de un mensaje enviado a un dispositivo.
   El tipo de variable “MensajeEstructurado” viene definido en el archivo “MemoriaCompartida.h” se trata de una variable de tipo “struct” que contiene un único campo: una cadena de caracteres de hasta 240 posiciones.

2. MensajeEstructurado MemoriaCompartida::MensajeRecibido;
   La variable de tipo “MensajeEstructurado” denominada “MensajeRecibido”, permite almacenar el contenido de un mensaje recibido por un dispositivo.
   El tipo de variable “MensajeEstructurado” viene definido en el archivo “MemoriaCompartida.h” se trata de una variable de tipo “struct” que contiene un único campo: una cadena de caracteres de hasta 240 posiciones.

3. uint8_t MemoriaCompartida::peerMAC[6];
   La variable “peerMAC” sirve para obtener el resultado al aplicar la función “ConvertirMacStringABytes()” a lo largo de todo el código.

4. bool MemoriaCompartida::RespuestaRecibida = false;

La variable “Respuestarecibida” sirve para, en el flujo del registro de dispositivos, saber si el dispositivo que se pretende registrar está en alcance o no.

5. std::vector<String> MemoriaCompartida::DispositivosConectados;
   El vector “DispositivosConectados” sirve para almacenar las direcciones MAC del resto de dispositivos registrados por un dispositivo.
   Ejem.: Para un dispositivo A registrado a B y C, el vector “DispositivosConectados” en el dispositivo A será:
   DispositivosConectados: MAC B, MAC C

6. std::vector<uint32_t*> MemoriaCompartida::MemoriaDispositivos;
   El vector “MemoriaDispositivos” almacena los punteros que apuntan a direcciones específicas dentro de bloques de memoria creados.

7. std::vector<String> MemoriaCompartida::DispositivosAlias;
   El vector “DispositivosAlias” almacena a pares la dirección MAC de un dispositivo junto al alias introducido por el usuario si lo hay.

8. std::vector<String> MemoriaCompartida::Bloques;
   El vector “Bloques” almacena a pares el nombre de un bloque de memoria reservado acompañado de su tamaño.
   Ejem.: Para un bloque de memoria llamado Ana de tamaño 50 direcciones de memoria y un bloque de memoria llamado Bea de 100 direcciones de memoria, el vector “Bloques” en el dispositivo donde se han creado será:
   Bloques: Ana, 50, Bea, 100

9. std::vector<String> MemoriaCompartida::VectoresVinculados;
   El vector “VectoresVinculados” almacena a pares el nombre de un bloque de memoria reservado junto a la dirección MAC del dispositivo que tenga ese mismo bloque de memoria vinculado.
   Ejm.: Para un bloque de memoria llamado “Ana” vinculado con el dispositivo B y C, un bloque de memoria llamado “Bea” vinculado con el dispositivo C y un bloque de memoria llamado Carla, no vinculado con ningún dispositivo, el contenido de “VectoresVinculados” en el dispositivo A es:
   VectoresVinculados: Ana, MAC B, Ana, MAC C, Bea, MAC C

10. std::vector<String> MemoriaCompartida::ValoresResponsabilizados;
    El vector “ValoresResponsabilizados” almacena en tríos de datos el nombre de un bloque vinculado con otro dispositivo, el índice menor (de ese bloque)

al cual el dispositivo tiene acceso, y el índice mayor (de ese bloque) al cual el dispositivo tiene acceso.
Ejm.: Para un bloque de memoria vinculado entre dos dispositivos: A y B llamado Ana de tamaño 50, donde A se responsabiliza de las posiciones 1 a la 15, el vector "VectoresVinculados" en A es:
VectoresVinculados: Ana, 1, 15

11. std::vector<String> MemoriaCompartida::ListaRespuestasPendientes;
    El vector "ListaRespuestasPendientes" almacena 0's ó 1's en función de si el dispositivo ha recibido una respuesta o no en el flujo de responsabilización de índices de un bloque. Si el vector solo contiene 0's no respondió nadie, si solo contiene 1's respondieron todos.

12. std::vector<int> MemoriaCompartida::Respuestas;
    El vector "Respuestas" almacena 0's ó 1's en función de si el dispositivo ha recibido una respuesta del tipo "ResponsError" o "ResponsExito" en el flujo de responsabilización de índices de un bloque, almacenando un 0 en caso de error y un 1 en caso de éxito.

13. bool MemoriaCompartida::SePuedeEscribir;
    La variable booleana "SePuedeEscribir" permite al sistema conocer, dado el flujo, si es posible escribir en un bloque de memoria creado en un dispositivo o no.

Contexto

Alcance

Esta librería permite la intercomunicación entre nodos para la simulación de una memoria compartida entre dispositivos ESP32 usando ESP-NOW. La librería proporciona mecanismos para el envío y recepción de estructuras de datos permitiendo sincronizar la información entre los diferentes nodos de forma eficiente y sin necesidad de una infraestructura de red.

Por ello, el alcance de este proyecto se limita a:

1. Uso exclusivo del protocolo ESP-NOW para la comunicación inalámbrica entre nodos. Este proyecto no incluye la integración con otros protocolos que no sean ESP-NOW como puede ser MQTT o LoRa o WiFi tradicional.
2. La implementación y validación de la librería únicamente sobre dispositivos ESP32. No se asegura que todas las funcionalidades

diseñadas en la librería deban funcionar con otro tipo de microprocesador.

Oportunidades

La librería puede usarse en cualquier entorno, ya sea profesional o de ocio, y con cualquier nivel de conocimiento técnico para el desarrollo de herramientas software donde la intercomunicación entre dispositivos en entornos sin infraestructura WI-Fi sea necesaria.

La librería es especialmente útil en: entornos IoT y de domótica donde la alta velocidad de transferencia de datos sea necesaria, y en proyectos de sensorización en zonas remotas.

Limitaciones

En cuanto a las limitaciones técnicas de la librería, cabe destacar:

1. Tamaño de los mensajes: el tamaño de los mensajes enviados entre dispositivos no deberá ser mayor a 250 bytes.
2. Cantidad de dispositivos conectados: la cantidad de dispositivos a registrar no deberá ser mayor a 20.
3. Memoria pequeña: los dispositivos esp32 suelen tener una memoria pequeña que puede no ser suficiente para el tipo de proyecto que se esté realizando. Aunque como solución sencilla podría usarse una memoria externa.
4. Distancia entre dispositivos máxima (para que ESP-NOW funcione): la distancia máxima, según el proveedor, en la que la tecnología esp-now funciona es de 200 metros.

Conclusiones e Implicaciones

En este proyecto se ha logrado desarrollar una librería funcional para la simulación de una memoria compartida entre dispositivos ESP32 usando el protocolo de comunicación inalámbrica ESP-NOW. Este objetivo principal ha sido cumplido, proporcionando una herramienta que permite la sincronización entre nodos para la compartición de estructuras de datos entre ellos, de forma eficiente y sin necesidad de infraestructura de red adicional.

La solución implementada ha demostrado que es posible abstraer la comunicación inalámbrica entre microprocesadores como si de una memoria compartida se tratara a través del uso de una librería, facilitando el uso y por ello, el desarrollo de herramientas en computación distribuida. La librería ha sido validada, confirmando su correcto funcionamiento mediante la implementación de pruebas empleando dispositivos ESP32.

Como posibles líneas de trabajo futuras, se propone:

1. Añadir integración con otros protocolos de comunicación como MQTT, LoRa o WiFi para una interoperabilidad híbrida.
2. Asegurar el soporte con otros microprocesadores.

En definitiva, este proyecto no solo presenta una solución concreta a un problema planteado, sino que también establece una base sólida para futuros desarrollos de proyectos relacionados con la intercomunicación entre dispositivos, de forma eficiente, sencilla y sin infraestructura externa.

Bibliografía

1. https://docs.keyestudio.com/projects/KS5016/en/latest/docs/
2. https://rmd.jcyl.es/web/es/territorio-rural-inteligente.html
3. https://wiki.keyestudio.com/KS5016_Keyestudio_ESP32_PLUS_Development_Board
4. https://www.arsys.es/blog/codigo-bloqueante-nodejs
5. https://www.electrosoftcloud.com/esp-now-conecta-dos-o-mas-esp32-esp8266/
6. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf
7. https://www.guiahardware.es/zigbee/
8. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-un-callback/
9. https://www.luisllamas.es/como-usar-esp-now-en-esp32/
10. https://www.matternet.com/
11. https://www.siemens.com/es/es/productos/software/mindsphere.html
12. https://www.silabs.com/wireless/technology
13. https://www.ti.com/lit/ds/symlink/cc2530.pdf?ts=1749028684669&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fes-mx%252FCC2530
14. https://www.youtube.com/watch?v=bEKjCDDUPaU
15. https://www.youtube.com/watch?v=QmvMtgNs9r8
16. https://www.youtube.com/watch?v=VGoiUk-jkjE&list=PL-Hb9zZP9qC40uOyYYYPrGeAClL2z97oU

Apéndice

Caso práctico de utilización:

En una red con tres dispositivos: A, B y C se quiere simular una memoria compartida de la siguiente forma:

- Entre A y B se creará un Bloque Ana de tamaño 50, con los siguientes accesos:
    - A tendrá acceso a los índices del 0 al 25
    - B tendrá acceso a los índices del 26 al 40
- Entre B y C se creará un Bloque Bea de tamaño 100, con los siguientes accesos:
    - B tendrá acceso a los índices del 0 al 50 y del 75 al 99.
    - C tendrá acceso a los índices del 51 al 74.
- Entre A, B y C se creará un Bloque Carla de tamaño 200, con los siguientes accesos:
    - A tendrá acceso a los índices del 0 al 25
    - B tendrá acceso del 26 al 75
    - C tendrá acceso del 76 al 150.

Se comienza con el registro de los dispositivos:
En A se registra B y C, y en B se registra C, de esta manera los 3 dispositivos estarán registrados entre sí. Para el registro se utilizarán las direcciones MAC sacadas por pantalla nada más ejecutar el programa:
El dispositivo A será el de la MAC: “EC:64:C9:99:CF:78”

```
18:43:30.212 -> MAC del dispositivo: EC:64:C9:99:CF:78
18:43:30.250 -> ESP-NOW iniciado correctamente.
18:43:32.256 -> ESP-32 listo.
```

El dispositivo B será el de la MAC: “EC:64:C9:99:D1:88”

```
18:44:29.858 -> MAC del dispositivo: EC:64:C9:99:D1:88
18:44:29.858 -> ESP-NOW iniciado correctamente.
18:44:31.857 -> ESP-32 listo.
```

El dispositivo C será el de la MAC: “34:98:7A:BC:6F:DC”

```
18:50:38.806 -> MAC del dispositivo: 34:98:7A:BC:6F:DC
18:50:38.806 -> ESP-NOW iniciado correctamente.
18:50:40.811 -> ESP-32 listo.
```

Para registrar B y C desde A:

```
18:54:25.430 -> Registrar EC:64:C9:99:D1:88
18:54:25.430 -> Iniciando conexión con: EC:64:C9:99:D1:88
18:54:25.477 -> Conectando...
18:54:25.477 -> No hay bloques creados para vincular.
18:54:29.773 -> Registrar 34:98:7A:BC:6F:DC
18:54:29.773 -> Iniciando conexión con: 34:98:7A:BC:6F:DC
18:54:29.811 -> Conectando...
18:54:29.811 -> No hay bloques creados para vincular.
```

Y para registrar C desde B:


```
18:56:10.049 -> Registrar 34:98:7A:BC:6F:DC
18:56:10.049 -> Iniciando conexión con: 34:98:7A:BC:6F:DC
18:56:10.049 -> Conectando...
18:56:10.049 -> No hay bloques creados para vincular.
```

Ahora, todos los dispositivos estarán registrados. Esto se comprobará ejecutando la función de Dispositivos en cada dispositivo:

En el dispositivo A:

```
18:58:53.468 -> Dispositivo local:
18:58:53.468 -> EC:64:C9:99:CF:78
18:58:53.469 -> Dispositivos conectados:
18:58:53.469 -> EC:64:C9:99:D1:88
18:58:53.469 -> 34:98:7A:BC:6F:DC
18:58:53.469 ->
```

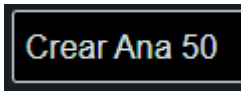
En el dispositivo B:

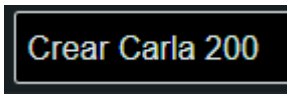
```
18:59:24.392 -> Dispositivo local:
18:59:24.392 -> EC:64:C9:99:D1:88
18:59:24.392 -> Dispositivos conectados:
18:59:24.392 -> EC:64:C9:99:CF:78
18:59:24.392 -> 34:98:7A:BC:6F:DC
18:59:24.392 ->
```

En el dispositivo C:

```
18:59:48.627 -> Dispositivo local:
18:59:48.627 -> 34:98:7A:BC:6F:DC
18:59:48.627 -> Dispositivos conectados:
18:59:48.627 -> EC:64:C9:99:CF:78
18:59:48.627 -> EC:64:C9:99:D1:88
18:59:48.627 ->
```

Una vez comprobado que todos los dispositivos están registrados entre sí, se procede con la creación de los bloques de memoria especificados:
En A se crea el bloque Ana de tamaño 50 y Carla de tamaño 200:

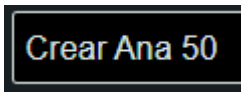


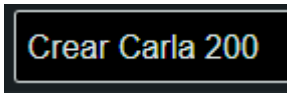


Ambos bloques se intentarán vincular con B y C nada más ser creados, pero no se vincularán por que estos dispositivos aún no tienen creado ningún bloque:

```
19:02:43.279 -> El vector: Ana se está intentando vincular con EC:64:C9:99:D1:88
19:02:43.279 -> El vector: Ana se está intentando vincular con 34:98:7A:BC:6F:DC
19:02:43.279 -> El vector: Carla se está intentando vincular con EC:64:C9:99:D1:88
19:02:43.327 -> El vector: Carla se está intentando vincular con 34:98:7A:BC:6F:DC
```

En B se crea el bloque Ana 50 (que se vinculará automáticamente con el bloque Ana de A), Bea 100 y Carla 200 (que se vinculará automáticamente con el bloque Carla de A):






```
19:06:37.332 -> Bloque de memoria 'Ana' creado con tamaño de 50 direcciones de memoria.
19:06:37.332 -> El vector: Ana se está intentando vincular con EC:64:C9:99:CF:78
19:06:37.332 -> El vector: Ana se está intentando vincular con 34:98:7A:BC:6F:DC
19:06:37.332 -> Vinculando Ana
19:06:37.332 -> El dispositivo EC:64:C9:99:CF:78 está intentando vincular el bloque Ana
19:06:37.377 -> Vinculando...
19:06:37.377 -> Vinculación completada.
19:06:44.000 -> Bloque de memoria 'Bea' creado con tamaño de 100 direcciones de memoria.
19:06:44.000 -> El dispositivo EC:64:C9:99:CF:78 ya tiene vinculado el bloque Ana
19:06:44.000 -> El vector: Ana se está intentando vincular con 34:98:7A:BC:6F:DC
19:06:44.000 -> El vector: Bea se está intentando vincular con EC:64:C9:99:CF:78
19:06:44.000 -> El vector: Bea se está intentando vincular con 34:98:7A:BC:6F:DC
```

```
19:06:50.166 -> Bloque de memoria 'Carla' creado con tamaño de 200 direcciones de memoria.
19:06:50.166 -> El dispositivo EC:64:C9:99:CF:78 ya tiene vinculado el bloque Ana
19:06:50.166 -> El vector: Ana se está intentando vincular con 34:98:7A:BC:6F:DC
19:06:50.166 -> El vector: Bea se está intentando vincular con EC:64:C9:99:CF:78
19:06:50.166 -> El vector: Bea se está intentando vincular con 34:98:7A:BC:6F:DC
19:06:50.209 -> El vector: Carla se está intentando vincular con EC:64:C9:99:CF:78
19:06:50.209 -> El vector: Carla se está intentando vincular con 34:98:7A:BC:6F:DC
19:06:50.209 -> Vinculando Carla
19:06:50.209 -> El dispositivo EC:64:C9:99:CF:78 está intentando vincular el bloque Carla
19:06:50.209 -> Vinculando...
19:06:50.209 -> Vinculación completada.
```

En C se crea el bloque Bea 100 (que se vinculará automáticamente con el bloque Bea de B) y el bloque Carla 200 (que se vinculará automáticamente con el bloque Carla de A y B):

```
Crear Bea 100
```

```
Crear Carla 200
```

```
19:13:05.023 -> Bloque de memoria 'Bea' creado con tamaño de 100 direcciones de memoria.
19:13:05.023 -> El vector: Bea se está intentando vincular con EC:64:C9:99:CF:78
19:13:05.023 -> El vector: Bea se está intentando vincular con EC:64:C9:99:D1:88
19:13:05.074 -> Vinculando Bea
19:13:05.074 -> El dispositivo EC:64:C9:99:D1:88 está intentando vincular el bloque Bea
19:13:05.074 -> Vinculando...
19:13:05.074 -> Vinculación completada.
```

```
19:14:02.596 -> Bloque de memoria 'Carla' creado con tamaño de 200 direcciones de memoria.
19:14:02.627 -> El vector: Bea se está intentando vincular con EC:64:C9:99:CF:78
19:14:02.627 -> El dispositivo EC:64:C9:99:D1:88 ya tiene vinculado el bloque Bea
19:14:02.627 -> El vector: Carla se está intentando vincular con EC:64:C9:99:CF:78
19:14:02.627 -> El vector: Carla se está intentando vincular con EC:64:C9:99:D1:88
19:14:02.675 -> Vinculando Carla
19:14:02.675 -> El dispositivo EC:64:C9:99:CF:78 está intentando vincular el bloque Carla
19:14:02.675 -> Vinculando...
19:14:02.675 -> Vinculando Carla
19:14:02.675 -> El dispositivo EC:64:C9:99:D1:88 está intentando vincular el bloque Carla
19:14:02.675 -> Vinculando...
19:14:02.675 -> Vinculación completada.
```

Los bloques ya están vinculados según se ha definido al inicio. A continuación, se procede con la obtención de acceso de cada dispositivo a cada bloque.
A debe tener acceso en el bloque Ana a los índices del 0 al 25 y en el bloque Carla igual:

```
Acceso Ana 0 25
```

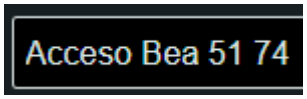
```
Acceso Carla 0  25
```

```
19:23:05.869 -> Para el bloque Ana este dispositivo solo podrá escribir entre las posiciones 0 y 25
19:23:53.148 -> Para el bloque Carla este dispositivo solo podrá escribir entre las posiciones 0 y 25
```

B debe tener acceso en el bloque Ana a los índices del 26 al 40, en el bloque Bea a los índices del 0 al 50 y del 75 al 99, y en el bloque Carla debe tener acceso a los índices del 26 al 75:


```
19:25:14.120 -> Para el bloque Ana este dispositivo solo podrá escribir entre las posiciones 26 y 40
19:25:54.985 -> Para el bloque Bea este dispositivo solo podrá escribir entre las posiciones 0 y 50
19:26:12.506 -> Para el bloque Bea este dispositivo solo podrá escribir entre las posiciones 75 y 99
19:26:55.556 -> Para el bloque Carla este dispositivo solo podrá escribir entre las posiciones 26 y 75
```

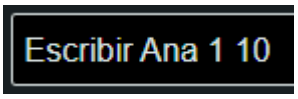
C debe tener acceso en el bloque Bea a los índices del 51 al 74, y en el bloque Carla los índices del 76 al 150:




```
19:28:01.123 -> Para el bloque Bea este dispositivo solo podrá escribir entre las posiciones 51 y 74
19:28:24.717 -> Para el bloque Carla este dispositivo solo podrá escribir entre las posiciones 76 y 150
```

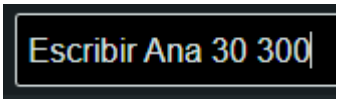
Como comprobación,
En A intentará escribir en:

- Bloque Ana, índice 1



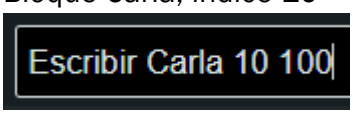

```
19:40:54.843 -> Escrito el valor 10 en la dirección de memoria reservada 1 del bloque de memoria Ana
```

- Bloque Ana, índice 30



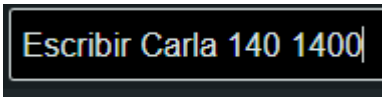

```
19:41:33.903 -> Error: Para el bloque Ana solo puedes escribir entre 0 y 25
```

- Bloque Carla, índice 10




```
19:43:15.489 -> Escrito el valor 100 en la dirección de memoria reservada 10 del bloque de memoria Carla
```

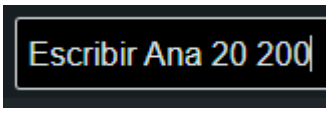
- Bloque Carla, índice 140




```
19:43:56.880 -> Error: Para el bloque Carla solo puedes escribir entre 0 y 25
```

En B se intentará escribir en:

- Bloque Ana, índice 20

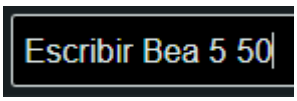



```
19:50:28.709 -> Error: Para el bloque Ana solo puedes escribir entre 26 y 40
```

- Bloque Ana, índice 35

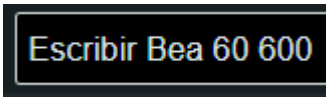

```
19:51:20.290 -> Escrito el valor 350 en la dirección de memoria reservada 35 del bloque de memoria Ana
```

- Bloque Bea, índice 5




```
19:54:21.898 -> Valor del bloque de memoria Bea en la dirección de memoria reservada 5: 50
```

- Bloque Bea, índice 60

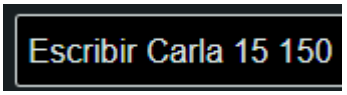



```
19:56:48.489 -> Error: Para el bloque Bea solo puedes escribir entre 0 y 50
19:56:48.489 -> Error: Para el bloque Bea solo puedes escribir entre 75 y 99
```

- Bloque Bea, índice 80


```
19:58:21.256 -> Escrito el valor 800 en la dirección de memoria reservada 80 del bloque de memoria Bea
```

- Bloque Carla, índice 15

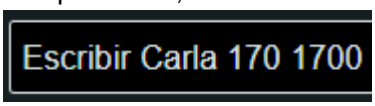



```
20:00:48.848 -> Error: Para el bloque Carla solo puedes escribir entre 26 y 75
```

- Bloque Carla, índice 40


```
20:01:38.034 -> Escrito el valor 400 en la dirección de memoria reservada 40 del bloque de memoria Carla
```

- Bloque Carla, índice 170




```
20:00:48.848 -> Error: Para el bloque Carla solo puedes escribir entre 26 y 75
```

En C se intentará escribir en:

- Bloque Bea, índice 45


```
20:04:43.912 -> Error: Para el bloque Bea solo puedes escribir entre 51 y 74
```

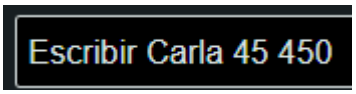
- Bloque Bea, índice 70


```
20:05:32.859 -> Escrito el valor 700 en la dirección de memoria reservada 70 del bloque de memoria Bea
```

- Bloque Carla, índice 0


```
20:08:02.431 -> Error: Para el bloque Carla solo puedes escribir entre 76 y 150
```

- Bloque Carla, índice 45




```
20:08:02.431 -> Error: Para el bloque Carla solo puedes escribir entre 76 y 150
```

- Bloque Carla, índice 110


```
20:10:03.290 -> Escrito el valor 1100 en la dirección de memoria reservada 110 del bloque de memoria Carla
```

Y se comprueba también, que en los casos donde sí se ha podido escribir, el cambio se refleja en el resto de bloques vinculados del resto de dispositivos: Cuando A escribe en Ana en el índice 1 un 10, en B aparece:


```
19:46:42.256 -> Valor del bloque de memoria Ana en la dirección de memoria reservada 1: 10
```

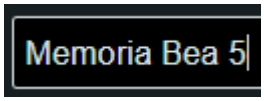
Cuando A escribe en Carla en el índice 10 un 100, en B y C aparece:


```
19:49:37.715 -> Valor del bloque de memoria Carla en la dirección de memoria reservada 10: 100
```

Cuando B escribe en Ana en el índice 35 un 350, en A aparece:

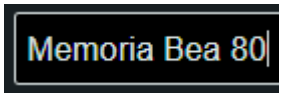

```
19:52:56.728 -> Valor del bloque de memoria Ana en la dirección de memoria reservada 35: 350
```

Cuando B escribe en Bea en el índice 5 un 50, en C aparece:




```
19:56:05.663 -> Valor del bloque de memoria Bea en la dirección de memoria reservada 5: 50
```

Cuando B escribe en Bea en el índice 80 un 800, en C aparece:

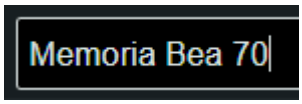



```
19:59:25.492 -> Valor del bloque de memoria Bea en la dirección de memoria reservada 80: 800
```

Cuando B escribe en Carla en el índice 40 un 400, en B y C aparece:

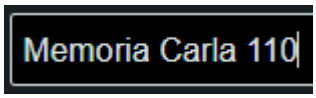

```
20:02:53.821 -> Valor del bloque de memoria Carla en la dirección de memoria reservada 40: 400
```

Cuando C escribe en Bea en el índice 70 un 700 aparece en B:




```
20:07:13.137 -> Valor del bloque de memoria Bea en la dirección de memoria reservada 70: 700
```

Cuando C escribe en Carla en el índice 110 un 1100 aparece en A y B:




```
20:11:29.500 -> Valor del bloque de memoria Carla en la dirección de memoria reservada 110: 1100
```

Anexo A: MemoriaCompartida.h

```
#ifndef MEMORIACOMPARTIDA_H
#define MEMORIACOMPARTIDA_H

#include <esp_now.h>
#include <WiFi.h>
#include <vector>
#include <string>
#include <stdlib.h>
#include <stdint.h>

typedef struct MensajeEstructurado{
  char text[240];
} Mensaje_Estructurado;

class MemoriaCompartida{
public:
  MemoriaCompartida();
  void init();
  void Registrar(const String& mac);
  void Borrar(String& Mac);
  void Dispositivos();
  static void EscribirMemoria(String NombreBloque, int Indice,
String Valor);
  static void EditarValorDeMemoria(String MacReceptor, String
NombreBloque, int Indice, int Valor);
  static void enRecepcionDeDatos(const esp_now_recv_info* info,
const uint8_t* DatosRecibidos, int tamaño);
  static void enEnvioDeDatos (const uint8_t* MAC,
esp_now_send_status_t status);
  static bool ConvertirMacStringABytes(const String& macString,
uint8_t* macBytes);
  static String NormalizarMac(const String& Mac);
  static uint32_t MostrarMemoria(String NombreBloque, int Indice);
  static std::vector<uint32_t*> MemoriaDispositivos;
  void Alias ();
  static bool esAlias(String Alias);
  static String ConvertirAliasAMacString(String Alias);
  void AsignarVectorMemoriaCompartida(String Mac, String
NombreBloque);
  static void CrearBloqueMemoria(String NombreBloque, uint32_t
TamanoBloque);
  static void VincularVector();
```

```
  static void EnviarValorVinculado(String NombreBloque, int Indice,
String Valor);
  static uint32_t InfoMostrarMemoria(String NombreBloque, int
Indice);
  static void CrearVectorValoresResponsabilizados(String
NombreBloque, int IndiceMin, int IndiceMax);

private:
  static MensajeEstructurado MensajeEnviado;
  static MensajeEstructurado MensajeRecibido;
  static uint8_t peerMAC[6];
  static bool RespuestaRecibida;
  static std::vector<String> DispositivosConectados;
  static bool esDireccionMac(const String& mac);
  static std::vector<String> DispositivosAlias;
  static std::vector<String> Bloques;

  static std::vector<String> VectoresVinculados;
  static std::vector<String> ValoresResponsabilizados;
  static std::vector<String> ListaRespuestasPendientes;
  static std::vector<int> Respuestas;
  static bool SePuedeEscribir;

};

#endif
```

Anexo B: MemoriaCompartida.cpp

```cpp
#include "Memoriacompartida.h"

MensajeEstructurado MemoriaCompartida::MensajeEnviado;
MensajeEstructurado MemoriaCompartida::MensajeRecibido;
uint8_t MemoriaCompartida::peerMAC[6];
bool MemoriaCompartida::RespuestaRecibida = false;
std::vector<String> MemoriaCompartida::DispositivosConectados;
std::vector<uint32_t*> MemoriaCompartida::MemoriaDispositivos;
std::vector<String> MemoriaCompartida::DispositivosAlias;
std::vector<String> MemoriaCompartida::Bloques;
std::vector<String> MemoriaCompartida::VectoresVinculados;
std::vector<String> MemoriaCompartida::ValoresResponsabilizados;
std::vector<String> MemoriaCompartida::ListaRespuestasPendientes;
std::vector<int> MemoriaCompartida::Respuestas;
bool MemoriaCompartida::SePuedeEscribir;

MemoriaCompartida::MemoriaCompartida() {}

bool MemoriaCompartida::esDireccionMac(const String& mac) {
  if (mac.length() != 17) {
    return false;
  }
  for (int i = 0; i < mac.length(); i++) {
    if (i % 3 == 2) {
      if (mac[i] != ':') {
        return false;
      }
    } else if (!isxdigit(mac[i])) {
      return false;
    }
  }
  return true;
}

bool MemoriaCompartida::ConvertirMacStringABytes(const String&
macString, uint8_t* macBytes) {
  int elementos[6];
  if (sscanf(macString.c_str(), "%x:%x:%x:%x:%x:%x", &elementos[0],
&elementos[1], &elementos[2], &elementos[3], &elementos[4],
&elementos[5]) == 6) {
    for (int i = 0; i < 6; i++) {
      macBytes[i] = (uint8_t)elementos[i];
    }
    return true;
  }
```

```
  return false;
}

void MemoriaCompartida::Registrar(const String& mac) {
  if (esDireccionMac(mac)) {
    if (!ConvertirMacStringABytes(mac, peerMAC)) {
      Serial.println("Error al convertir la Mac.");
      return;
    }
    uint8_t MacBytesLocal[6];
    WiFi.macAddress(MacBytesLocal);
    bool MismaMac = true;
    for (int i = 0; i < 6; i++) {
      if (peerMAC[i] != MacBytesLocal[i]) {
        MismaMac = false;
        break;
      }
    }
    if (MismaMac) {
      Serial.println("Error: La MAC introducida es la de este
dispositivo.");
      return;
    }
    esp_now_peer_info_t InfoPeer = {};
    memcpy(InfoPeer.peer_addr, peerMAC, 6);
    InfoPeer.channel = 0;
    InfoPeer.encrypt = false;
    if (esp_now_is_peer_exist(peerMAC)) {
      Serial.println("Error: Este dispositivo ya está registrado.");
      return;
    } else {
      esp_err_t Resultado = esp_now_add_peer(&InfoPeer);
      if (Resultado != ESP_OK) {
        Serial.println("Error al agregar peer temporalmente");
        return;
      }
    }
    strcpy(MensajeEnviado.text, "Prueba de conexión");
    Serial.print("Iniciando conexión con: ");
    Serial.println(mac);
    esp_err_t ResultadoDeEnvio = esp_now_send(peerMAC,
(uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
    if (ResultadoDeEnvio == ESP_OK) {
      Serial.println("Conectando...");
      RespuestaRecibida = false;
      unsigned long Contador = millis();
      while (millis() - Contador < 20000) {
```

```
        if (RespuestaRecibida == true) {
          esp_now_add_peer(&InfoPeer);
          break;
        }
      }
      if (!RespuestaRecibida) {
        Serial.println("No se pudo conectar. Eliminando
dispositivo.");
        esp_now_del_peer(peerMAC);
      }
    } else {
      Serial.print("Error al conectar. Código de error: ");
      Serial.println(ResultadoDeEnvio);
      esp_now_del_peer(peerMAC);
    }
  } else {
    Serial.println("Dirección MAC no válida.");
  }
}

void MemoriaCompartida::enRecepcionDeDatos(const esp_now_recv_info*
info, const uint8_t* DatosRecibidos, int tamaño) {
  memcpy(&MensajeRecibido, DatosRecibidos, sizeof(MensajeRecibido));
  if (strcmp(MensajeRecibido.text, "Prueba de conexión") == 0) {
    String MacString = "";
    for (int i = 0; i < 6; i++) {
      MacString += String(info->src_addr[i], HEX);
      if (i < 5) {
        MacString += ":";
      }
    }
    if (find(DispositivosConectados.begin(),
DispositivosConectados.end(), MacString) ==
DispositivosConectados.end()) {
      DispositivosConectados.push_back(MacString);
    }
    if (!esp_now_is_peer_exist(info->src_addr)) {
      esp_now_peer_info_t InfoPeer = {};
      memcpy(InfoPeer.peer_addr, info->src_addr, 6);
      InfoPeer.channel = 0;
      InfoPeer.encrypt = false;
      if (esp_now_add_peer(&InfoPeer) != ESP_OK) {
        Serial.println("Error al agregar el peer temporalmente.");
        return;
      }
    }
    strcpy(MensajeEnviado.text, "Recibido");
```

```
    esp_err_t ResultadoDeEnvio = esp_now_send(info->src_addr,
(uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
    Serial.print(NormalizarMac(MacString));
    Serial.println(" está intentando conectarse.");
    if (ResultadoDeEnvio == ESP_OK) {
      Serial.println("Iniciando conexión...");
    } else {
      Serial.print("Error al iniciar conexión. Código de error: ");
      Serial.println(ResultadoDeEnvio);
    }
  } else if (strcmp(MensajeRecibido.text, "Recibido") == 0) {
    RespuestaRecibida = true;
    String MacString = "";
    for (int i = 0; i < 6; i++) {
      MacString += String(info->src_addr[i], HEX);
      if (i < 5) {
        MacString += ":";
      }
    }
    if (find(DispositivosConectados.begin(),
DispositivosConectados.end(), MacString) ==
DispositivosConectados.end()) {
      DispositivosConectados.push_back(MacString);
      VincularVector();
    }
  } else if (strncmp(MensajeRecibido.text, "Borrado", 7) == 0) {
    String mensajeRecibido = MensajeRecibido.text;
    int espacioPos = mensajeRecibido.indexOf(' ');
    if (espacioPos != -1) {
      String MacString = mensajeRecibido.substring(espacioPos + 1);
      Serial.print("El dispositivo ");
      Serial.print(MacString);
      Serial.println(" ha solicitado borrar su registro.");
      if (DispositivosConectados.empty()) {
        Serial.println("No hay dispositivos para borrar.");
      } else {
        for (auto it = DispositivosConectados.begin(); it !=
DispositivosConectados.end(); ++it) {
          if (NormalizarMac(*it) == NormalizarMac(MacString)) {
            ConvertirMacStringABytes(MacString, peerMAC);
            DispositivosConectados.erase(it);
            esp_now_del_peer(peerMAC);
            Serial.print("El dispositivo ");
            Serial.print(MacString);
            Serial.println(" ha sido eliminado.");
```

```
            for (size_t i = 0; i < VectoresVinculados.size(); i++) {
              if(NormalizarMac(VectoresVinculados[i]) ==
NormalizarMac(*it)) {
                VectoresVinculados.erase(VectoresVinculados.begin()
+ (i - 1));
                VectoresVinculados.erase(VectoresVinculados.begin()
+ (i - 1));
                i -= 1;
                }
              }


            break;
          }
        }
      }
    }
  } else if (strncmp(MensajeRecibido.text, "Editar", 6) == 0) {
    String Mensaje = String(MensajeRecibido.text);

    int PrimerEspacio = Mensaje.indexOf(' ');
    int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio + 1);
    int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio + 1);

    String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
SegundoEspacio);
    String IndiceStr = Mensaje.substring(SegundoEspacio + 1,
TercerEspacio);
    String ValorStr = Mensaje.substring(TercerEspacio + 1);

    int IndiceBytes = IndiceStr.toInt();

    String MacStr = "";
    for (int i = 0; i < 6; i++) {
      MacStr += String(info->src_addr[i], HEX);
      if (i < 5) MacStr += ":";
    }
    for (size_t i = 0; i < Bloques.size(); ++i) {
      for (size_t j = 0; j < VectoresVinculados.size(); ++j) {
        if (Bloques [i] == VectoresVinculados [j] && Bloques[i] ==
NombreBloque) {
          uint32_t TamanoBloque = Bloques [i + 1].toInt();
          if (IndiceBytes >= 0 && IndiceBytes < TamanoBloque) {

          } else {
            Serial.println("Enviando error a emisor...");
            uint8_t MacBytes[6];
```

```
            ConvertirMacStringABytes(MacStr, MacBytes);
            strcpy(MensajeEnviado.text, "Error 3: Índice fuera de
rango");
            esp_err_t Result = esp_now_send(info->src_addr,
(uint8_t*)&MensajeEnviado, sizeof(MensajeEnviado));
            return;
          }
        }
      }
    }
    EscribirMemoria(NombreBloque, IndiceBytes, ValorStr);
    delayMicroseconds(200);
  } else if (strncmp(MensajeRecibido.text, "Error 1", 7) == 0) {
    Serial.print("Mensaje recibido: ");
    Serial.println(MensajeRecibido.text);
  } else if (strncmp(MensajeRecibido.text, "Error 2", 7) == 0) {
    Serial.print("Mensaje recibido: ");
    Serial.println(MensajeRecibido.text);
  } else if (strncmp(MensajeRecibido.text, "Error 3", 7) == 0) {
    Serial.print("Mensaje recibido: ");
    Serial.println(MensajeRecibido.text);
  } else if (strncmp(MensajeRecibido.text, "Vincular", 8) == 0){
    String Mensaje = String(MensajeRecibido.text);
    int PrimerEspacio = Mensaje.indexOf(' ');
    int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio + 1);
    String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
SegundoEspacio);
    String TamanoBloque = Mensaje.substring(SegundoEspacio + 1);
    int TamanoBloqueBytes = TamanoBloque.toInt();
    for (size_t i = 0; i < Bloques.size(); i++){
      if(Bloques[i] == NombreBloque){
        if (Bloques[i+1] == TamanoBloque){
          String MacStr = "";
          for (int i = 0; i < 6; i++) {
            MacStr += String(info->src_addr[i], HEX);
            if (i < 5) MacStr += ":";
          }
          ConvertirMacStringABytes(MacStr,peerMAC);
          NormalizarMac(MacStr);
          VectoresVinculados.push_back(NombreBloque);
          VectoresVinculados.push_back(MacStr);

          char Texto[50];
          sprintf(Texto, "Vinculando %s ", NombreBloque.c_str());
          strcpy(MensajeEnviado.text, Texto);
          esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
```

```
          Serial.print("El vector: ");
          Serial.print(NombreBloque);
          Serial.print(" se está intentando vincular con ");
          Serial.println(NormalizarMac(MacStr));
          Serial.println("Vinculando...");
          for (int i = 0; i < TamanoBloqueBytes; i++) {
            if(MostrarMemoria(NombreBloque, i) != 0) {
              EditarValorDeMemoria(MacStr, NombreBloque, i,
MostrarMemoria(NombreBloque, i));
              delayMicroseconds(1000);
            }
          }
          Serial.println("Vinculación completada");
          sprintf(Texto, "Vinculado %s", NombreBloque.c_str());
          strcpy(MensajeEnviado.text, Texto);
          esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
        }
      }
    }
  } else if (strncmp(MensajeRecibido.text, "Vinculando", 10) == 0) {
    String Mensaje = String(MensajeRecibido.text);
    Serial.println(Mensaje);
    int PrimerEspacio = Mensaje.indexOf(' ');
    String NombreBloque = Mensaje.substring(PrimerEspacio + 1);
    String MacStr = "";
    for (int i = 0; i < 6; i++) {
      MacStr += String(info->src_addr[i], HEX);
      if (i < 5) MacStr += ":";
    }
    Serial.print("El dispositivo ");
    Serial.print(NormalizarMac(MacStr));
    Serial.print(" está intentando vincular el bloque ");
    Serial.println(NombreBloque);
    Serial.println("Vinculando...");
  } else if (strncmp(MensajeRecibido.text, "Vinculado", 9) == 0) {
    Serial.println("Vinculación completada.");
    String Mensaje = String(MensajeRecibido.text);
    int PrimerEspacio = Mensaje.indexOf(' ');
    String NombreBloque = Mensaje.substring(PrimerEspacio + 1);
    String MacStr = "";
    for (int i = 0; i < 6; i++) {
      MacStr += String(info->src_addr[i], HEX);
      if (i < 5) MacStr += ":";
    }
    VectoresVinculados.push_back(NombreBloque);
    VectoresVinculados.push_back(MacStr);
```

```
  } else if (strncmp(MensajeRecibido.text, "Responsable", 11) == 0)
{
    String Mensaje = String(MensajeRecibido.text);
    int PrimerEspacio = Mensaje.indexOf(' ');
    int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio + 1);
    int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio + 1);

    String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
SegundoEspacio);
    String StrIndiceMin = Mensaje.substring(SegundoEspacio + 1,
TercerEspacio);
    String StrIndiceMax = Mensaje.substring(TercerEspacio + 1);
    uint32_t IndiceMin = StrIndiceMin.toInt();
    uint32_t IndiceMax = StrIndiceMax.toInt();

    String MacStr = "";
      for (int i = 0; i < 6; i++) {
        MacStr += String(info->src_addr[i], HEX);
        if (i < 5) MacStr += ":";
      }
    ConvertirMacStringABytes(MacStr,peerMAC);
    for (size_t i = 0; i < ValoresResponsabilizados.size(); i++) {
      if (NombreBloque == ValoresResponsabilizados[i]) {
        if ((IndiceMin >= ValoresResponsabilizados[i+1].toInt() &&
IndiceMin <= ValoresResponsabilizados[i+2].toInt()) || (IndiceMax >=
ValoresResponsabilizados[i+1].toInt() && IndiceMax <=
ValoresResponsabilizados[i+2].toInt()) || (IndiceMin <=
ValoresResponsabilizados[i+1].toInt() && IndiceMax >=
ValoresResponsabilizados[i+2].toInt())) {
          char Texto[100];
          sprintf(Texto, "ResponsError %s %d %d",
NombreBloque.c_str(), ValoresResponsabilizados[i+1].toInt(),
ValoresResponsabilizados[i+2].toInt());
          strcpy(MensajeEnviado.text, Texto);
          esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
        } else {
          char Texto[100];
          sprintf(Texto, "ResponsExito %s %d %d",
NombreBloque.c_str(), IndiceMin, IndiceMax);
          strcpy(MensajeEnviado.text, Texto);
          esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
        }
      }
    }
    if (ValoresResponsabilizados.size() == 0) {
```

```
      char Texto[100];
      sprintf(Texto, "ResponsExito %s %d %d", NombreBloque.c_str(),
IndiceMin, IndiceMax);
      strcpy(MensajeEnviado.text, Texto);
      esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
    }
  } else if ((strncmp(MensajeRecibido.text, "ResponsError", 12) ==
0) || ((strncmp(MensajeRecibido.text, "ResponsExito", 12) == 0))) {
    String Mensaje = String(MensajeRecibido.text);
    int PrimerEspacio = Mensaje.indexOf(' ');
    int SegundoEspacio = Mensaje.indexOf(' ', PrimerEspacio + 1);
    int TercerEspacio = Mensaje.indexOf(' ', SegundoEspacio + 1);

    String NombreBloque = Mensaje.substring(PrimerEspacio + 1,
SegundoEspacio);
    String StrIndiceMin = Mensaje.substring(SegundoEspacio + 1,
TercerEspacio);
    String StrIndiceMax = Mensaje.substring(TercerEspacio + 1);
    uint32_t IndiceMin = StrIndiceMin.toInt();
    uint32_t IndiceMax = StrIndiceMax.toInt();
    if (strncmp(MensajeRecibido.text, "ResponsError", 12) == 0) {
      Respuestas.push_back(0); // 0 = Error
    }
    if (strncmp(MensajeRecibido.text, "ResponsExito", 12) == 0) {
      Respuestas.push_back(1); // 1 = Exito
    }
    String MacStr = "";
      for (int i = 0; i < 6; i++) {
        MacStr += String(info->src_addr[i], HEX);
        if (i < 5) MacStr += ":";
      }
    ConvertirMacStringABytes(MacStr,peerMAC); //No hace falta
    for (size_t i = 0; i < ListaRespuestasPendientes.size(); i++) {
      if (ListaRespuestasPendientes[i] == MacStr) {
        ListaRespuestasPendientes[i+1] = "1";
        break;
      }
    }
    bool TodosRespondieron = true;
    for (size_t i = 1; i < ListaRespuestasPendientes.size(); i += 2)
{
      if (ListaRespuestasPendientes[i] != "1") {
        TodosRespondieron = false;
        break;
      }
    }
```

```
    bool ExisteError = false;
    if (TodosRespondieron) {
      for (size_t i = 0; i < Respuestas.size(); i++) {
        if (Respuestas[i] == 0) {
          ExisteError = true;
          break;
        }
      }
      if (ExisteError) {
        Serial.print("Error: Para el bloque ");
        Serial.print(NombreBloque);
        Serial.print(" no puedes responsabilizarte de unos índices
entre ");
        Serial.print(IndiceMin);
        Serial.print(" y ");
        Serial.println(IndiceMax);
      } else {
        Serial.print("Para el bloque ");
        Serial.print(NombreBloque);
        Serial.print(" este dispositivo solo podrá escribir entre
las posiciones ");
        Serial.print(IndiceMin);
        Serial.print(" y ");
        Serial.println(IndiceMax);
        ValoresResponsabilizados.push_back(NombreBloque);
        ValoresResponsabilizados.push_back(StrIndiceMin);
        ValoresResponsabilizados.push_back(StrIndiceMax);
      }
      Respuestas.clear();
      ListaRespuestasPendientes.clear();
    }
  }
}

void MemoriaCompartida::enEnvioDeDatos(const uint8_t* MAC,
esp_now_send_status_t status) {
  if (status == ESP_NOW_SEND_SUCCESS) {

  } else {
    Serial.println("Error al enviar el mensaje");
  }
}

void MemoriaCompartida::Dispositivos() {
  if (DispositivosConectados.empty()) {
    Serial.println("No hay dispositivos conectados.");
    Serial.println("Dispositivo local: ");
```

```
    Serial.println(WiFi.macAddress());
    delay(100);
  } else {
    Serial.println("Dispositivo local:");
    Serial.println(WiFi.macAddress());
    Serial.println("Dispositivos conectados: ");
    for (size_t i = 0; i < DispositivosConectados.size(); ++i) {
      Serial.println(NormalizarMac(DispositivosConectados[i]));
    }
  }
  Serial.println();
}

String MemoriaCompartida::NormalizarMac(const String& Mac) {
  String MacNormalizada = Mac;
  MacNormalizada.toUpperCase();
  return MacNormalizada;
}

void MemoriaCompartida::Borrar(String& Mac) {
  bool Registrado = false;
  if (!esDireccionMac(Mac)) {
    if (!esAlias(Mac)){
      Serial.println("Error: Formato de MAC inválido.");
      return;
    }
    if(esAlias(Mac)){
      ConvertirAliasAMacString(Mac);
      String mac = ConvertirAliasAMacString(Mac);
      Mac = mac;
    }
  }
  if (NormalizarMac(Mac) == NormalizarMac(WiFi.macAddress())) {
    Serial.println("Error: La MAC introducida es la de este
dispositivo.");
    return;
  }
  if (!ConvertirMacStringABytes(Mac, peerMAC)) {
    Serial.println("Error al convertir la MAC");
    return;
  }
  for (auto it = DispositivosConectados.begin(); it !=
DispositivosConectados.end(); ++it) {
    if (NormalizarMac(*it) == NormalizarMac(Mac)) {
      char Texto[30];
      sprintf(Texto, "Borrado %s", WiFi.macAddress().c_str());
      strcpy(MensajeEnviado.text, Texto);
```

```
      esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
      DispositivosConectados.erase(it);
      esp_now_del_peer(peerMAC);
      Serial.print("El dispositivo ");
      Serial.print(Mac);
      Serial.println(" ha sido elimindado.");
      Registrado = true;
      for (size_t i = 0; i < VectoresVinculados.size(); i++) {
        if(NormalizarMac(VectoresVinculados[i]) ==
NormalizarMac(*it)) {
          VectoresVinculados.erase(VectoresVinculados.begin() + (i -
1));
          VectoresVinculados.erase(VectoresVinculados.begin() + (i -
1));
          i -= 1;
        }
      }
      break;
    }
  }
  for (int i = 0; i < DispositivosAlias.size(); i += 2) {
    if (NormalizarMac(DispositivosAlias[i]) == NormalizarMac(Mac)) {
      DispositivosAlias.erase(DispositivosAlias.begin() + i);
      DispositivosAlias.erase(DispositivosAlias.begin() + i);
      Registrado = true;
      break;
    }
  }
  delay(100);
  esp_now_del_peer(peerMAC);
  Registrado = true;
  if (!Registrado) {
    Serial.println("Error: Dispositivo no registrado.");
  }
}

void MemoriaCompartida::CrearBloqueMemoria(String NombreBloque,
uint32_t TamanoBloque) { // Actualizada
  if (NombreBloque.length() == 0) {
    Serial.println("El nombre del bloque no puede estar vacío.");
    return;
  }
  for (size_t i = 0; i < Bloques.size(); i += 2) {
    if (Bloques[i] == NombreBloque) {
      Serial.println("Error: El nombre del bloque ya existe.");
      return;
```

```
    }
  }
  if (TamanoBloque == 0){
    Serial.println("Error: El tamaño deber ser un número y mayor a
0");
    return;
  }
  uint32_t EspacioGastado = 0;
  for (size_t i = 0; i < Bloques.size(); i += 2){
    EspacioGastado += (Bloques[i + 1].toInt()) * 4;
  }
  uint32_t MemoriaLibre = ESP.getFreeHeap();
  MemoriaLibre -= EspacioGastado;
  if (TamanoBloque * 4 > MemoriaLibre) {
    Serial.print("Error: No hay suficiente memoria disponible para
este bloque. Espacio libre disponible: ");
    Serial.println(MemoriaLibre/4);
    return;
  }
  Bloques.push_back(NombreBloque);
  Bloques.push_back(String(TamanoBloque));
  Serial.print("Bloque de memoria '");
  Serial.print(NombreBloque);
  Serial.print("' creado con tamaño de ");
  Serial.print(TamanoBloque);
  Serial.println(" direcciones de memoria.");
  uint32_t* DireccionMemoria = (uint32_t*)malloc(TamanoBloque *
sizeof(uint32_t));
  if (DireccionMemoria != NULL) {
    MemoriaDispositivos.push_back(DireccionMemoria);
    for (size_t i = 0; i < TamanoBloque; i++) {
      DireccionMemoria[i] = 0;
    }
    VincularVector();
  } else {
    Serial.println("Error al reservar memoria para el bloque.");
  }
}

void MemoriaCompartida::VincularVector() {
  if (DispositivosConectados.size() == 0) {
    Serial.println("No hay dispositivos conectados para vincular.");
    return;
  }
  if (Bloques.size() == 0) {
    Serial.println("No hay bloques creados para vincular.");
    return;
```

```
  }
  for (size_t i = 1; i < Bloques.size(); i += 2) {
    std::vector<int> IndiceDispositivo;
    for (size_t j = 0; j < VectoresVinculados.size(); j++) {
      if (Bloques[i - 1] == VectoresVinculados[j]) {
        IndiceDispositivo.push_back(j + 1);
      }
    }
    for (size_t j = 0; j < DispositivosConectados.size(); j++) {
      bool yaVinculado = false;
      for (size_t k = 0; k < IndiceDispositivo.size(); k++) {
        if (DispositivosConectados[j] ==
VectoresVinculados[IndiceDispositivo[k]]) {
          yaVinculado = true;
          Serial.print("El dispositivo ");
          Serial.print(NormalizarMac(DispositivosConectados[j]));
          Serial.print(" ya tiene vinculado el bloque ");
          Serial.println(Bloques[i - 1]);
          break;
        }
      }
      if (!yaVinculado) {
        char Texto[100];
        sprintf(Texto, "Vincular %s %lu", Bloques[i - 1].c_str(),
Bloques[i].toInt());
        strcpy(MensajeEnviado.text, Texto);
        ConvertirMacStringABytes(DispositivosConectados[j],
peerMAC);
        esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));

        Serial.print("El vector: ");
        Serial.print(Bloques[i - 1]);
        Serial.print(" se está intentando vincular con ");
        Serial.println(NormalizarMac(DispositivosConectados[j]));
      }
    }
    IndiceDispositivo.clear();
  }
}

void MemoriaCompartida::CrearVectorValoresResponsabilizados(String
NombreBloque, int IndiceMin, int IndiceMax) {
  bool BloqueEncontrado = false;
  bool BloqueVinculado = false;
  uint32_t TamanoBloque;
  for (size_t i = 0; i < Bloques.size(); i += 2) {
```

```
    if (NombreBloque == Bloques [i]) {
      BloqueEncontrado = true;
      TamanoBloque = Bloques[i+1].toInt();
      break;
    }
  }
  if (!BloqueEncontrado) {
    Serial.print("Error: El vector ");
    Serial.print(NombreBloque);
    Serial.println(" no está creado en este dispositivo");
    return;
  } else {
    for (size_t i = 0; i < VectoresVinculados.size(); i++) {
      if (NombreBloque == VectoresVinculados[i]) {
        BloqueVinculado = true;
        break;
      }
    }
    if (!BloqueVinculado) {
      Serial.println("Error: Primero hay que vincular el vector.");
      return;
    }
  }
  if (IndiceMin > IndiceMax || IndiceMax > TamanoBloque) {
    Serial.println("Error: Indices mal definidos.");
    return;
  }
  for (size_t i = 0; i < VectoresVinculados.size(); i++) {
    if (NombreBloque == VectoresVinculados[i]) {
      ListaRespuestasPendientes.push_back(VectoresVinculados[i+1]);
      ListaRespuestasPendientes.push_back("0");
      char Texto[100];
      sprintf(Texto, "Responsable %s %d %d", NombreBloque.c_str(),
IndiceMin, IndiceMax);
      strcpy(MensajeEnviado.text, Texto);
      ConvertirMacStringABytes(VectoresVinculados[i+1], peerMAC);
      esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
    }
  }
}

uint32_t MemoriaCompartida::MostrarMemoria(String NombreBloque, int
Indice) { //Editada
  size_t index = -1;
  for (size_t i = 0; i < Bloques.size(); i += 2) {
    if (Bloques[i] == NombreBloque) {
```

```
      index = i/2;
    }
  }
  if (index == -1) {
    return -1;
  }
  uint32_t* DireccionMemoria = MemoriaDispositivos[index];
  if (DireccionMemoria != nullptr) {
    for (size_t i = 0; i < Bloques.size(); ++i) {
      if (Bloques[i] == NombreBloque) {
        uint32_t TamanoBloque = Bloques [i + 1].toInt();
        if (Indice >= 0 && Indice < TamanoBloque) {
          return DireccionMemoria[Indice];
        } else {
          return -1;
        }
      }
    }
  }
}

uint32_t MemoriaCompartida::InfoMostrarMemoria(String NombreBloque,
int Indice) {
  size_t index = -1;
  for (size_t i = 0; i < Bloques.size(); i += 2) {
    if (Bloques[i] == NombreBloque) {
      index = i/2;
    }
  }
  if (index == -1) {
    Serial.println ("Error: El bloque de memoria no existe");
    return -1;
  }
  uint32_t* DireccionMemoria = MemoriaDispositivos[index];
  if (DireccionMemoria != nullptr) {
    for (size_t i = 0; i < Bloques.size(); ++i) {
      if (Bloques[i] == NombreBloque) {
        uint32_t TamanoBloque = Bloques [i + 1].toInt();
        if (Indice >= 0 && Indice < TamanoBloque) {
          Serial.print("Valor del bloque de memoria ");
          Serial.print(NombreBloque);
          Serial.print(" en la dirección de memoria reservada ");
          Serial.print(Indice);
          Serial.print(": ");
          Serial.println(DireccionMemoria[Indice]);
          return DireccionMemoria[Indice];
        } else {
```

```
          Serial.println("Error: Índice fuera de rango.");
          return -1;
        }
      }
    }
  }
}

void MemoriaCompartida::EscribirMemoria(String NombreBloque, int
Indice, String Valor) {
  std::vector<String> Dispositivos = DispositivosConectados;
  Dispositivos.push_back(WiFi.macAddress());
  size_t index = -1;
  for (size_t i = 0; i < Bloques.size(); i += 2) {
    if (Bloques[i] == NombreBloque) {
      index = i/2;
    }
  }
  if (index == -1) {
    Serial.println ("Error: El bloque de memoria no existe");
    return;
  }
  uint32_t* DireccionMemoria = MemoriaDispositivos[index];
  uint32_t ValorBytes = Valor.toInt();
  if (ValorBytes == 0 && Valor != "0") {
    Serial.println("Error: Valor no válido. Asegúrate de que el
valor sea un número.");
    return;
  }
  bool BloqueVinculado = false;
  if (DireccionMemoria != nullptr) {
    for (size_t i = 0; i < Bloques.size(); ++i) {
      if (Bloques[i] == NombreBloque) {
        uint32_t TamanoBloque = Bloques [i + 1].toInt();
        if (Indice >= 0 && Indice < TamanoBloque) {
          DireccionMemoria[Indice] = ValorBytes;
          Serial.print("Escrito el valor ");
          Serial.print(DireccionMemoria[Indice]);
          Serial.print(" en la dirección de memoria reservada ");
          Serial.print(Indice);
          Serial.print(" del bloque de memoria ");
          Serial.println(NombreBloque);
          SePuedeEscribir = true;
        }
      }
    }
  }
```

```
}

void MemoriaCompartida::EditarValorDeMemoria(String MacReceptor,
String NombreBloque, int Indice, int Valor) { //Editada
  if (!esDireccionMac(MacReceptor)) {
    if (!esAlias(MacReceptor)){
      Serial.println("Error: Formato de MAC inválido.");
      return;
    }
    if(esAlias(MacReceptor)){
      ConvertirAliasAMacString(MacReceptor);
      String mac = ConvertirAliasAMacString(MacReceptor);
      MacReceptor = mac;
    }
  }
  if (NormalizarMac(MacReceptor) ==
NormalizarMac(WiFi.macAddress())) {
    Serial.println("Error: La MAC introducida es la de este
dispositivo.");
    return;
  }
  if (!ConvertirMacStringABytes(MacReceptor, peerMAC)) {
    Serial.println("Error al convertir la MAC");
    return;
  }
  char Texto[100];
  sprintf(Texto, "Editar %s %d %d", NombreBloque.c_str(), Indice,
Valor);
  strcpy(MensajeEnviado.text, Texto);
  ConvertirMacStringABytes(MacReceptor, peerMAC);
  esp_now_send(peerMAC, (uint8_t*)&MensajeEnviado,
sizeof(MensajeEnviado));
}


void MemoriaCompartida::EnviarValorVinculado(String NombreBloque,
int Indice, String Valor){
  uint32_t ValorBytes = Valor.toInt();
  bool BloqueVinculado = false;
  bool BloqueResponsabilizado = false;
  for (size_t i = 0; i < VectoresVinculados.size(); i++) {
    if(NombreBloque == VectoresVinculados[i]) {
      BloqueVinculado = true;
    }
  }
  if(BloqueVinculado) {
    if(ValoresResponsabilizados.size() == 0) {
```

```
      Serial.println("Error: Antes debes responsabilizarte de unos
índices.");
    }
    else{
      for(size_t i = 0; i < ValoresResponsabilizados.size(); i++) {
        if (NombreBloque == ValoresResponsabilizados[i]) {
          BloqueResponsabilizado =  true;
          if (Indice < ValoresResponsabilizados[i+1].toInt() ||
Indice > ValoresResponsabilizados[i+2].toInt()) {
            Serial.print("Error: Para el bloque ");
            Serial.print(NombreBloque);
            Serial.print(" solo puedes escribir entre ");
            Serial.print(ValoresResponsabilizados[i+1]);
            Serial.print(" y ");
            Serial.println(ValoresResponsabilizados[i+2]);
          } else {
            EscribirMemoria(NombreBloque, Indice, Valor); //Se llama
a escribir desde enviar porque si se hace al reves entra en bucle:
Escribir->Enviar->Editar->Escribir...
            if (VectoresVinculados.size() > 0) {
              for (size_t i = 0; i < VectoresVinculados.size(); i++)
{
                if (NombreBloque == VectoresVinculados[i]) {
                  EditarValorDeMemoria(VectoresVinculados[i+1],
NombreBloque, Indice, ValorBytes);
                  delayMicroseconds(200);
                }
              }
            }
          }
        }
      }
    }
  } else {
    EscribirMemoria(NombreBloque, Indice, Valor);
  }
}

void MemoriaCompartida::Alias() {
  Serial.print("Introduce alias para ");
  Serial.print(WiFi.macAddress());
  Serial.print(" (dispositivo local): ");
  while (Serial.available() == 0) {}
  String AliasLocalIntroducido = Serial.readStringUntil('\n');
  AliasLocalIntroducido.trim();
  Serial.println(AliasLocalIntroducido);
  if (AliasLocalIntroducido.length() > 0) {
```

```
    bool esAliasLocalConMacVinculada = false;
    for (int i = 0; i < DispositivosAlias.size(); i++) {
      if (DispositivosAlias[i] == WiFi.macAddress()) {
        DispositivosAlias[i + 1] = AliasLocalIntroducido;
        esAliasLocalConMacVinculada = true;
        break;
      }
    }
    if (!esAliasLocalConMacVinculada) {
      DispositivosAlias.push_back(WiFi.macAddress());
      DispositivosAlias.push_back(AliasLocalIntroducido);
    }
  }

  for (int i = 0; i < DispositivosConectados.size(); i++) {
    Serial.print("Introduce alias para ");
    Serial.print(NormalizarMac(DispositivosConectados[i]));
    Serial.print(": ");
    while (Serial.available() == 0) {}
    String AliasConectadoIntroducido = Serial.readStringUntil('\n');
    AliasConectadoIntroducido.trim();
    Serial.println(AliasConectadoIntroducido);

    if (AliasConectadoIntroducido.length() > 0) {
      bool esAliasConectadoRepetido = false;
      for (int j = 0; j < DispositivosAlias.size(); j++) {
        if (DispositivosAlias[j + 1] == AliasConectadoIntroducido) {
          esAliasConectadoRepetido = true;
          break;
        }
      }
      if (esAliasConectadoRepetido) {
        Serial.println("Error: El alias ya está en uso. Introduzca
otro alias.");
        i--;
        continue;
      }
      bool esAliasConectadoConMacVinculada = false;
      for (int j = 0; j < DispositivosAlias.size(); j++) {
        if (DispositivosAlias[j] == DispositivosConectados[i]) {
          DispositivosAlias[j + 1] = AliasConectadoIntroducido;
          esAliasConectadoConMacVinculada = true;
          break;
        }
      }
      if (!esAliasConectadoConMacVinculada) {
        DispositivosAlias.push_back(DispositivosConectados[i]);
```

```
        DispositivosAlias.push_back(AliasConectadoIntroducido);
      }
    }
  }
  Serial.println("Alias asignado a los dispositivos:");
  for (int i = 0; i < DispositivosAlias.size(); i++) {
    Serial.println(NormalizarMac(DispositivosAlias[i]));
  }
}

bool MemoriaCompartida::esAlias(String Alias) {
  for (int i = 1; i < DispositivosAlias.size(); i += 2) {
    if (DispositivosAlias[i] == Alias) {
      return true;
    }
  }
  return false;
}

String MemoriaCompartida::ConvertirAliasAMacString(String Alias) {
    for (int i = 1; i < DispositivosAlias.size(); i += 2) {
        if (DispositivosAlias[i] == Alias) {
            return DispositivosAlias[i - 1];
        }
    }
    return "";
}

void MemoriaCompartida::init() {
  WiFi.mode(WIFI_STA);
  delay(2000);
  Serial.print("MAC del dispositivo: ");
  Serial.println(WiFi.macAddress());
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error al inicializar ESP-NOW. Reiniciando...");
    ESP.restart();
  }
  Serial.println("ESP-NOW iniciado correctamente.");
  esp_now_register_recv_cb(enRecepcionDeDatos);
  esp_now_register_send_cb(enEnvioDeDatos);
  delay(2000);
  Serial.println("ESP-32 listo.");
}
```