Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Comunicación entre un ordenador y una FPGA empleando un ESP32

Autor:
Calvo Sánchez, Javier

Tutor(es):
De Pablo Gómez, Santiago
Departamento de Tecnología Electrónica

Valladolid, julio 2024.

RESUMEN

En este trabajo se ha perseguido el objetivo de establecer una comunicación wifi entre un ordenador y una FPGA a través de un procesador ESP32. Para ello se ha optado por el uso de una comunicación TCP/IP personalizada entre dos módulos distintos ESP32 y una comunicación en serie SPI entre un ESP32 y una FPGA. Además, se ha empleado un tipo de mensaje propio basado en hexadecimal y se ha probado físicamente a través de una conexión WIFI entre los dos módulos ESP32.

Para ello se ha requerido del uso del entorno de programación Arduino IDE y del uso de conocimientos básicos de electrónica para la conexión física del ESP32 y la FPGA, además de un analizador lógico. Se ha optado por una programación sencilla y con el uso de funciones fuera de las funciones principales para no sobrecargar el código general.

PALABRAS CLAVE

ESP32, FPGA, ARDUINO, TCP/IP, SPI.

ABSTRACT

In this work, the objective has been to establish Wi-Fi communication between a computer and an FPGA through an ESP32 processor. To achieve this, a custom TCP/IP communication has been chosen between two different ESP32 modules, as well as SPI serial communication between an ESP32 and an FPGA. Additionally, a custom message format based on hexadecimal has been used and physically tested via a Wi-Fi connection between the two ESP32 modules.

To accomplish this, the Arduino IDE programming environment was used, along with basic electronics knowledge for the physical connection between the ESP32 and the FPGA, as well as a logic analyzer. The approach involved simple programming and the use of functions outside the main code to avoid overloading it.

KEYWORDS

ESP32, FPGA, ARDUINO, TCP/IP, SPI.

AGRADECIMIENTOS

A mi padre y a mi tío por haberme apoyado en los momentos difíciles y brindar de puntos de vista alternativos y necesarios durante esta etapa.

Debo agradecerles también a las personas que he ido conociendo en el camino. En especial, a aquellos que puedo considerar mis amigos. Tanto de Canarias, como de León, sin vosotros este camino habría sido mucho más difícil y aburrido.

Por último, mamá y abu, espero que haya donde estéis, estéis orgullosas de la persona en la que me he convertido durante este largo viaje. Ojalá pudieseis estar aquí para verlo.

ÍNDICE

TABLA DE ILUSTRACIONES

INTRODUCCIÓN Y OBJETIVOS

En el contexto actual de la tecnología de la información, la comunicación inalámbrica se ha convertido en un elemento esencial para la interconexión de dispositivos. Uno de los desafíos más interesantes es establecer una comunicación eficiente entre un ordenador y una FPGA (*Field-Programmable Gate Array*) utilizando tecnologías inalámbricas. En este sentido, el procesador ESP32 se presenta como una solución versátil y potente que facilita implementar conexiones Wi-Fi de alta velocidad.

El objetivo principal de este trabajo consiste en diseñar e implementar una solución que permita la comunicación inalámbrica entre un ordenador y una FPGA mediante un procesador ESP32. Para lograrlo, se han explorado dos enfoques: una comunicación TCP/IP personalizada entre dos módulos ESP32 y una comunicación en serie SPI entre un ESP32 y la FPGA. Además, se ha desarrollado un protocolo de mensajes propio basado en representación hexadecimal para facilitar la transmisión de datos.

En este proyecto se llevará a cabo la configuración inicial de los módulos ESP32. Esto implica establecer los parámetros necesarios para su funcionamiento, como la configuración de la red Wi-Fi y la asignación de direcciones IP. Además, se establecerá una conexión Wi-Fi entre los dos módulos ESP32, permitiendo así la comunicación entre el ordenador y la FPGA a través de estos dispositivos.

Como objetivos más concretos se tiene:

1. **La implementación de la comunicación TCP/IP personalizada**: el siguiente objetivo consiste en diseñar un protocolo de comunicación personalizado basado en TCP/IP. Se explorarán las características de este protocolo y se adaptarán a las necesidades específicas del proyecto. La fiabilidad y la integridad de los datos transmitidos serán aspectos clave a considerar durante la implementación.

2. **Desarrollo de la comunicación en serie SPI**: para establecer la comunicación con la FPGA, se utilizará el protocolo SPI (*Serial Peripheral Interface*). Se definirán los parámetros de comunicación y la configuración de los pines. Este objetivo busca garantizar una conexión eficiente y rápida entre el ESP32 y la FPGA.

3. **Creación y validación del protocolo de mensajes hexadecimal**: se diseñará un formato de mensaje propio basado en representación hexadecimal. Este protocolo permitirá la transmisión de datos entre los dispositivos de manera eficiente y compacta. Se realizarán pruebas físicas mediante una conexión Wi-Fi entre los módulos ESP32 para validar su funcionamiento.

4. **Documentación**: finalmente, se generará una documentación detallada que permita replicar el proyecto y comprender su funcionamiento. La claridad en la documentación facilitará futuras investigaciones y mejoras en el sistema.

1.0 ANTECEDENTES

1.1 ORIGEN MÓDULO ESP32

El módulo ESP32 consiste en un microcontrolador, siendo esto un circuito integrado programable que puede llevar a cabo órdenes almacenadas en la memoria, incluyendo en su estructura la unidad procesadora, la memoria y los periféricos tanto de entrada como de salida [1]. Se trata de un microcontrolador de bajo coste y que tiene un consumo de energía muy bajo. Tiene su origen en la empresa *Espressif Systems,* en la ciudad de Shanghái en el año 2008.

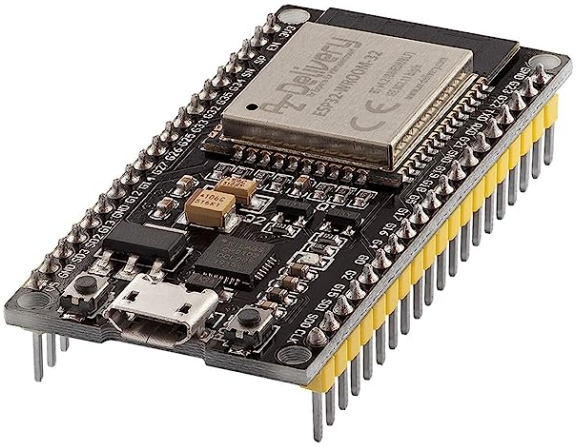


Ilustración 1. Imagen del módulo ESP32.

Utiliza como microprocesador el modelo *Tensilica Xtensa LX6* tanto de uno como de dos núcleos, que suelen trabajar en una frecuencia de operación entre 80-240 MHz. Incluye también un microprocesador de baja energía (ULP) [1]. Las últimas versiones del ESP32 se basan en la arquitectura RISC-V, sin royalties.

Se podrá apreciar en algunas literaturas, tales como *"Xtensa LX Microprocessor"* [2] o la página *"Cadence"* [3], que se le denomina de serie SoC (System on Chip). Este módulo tiene como antecesor al ESP8266, ambos con una arquitectura de 32 bits, añadiendo ambos la posibilidad de conexiones por tecnología Bluetooth y Wifi.

Las diferencias evolutivas del nuevo módulo ESP32 son: La incorporación del nuevo núcleo y las nuevas frecuencias de

funcionamiento, ya que el ESP8266 contaba con un solo núcleo de 160MHz. Entre los avances se encuentran la mejora de la conectividad WIFI, la incorporación del Bluetooth 4.2 y el de baja energía (BLE), más entradas y salidas de propósito general (GPIO), sensores de tacto, de temperatura y de efecto hall [4].

1.2 RELACIÓN ESP32-ARDUINO

El módulo ESP32 y Arduino son dos microcontroladores que tienen una arquitectura muy diferente de hardware, aparte de diferir en capacidad de procesamiento, número de GPIO, capacidades integradas, características de comunicación, etc [5].

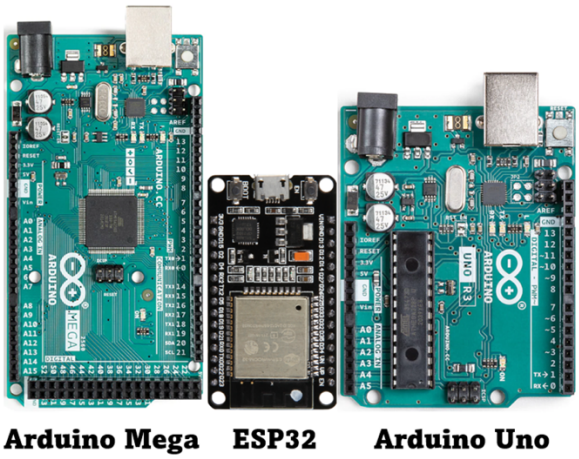


Ilustración 2. Comparación ESP32 y Arduino. [6]

A pesar de estar diseñados por distintas compañías, *Espressif*, que es la diseñadora del módulo ESP32, se ha encargado de diseñar un microcontrolador casi totalmente compatible con el entorno Arduino IDE y todas sus librerías. Esto se debe al software ESP32-Arduino Core [5].

1.3 ORIGEN DE *EXPRESSIF*

La empresa responsable del diseño y manufactura del módulo tratado de ESP32 es la multinacional china *Espressif*, con sede en Shanghai. Se trata de una empresa de semiconductores que comenzó como una pequeña *startup* en China [7].

Ilustración 3. Logo Espressif. [8]

Adquirió bastante auge con el desarrollo del SoC ESP8266 y culminó con la aparición del ESP32. Tras ello han ido innovando con la aparición de nuevos modelos de ESP32 como el modelo “S”, “C” y “H” [7].

1.4 APLICACIONES INDUSTRIALES DEL ESP32

La empresa TUPUNATRON ha conseguido la creación de un PLC basado en el módulo ESP32. Cuenta con un total de 58 E/S y varios puertos de comunicación.

Ilustración 4. Ejemplo de PLC con ESP32. [9]

Se pueden conectar por I2C hasta 127 módulos con conexión maestro-esclavo. Y tiene aplicaciones para el campo de las energías renovables mediante la monitorización de los KPI (“Key Performance Indicator”, métrica cuantitativa que mide la progresión hacia los objetivos), para invernaderos controlando a distancia la humedad, el CO2 y la luminosidad, etc [9].

1.5 ORIGEN DE LA FPGA

Las FPGA, *Field Programmable Gate Array*, son dispositivos semiconductores que se basan en una matriz de bloques lógicos configurables (CLBs) conectados a través de interconexiones programables [10]. Estos bloques lógicos pueden ser configurados para realizar funciones complejas, o actuar como simples puertas lógicas como AND y XOR [11] [12]. En la mayoría de las FPGA, los bloques lógicos también incluyen elementos de memoria, que pueden ser simples flip-flops o bloques de memoria más completos [11] [12].

Las FPGA son el resultado de la unión de dos tecnologías diferentes: los dispositivos lógicos programables (PLD) y los circuitos integrados de aplicación específica (ASIC) [10] [12]. Los PLD son dispositivos que contienen una matriz de puertas lógicas programables, mientras que los ASIC son circuitos integrados diseñados para una aplicación específica [10] [12].

Ilustración 5. Tarjeta de desarrollo para una FPGA. [13]

La principal ventaja de las FPGA sobre los ASIC es su capacidad para ser reprogramadas, lo que les confiere una gran flexibilidad en el diseño. Además, sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo también es menor [10] [11]. Sin embargo, las FPGA son más lentas, tienen un mayor consumo de energía y no pueden abarcar sistemas tan complejos como los ASIC [10] [11] [12].

1.6 APLICACIONES DE LA FPGA

Algunas de las aplicaciones más comunes de las FPGA incluyen:

- *Procesamiento de vídeo e imagen*: Las FPGA son muy útiles para el procesamiento en tiempo real de vídeo e imágenes debido a su capacidad para realizar operaciones en paralelo [14].
- *Telecomunicaciones y Datacom*: Las FPGA se utilizan tanto en comunicaciones cableadas como inalámbricas. En comunicaciones cableadas, se utilizan en *backplanes* serie; en comunicaciones inalámbricas, se utilizan para soluciones de red y para cumplir con los estándares de comunicación [14].
- *Servidores y nube*: Las FPGA se utilizan para acelerar el hardware y los algoritmos, así como para acelerar redes neuronales artificiales o aplicaciones de aprendizaje automático, como lo está haciendo *Microsoft* en su *Proyecto Catapult* [15].
- *Defensa y espacio*: Las FPGA tienen aplicaciones en el espacio para la transmisión de datos desde el suelo hasta la unidad espacial, como un satélite, y desde el satélite hasta la estación terrestre. Las FPGA pueden utilizarse para el procesamiento de datos ópticos de alta resolución y para imágenes de radar. También pueden utilizarse en el espacio para el control de trayectorias y para interactuar con los sensores [14].
- *Médico*: Las FPGA también se utilizan en aplicaciones médicas, como en equipos de diagnóstico por imagen y en dispositivos médicos portátiles [14].

Ilustración 6. Aplicaciones FPGA. [14]

1.7 PROGRAMACIÓN DE LA FPGA

Para programar una FPGA hay que seguir un proceso que tiene varios pasos. En primer lugar, se diseña la arquitectura de hardware utilizando un lenguaje de descripción de hardware (HDL) como VHDL o Verilog [16]. Tras eso, se simulará y se verificará el diseño para asegurarse de que se cumple con las especificaciones y no contiene errores [16].

Una vez validado el diseño, se sintetiza para generar una secuencia de bits, es decir, un archivo de configuración que puede ser cargado en la FPGA [16]. Este archivo de configuración tiene toda la información necesaria que permite programar los bloques lógicos y las interconexiones de la FPGA para implementar el diseño [17].

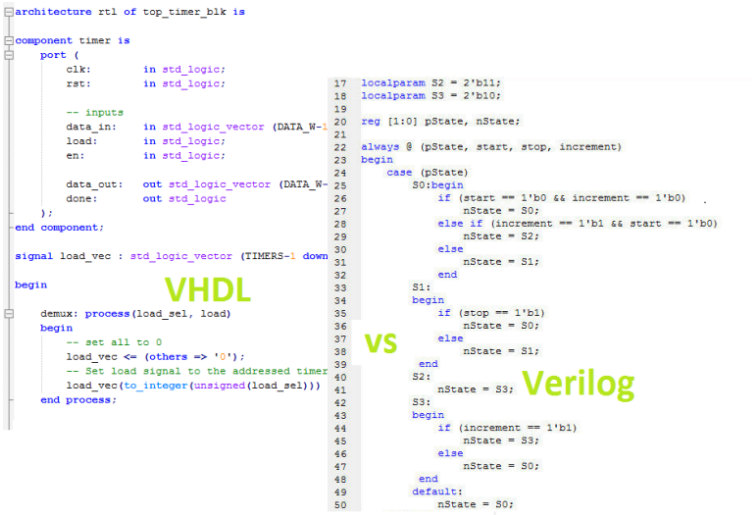


Ilustración 7. Comparación VHDL y VERILOG. [18]

Como paso final, se carga el archivo de configuración en la FPGA utilizando una herramienta de programación específica para el dispositivo [17]. Una vez cargado el archivo, la FPGA está programada y lista para funcionar según el diseño especificado [17].

1.8 PROTOCOLOS DE ENVÍO DE PAQUETES DE DATOS

Están los protocolos de nivel de transporte TCP/IP, que posibilitan la comunicación entre los programas de aplicación.

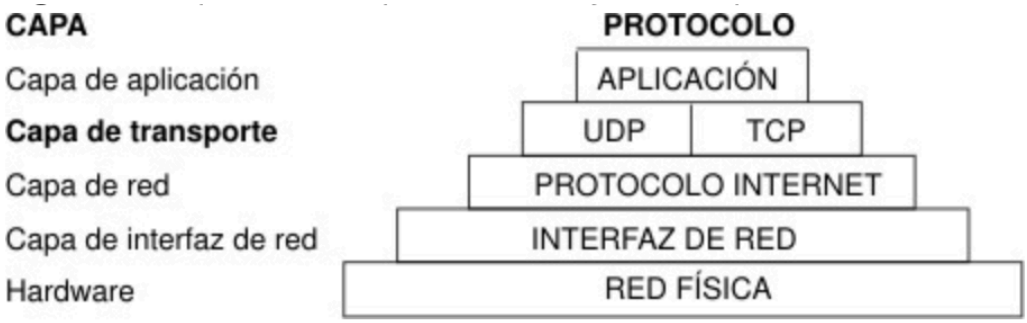


Ilustración 8. Capa de transporte del conjunto de protocolos TCP/IP. [17]

En la figura anterior se puede apreciar las distintas capas de los protocolos TCP/IP. Como se aprecia, la capa de aplicación está formada por la aplicación; la capa de transporte se divide en TCP y UDP. La siguiente capa de red está formada por el protocolo Internet y la capa de interfaz de red. Finalmente está la capa de Hardware, que contiene la red física. [19]

Los protocolos de envío de paquetes de datos son utilizados para poder transmitir datos a través de una red de ordenadores. En una red de conmutación de paquetes, como en el caso de Internet, los datos son divididos en pequeñas unidades denominadas paquetes. Estos paquetes contienen una carga útil, que son los datos, y una cabecera que es la encargada de describir al paquete [20].

Existen varios protocolos distintos empleados para el envío de paquetes en Internet. Dentro de todos ellos, los dos más destacables y empleados son UDP y TCP.

1.8.1 UDP

Se trata del protocolo de Datagramas de Usuario. Es un protocolo sin conexión, prioriza el tiempo sobre la fiabilidad. Es decir, se prioriza la entrega de todos los mensajes a tiempo, aunque se puedan "perder" algunos por el camino. Son de gran utilidad en aplicaciones que no requieren un servicio totalmente diable de transmisión de datos, como videojuegos en línea o transmisiones en vivo [21].

Los mensajes se transmiten como datagramas en paquetes. Se emplea también en conexiones VPN ("*Virtual Private Network*", mecanismo que permite crear una conexión segura empleando un medio de comunicación inseguro) [21].

1.8.2 TCP

Consiste en un Protocolo de Control de Transmisión. Este protocolo está orientado a una conexión con entrega de mensajes fiable, ordenada y con control de errores en un flujo de octetos, bytes, entre aplicaciones ejecutadas en hosts comunicados mediante una red IP [22].

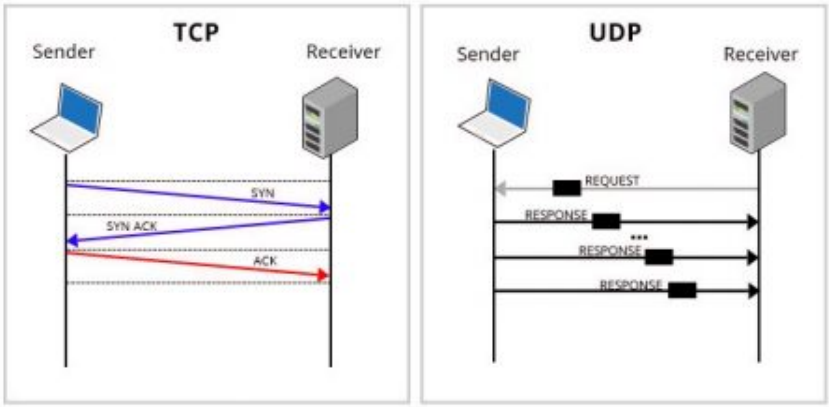


Ilustración 9. Comparación TCP-UDP. [23]

De este modo, este protocolo es ideal para el caso tratado en este trabajo, ya que se desea tener un intercambio de datos seguro y fiable en el que no se pierda información entre emisor y receptor.

Establece y mantiene una conexión entre emisor y receptor durante el proceso de transferencia de datos garantizando que todos los paquetes sean entregados de forma adecuada. Ofrece una transmisión fluida y fiable a través de varios dispositivos [24].

Se emplea en servicios como correos electrónicos y métodos de intercambio entre pares como SSH (*Secure Shell*) y FTP (*File Transfer Protocol*) [24].

2.0 CONEXIÓN CON EL ESP32

Todos los detalles más técnicos, tanto funcionalidades de los pines como características técnicas del módulo ESP-32, están contenidos en el Anexo D.

Se va a proceder con la explicación en pasos de cómo conectarse al ESP32 desde el ordenador/arduino. Será imprescindible disponer del programa Arduino IDE, que en el caso tratado será la versión 2.1.1. Se pretende enviar los mensajes desde un Arduino que hará las funciones de PC.

2.1 INSTALACIÓN *ADD-ON* ESP-32

En este apartado se va a indicar cómo realizar los primeros pasos para poder conectar la placa del ESP-32 al ordenador y poder programarla mediante Arduino IDE.

Se comienza accediendo a la aplicación Arduino IDE y a preferencias:

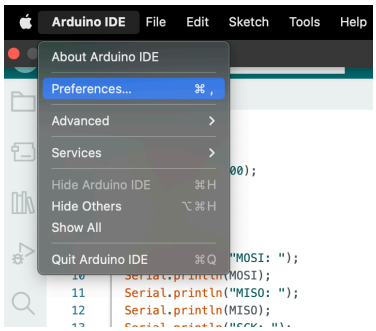


Ilustración 10. Acceso a preferencias.

Tras ello se debe incluir el siguiente enlace como se indica en la figura a continuación para poderse descargar el paquete del ESP32 para Arduino:

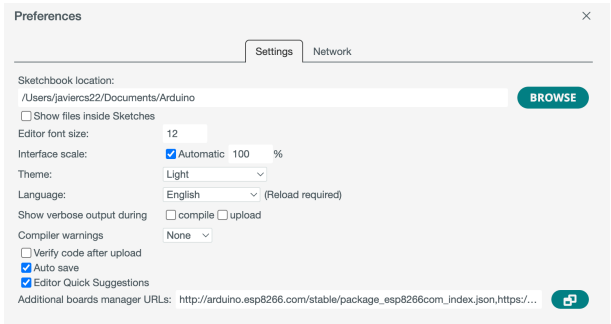


Ilustración 11. URL ESP32.

Una vez descargado, se debe acceder a herramientas, placa y gestor de tarjetas en donde aparecerá lo siguiente, y ahí se procederá a la instalación del *add-on*:

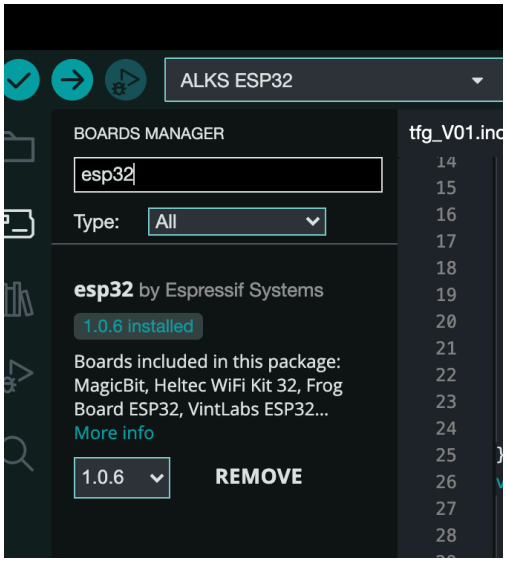


Ilustración 12. Librería ESP32.

Tras la instalación se debe volver a placas y seleccionar el modelo que se disponga, en este caso es el ESP32 Wrover Module.

Finalmente quedaría seleccionar el puerto de la placa con el que se va a establecer la conexión en la siguiente sección:

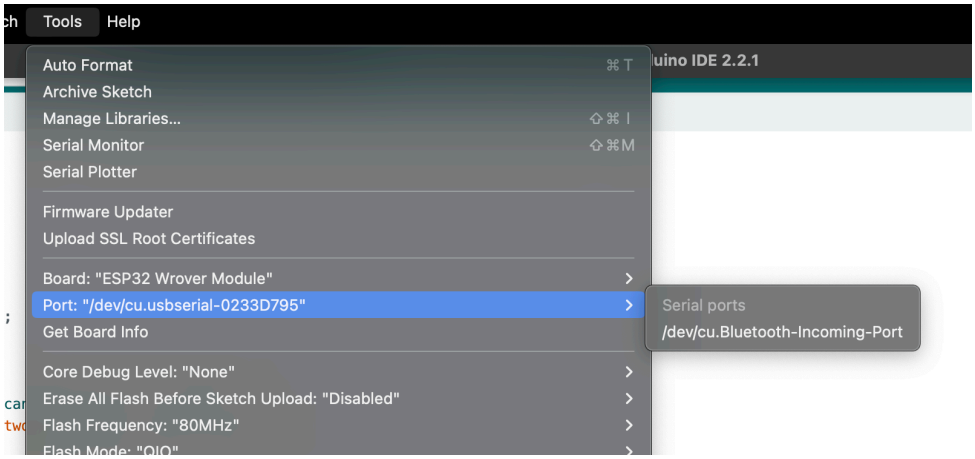
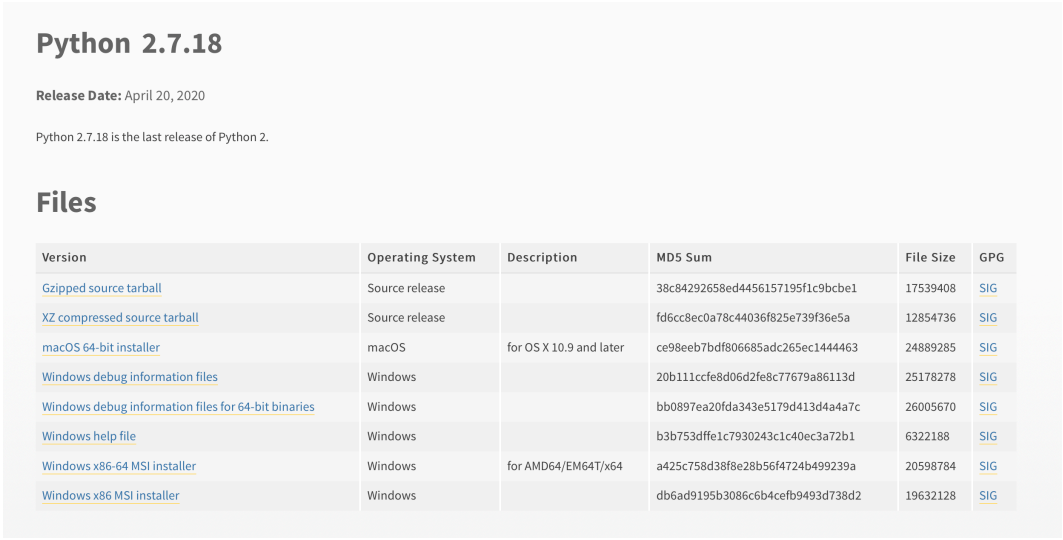


Ilustración 13. Selección de puerto.

Con esto ya estarían realizados los primeros pasos para conectar la placa al ordenador.

Además, se necesita instalar python 2 para poder ejecutar los programas y compilarlos.



Python 2.7.18

Release Date: April 20, 2020

Python 2.7.18 is the last release of Python 2.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		38c84292658ed4456157195f1c9bcbe1	17539408	SIG
XZ compressed source tarball	Source release		fd6cc8ec0a78c44036f825e739f36e5a	12854736	SIG
macOS 64-bit installer	macOS	for OS X 10.9 and later	ce98eeb7bdf806685adc265ec1444463	24889285	SIG
Windows debug information files	Windows		20b111ccfe8d06d2fe8c77679a86113d	25178278	SIG
Windows debug information files for 64-bit binaries	Windows		bb0897ea20fda343e5179d413d4a4a7c	26005670	SIG
Windows help file	Windows		b3b753dffe1c7930243c1c40ec3a72b1	6322188	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64	a425c758d38f8e28b56f4724b499239a	20598784	SIG
Windows x86 MSI installer	Windows		db6ad9195b3086c6b4cefb9493d738d2	19632128	SIG

Ilustración 14. Descarga de python.

2.2 MODOS DE CONEXIÓN

Lo primero que se debe hacer para poder usar el modo de funcionamiento Wifi del módulo ESP32 es añadir la librería Wifi.h, con el código **include <Wifi.h>**. Esta librería se instala de manera automática al añadir el *add-on* de ESP-32 en el Arduino IDE.

El módulo ESP32 es capaz de funcionar con tres modos wifi: estación, punto de acceso o ambos. Para establecer el modo de wifi que se desea utilizar se debe emplear el comando ***Wifi.mode()*** incluyendo el argumento necesario en función del modo deseado:

WIFI_STA	Estación: el ESP32 se conecta a un punto de acceso.
WIFI_AP	Punto de Acceso: el módulo sirve como punto de acceso.
WIFI_STA_AP	Punto de Acceso y Estación: conectados a otro punto de acceso.

2.2.1 MODO ESTACIÓN WIFI

En este modo el ESP32 puede conectarse a otras redes, siendo asignado el ESP32 con una dirección IP por parte del router de esa red. Con este modo, si se dispone de más dispositivos conectados a la misma red, se puede establecer una comunicación entre estos dispositivos y el ESP32 con sólo establecer la dirección IP asignada a la tarjeta. A parte de comunicarse entre

dispositivos conectados a la misma red, si el router está conectado a internet, también se puede acceder a información en la red mediante el ESP32.

Ilustración 15. Modo estación Wi-Fi. [23]

2.2.2 MODO PUNTO DE ACCESO

Este modo se emplea cuando se desea disponer del ESP32 como si fuese un router. Es decir, cuando se desea conectar el ESP32 con otros dispositivos a través de "WIFI" prescindiendo de un router. Este modo es muy útil cuando se dispone de varias placas y se desea establecer comunicación entre las mismas. Cabe destacar que, en este modo, como no se dispone de un router conectado a internet, no se pueden hacer peticiones de información ni demás acciones que involucren el uso de la red.

Los pasos que habría que seguir para establecer el módulo como punto de acceso son:

- Introducir el comando **Wifi.mode(WIFI_AP)** y **Wifi.softAP(ssid, password)** siendo ssid el nombre que se le desea dar al módulo como punto de acceso y la contraseña del mismo, siendo necesarios mínimo 8 caracteres. En caso de no desear establecer una contraseña se escribiría NULL.
- A parte de esos parámetros básicos, se pueden establecer más dentro del comando **Wifi.softAP(const char ssid, const char password, int channel, int ssid_hidden, int max_connection)** siendo cada uno de los nuevos:
  - channel
  - número de canal de WIFI (1-13)
  - ssid_hidden (0 para broadcast SSID y 1 para hide SSID)
  - max_connection para número máximo de conexiones simultáneas con clientes (1-4).

Ilustración 16. Modo access point. [25]

2.2.3 MODO COMBINADO

Con este modo se puede utilizar el módulo ESP32 tanto como punto de acceso como estación WIFI a la vez.

2.3 CONEXIÓN A UNA RED WIFI

Dado el objetivo final, que es poder conectar el ESP32 a una red en la que se encontrarán otro número de dispositivos ESP32 o de otra índole, el mejor modo para conectarse con el ESP32 desde el ordenador es a través de una misma red WIFI.
Como aclaración previa, el ordenador se diseña de tal forma que no tiene una dirección IP estática, si no que es variable, de este modo se establecerá una dirección estática permanente al módulo ESP32 de entre las disponibles.
Se ha escogido que la dirección del módulo sea 192.168.0.13, que se detalla de manera más profunda en el Anexo A de este documento.
En cada aplicación se deberá arbitrar algún mecanismo que permite definir esa dirección.

2.3.1 ESCANEAR REDES WIFI

Se procederá a desarrollar un programa que permita el escaneo de todas las redes wifi disponibles con el módulo ESP32. Se adjunta la imagen del código y, tras ello, se procede a la explicación del mismo:

```
#include <WiFi.h>

String ssid;
String password;

void setup() {
  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Buscando redes Wi-Fi disponibles...");
  int n = WiFi.scanNetworks();
  if (n == 0) {
    Serial.println("No se encontraron redes Wi-Fi disponibles");
  } else {
    Serial.print(n);
    Serial.println(" redes Wi-Fi encontradas:");
    for (int i = 0; i < n; ++i) {
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.println(" dBm)");
      delay(10);
    }
  }
```

Ilustración 17. Programa para escanear redes Wi-Fi.

Primero se establece la conexión en serie con el ordenador que controla el ESP32 a una velocidad de 9600 baudios. Tras ello se determina el modo de conexión wifi en estación "WIFI_STA" mediante la función **Wifi.mode()**.

Tras esto se desconecta de cualquier conexión wifi existente mediante el uso de la función **Wifi.disconnect()**. Se deja un tiempo de espera, y se imprime por pantalla que se están buscando las redes Wifi disponibles.

Se inicia la búsqueda de redes Wifi a través de la función **Wifi.scanNetworks()** y se almacena el valor de las escaneadas en una variable denominada *n*. En el caso de que ese valor sea 0, se indica por pantalla que no se han encontrado redes Wifi disponibles. En caso de que sea distinto de 0, se imprime el número de redes encontradas por pantalla, el nombre de cada una y el nivel de señal, que será una medida de la intensidad que se recibe de esa señal wifi.

En el apartado 2.4 se tratará en más profundidad cómo medir la velocidad y calidad de conexión a la red wifi.

2.4 ENVÍO DE PAQUETES TCP ARDUINO al ESP32

En este apartado se tratará el envío de paquetes TCP entre el módulo Arduino Emisor y el ESP32 Receptor. Estos mensajes pueden ser:

- *ENVÍO DE DATOS*: El ordenador/Arduino Emisor envía una serie de datos al módulo ESP32 Receptor.
- *PETICIÓN DE DATOS*: El PC/Arduino Emisor solicita datos al ESP32 Receptor.
- *RESPUESTA*: Mensaje de respuesta del módulo receptor a la petición de datos.

Todos los tipos de mensajes tendrán la misma estructura y estarán en hexadecimal, que se detallará en el ***Anexo J***. Cada carácter representa **4** bits. De acuerdo con la lógica de la FPGA los valores de dirección y longitud serán enviados desde el byte menos significativo al más significativo al igual que los datos que serán interpretados de la misma forma.

Aquí un ejemplo detallado:

Supongamos que tienes el siguiente mensaje: **<WAABBCCDD000C;Hello World!>**

- **"<" y ">"**: Estos son los delimitadores del mensaje.
- **"W"**: Este es el tipo de comando. En este caso, 'W' significa que es un comando de escritura.
- **"AABBCCDD"**: Esta es la dirección de 32 bits del módulo al que se conectan.
- **"000C"**: Esta es la longitud de 16 bits del mensaje en hexadecimal. '000C' en hexadecimal es 12 en decimal, lo que significa que el mensaje tiene una longitud de 12 caracteres.
- **";"** : Este es el delimitador entre la cabecera del mensaje y el cuerpo del mensaje.
- **"Hello World!"**: Este es el cuerpo del mensaje. Debe tener una longitud igual a la especificada en la cabecera del mensaje.

Tiene un primer carácter que puede ser: **W (write), R (read), A (answer)** en función si es para enviar datos, recibir datos o es la respuesta del ESP32 respectivamente.

A continuación, se indica la dirección con 8 caracteres, utilizando 32 bits. En la FPGA se emplearán 8 bits para identificar el módulo interno con el que se

quiere comunicar. Tras la dirección, están 4 caracteres que indican en hexadecimal la longitud del mensaje que viene después. Se detalla más sobre el formato hexadecimal en el ***Anexo I***. Es decir, un total de 16 bits para indicar la longitud del mensaje, lo que hace que los mensajes puedan tener una longitud máxima de 65535 bytes. De manera que los otros 24 bits restantes sirven para indicar la dirección real, en ese módulo, del byte que se desea leer o escribir. Acompañado de un punto y coma que delimitará el final de la parte de detalles del mensaje para que tras él venga el propio mensaje.

Todo el mensaje estará delimitado por "<" al principio y ">" al final del mensaje.

2.5 CONEXIÓN RED WIFI

```
#include <WiFi.h>

String ssid;
String password;

void setup() {
  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Buscando redes Wi-Fi disponibles...");
  int n = WiFi.scanNetworks();
  if (n == 0) {
    Serial.println("No se encontraron redes Wi-Fi disponibles");
  } else {
    Serial.print(n);
    Serial.println(" redes Wi-Fi encontradas:");
    for (int i = 0; i < n; ++i) {
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.println(" dBm)");
      delay(10);
    }
  }
```

Ilustración 18. Primer programa para buscar redes Wi-Fi.

En este programa se incluye primero para poder incorporar las funciones wifi la librería Wifi.h. Tras ello se configura en el ***setup()*** en este sketch porque es para que realice la prueba una vez el programa en cuestión.

Se inicia la conexión a 9600 baudios con el serial. Se inicia el modo de Estación Wifi y se desconecta del wifi que estuviese anteriormente. Tras ello se espera 100 ms. Se imprime por pantalla que se están buscando las redes disponibles y se crea una variable que almacene el número total de redes disponibles.

Adjunta el nombre con SSID y la calidad de conexión con RSSI en dBm.

```
????????????????22 redes Wi-Fi encontradas:
1: EM_BD (-45 dBm)
2: Vodafone-3854 (-54 dBm)
3: Vodafone-3854 (-67 dBm)
4: MOVISTAR_F100 (-69 dBm)
5: EM_3D (-72 dBm)
6: DIRECT-2A-HP DeskJet 2700 series (-75 dBm)
7: WIFI EM 09 (-76 dBm)
8: MiFibra-9BF0 (-78 dBm)
9: EM_2C_WIFI (-80 dBm)
10: WIFI EM 02 (-81 dBm)
11: MiFibra-B4D6 (-81 dBm)
12: EM_4C (-84 dBm)
13: vodafoneF0D0 (-85 dBm)
14: DIGIFIBRA-2CA9 (-85 dBm)
15: MOVISTAR_C1C0 (-89 dBm)
16: DIRECT-AD-HP OfficeJet 6950 (-89 dBm)
17: vodafone7EFB (-89 dBm)
18: MiFibra-C3D0 (-90 dBm)
19: Livebox6-EF40 (-90 dBm)
20: MOVISTAR_EF20 (-92 dBm)
21: KIDSANDUS_LEON  (-93 dBm)
22: MOVISTAR_FD3C (-93 dBm)
```

Ilustración 19. Muestra de la lista de redes disponibles.

2.6 ASIGNACIÓN IP ESTÁTICA

Una vez conseguido que funcione esta primera parte del programa, se pretende mejorarlo asignando la IP física al módulo ESP32 y que se pida por pantalla el nombre de la red wifi a conectar y su contraseña.

```
#include <WiFi.h>

String ssid;
String password;

void setup() {
  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Buscando redes Wi-Fi disponibles...");
  int n = WiFi.scanNetworks();
  if (n == 0) {
    Serial.println("No se encontraron redes Wi-Fi disponibles");
  } else {
    Serial.print(n);
    Serial.println(" redes Wi-Fi encontradas:");
    for (int i = 0; i < n; ++i) {
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.println(" dBm)");
      delay(10);
    }
  }

  Serial.println("Por favor, introduce el número de la red a la que te quieres conectar:");
  while (Serial.available() == 0) {}
  int networkNumber = Serial.parseInt();
  ssid = WiFi.SSID(networkNumber - 1);

  Serial.println("Por favor, introduce la contraseña de la red:");
  while (Serial.available() == 0) {}
  password = Serial.readString();

  WiFi.begin(ssid.c_str(), password.c_str());

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

Ilustración 20. Primera parte de la modificación del primer código añadiendo las funciones anteriormente mencionadas.

```
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi conectado.");

  Serial.println("Por favor, introduce la dirección IP que deseas asignar al ESP32 (formato: xxx.xxx.xxx.xxx):");
  while (Serial.available() == 0) {}
  String requestedIp = Serial.readString();
  IPAddress staticIp;
  staticIp.fromString(requestedIp);

  IPAddress gateway(192, 168, 1, 1);  // Tu gateway predeterminado. Modifícalo si es necesario.
  IPAddress subnet(255, 255, 255, 0);  // Tu máscara de subred predeterminada. Modifícalo si es necesario.

  WiFi.config(staticIp, gateway, subnet);

  Serial.println("Dirección IP estática asignada.");
  Serial.println("Dirección IP: ");
  Serial.println(WiFi.localIP());
}

void loop() {}
```

Ilustración 21. Segunda parte de la modificación del primer código añadiendo las funciones anteriormente mencionadas.

Se puede ver que funciona correctamente esta primera introducción al código general. Cabe destacar que este código se empleará en el ESP32 emisor y que en el receptor se le asignará mediante una variable el nombre, contraseña del wifi y la dirección estática. Además, se indicará ya el puerto de conexión entre los dos ESP32. También indica el código la dirección IP asignada al ESP32.

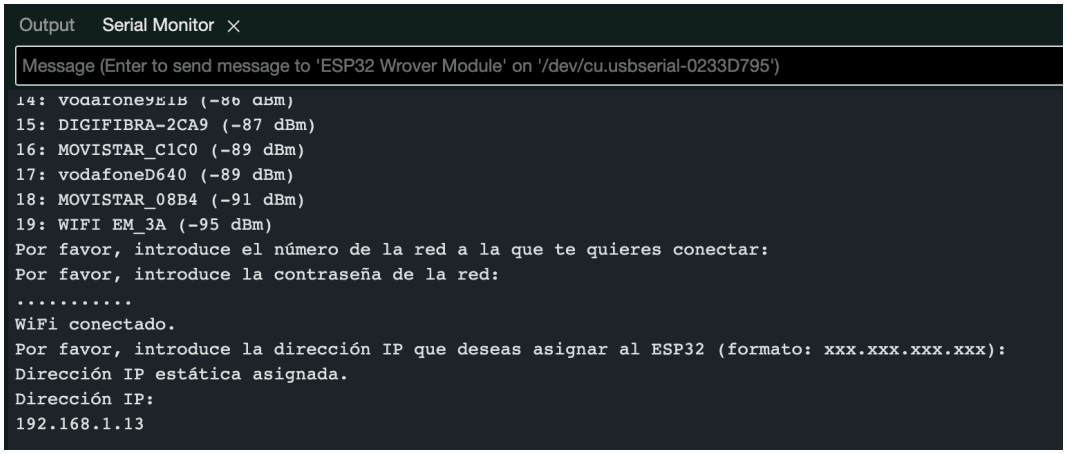


Ilustración 22. Demostración de funcionamiento del código anterior.

Se han dejado indicadas las Gateway y máscara de subred de forma general.

2.7 MOSTRAR POR PANTALLA

Una vez se ha conseguido realizar la parte del código que se emplea para conectarse a una red wifi seguiría la creación de las funciones para formar los mensajes a enviar por TCP, las funciones para pasar a hexadecimal o a código ASCII y el tratamiento de los datos recogidos por pantalla. Esto se hace así para que por pantalla el usuario introduzca si quiere enviar un mensaje de lectura o escritura, el mensaje en sí, la dirección de la FPGA a la que quiere acceder. Y, una vez realizado eso se volvería a empezar de nuevo el bucle.

Primero, en la parte de set up se incluye una función ***ConectarWifi()*** que se ha creado incluyendo las partes del código mencionadas en los apartados anteriores para poderse conectar a una red wifi. Además, se asignaría la dirección IP estática al ESP32.

Se emplea una función ***JoinHexStrings()*** que es la encargada de juntar todo en un mensaje final. Lo crea de acuerdo con lo comentado anteriormente.

Se crea una función para pasar de hexadecimal a código ASCII. Esta vendrá bien sobre todo para imprimir caracteres por pantalla y que no estén en hexadecimal. ***HexToAscii()*** funciona de la siguiente manera: Calcula la longitud del mensaje en hexadecimal. Ahora se actualiza la longitud con la de la cadena entre dos, para obtener los caracteres hexadecimales del mismo. Luego se reserva un espacio en *output* para ese número de caracteres. Tras ello se tiene un bucle, que se ejecuta a través de *hexStr* de dos en dos caracteres. Con *buf* se almacenarán dos caracteres de *hexStr*, para luego convertirlos en decimal y almacenarlos en 'c'. Tras todo esto, se convierte 'c' de decimal a ASCII y se añade a output. Con esto se devuelve *output* que es la representación en ASCII de *hexStr*.

A continuación, se define **AsciiToHex**(). Esta función convierte una cadena de caracteres ASCII en una de caracteres hexadecimal. Se vuelve a optar por inicializar una cadena vacía. Se obtiene la longitud de la cadena argumento y se almacena en una variable. Tras ello se ejecuta un bucle, carácter por carácter de la cadena argumento, para extraer el i-ésimo carácter y almacenarlo en 'c'. Si el valor ASCII de 'c' es menor que 16, se añade un cero al principio. Se convierte 'c' a hexadecimal y se añade a la cadena creada al principio, que es la que será devuelta por la función.

Con esto se tendrían detalladas cada una de las funciones individuales y el ***main***() ya que sólo llama a la función para conectarse a una red wifi. Faltaría definir lo que ocurre dentro de la función ***loop***().

- Serial.println("¿Deseas enviar un mensaje al ESP32? (s/n)");: Esta línea imprime un mensaje en el monitor serie preguntando al usuario si desea enviar un mensaje al ESP32.
- while (Serial.available() == 0) {}: Este bucle se ejecuta mientras no haya datos disponibles para leer desde el monitor serie.
- char respuesta = Serial.read();: Esta línea lee un carácter del monitor serie y lo almacena en respuesta.
- if (respuesta == 'n') { return; }: Si la respuesta es 'n', la función ***loop()*** termina y se vuelve a ejecutar desde el principio.
- tipoMensaje = Serial.readString();: Esta línea lee una cadena de caracteres del monitor serie y la almacena en ***tipoMensaje***.
- String tipoMensaje_hex = AsciiToHex(tipoMensaje);: Esta línea convierte tipoMensaje de ASCII a hexadecimal y lo almacena en ***tipoMensaje_hex***.
- direccionFPGA = Serial.readString();: Esta línea lee una cadena de caracteres del monitor serie y la almacena en ***direccionFPGA***.
- mensaje = Serial.readString();: Esta línea lee una cadena de caracteres del monitor serie y la almacena en mensaje.
- int longitud_hex = (mensaje.length())*2;: Esta línea calcula la longitud de mensaje en hexadecimal y la almacena en ***longitud_hex***.
- longitud_hex_s = String(longitud_hex);: Esta línea pasa el entero longitud_hex a un string almacenado en ***longitud_hex_s***.
- mensaje_hex = AsciiToHex(mensaje);: Esta línea convierte mensaje de ASCII a hexadecimal y lo almacena en ***mensaje_hex***.
- String joined = JoinHexStrings(tipoMensaje_hex ,hexStrw, longitud_hex_s, mensaje_hex);: Esta línea une ***tipoMensaje_hex***, ***hexStrw***, ***longitud_hex_s*** y ***mensaje_hex*** en una sola cadena de caracteres y la almacena en ***joined***.

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>

//Variables Wifi
const char* host = "192.168.1.113";

// Variables para almacenar los datos del mensaje
String tipoMensaje;
String direccionFPGA;
String mensaje;
String ssid;
String password;
String mensaje_hex;
int longitud_mensaje;
int ESP32_TCP_PORT = 80;

String JoinHexStrings(String hex1, String hex2, String hex3, String hex4) {
  String result = "<";  // Añade el carácter '<' al principio
  result += hex1;
  result += hex2;
  result += hex3;
  result += ";";  // Añade el símbolo ';' entre la tercera y la cuarta cadena
  result += hex4;
  result += ">";  // Añade el carácter '>' al final
  return result;
}

void SplitHexStrings(String message, String &hex1, String &hex2, String &hex3, String &hex4) {

  // Elimina los caracteres '<' y '>' del principio y del final
  message = message.substring(1, message.length() - 1);

  // Encuentra la posición del símbolo ';'
  int semicolonPos = message.indexOf(';');
  if (semicolonPos == -1) {
    Serial.println("El mensaje no tiene el formato correcto");
    return;
  }

```

Ilustración 23. Primera parte del código para mostrar por pantalla.

```
41    // Divide el mensaje en las cuatro cadenas hexadecimales
42    hex1 = message.substring(0, 2);  // Los dos primeros caracteres hexadecimales
43    hex2 = message.substring(2, 10);  // Los 8 siguientes caracteres hexadecimales
44    hex3 = message.substring(10, semicolonPos);  // Los caracteres restantes hasta el ';'
45    hex4 = message.substring(semicolonPos + 1);  // Los caracteres después del ';'
46  }
47
48  String HexToAscii(String hexStr) {
49    int len = hexStr.length();
50
51    len = hexStr.length() / 2;
52    String output;
53    output.reserve(len);
54    char buf[5] = {'\0', '\0', '\0', '\0', '\0'};
55    unsigned int c;
56
57    for (unsigned int i = 0; i < hexStr.length(); i += 2) {
58      buf[0] = hexStr[i];
59      buf[1] = hexStr[i + 1];
60      sscanf(buf, "%x", &c);
61      output += (char)c;
62    }
63
64    return output;
65  }
66
67  //código para conectarse a la red wifi
68  void Conectarwifi(){
69
70    Serial.begin(9600);
71
72    WiFi.mode(WIFI_STA);
73    WiFi.disconnect();
74    delay(100);
75
76    Serial.println("Buscando redes Wi-Fi disponibles...");
77    int n = WiFi.scanNetworks();
78    if (n == 0) {
79      Serial.println("No se encontraron redes Wi-Fi disponibles");
80    } else {
```

Ilustración 24. Segunda parte del código para mostrar por pantalla.

```
80    } else {
81      Serial.print(n);
82      Serial.println(" redes Wi-Fi encontradas:");
83      for (int i = 0; i < n; ++i) {
84        Serial.print(i + 1);
85        Serial.print(": ");
86        Serial.print(WiFi.SSID(i));
87        Serial.print(" (");
88        Serial.print(WiFi.RSSI(i));
89        Serial.println(" dBm)");
90        delay(10);
91      }
92    }
93
94    Serial.println("Por favor, introduce el número de la red a la que te quieres conectar:");
95    while (Serial.available() == 0) {}
96    int networkNumber = Serial.parseInt();
97    ssid = WiFi.SSID(networkNumber - 1);
98
99    Serial.println("Por favor, introduce la contraseña de la red:");
100   while (Serial.available() == 0) {}
101   password = Serial.readString();
102
103   WiFi.begin(ssid.c_str(), password.c_str());
104
105   while (WiFi.status() != WL_CONNECTED) {
106     delay(500);
107     Serial.print(".");
108   }
109
110   Serial.println("");
111   Serial.println("WiFi conectado.");
112
113   Serial.println("Por favor, introduce la dirección IP que deseas asignar al ESP32 (formato: xxx.xxx.xxx.xxx):");
114   while (Serial.available() == 0) {}
115   String requestedIp = Serial.readString();
116   IPAddress staticIp;
117   staticIp.fromString(requestedIp);
118
119   IPAddress gateway(192, 168, 1, 1);  // gateway predeterminado.
120   IPAddress subnet(255, 255, 255, 0);  // máscara de subred predeterminada.
```

Ilustración 25. Tercera parte del código para mostrar por pantalla.

```
121
122     WiFi.config(staticIp, gateway, subnet);
123
124     Serial.println("Dirección IP estática asignada.");
125     Serial.println("Dirección IP: ");
126     Serial.println(WiFi.localIP());
127   }
128
129   String AsciiToHex(String asciiStr) {
130     String hexStr = "";
131     int len = asciiStr.length();
132
133     for (unsigned int i = 0; i < asciiStr.length(); i++) {
134       char c = asciiStr[i];
135       if (c < 16) hexStr += '0';  // Se asegura de que los caracteres hexadecimales tengan dos dígitos
136       hexStr += String(c, HEX);
137     }
138     return hexStr;
139   }
140
141   void setup() {
142    Conectarwifi();
143    //server.begin();  // Inicia el servidor
144   }
145
146   void loop() {
147     // Preguntar al usuario si desea enviar un mensaje
148     Serial.println("¿Deseas enviar un mensaje al ESP32? (s/n)");
149     while (Serial.available() == 0) {}
150     char respuesta = Serial.read();
151     if (respuesta == 'n') {
152       return;
153     }
154
155     // Preguntar al usuario si el mensaje es de lectura o escritura
156     Serial.println("¿El mensaje es de lectura (R) o escritura (W)?");
157     while (Serial.available() == 0) {}
158     tipoMensaje = Serial.readString();
159     String tipoMensaje_hex = AsciiToHex(tipoMensaje);
160
```

Ilustración 26. Cuarta parte del código para mostrar por pantalla.

```
  // Preguntar al usuario la dirección de la FPGA
  Serial.println("Introduce la dirección de la FPGA del siguiente modo (AABBCCDD):");
  while (Serial.available() == 0) {}
  direccionFPGA = Serial.readString();
  String direccionFPGA_hex = AsciiToHex(direccionFPGA);
  Serial.println("La dirección de la FPGA es la siguiente:");
  Serial.println(direccionFPGA);  // Imprime la dirección de la FPGA para verificarla

  // Preguntar al usuario el contenido del mensaje
  Serial.println("Introduce el mensaje a enviar:");
  while (Serial.available() == 0) {}
  mensaje = Serial.readString();

 // Calcular la longitud en bits del mensaje
  int longitud_hex = (mensaje.length())*2; //mensaje.length() cuenta el número de caracteres, luego se multiplica por 2
  //ya que cada caracter son 2 números hexadecimales y te da la longitud en hexadecimal (1---4)
  String longitud_hex_s;
  longitud_hex_s = String(longitud_hex);

  // Crear el mensaje en formato hexadecimal
  mensaje_hex = AsciiToHex(mensaje);
  Serial.println("El mensaje a enviar es el siguiente");
  Serial.println(HexToAscii(AsciiToHex(mensaje)));

  //UNIR CADENAS DE TEXTO
  String joined = JoinHexStrings(tipoMensaje_hex ,direccionFPGA_hex, longitud_hex_s, mensaje_hex);
  Serial.println(joined);
}
```

Ilustración 27. Quinta parte del código para mostrar por pantalla.

```
Output  Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on '/dev/cu.usbserial-0233D7FF')

19: EM_3D (-90 dBm)
20: Tapo_Cam_DF3B (-91 dBm)
21: MOVISTAR_EF20 (-91 dBm)
22: KIDSANDUS_LEON  (-92 dBm)
23: MOVISTAR_EE10 (-92 dBm)
24: MOVISTAR_08B4 (-93 dBm)
25: DIRECT-AD-HP OfficeJet 6950 (-94 dBm)
Por favor, introduce el número de la red a la que te quieres conectar:
Por favor, introduce la contraseña de la red:
........
WiFi conectado.
Por favor, introduce la dirección IP que deseas asignar al ESP32 (formato: xxx.xxx.xxx.xxx):
Dirección IP estática asignada.
Dirección IP:
192.168.1.112
¿Deseas enviar un mensaje al ESP32? (s/n)
¿El mensaje es de lectura (R) o escritura (W)?
Introduce la dirección de la FPGA del siguiente modo (AABBCCDD):
La dirección de la FPGA es la siguiente:
12456789
Introduce el mensaje a enviar:
El mensaje a enviar es el siguiente
probando tfg javier
<57313234353637383938;70726f62616e646f20746667206a6176696572>
¿Deseas enviar un mensaje al ESP32? (s/n)
```

Ilustración 28. Funcionamiento del código mostrar por pantalla.

Se puede apreciar en la imagen anterior que se asigna bien la dirección IP al ESP32, que se conecta bien a la red Wifi. Se puede ver cuál es la dirección de la FPGA a la que se va a conectar, cómo se traduce bien a hexadecimal y cómo se vuelve a pasar bien a ASCII. También se aprecia cómo se calcula bien el tamaño del mensaje en hexadecimal. Se aprecia el mensaje que se ha enviado y el mensaje final formado por la función creada explícitamente para ello.

Tras todo esto se ve cómo se inicia de nuevo el bucle en la selección de mensaje de lectura y escritura.

A la hora del código final bastantes cosas de las que se muestran por pantalla no son necesarias. Se han empleado ahora para comprobar que cada una de las partes del código funciona bien y sin errores. De este modo, para su implementación final se puede asegurar su correcto funcionamiento.

2.8 COMUNICACIÓN TCP/IP

Los códigos proporcionados permiten una comunicación bidireccional entre dos ESP32 a través de TCP/IP. Así es cómo funcionan:

- **Emisor (Cliente)**: El ESP32 cliente se conecta al servidor y envía un mensaje iniciando la comunicación. Luego espera y recibe una respuesta del servidor.
- **Receptor (Servidor)**: El ESP32 servidor espera una conexión del cliente. Cuando recibe un mensaje del cliente, lo maneja (en este caso, simplemente lo imprime en el Monitor Serie), y luego envía un mensaje de confirmación de recepción al cliente.

Ahora se va a ver un ejemplo de cómo se aplica el código en ambos ESP32. El código final será añadir esta parte a todo lo demás. Aquí está sólo el funcionamiento correcto de la comunicación TCP/IP, con mensajes simples, pero con exactamente la misma estructura necesaria y que se empleará en el código final.

A la izquierda se puede contemplar el servidor y a la derecha el cliente. Se puede contemplar que el cliente manda el mensaje *"Hello from client"* que será lo que se deba modificar en el código final y que el servidor envía un mensaje de confirmación de recepción. Aquí se modificará para adecuarlo al código final. Ya que este mensaje de confirmación de recepción será algo distinto y se empleará como respuesta del receptor o servidor con los datos obtenidos de la

FPGA en caso de lectura. En el caso de escritura será básicamente algo parecido a la confirmación de recepción del mensaje.

Respecto al código, se puede apreciar que la parte correspondiente a la comunicación TCP/IP se ha añadido en el *loop*(). Sólo hay a mayores esta línea de código *"WiFiServer server(80);"* se utiliza para crear un servidor que escucha las conexiones entrantes en el puerto especificado, en este caso, el puerto 80 (puerto estándar de las comunicaciones http, aunque para TCP en general se use otro puerto) en el código del Server.

Primero se va a detallar el código del servidor y después el del cliente. En el caso del ***servidor***, dentro del **loop**() se tiene lo siguiente:

- WiFiClient client = server.available();: Esta línea de código verifica si hay algún cliente que intenta conectarse al servidor Wifi. Si hay un cliente, server.available() devuelve una referencia a ese cliente.
- if (client): Este condicional verifica si un cliente está intentando conectarse al servidor.
- while (client.connected()): Este bucle se ejecuta mientras el cliente esté conectado al servidor.
- while (client.available()>0): Este bucle se ejecuta mientras haya datos disponibles para leer desde el cliente.
- char c = client.read();: Esta línea de código lee un carácter del cliente.
- Serial.write(c);: Esta línea de código escribe el carácter leído en el monitor serie. Esto es útil para depurar y ver qué datos se están recibiendo del cliente.
- client.println("Mensaje recibido");: Esta línea de código envía un mensaje de confirmación de recepción al cliente. Esto le permite al cliente saber que el servidor ha recibido su mensaje.
- client.stop();: Esta línea de código detiene la conexión con el cliente una vez que se han leído todos los datos.

Mientras que en el caso del ***cliente*** se tiene lo siguiente:

- WiFiClient client;: Esta línea de código crea un objeto client de la clase WiFiClient.
- if (!client.connect(host, 80)): Esta línea de código intenta conectar el cliente al servidor especificado por host en el puerto 80. Si la conexión falla, se ejecuta el bloque de código dentro de este condicional.
- Serial.println("Connection to host failed");: Si la conexión falla, esta línea de código imprime un mensaje en el monitor serie indicando que la conexión ha fallado.

- **delay(1000);:** Esta línea de código hace una pausa en la ejecución del programa durante 1000 milisegundos (1 segundo). Tiempo suficiente para recuperarse de un fallo.
- **return;:** Si la conexión falla, esta línea de código sale de la función loop. Esto evita que se ejecute el resto del código en loop hasta la próxima iteración.
- **client.println("Hello from client");:** Si la conexión tiene éxito, esta línea de código envía un mensaje al servidor.
- **while(client.connected() && !client.available()) delay(1);:** Esta línea de código hace una pausa en la ejecución del programa hasta que haya datos disponibles para leer desde el servidor.
- **while (client.available()):** Este bucle se ejecuta mientras haya datos disponibles para leer desde el servidor.
- **char c = client.read();**: Esta línea de código lee un carácter del servidor.
- **Serial.write(c);**: Esta línea de código escribe el carácter leído en el monitor serie. Esto es útil para depurar y ver qué datos se están recibiendo del servidor.
- **client.stop();:** Esta línea de código detiene la conexión con el servidor una vez que se han leído todos los datos.

```ino
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>

const char* ssid = "EM_BD";
const char* password = "46528575";

WiFiServer server(80);

void setup()
{
  Serial.begin(9600);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }

  server.begin();
  Serial.println(WiFi.localIP());
}

void loop()
{
  WiFiClient client = server.available();

  if (client) {
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
      }
    }
    // Envía un mensaje al cliente.
    client.println("mensaje recibido");
    client.stop();
  }
}
```

```ino
#include <WiFi.h>

const char* ssid = "EM_BD";
const char* password = "46528575";
const char* host = "192.168.2.113";

void setup()
{
  Serial.begin(9600);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println(WiFi.localIP());
}

void loop()
{
  WiFiClient client;

  if (!client.connect(host, 80)) {
    Serial.println("Connection to host failed");
    delay(1000);
    return;
  }

  // Envía un mensaje al servidor.
  client.println("Hello from client, 11");

  // Espera la respuesta del servidor.
  while(client.connected() && !client.available()) delay(1);

  while (client.available()) {
    char c = client.read();
    Serial.write(c);
  }

  client.stop();
}
```

Ilustración 29. Códigos de ejemplo de servidor y cliente.

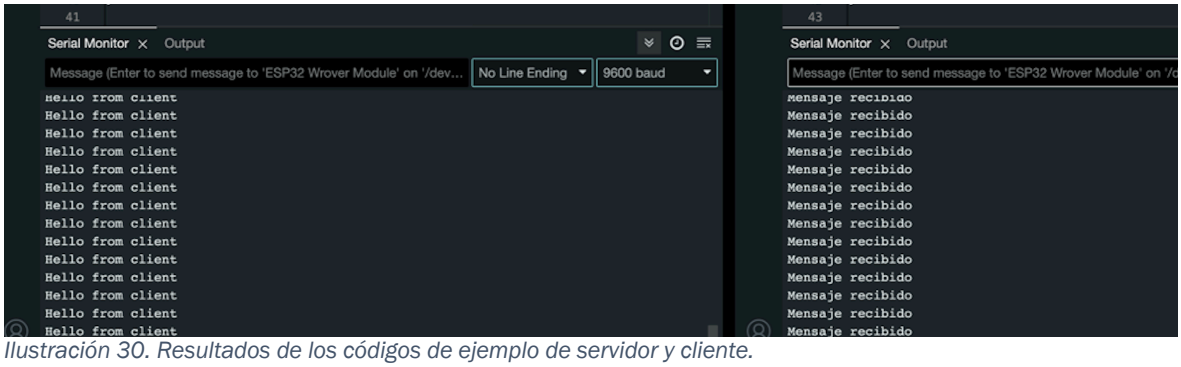


Ilustración 30. Resultados de los códigos de ejemplo de servidor y cliente.

2.9 DESCODIFICAR EL MENSAJE

Para descodificar el mensaje se empleará una función que hará el trabajo inverso al mencionado anteriormente para crear el mensaje. Esto vendrá bien para distinguir casos entre escritura, lectura o para comprobar el mensaje de respuesta. También vendrá bien para manejar el mensaje para la comunicación SPI y para poder mostrar por pantalla las respuestas. Será necesario para saber la dirección de la FPGA y poder configurar bien la comunicación en serie.

```
29  void SplitHexStrings(String message, String &hex1, String &hex2, String &hex3, String &hex4) {
30
31    // Elimina los caracteres '<' y '>' del principio y del final
32    message = message.substring(1, message.length() - 1);
33
34    // Encuentra la posición del símbolo ';'
35    int semicolonPos = message.indexOf(';');
36    if (semicolonPos == -1) {
37      Serial.println("El mensaje no tiene el formato correcto");
38      return;
39    }
40
41    // Divide el mensaje en las cuatro cadenas hexadecimales
42    hex1 = message.substring(0, 2);  // Los dos primeros caracteres hexadecimales
43    hex2 = message.substring(2, 10);  // Los 8 siguientes caracteres hexadecimales
44    hex3 = message.substring(10, semicolonPos);  // Los caracteres restantes hasta el ';'
45    hex4 = message.substring(semicolonPos + 1);  // Los caracteres después del ';'
46  }
47
```

Ilustración 31. Código de la función para desglosar mensajes.

La primera parte del código sirve para eliminar ‘<’ y ‘>’ del principio y del final del mensaje, respectivamente. Una vez hecho esto, se procede a buscar la posición de “;” dentro del mensaje y se almacena en una variable. Se añade una parte en la que si no hay ese carácter se imprime por pantalla que el mensaje no tiene el formato correcto.

Tras todo esto se almacenan en cuatro variables los datos de: Tipo de mensaje, los dos primeros caracteres, Dirección FPGA, con los siguientes 8 caractereres,

Tamaño del mensaje, con los caracteres restantes hasta el ";" y Mensaje, con los caracteres desde ";" hasta el final.

De este modo ya se tendría el mensaje desglosado en cuatro variables como se deseaba.

3.0 CONEXIÓN ESP32-FPGA

3.1 ¿QUÉ ES EL SPI?

El Bus de comunicación SPI, *Serial Peripheral Interface*, es un estándar de comunicaciones desarrollado por *Motorola* en el año 1970. Es empleado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj, comunicación síncrona. Incluye una línea de reloj, dato entrante, dato saliente y una señal de chip select, que indica con qué dispositivo se desea comunicar una señal. De esta forma, este estándar permite multiplexar las líneas de reloj, es decir, convertir una señal en varias como si de un multiplexor se tratase. [26]

La comunicación síncrona es aquella que se lleva a cabo mediante el control de una señal de reloj, mientras que la comunicación asíncrona es aquella que no necesita de dicha señal. Un ejemplo de comunicación síncrona sería *I2C* o *SPI*, y uno de asíncrona las *UART* o *1-WIRE*.

3.2 CONFIGURACIÓN SPI (MASTER-SLAVE)

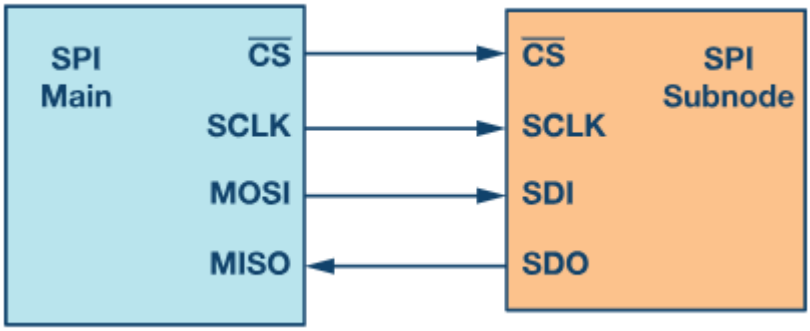


Ilustración 32. Configuración SPI con master y un subnodo. [27]

En la figura anterior se aprecia la configuración de una conexión SPI entre un maestro y un esclavo. Se trata del caso en el que existen 4 señales y 4 cables [27]:

- Reloj (SPI CLK, SCLK)
- Selector de chip (CS ó SS)
- Master OUT Slave IN (MOSI)
- Master IN Slave OUT (MISO)

El dispositivo encargado de general la señal de reloj es el principal o master. La información que se transmite entre ese principal/master y slave/subnode se sincroniza al reloj. Se escoge SPI frente a I2C porque son compatibles con frecuencias de reloj mucho más altas. [27]

El selector de chip, del principal, se emplea para seleccionar el subnodo o slave con el que comunicarse. Como norma general se suele emplear una señal cero o baja, para que cuando se envíe un uno o señal alta se desconecte del subnodo. En el caso de muchos subnodos será necesario una señal CS por cada uno. [27]

Finalmente, MOSI y MISO son líneas de datos. MOSI es para enviar datos del principal al periférico y MISO al revés. [27]

3.3 FUNCIONAMIENTO SPI

El principal o master debe iniciar la señal de reloj y seleccionar el slave escogiendo la señal cs a emitir. Como se indicó anteriormente es una señal de nivel bajo para activación, es decir, se envía un 0. Se trata de una comunicación en dos bandas, porque tanto el principal como el slave pueden enviar datos al mismo tiempo por MOSI y MISO. El reloj se encarga de sincronizar el desplazamiento y muestreo de los datos. Se tiene libre elección de escoger el borde superior o inferior del reloj. [27]

3.4 SPI EN ESP32

El módulo ESP32 dispone de 4 puertos SPI, siendo dos primarios y dos slaves. Los primeros son HSPI (SPI2) y VSPI (SPI3). Que se tratan de controladores SPI con propósito de carácter general. SP0 y SP1 se emplean de manera interna para comunicarse con la memoria flash incorporada, de este modo, no se deben emplear. Los dos primarios pueden llegar a controlar hasta 3 dispositivos esclavos. [28]. La comunicación SPI funciona con 3.3V.

SPI	MOSI	MISO	SCLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

Ilustración 33. Correspondencia pines ESP32 SPI. [28]

Los tres primeros son pines fijos, pero el último pin ***cs*** se trata de un pin variable. Esto se debe a que las señales de datos y de reloj emplean los comandos rápidos de modificación de pines, que requieren pines fijos. La señal ***cs*** en cambio puede ser más lenta, porque sólo se activa una vez por cada byte transferido, y puede usar los comandos habituales de *digitalWrite* que son más lentos, pero más flexibles.

Se va a emplear el código mostrado a continuación para averiguar los pines del ESP32 empleados para la comunicación SPI:

```
#include <SPI.h>

void setup() {
  Serial.begin(9600);
}

void loop() {

  Serial.println("MOSI: ");
  Serial.println(MOSI);
  Serial.println("MISO: ");
  Serial.println(MISO);
  Serial.println("SCK: ");
  Serial.println(SCK);
  Serial.println("SS: ");
  Serial.println(SS);
  delay(1000);
}

```

Ilustración 34. Código para comprobar pines SPI.

Obteniéndose por pantalla los siguientes resultados:

```
MOSI:
23
MISO:
19
SCK:
18
SS:
5
```

Ilustración 35. Pines SPI.

Existiría la posibilidad de customizar los pines empleados para la comunicación SPI, o de usar dos interfaces de bus SPI. En este caso no sería necesario. Se van a emplear los pines por defecto del ESP32.

3.5 CÓDIGO SPI

Ahora, sabiendo cuales son los pines se procede a crear el código para realizar la comunicación SPI con el ESP32. Se enseña una imagen del mismo a continuación y se procederá a la explicación paso a paso, detallada, del mismo.

```
#include <SPI.h>

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23

// Se crean los pines para dirección y datos
#define SPI_AS 26 //Dirección
#define SPI_DS 33 //Datos

//Se crea el pin para distinguir entre lectura y escritura
#define RW 27

//Ejemplo de mensaje de escritura
//String mensaje = "<57313234353637383938;70726f62616e646f20746667206a6176696572>";
//Ejemplo de mensaje de lectura
String mensaje = "<52313234353637383938;70726f62616e646f20746667206a6176696572>";

SPIClass spi(HSPI);

void setup() {
  Serial.begin(9600);

  spi.begin(SPI_SCK, SPI_MISO, SPI_MOSI, -1); // SCK, MISO, MOSI, SS (no se utiliza)
  // Configura los pines SS como salidas.
  pinMode(SPI_AS, OUTPUT);
  pinMode(SPI_DS, OUTPUT);
  pinMode(RW, OUTPUT);
}

void SplitHexStrings(String message, String &hex1, String &hex2, String &hex3, String &hex4) {

  // Elimina los caracteres '<' y '>' del principio y del final
  message = message.substring(1, message.length() - 1);

  // Encuentra la posición del símbolo ';'
  int semicolonPos = message.indexOf(';');
  if (semicolonPos == -1) {
    Serial.println("El mensaje no tiene el formato correcto");
```

Ilustración 36. Primera parte del código de prueba de comunicación SPI.

```
    return;
  }
  // Divide el mensaje en las cuatro cadenas hexadecimales
  hex1 = message.substring(0, 2);  // Los dos primeros caracteres hexadecimales
  hex2 = message.substring(2, 10);  // Los 8 siguientes caracteres hexadecimales. Esta es la dirección de la FPGA.
  hex3 = message.substring(10, semicolonPos);  // Los caracteres restantes hasta el ';'
  hex4 = message.substring(semicolonPos + 1);  // Los caracteres después del ';'
}

String HexToAscii(String hexStr) {
  int len = hexStr.length();

  len = hexStr.length() / 2;
  String output;
  output.reserve(len);
  char buf[5] = {'\0', '\0', '\0', '\0', '\0'};
  unsigned int c;

  for (unsigned int i = 0; i < hexStr.length(); i += 2) {
    buf[0] = hexStr[i];
    buf[1] = hexStr[i + 1];
    sscanf(buf, "%x", &c);
    output += (char)c;
  }

  return output;
}

void loop() {
  spi.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));

  String hex1, hex2, hex3, hex4;
  SplitHexStrings(mensaje,hex1,hex2,hex3,hex4);


```

Ilustración 37. Segunda parte del código de prueba de comunicación SPI.

```
75    if (HexToAscii(hex1) == "R") {
76    digitalWrite(RW, HIGH);
77
78    // Activa SPI_AS durante 8 pulsos para enviar hex2 por MOSI
79    digitalWrite(SPI_AS, LOW);
80    for (int i = 0; i < hex2.length(); i += 2) {
81    String byteString = hex2.substring(i, i+2);
82    byte b = (byte) strtol(byteString.c_str(), NULL, 16);
83    spi.transfer(b);
84    Serial.println(b, HEX);
85    }
86    digitalWrite(SPI_AS, HIGH);
87    digitalWrite(RW, LOW);
88
89    delay(100); // Espera una décima de segundo antes de la próxima operación.
90
91    // Activa SPI_DS durante el número de pulsos necesarios para recibir datos por MISO
92    digitalWrite(SPI_DS, LOW);
93    for (int i = 0; i < 8; i++) {
94      byte receivedData = spi.transfer(0x00); // Dato "dummy" para recibir datos del esclavo
95      Serial.println("Haciendo lectura del SPI...");
96    }
97    digitalWrite(SPI_DS, HIGH);
98
99    } else if (HexToAscii(hex1) == "W") {
100
101     //Se muestra por el monitor serie el mensaje a escribir
102     Serial.println("Se va a escribir en la FPGA");
103     Serial.println(hex4);
104
105   // Activa SPI_AS durante 8 pulsos para enviar hex2 por MOSI
106   digitalWrite(SPI_AS, LOW);
107   digitalWrite(RW, HIGH);
108   for (int i = 0; i < hex2.length(); i += 2) {
109   String byteString = hex2.substring(i, i+2);
110   byte b = (byte) strtol(byteString.c_str(), NULL, 16);
111   spi.transfer(b);
112   Serial.println(b, HEX);
```

Ilustración 38. Tercera parte del código de prueba de comunicación SPI.

```
113     }
114     digitalWrite(SPI_AS, HIGH);
115
116     delay(100); // Espera una décima de segundo antes de la próxima operación.
117
118     // Activa SPI_DS durante el número de pulsos necesarios para enviar datos por MOSI
119
120     digitalWrite(SPI_DS, LOW);
121     digitalWrite(RW, HIGH);
122     for (int i = 0; i < hex4.length(); i += 2) {
123     String byteString = hex4.substring(i, i+2);
124     byte b = (byte) strtol(byteString.c_str(), NULL, 16);
125     spi.transfer(b);
126     Serial.println(b, HEX);
127     }
128     digitalWrite(SPI_DS, HIGH);
129   }
130     spi.endTransaction();
131
132     delay(100); // Espera una décima de segundo antes de la próxima operación.
133   }
```

Ilustración 39. Cuarta parte del código de prueba de comunicación SPI.

Este código se centra en la parte de la comunicación SPI. Que formará una de las partes del código final.

La primera parte de código consiste en la definición de los pines básicos de la comunicación SPI, que son el **MOSI, MISO, CLOCK y SS**. En este caso, se ha añadido una variación en el **ss**. Se tienen dos señales de ese tipo, una para datos **SPI_DS** y otra para dirección **SPI_AS**. Además, se define otro a mayores que servirá para distinguir entre operación de lectura y escritura, **RW**. Se definen con la función *#define* y el número del pin.

Después de esto, se tienen un ejemplo de mensaje de escritura y otro de lectura que fueron empleados para la prueba del programa y su posterior lectura tanto en el monitor serie como con el analizador lógico.

En la línea 19 del código se instancia un objeto de la clase 'SPIClass' utilizando el bus HSPI. Este objeto se utilizará para realizar operaciones SPI.

En la parte de setup se inicia la comunicación SPI con esas variables, y se configuran como salida los pines SS y el pin RW. **SPI.beginTransaction** se emplea para iniciar una transacción SPI, aunque en el código está dentro de ***loop()***. Se emplea después de **SPI.begin**(); además, este comando toma un objeto *SPISettings* como parámetro. Este objeto define la configuración para la transacción SPI, incluyendo la velocidad del reloj, el orden de los bits y el modo SPI [29].

Además, durante la transacción SPI, cualquier interrupción que utilice SPI se desactivará automáticamente. Esto ayuda a evitar la corrupción de datos si otro dispositivo SPI está comunicando al mismo tiempo [30]. Este comando iría con **SPI.endTransiction()** al acabar la transmisión para reactivar cualquier interrupción que emplee SPI [30].

Se emplean las funciones descritas con anterioridad ***SplitHexStrin*** y ***HexToAscii.*** Ahora en ***loop()***, tras el inicio de la transacción SPI como se ha indicado, se definen 4 variables string para almacenar las cuatro partes del mensaje al llamar a la función *SplitHexStrings*. Y, después de esto ya comienzan los bucles que distinguirán entre un mensaje de escritura y lectura, para ello se compara el valor de *hex1* con "R" y "W". Ambos casos tienen una parte similar y otra distinta. La parte similar será el establecer en HIGH el pin *RW* para indicar que se procede a la escritura en la FPGA, y HIGH del pin AS para indicar que se va a transferir la dirección. Tras ello un bucle que se repite hasta *hex2.length()*, que el el tamaño de la dirección FPGA en *hex2,* de 2 en 2 para pasar cada par de caracteres hexadecimales en un byte. Dentro del bucle se emplea *spi.transfer(b)* para enviar cada byte por *MOSI* y también por el monitor serie con la siguiente línea de código. Se vuelve a poner a HIGH AS para indicar que ya no se transfiere más dirección. En el caso de lectura, RW se pone a LOW para indicar que se deja de escribir.

En ambos casos el pin DS se pone a LOW para indicar que se trata de datos en la siguiente transacción. En el caso de lectura, se crea un bucle para enviar un byte "dummy" (0x00) para generar los pulsos de reloj necesarios para recibir datos. Estos datos se almacenan en *receivedData*, el cómo se envían los datos de la FPGA por MISO no es el caso de estudio de este proyecto pues no está entre las especificaciones demandadas. Se imprime por monitor serie el mensaje "Haciendo lectura del SPI…" para comprobar que funciona adecuadamente. Finalizada la operación, se vuelve a poner en HIGH el pin DS.

En el caso de escritura, RW sigue en HIGH y se inicia un bucle cuyo final será el de la longitud de la variable *string hex4*. En él, se transfiere el mensaje de igual forma que se ha transferido la dirección. Finalizada la transferencia, DS vuelve a estar en HIGH.

Acabados los bucles, se finaliza la transacción y se esperan 0,1s.

Una vez hecho esto, se ha procedido a comprobar lo que se estaba transmitiendo mediante un analizador lógico. Se han conectado los pines empleados en la comunicación SPI al analizador y se ha comprobado lo que se ha emitido a través de ellos.

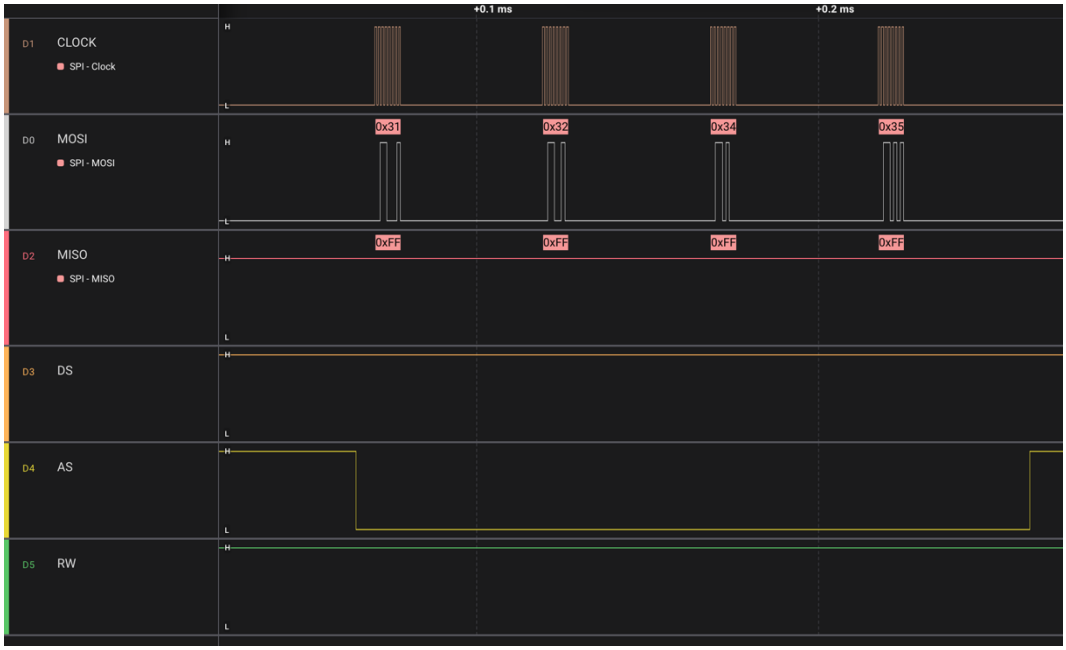


Ilustración 40. Imagen de los datos obtenidos por el analizador lógico en el caso de transferencia de dirección a la FPGA en caso de escritura.

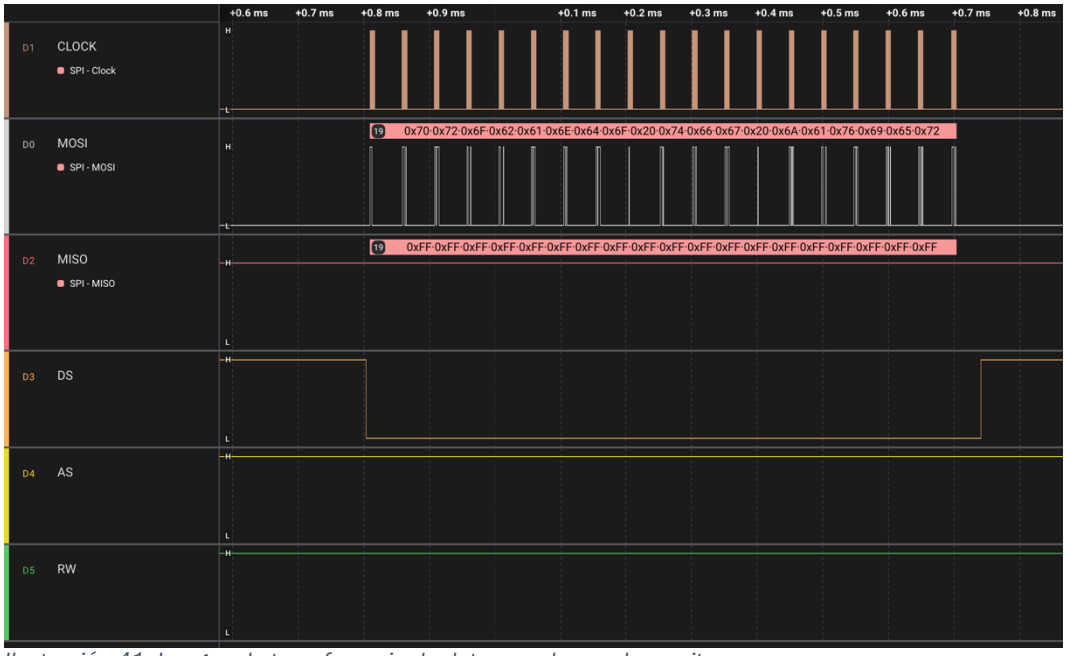


Ilustración 41. Imagen de transferencia de datos en el caso de escritura.

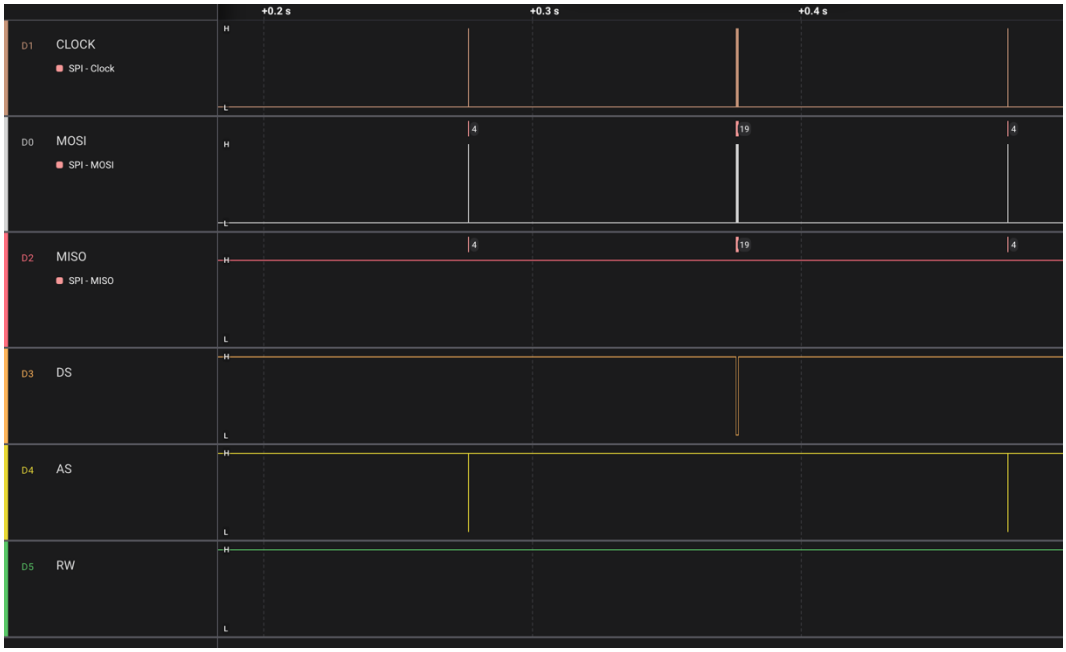


Ilustración 42. Imagen global del funcionamiento de un mensaje de escritura.

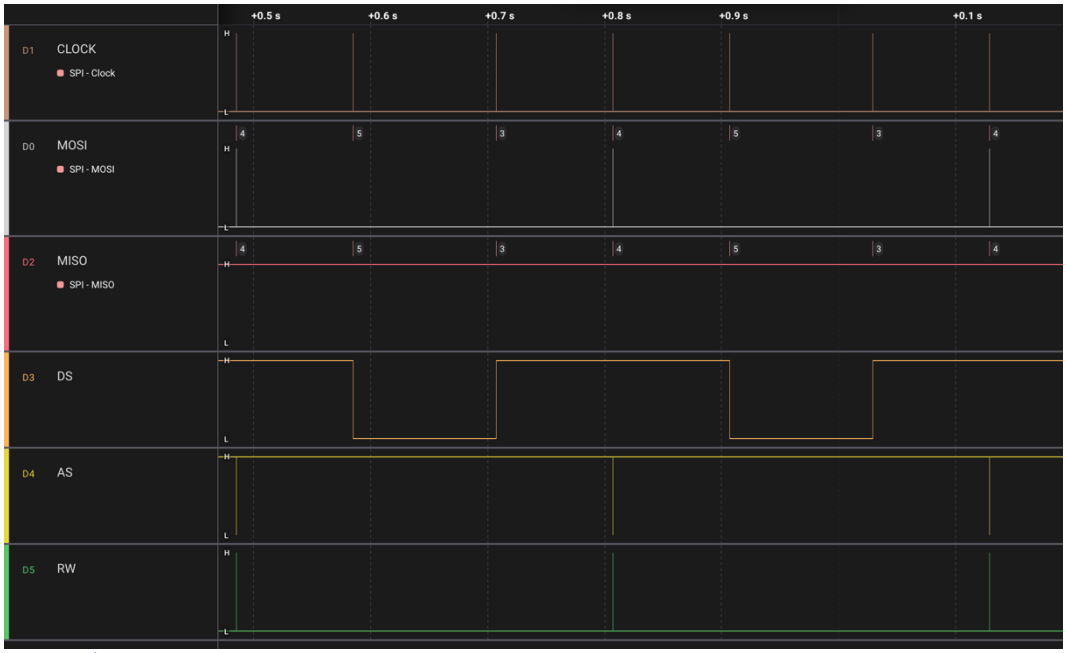


Ilustración 43. Imagen global del funcionamiento de un mensaje de lectura.

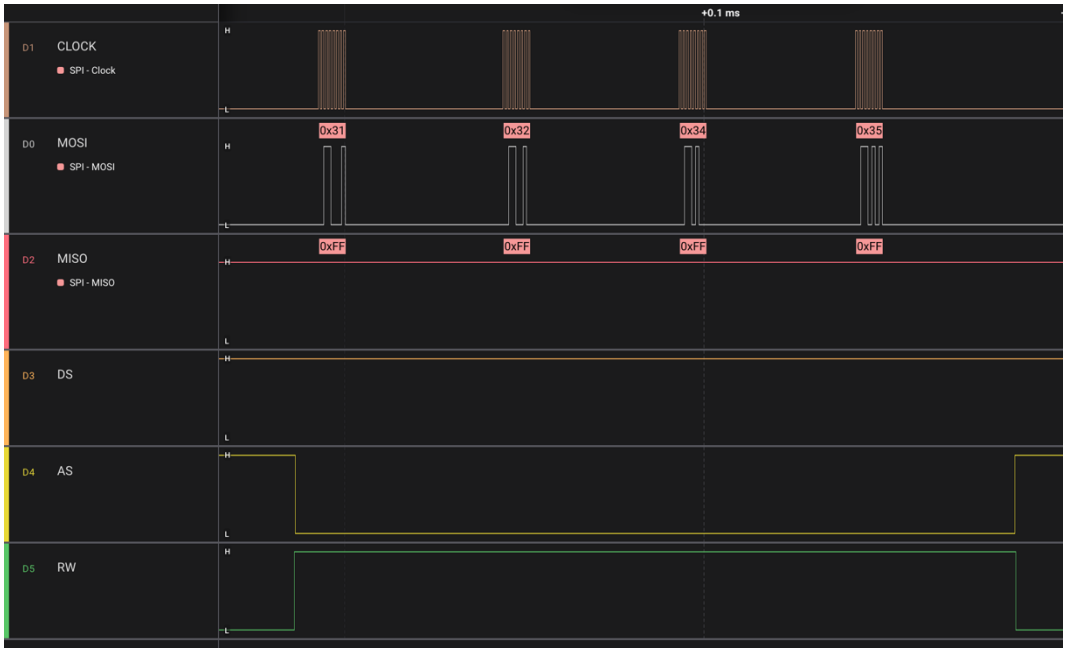


Ilustración 44. Imagen de transferencia de dirección en un mensaje de lectura.

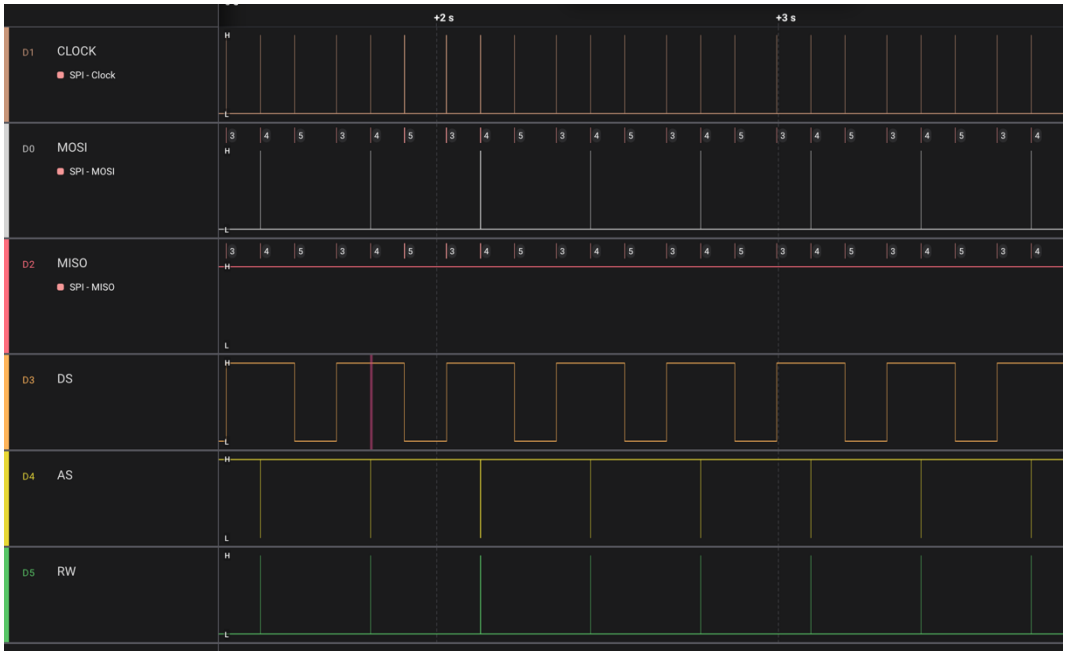


Ilustración 45. Imagen global alejada de la transferencia en caso de mensaje de lectura.

	Duration	mosi	miso
8 ms	7.542 µs	0x00	0xFF
ms	7.542 µs	0x00	0xFF
2 ms	7.542 µs	0x00	0xFF
ns	7.583 µs	0x31	0xFF
17 ms	7.542 µs	0x32	0xFF
58 ms	7.542 µs	0x34	0xFF
ms	7.542 µs	0x35	0xFF
67 ms	7.542 µs	0x00	0xFF
58 ms	7.542 µs	0x00	0xFF
5 ms	7.542 µs	0x00	0xFF
42 ms	7.542 µs	0x00	0xFF
33 ms	7.542 µs	0x00	0xFF
92 ms	7.542 µs	0x00	0xFF
83 ms	7.542 µs	0x00	0xFF
75 ms	7.542 µs	0x00	0xFF
33 ms	7.542 µs	0x31	0xFF
5 ms	7.542 µs	0x32	0xFF
92 ms	7.542 µs	0x34	0xFF
33 ms	7.542 µs	0x35	0xFF
ms	7.542 µs	0x00	0xFF

Ilustración 46. Tabla de datos transferidos en caso de lectura.

	Duration	mosi	miso
8 ms	7.542 µs	0x31	0xFF
s	7.542 µs	0x32	0xFF
2 ms	7.542 µs	0x34	0xFF
3 ms	7.542 µs	0x35	0xFF
33 ms	7.542 µs	0x70	0xFF
17 ms	7.542 µs	0x72	0xFF
83 ms	7.542 µs	0x6F	0xFF
08 ms	7.542 µs	0x62	0xFF
33 ms	7.542 µs	0x61	0xFF
58 ms	7.542 µs	0x6E	0xFF
83 ms	7.542 µs	0x64	0xFF
08 ms	7.542 µs	0x6F	0xFF
33 ms	7.542 µs	0x20	0xFF
58 ms	7.542 µs	0x74	0xFF
42 ms	7.542 µs	0x66	0xFF
67 ms	7.542 µs	0x67	0xFF
92 ms	7.542 µs	0x20	0xFF
17 ms	7.542 µs	0x6A	0xFF
42 ms	7.542 µs	0x61	0xFF
67 ms	7.542 µs	0x76	0xFF
92 ms	7.542 µs	0x69	0xFF
75 ms	7.542 µs	0x65	0xFF

Ilustración 47. Datos obtenidos en el mensaje de escritura.

En las imágenes anteriores se puede observar cómo se han obtenido correctamente los datos emitidos por el MOSI, MISO y el funcionamiento de RW, AS, DS. De este modo, se puede concluir que la transmisión mediante SPI del mensaje hacia la FPGA es correcta.

En las tablas, en la parte de la izquierda se hace referencia al instante de tiempo en el que se han tomado las medidas, aunque se ve un poco cortado.

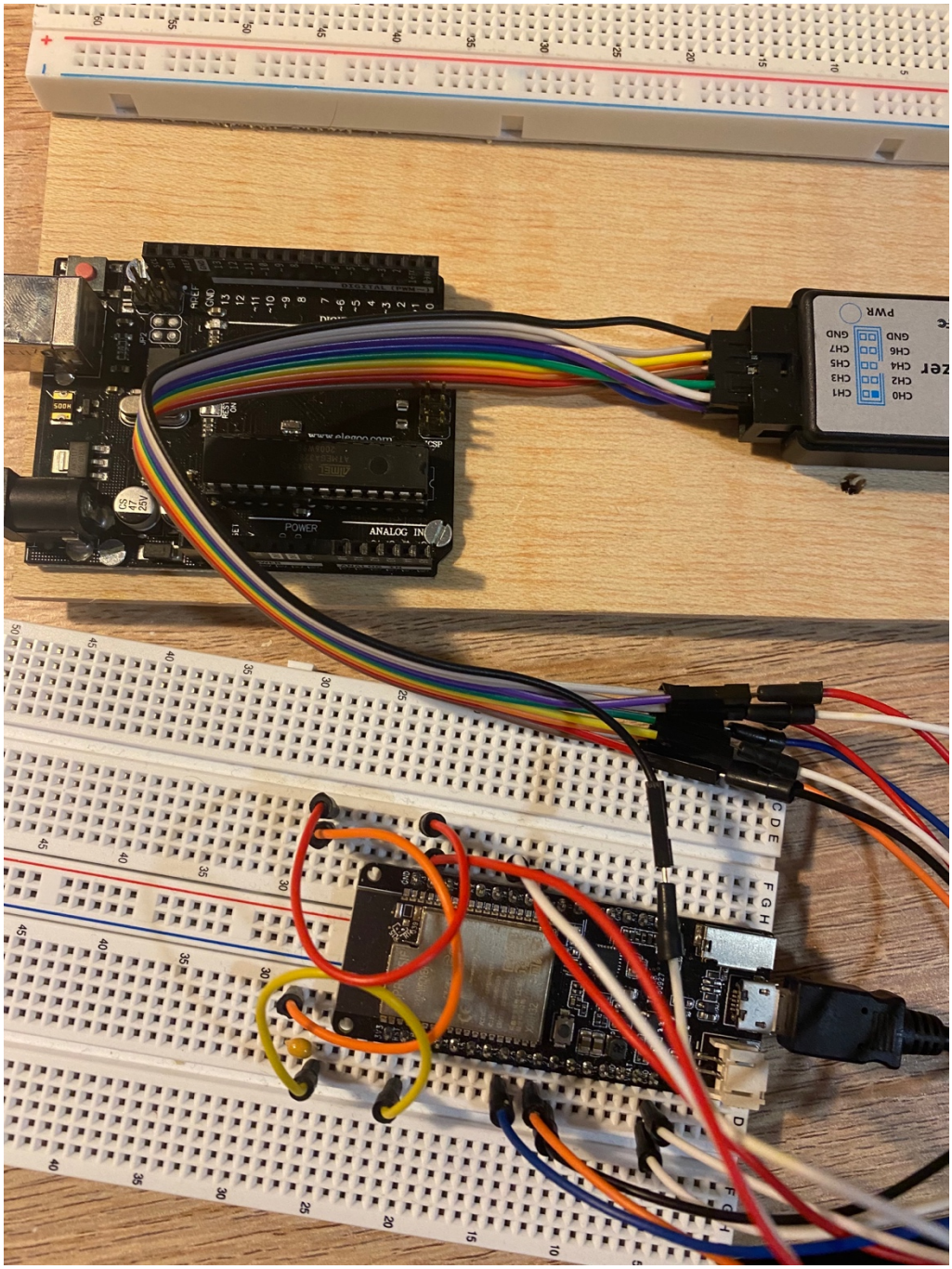


Ilustración 48. Imagen de la conexión real del ESP32 con el analizador lógico.

En la imagen anterior se puede observar cómo se han conectado el ESP32 y el analizador lógico. La conexión entre la placa y la FPGA emplearía los mismos cables.

CONCLUSIONES

A lo largo de este trabajo de fin de grado, se ha explorado en profundidad la comunicación entre dos módulos ESP32 y una FPGA. Se ha demostrado que es posible establecer una comunicación efectiva y eficiente entre estos dispositivos, lo que abre un amplio abanico de posibilidades para futuras aplicaciones.

En primer lugar, se ha logrado configurar y programar los módulos ESP32 para que puedan comunicarse entre sí. Esto ha requerido un entendimiento profundo de los protocolos de comunicación TCP y una cuidadosa programación para garantizar la fiabilidad de la comunicación.

En segundo lugar, se ha conseguido integrar una FPGA en este sistema de comunicación, aunque haya sido un poco más desde el marco teórico. Esto ha añadido una capa adicional de complejidad al proyecto, pero también ha permitido una mayor flexibilidad y potencia en el procesamiento de datos, dando un sentido a la comunicación con el ESP32 a través de SPI.

A pesar de los desafíos que se han encontrado, los resultados obtenidos son prometedores. La comunicación entre los módulos ESP32 y la FPGA ha sido exitosa y se ha podido transmitir y procesar datos de manera correcta.

Este proyecto ha demostrado que la combinación de ESP32 y FPGA puede ser una solución adecuada para diversas aplicaciones, desde sistemas embebidos hasta redes de sensores y más allá. Sin embargo, también se han identificado áreas de mejora y oportunidades para futuras investigaciones.

El orden seguido para la bibliografía se corresponde a la estructuración de los contenidos según el índice.

BIBLIOGRAFÍA

[1] Wikipedia, «Wikipedia - Microcontrolador,» [En línea]. Available: https://es.wikipedia.org/wiki/Microcontrolador. [Último acceso: 14 Junio 2024].

[2] Loboris, «Loboris - ESP32 handbook,» [En línea]. Available: https://loboris.eu/ESP32/Xtensa_lx%20Overview%20handbook.pdf. [Último acceso: 12 Junio 2024].

[3] Cadence, «Cadence.com - Microcontroladores,» [En línea]. Available: https://www.cadence.com/en_US/home/tools/silicon-solutions/compute-ip/tensilica-xtensa-controllers-and-extensible-processors/xtensa-lx-processor-platform.html. [Último acceso: 02 Junio 2024].

[4] Descubrearduino, «Descubrearduino - ESP32 vs ESP8266,» [En línea]. Available: https://descubrearduino.com/esp32-vs-esp8266/. [Último acceso: 5 Junio 2024].

[5] H. Talk, «Techexplorations - Guía ESP32,» [En línea]. Available: https://techexplorations.com/guides/esp32/begin/esp32ard/. [Último acceso: 15 Junio 2024].

[6] Kashif, «Linuxhint - Comparación ESP32 y Arduino,» [En línea]. Available: https://linuxhint.com/esp32-vs-arduino/. [Último acceso: 4 Junio 2024].

[7] Espressif, «Espressif - Información de la empresa,» [En línea]. Available: https://www.espressif.com/en/company/about-espressif. [Último acceso: 15 Junio 2024].

[8] Espressif, «csa-iot.org - Expressif,» [En línea]. Available: https://csa-iot.org/member/espressif-systems/. [Último acceso: 17 Junio 2024].

[9] Tupunatron, «Tupunatron - PLC de ESP32,» [En línea]. Available: https://tupunatron.com/wp-content/uploads/2021/03/plc-esp32-1.jpg. [Último acceso: 16 Junio 2024].

[10] Wikipedia, «Wikipedia - Matriz de puerta programable en campo,» [En línea]. Available: https://es.wikipedia.org/wiki/Matriz_de_puerta_programable_en_campo. [Último acceso: 17 Junio 2024].

[11] Wikipedia, «Wikipedia - Field programmable gate array,» [En línea]. Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array. [Último acceso: 17 Junio 2024].

[12] J. M. M. d. l. Rosa, «Biblus.us - FPGA,» [En línea]. Available: https://biblus.us.es/bibing/proyectos/abreproy/11375/fichero/MEMORIA%252FFPGAs.pdf. [Último acceso: 14 Junio 2024].

[13] J. López, «Hardzone - Características FPGA,» [En línea]. Available: https://hardzone.es/reportajes/que-es/fpga-caracteristicas-utilidad/. [Último acceso: 02 Junio 2024].

[14] Hardwarebee, «Hardwarebee - Aplicaciones FPGA,» [En línea]. Available: https://hardwarebee.com/fpga-common-applications/. [Último acceso: 17 Junio 2024].

[15] Wikipedia, «Wikipedia - Field programmable gate array,» [En línea]. Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array. [Último acceso: 17 Junio 2024].

[16] Mathworks, «Mathworks - Programación FPGA,» [En línea]. Available: https://www.mathworks.com/discovery/fpga-programming.html. [Último acceso: 13 Junio 2024].

[17] Xilink, «Xilinx - Programming an FPGA,» [En línea]. Available: https://www.xilinx.com/products/silicon-devices/resources/programming-an-fpga-an-introduction-to-how-it-works.html. [Último acceso: 14 Junio 2024].

[18] admin, «Fpgear - VDHL or Verilog,» [En línea]. Available: https://fpgaer.tech/?p=168. [Último acceso: 17 Junio 2024].

[19] IBM, «IBM - Protocolos de Internet,» [En línea]. Available: https://www.ibm.com/docs/es/aix/7.1?topic=protocols-internet-transport-level. [Último acceso: 17 Junio 2024].

[20] Wikipedia, «Wikipedia - Comparasion of life transfer protocols,» [En línea]. Available: https://en.wikipedia.org/wiki/Comparison_of_file_transfer_protocols. [Último acceso: 17 Junio 2024].

[21] Wikipedia, «Wikipedia - Protocolo Tcp,» [En línea]. Available: https://www.hostinger.es/tutoriales/protocolo-tcp. [Último acceso: 17 Junio 2024].

[22] Wikipedia, «Wikipedia - Transmission Control Protocol,» [En línea]. Available: https://en.wikipedia.org/wiki/Transmission_Control_Protocol. [Último acceso: 17 Junio 2024].

[23] Prometec, «Promotec - UDP,» [En línea]. Available: https://www.prometec.net/protocolo-udp/. [Último acceso: 13 Junio 2024].

[24] D. V., «Hostinger - Protocolo TCP,» [En línea]. Available: https://www.hostinger.es/tutoriales/protocolo-tcp. [Último acceso: 17 Junio 2024].

[25] Prometec, «Prometec - Modo Acces Point,» [En línea]. Available: https://www.prometec.net/wifi-modo-acces-point/. [Último acceso: 12 Junio 2024].

[26] Wikipedia, «Wikipedia - Serial Peripheral Interface,» [En línea]. Available: https://es.wikipedia.org/wiki/Serial_Peripheral_Interface. [Último acceso: 14 Junio 2024].

[27] SPI, «SPI - Introducción al SPI,» [En línea]. Available: https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html.

[28] S. Santos, «Randomnerdtutorials - ESP32 comunicación SPI,» [En línea]. Available: https://randomnerdtutorials.com/esp32-spi-communication-arduino/#SPI-intro. [Último acceso: 07 Junio 2024].

[29] Arduino CC, «Arduino.cc - SPI,» [En línea]. Available: https://www.arduino.cc/reference/en/language/functions/communication/spi/begintransaction/. [Último acceso: 10 Junio 2024].

[30] Arduino CC, «Arduin.cc - Comunicacion SPI,» [En línea]. Available: https://docs.arduino.cc/tutorials/communication/SPITransaction/. [Último acceso: 12 Junio 2024].

[31] Espressif, «Mouser - ESP32 datasheet,» [En línea]. Available: https://www.mouser.es/datasheet/2/891/esp32_wrover_b_datasheet_en-1384674.pdf. [Último acceso: 14 Junio 2024].

[32] Downs, «ipcb - PCB,» [En línea]. Available: https://www.ipcb.com/es/electronic-design/9160.html#:~:text=¿¿%20qué%20es%20una%20antena%20de%20pcb%3F%20El,Casi%20todos%20los%20dispositivos%20inalámbricos%20utilizan%20PCB%20inalámbricos.. [Último acceso: 14 Junio 2024].

[33] Espressif, «Espressif - Documentación ESP32,» [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wrover-b_datasheet_en.pdf. [Último acceso: 06 Junio 2024].

[34] Arduino cc, «Docs.arduino.cc - Creación de librería,» [En línea]. Available: https://docs.arduino.cc/learn/contributions/arduino-creating-library-guide. [Último acceso: 17 Junio 2024].

[35] Arduino.cc, «Support.arduino.cc - Librería Arduino,» [En línea]. Available: https://support.arduino.cc/hc/en-us/articles/5145457742236-Add-libraries-to-Arduino-IDE. [Último acceso: 17 Junio 2024].

[36] Wikipedia, «Wikipedia - ASCII-Table,» [En línea]. Available: https://es.m.wikipedia.org/wiki/Archivo:ASCII-Table-wide.svg. [Último acceso: 14 Junio 2024].

[37] Wikipedia, «Wikipedia - Silvicultura,» [En línea]. Available: https://es.wikipedia.org/wiki/Silvicultura. [Último acceso: 11 Junio 2024].

[38] Amazon, «Amazon - ESP32,» [En línea]. Available: https://www.amazon.es/AZDelivery-NodeMCU-ESP32-Development-Parent/dp/B0821JM6J7. [Último acceso: 15 Junio 2024].

[39] Prometec, «Prometec - Conexiones Arduino,» [En línea]. Available: https://www.prometec.net/wp-content/uploads/2020/10/Sin-título-768x640.jpg. [Último acceso: 11 Junio 2024].

ANEXOS

ANEXO A – DIRECCIÓN IP ESTÁTICA

Una dirección IP estática es una dirección IP que se asigna de forma manual a un dispositivo en una red y que no varía con el tiempo. Esto difiere de una dirección IP dinámica, que se asigna de manera automática por un servidor DHCP (Protocolo de configuración dinámica de Host) y que puede variar con el tiempo.

La dirección IP estática es útil ya que si se necesita acceder a un dispositivo en una red de forma remota es de gran utilidad, como sería el caso de estudio. De este modo, estableciendo esa dirección estática se podría acceder al módulo a distancia con gran facilidad.

Las direcciones IP de las redes wifi se dividen en 4 clases:

1. Clase A: 1-126 // máscara de subred 255.0.0.0
2. Clase B: 128-191 // máscara de subred 255.255.0.0
3. Clase C: 192-223 // máscara de subred 255.255.255.0
4. Clase D:

Cabe destacar la dirección IP: 127.0.0.1 que es el local host o dirección IP local. Dependiendo del tipo de clase, se utiliza un número determinado de bits para la dirección de red y otro número para la dirección del host. El total son 32 bits.

En la clase A se utilizan los 8 primeros bits para la dirección de la red y los 24 restantes para distinguir a los host de esa red. En la clase B 16 bits de red y 16 bits de host. En la clase C 24 bits de red y 8 bits de host, etc.

Como nuestro caso es una red de clase C, se dispone de 2^8-2 = 254 posibles direcciones para los host de esa red. De este modo se escogen al azar dos direcciones, una para cada módulo ESP32.

192.168.1.184 para el ESP32 emisor de mensajes y 192.168.1.185 para el ESP32 receptor de mensajes.

En este apartado se detallará la parte del programa correspondiente al establecimiento de una dirección IP estática del módulo ESP32 conectado a una red WIFI. Se utilizarán las variables “your_SSID” y “your_PASSWORD”, sólo habría que sustituir el nombre por el necesario en el caso particular.

También se deberá reemplazar “your_IP”, “your_SUBNET” y “your_DNS” por la dirección IP deseada para establecer como estática, la máscara de red y el servidor DNS.

```
#include <WiFi.h>

const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

IPAddress local_IP(192, 168, 1, 184);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(8, 8, 8, 8);

void setup() {
  Serial.begin(115200);

  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS)) {
    Serial.println("Error al configurar la dirección IP estática");
  }

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }

  Serial.println("Conectado a WiFi");
  Serial.print("Dirección IP: ");
  Serial.println(WiFi.localIP());
}

void loop() {}
```

Ilustración 49. Programa para determinar ip estática.

Como se aprecia en la imagen anterior, habría que situar el código en la parte del setup o fuera de las dos partes, sin necesidad de utilizar el loop.

Desglosando el código se ve que primero se incluye la librería Wifi.h como en todos los casos anteriores, y que se definen el nombre y contraseña de la red. A parte de eso, se define la dirección local escogida, la máscara y la puerta de enlace y el DNS.

Tras esto, en la parte del setup se establece la comunicación en serie con una velocidad de 115200 baudios (posteriormente se modifica a 9600 baudios). Tras ello se crea un if llamando a la función Wifi.config(), en el que si la respuesta es false se imprime por pantalla un mensaje de error. Esta función

se llama antes que a Wifi.begin para que configure la red deseada, con los parámetros deseados. En el caso de que se haya conectado se procede a llamar a la función Wifi.begin() para iniciar la conexión a la red wifi.

Una vez completados los pasos anteriores se procede a la creación de un bucle while que se emplea para esperar al establecimiento correcto de la conexión a la red wifi. Durante este bucle se imprime por pantalla cada segundo el mensaje de "Conectando a una red Wifi...". Una vez se haya procedido a conectar se envía el mensaje por pantalla de "Conectado a Wifi" junto a la dirección IP local asignada al módulo.

Con este código ya se tendría establecida la conexión a la red Wifi con la dirección estática IP deseada.

ANEXO B – DETALLE TÉCNICOS MÓDULO ESP-32

En este caso de estudio se está utilizando una placa ESP-32 de la marca comercial Espressif, denominada Module ESP32-WROVER-B, pero de TTGO que tiene ligeras diferencias en la disposición de los pines. Se procederá a determinar todo para un módulo básico del mismo modelo, pero de la marca original y al final se procederá a detallar en una imagen las diferencias en los pines. Se puede apreciar en la imagen siguiente de qué placa se trata:

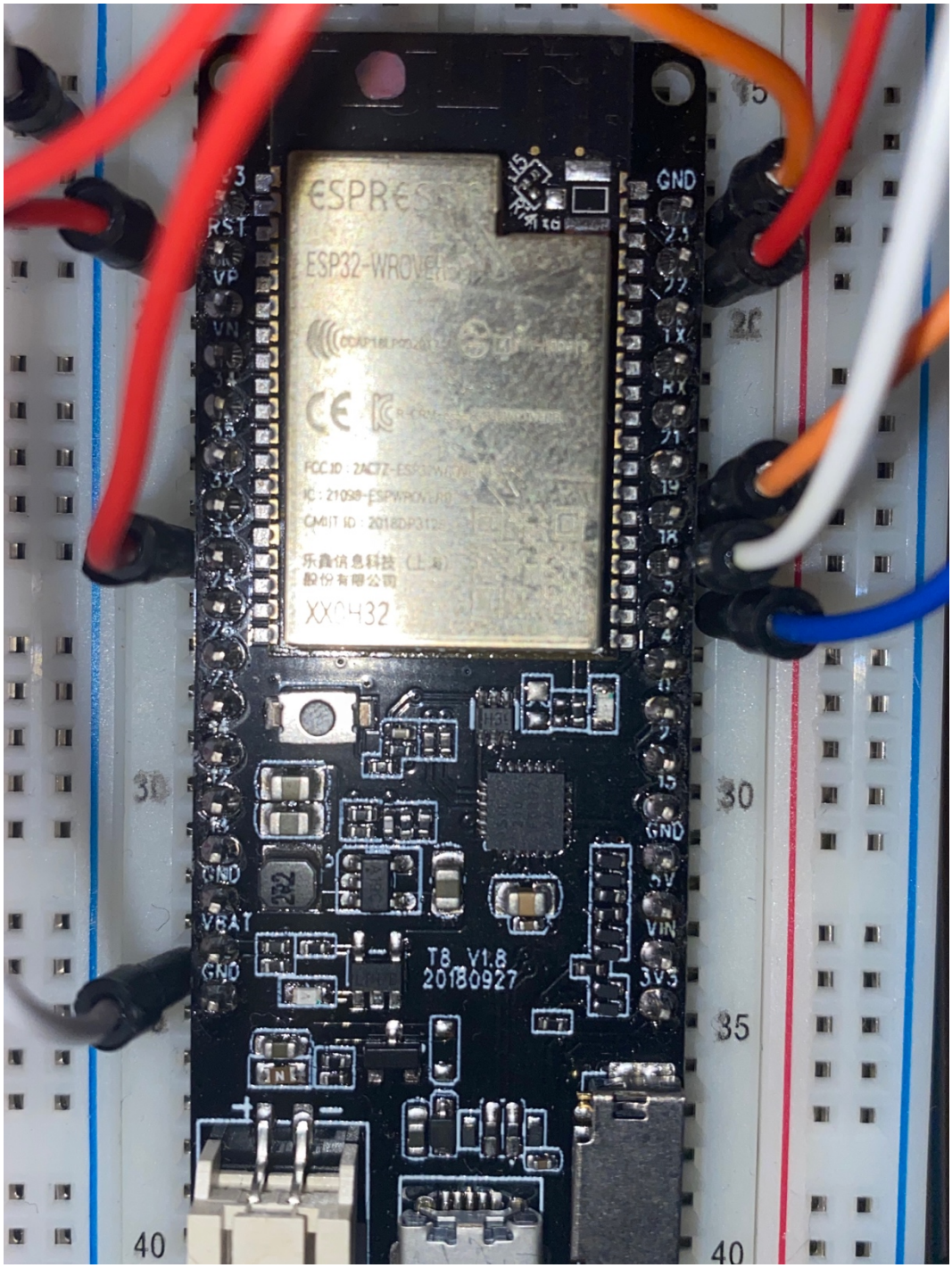


Ilustración 50. Imagen del módulo de trabajo.

A continuación, se aprecia en la siguiente tabla unas especificaciones del módulo extraída directamente del datasheet:

Categories	Items	Specifications
Certification	RF certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Bluetooth certification	BQB
	Green certification	RoHS, REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWAI®), compatible with ISO11898-1 (CAN Specification 2.0)
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Integrated PSRAM	8 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range	–40 °C ~ 85 °C
	Package size	(18.00±0.10) mm × (31.40±0.10) mm × (3.30±0.10) mm
	Moisture sensitivity level (MSL)	Level 3

Ilustración 51. Características generales del módulo ESP32. [31]

Este modelo incorpora módulos de Wifi, Bluetooth y Bluetooth LE MCU, que permiten realizar aplicaciones que engloban desde el uso de sensores más sencillos hasta tareas de alto rendimiento.
El módulo tratado incluye una antena PCB, que se trata de un dispositivo inalámbrico que permite enviar y recibir señales. [32]

En el corazón del módulo se cuenta con un chip ESP32-D0WD, escalable y adaptativo. Se cuenta con un total de dos núcleos que se pueden controlar cada uno de manera individual, además de poderse modificar la frecuencia del reloj con un rango comprendido entre 80 MHz y 240MHz.
Además, el chip cuenta con otro coprocesador que está diseñado para trabajar en bajo consumo y sustituir al CPU permitiendo así ahorrar batería al realizar tareas que no necesitan mucho procesamiento, dentro de las cuales pueden incluirse monitorizar periféricos [33].

Se incluye a continuación una imagen de la distribución de los pines extraída del datasheet:

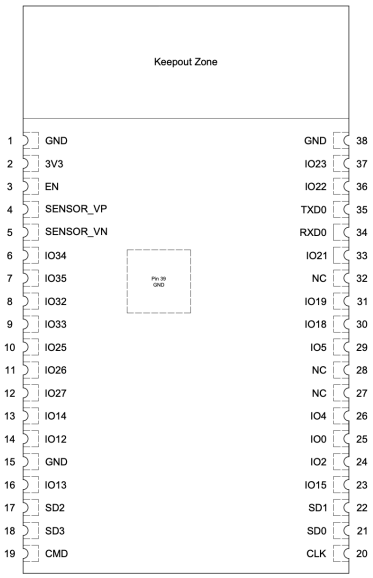


Ilustración 52. Pin layout del móduglo ESP32 Espressif. [33]

Se aprecia que cuenta con 38 pines, cuyo propósito se detalla en la tabla siguiente:

Name	No.	Type	Function
GND	1	P	Ground
3V3	2	P	Power supply
EN	3	I	Module-enable signal. Active high.

SENSOR_VP	4	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_VN	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
IO34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4
IO35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2*	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3*	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD*	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK*	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0*	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1*	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
NC1	27	-	-
NC2	28	-	-
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground

Ilustración 53. Distribución de los pines del módulo ESP32 de Espressif.[30]

Cabe destacar que los pines SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, es decir, del pin GPIO6 al GPIO11 están conectados al flash SPI. No se recomienda su uso. [33]

Se puede apreciar también en el datasheet que existen unos pines predeterminados para una determinada función, es decir, que se pueden emplear para otros fines, pero se diseñaron inicialmente para funcionar de una manera determinada. Como una especie de configuración por hardware, así que será aconsejable dejarlos libres para tal uso. Se trata de los pines: MTDI, GPIO0, GPIO2, MTDO, GPIO5.

Cabe destacar el tema de la alimentación del módulo. Se muestran en la siguiente tabla los valores máximos de operación a no superar:

Symbol	Parameter	Min	Max	Unit
VDD33	Power supply voltage	–0.3	3.6	V
I_{output}[1]	Cumulative IO output current	-	1,100	mA
T_{store}	Storage temperature	–40	105	°C

Ilustración 54. Valores máximos de operación. [33]

Symbol	Parameter	Min	Typical	Max	Unit
VDD33	Power supply voltage	3.0	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T	Operating ambient temperature	–40	-	85	°C

Ilustración 55. Valores recomendados de operación. [33]

Cabe destacar como cambio que este módulo incorpora una alimentación de 5V.

ANEXO C – OBTENCIÓN DIRECCIÓN IP DE LA RED WIFI

En este anexo se trata el método de obtención de la dirección ip de la red Wifi a la que está conectado el ordenador. Es de gran ayuda ya que esa red wifi es la que se va a emplear para conectar el módulo ESP32.

Primero se debe abrir el símbolo del sistema o terminal del pc, escribir el comando ipconfig/ifconfig en función de si es Windows o Mac y presionar el botón enter.

Se deberá buscar la sección correspondiente al adaptador de red inalámbrica. La dirección IP de la red wifi se mostrará junto a “Dirección IPv4” en Windows o “inet” en macOS.

En el caso de sólo disponer de un dispositivo inteligente o tableta, también se podría acceder a la dirección IP de la red Wifi. Se debería acceder a la configuración de Wifi en el dispositivo en la sección Avanzado o Detalles.

También se puede acceder en mac desde Ajustes del Sistema > Red > Wifi > detalles como aparece a continuación:

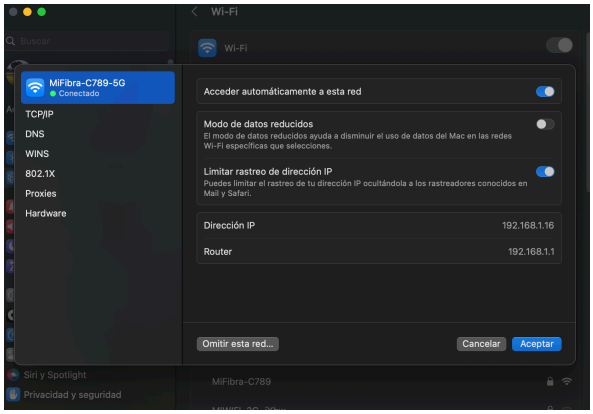


Ilustración 56. Obtención ip de la red Wi-Fi.

ANEXO D - CÓDIGO ARDUINO SERVIDOR

```
#include <WiFi.h>
#include <SPI.h>

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23

// Se crean los pines para dirección y datos
#define SPI_AS 26 //Dirección
#define SPI_DS 33 //Datos

//Se crea el pin para distinguir entre lectura y escritura
#define RW 27

//CONSTANTES DE WIFI
const char *ssid = "EM_BD";
const char *password = "46528575";

// Configuración de la dirección IP estática
IPAddress local_IP(192, 168, 1, 113);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 0, 0);

WiFiServer server(80);
SPIClass spi(HSPI);

String JoinHexStrings(String hex1, String hex2, String hex3, String hex4) {
  String result = "<";  // Añade el carácter '<' al principio
  result += hex1;
  result += hex2;
  result += hex3;
  result += ";";  // Añade el símbolo ';' entre la tercera y la cuarta cadena
  result += hex4;
  result += ">";  // Añade el carácter '>' al final
  return result;
}

void SplitHexStrings(String message, String &hex1, String &hex2, String &hex3, String &hex4) {

  // Elimina los caracteres '<' y '>' del principio y del final
  message = message.substring(1, message.length() - 1);

```

Ilustración 57. Primera parte del código del servidor

```
43    // Encuentra la posición del símbolo ';'
44    int semicolonPos = message.indexOf(';');
45    if (semicolonPos == -1) {
46      Serial.println("El mensaje no tiene el formato correcto");
47      return;
48    }
49    // Divide el mensaje en las cuatro cadenas hexadecimales
50    hex1 = message.substring(0, 2);  // Los dos primeros caracteres hexadecimales
51    hex2 = message.substring(2, 10);  // Los 8 siguientes caracteres hexadecimales. Esta es la dirección de la FPGA.
52    hex3 = message.substring(10, semicolonPos);  // Los caracteres restantes hasta el ';'
53    hex4 = message.substring(semicolonPos + 1);  // Los caracteres después del ';'
54  }
55
56  String HexToAscii(String hexStr) {
57    int len = hexStr.length();
58
59    len = hexStr.length() / 2;
60    String output;
61    output.reserve(len);
62    char buf[5] = {'\0', '\0', '\0', '\0', '\0'};
63    unsigned int c;
64
65    for (unsigned int i = 0; i < hexStr.length(); i += 2) {
66      buf[0] = hexStr[i];
67      buf[1] = hexStr[i + 1];
68      sscanf(buf, "%x", &c);
69      output += (char)c;
70    }
71
72    return output;
73  }
74
75  String AsciiToHex(String asciiStr) {
76    String hexStr = "";
77    int len = asciiStr.length();
78
79    for (unsigned int i = 0; i < asciiStr.length(); i++) {
80      char c = asciiStr[i];
81      if (c < 16) hexStr += '0';  // Asegura que los caracteres hexadecimales tengan dos dígitos
82      hexStr += String(c, HEX);
83    }
84    return hexStr;
```

Ilustración 58. Segunda parte del código del servidor.

```
84    return hexStr;
85  }
86
87  void setup() {
88    Serial.begin(9600);
89    if (!WiFi.config(local_IP, gateway, subnet)) {
90      Serial.println("STA Failed to configure");
91    }
92    WiFi.begin(ssid, password);
93
94    while (WiFi.status() != WL_CONNECTED) {
95      delay(1000);
96      Serial.println("Connecting to WiFi...");
97    }
98
99    Serial.println(WiFi.localIP());
100
101   server.begin();
102
103   spi.begin(SPI_SCK, SPI_MISO, SPI_MOSI, -1);  // SCK, MISO, MOSI, SS (no se utiliza)
104
105   // Configura los pines SS como salidas.
106   pinMode(SPI_AS, OUTPUT);
107   pinMode(SPI_DS, OUTPUT);
108   pinMode(RW, OUTPUT);
109 }
110
111 void loop() {
112   WiFiClient client = server.available();
113   String operacion, direccion, longitud;
114   String mensaje;
115   bool mensajeEnviadobool = false;
116   String s;
117
118   if (client) {
119     while (client.connected()) {
120       while (client.available() > 0) {
121         char c = client.read();
122         if (c != '\n') {
123           s += c;
124           continue;
```

Ilustración 59. Tercera parte del código del servidor.

```
125         }
126         Serial.println(s);
127
128         spi.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
129
130       String hex1, hex2, hex3, hex4;
131       SplitHexStrings(mensaje,hex1,hex2,hex3,hex4);
132
133
134       if (HexToAscii(hex1) == "R") {
135       digitalWrite(RW, HIGH);
136
137       // Activa SPI_AS durante 8 pulsos para enviar hex2 por MOSI
138       digitalWrite(SPI_AS, LOW);
139       for (int i = 0; i < hex2.length(); i += 2) {
140     String byteString = hex2.substring(i, i+2);
141     byte b = (byte) strtol(byteString.c_str(), NULL, 16);
142     spi.transfer(b);
143     Serial.println(b, HEX);
144     }
145     digitalWrite(SPI_AS, HIGH);
146     digitalWrite(RW, LOW);
147
148     delay(100); // Espera una décima de segundo antes de la próxima operación.
149
150       // Activa SPI_DS durante el número de pulsos necesarios para recibir datos por MISO
151       digitalWrite(SPI_DS, LOW);
152       for (int i = 0; i < 8; i++) {
153         byte receivedData = spi.transfer(0x00); // Dato "dummy" para recibir datos del esclavo
154         Serial.println("Haciendo lectura del SPI...");
155     }
156     digitalWrite(SPI_DS, HIGH);
157
158       } else if (HexToAscii(hex1) == "W") {
159
160       //Se muestra por el monitor serie el mensaje a escribir
161       Serial.println("Se va a escribir en la FPGA");
162       Serial.println(hex4);
163
164     // Activa SPI_AS durante 8 pulsos para enviar hex2 por MOSI
165       digitalWrite(SPI_AS, LOW);
166     digitalWrite(RW, HIGH);
167       for (int i = 0; i < hex2.length(); i += 2) {
```

Ilustración 60. Cuarta parte del código del servidor.

```
      for (int i = 0; i < hex2.length(); i += 2) {
      String byteString = hex2.substring(i, i+2);
     byte b = (byte) strtol(byteString.c_str(), NULL, 16);
     spi.transfer(b);
     Serial.println(b, HEX);
    }
     digitalWrite(SPI_AS, HIGH);

     delay(100); // Espera una décima de segundo antes de la próxima operación.

     // Activa SPI_DS durante el número de pulsos necesarios para enviar datos por MOSI

     digitalWrite(SPI_DS, LOW);
      digitalWrite(RW, HIGH);
      for (int i = 0; i < hex4.length(); i += 2) {
      String byteString = hex4.substring(i, i+2);
     byte b = (byte) strtol(byteString.c_str(), NULL, 16);
     spi.transfer(b);
      Serial.println(b, HEX);
     }
     digitalWrite(SPI_DS, HIGH);
    }
     spi.endTransaction();

     delay(100); // Espera una décima de segundo antes de la próxima operación.

       if (!mensajeEnviadobool) {
         String mensaje_hex = AsciiToHex(mensaje);
         String respuesta = JoinHexStrings(AsciiToHex("A"), direccion, longitud, mensaje_hex);
         Serial.println(respuesta);
         client.print(respuesta);
         mensajeEnviadobool = true;
       }
       s = "";  //Vacía s
      }
    }
    client.stop();
  }
}
```

Ilustración 61. Quinta parte del código del servidor.

```
Connecting to WiFi...
192.168.1.113
<57c0a8010254;5072696d65726120707275656261206465206573637269747572 61>
```

Ilustración 62. Ejemplo de mensaje recibido "primera prueba de escritura".

ANEXO E - CÓDIGO ARDUINO CLIENTE

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>

//Variables Wifi
const char* host = "192.168.1.113";

// Variables para almacenar los datos del mensaje
String tipoMensaje;
String direccionFPGA;
String mensaje;
String ssid;
String password;
String mensaje_hex;
int longitud_mensaje;
int ESP32_TCP_PORT = 80;

String JoinHexStrings(String hex1, String hex2, String hex3, String hex4) {
  String result = "<";  // Añade el carácter '<' al principio
  result += hex1;
  result += hex2;
  result += hex3;
  result += ";";  // Añade el símbolo ';' entre la tercera y la cuarta cadena
  result += hex4;
  result += ">";  // Añade el carácter '>' al final
  return result;
}

void SplitHexStrings(String message, String &hex1, String &hex2, String &hex3, String &hex4) {

  // Elimina los caracteres '<' y '>' del principio y del final
  message = message.substring(1, message.length() - 1);

  // Encuentra la posición del símbolo ';'
  int semicolonPos = message.indexOf(';');
  if (semicolonPos == -1) {
    Serial.println("El mensaje no tiene el formato correcto");
    return;
  }

```

Ilustración 63. Primera parte del código del cliente.

```
  // Divide el mensaje en las cuatro cadenas hexadecimales
  hex1 = message.substring(0, 2);  // Los dos primeros caracteres hexadecimales
  hex2 = message.substring(2, 10);  // Los 8 siguientes caracteres hexadecimales
  hex3 = message.substring(10, semicolonPos);  // Los caracteres restantes hasta el ';'
  hex4 = message.substring(semicolonPos + 1);  // Los caracteres después del ';'
}

String HexToAscii(String hexStr) {
  int len = hexStr.length();

  len = hexStr.length() / 2;
  String output;
  output.reserve(len);
  char buf[5] = {'\0', '\0', '\0', '\0', '\0'};
  unsigned int c;

  for (unsigned int i = 0; i < hexStr.length(); i += 2) {
    buf[0] = hexStr[i];
    buf[1] = hexStr[i + 1];
    sscanf(buf, "%x", &c);
    output += (char)c;
  }

  return output;
}

//código para conectarse a la red wifi
void Conectarwifi(){

  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Buscando redes Wi-Fi disponibles...");
  int n = WiFi.scanNetworks();
  if (n == 0) {
    Serial.println("No se encontraron redes Wi-Fi disponibles");
  } else {
    Serial.print(n);
    Serial.println(" redes Wi-Fi encontradas:");
```

Ilustración 64. Segunda parte del código del cliente.

```
  Serial.println(" redes Wi-Fi encontradas:");
  for (int i = 0; i < n; ++i) {
    Serial.print(i + 1);
    Serial.print(": ");
    Serial.print(WiFi.SSID(i));
    Serial.print(" (");
    Serial.print(WiFi.RSSI(i));
    Serial.println(" dBm)");
    delay(10);
  }
}

Serial.println("Por favor, introduce el número de la red a la que te quieres conectar:");
while (Serial.available() == 0) {}
int networkNumber = Serial.parseInt();
ssid = WiFi.SSID(networkNumber - 1);

Serial.println("Por favor, introduce la contraseña de la red:");
while (Serial.available() == 0) {}
password = Serial.readString();

WiFi.begin(ssid.c_str(), password.c_str());

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}

Serial.println("");
Serial.println("WiFi conectado.");

Serial.println("Por favor, introduce la dirección IP que deseas asignar al ESP32 (formato: xxx.xxx.xxx.xxx):");
while (Serial.available() == 0) {}
String requestedIp = Serial.readString();
IPAddress staticIp;
staticIp.fromString(requestedIp);

IPAddress gateway(192, 168, 1, 1);  // gateway predeterminado.
IPAddress subnet(255, 255, 255, 0);  // máscara de subred predeterminada.

WiFi.config(staticIp, gateway, subnet);

```

Ilustración 65. Tercera parte del código del cliente.

```
123
124   Serial.println("Dirección IP estática asignada.");
125   Serial.println("Dirección IP: ");
126   Serial.println(WiFi.localIP());
127 }
128
129 String AsciiToHex(String asciiStr) {
130   String hexStr = "";
131   int len = asciiStr.length();
132
133   for (unsigned int i = 0; i < asciiStr.length(); i++) {
134     char c = asciiStr[i];
135     if (c < 16) hexStr += '0';  // Asegúrate de que los caracteres hexadecimales tengan dos dígitos
136     hexStr += String(c, HEX);
137   }
138   return hexStr;
139 }
140
141 void setup() {
142  Conectarwifi();
143  //server.begin();  // Inicia el servidor
144 }
145
146 void loop() {
147   // Preguntar al usuario si desea enviar un mensaje
148   Serial.println("¿Deseas enviar un mensaje al ESP32? (s/n)");
149   while (Serial.available() == 0) {}
150   char respuesta = Serial.read();
151   if (respuesta == 'n') {
152     return;
153   }
154
155   // Preguntar al usuario si el mensaje es de lectura o escritura
156   Serial.println("¿El mensaje es de lectura (R) o escritura (W)?");
157   while (Serial.available() == 0) {}
158   tipoMensaje = Serial.readString();
159   String tipoMensaje_hex = AsciiToHex(tipoMensaje);
160
161   // Preguntar al usuario la dirección de la FPGA
162   Serial.println("Introduce la dirección de la FPGA del siguiente modo (AABBCCDD):");
163   while (Serial.available() == 0) {}
164   direccionFPGA = Serial.readString();
```

Ilustración 66. Cuarta parte del código del cliente.

```
164   direccionFPGA = Serial.readString();
165   String direccionFPGA_hex = AsciiToHex(direccionFPGA);
166   Serial.println("La dirección de la FPGA es la siguiente:");
167   Serial.println(direccionFPGA);  // Imprime la dirección de la FPGA para verificarla
168
169   // Preguntar al usuario el contenido del mensaje
170   Serial.println("Introduce el mensaje a enviar:");
171   while (Serial.available() == 0) {}
172   mensaje = Serial.readString();
173
174  // Calcular la longitud en bits del mensaje
175   int longitud_hex = (mensaje.length())*2; //mensaje.length() cuenta el número de caracteres, luego se multiplica por 2
176   //ya que cada caracter son 2 números hexadecimales y te da la longitud en hexadecimal (1---4)
177   String longitud_hex_s;
178   longitud_hex_s = String(longitud_hex);
179
180   // Crear el mensaje en formato hexadecimal
181   mensaje_hex = AsciiToHex(mensaje);
182   Serial.println("El mensaje a enviar es el siguiente");
183   Serial.println(HexToAscii(AsciiToHex(mensaje)));
184
185   //UNIR CADENAS DE TEXTO
186   String joined = JoinHexStrings(tipoMensaje_hex ,direccionFPGA_hex, longitud_hex_s, mensaje_hex);
187   Serial.println(joined);
188
189   //Conexión con servidor y envío de mensaje
190   WiFiClient client;
191
192   if (!client.connect(host, 80)) {
193     Serial.println("Connection to host failed");
194     delay(1000);
195     return;
196   }
197
198  // Envía un mensaje al servidor.
199   client.println(joined);
200
201   String s;
202
```

Ilustración 67. Quinta parte del código del cliente.

```
203   while (client.connected()) {
204     if (client.available() > 0){
205       char c = client.read();
206       s += c;
207
208   Serial.println(s);  // Imprimirá 'b'
209
210   // Dividir mensaje
211   String primer, segundo, tercero, cuarto;
212   SplitHexStrings(s, primer, segundo, tercero, cuarto);
213   Serial.println("Letra inicial: " + HexToAscii(primer));
214   Serial.println("Dirección IP de la FPGA: " + HexToAscii(segundo));
215   Serial.println("Tamaño del mensaje: " + HexToAscii(tercero));
216   Serial.print("Mensaje en ASCII: " + HexToAscii(cuarto));
217     }
218   }
219   client.stop();
220   }
```

Ilustración 68. Sexta parte del código del cliente.

```
Introduce la dirección de la FPGA del siguiente modo (AAA.BBB.CCC.DDD):
192.168.1.2
c0a80102
192.168.1.2
Introduce el mensaje a enviar:
54
216
El mensaje a enviar es el siguiente
Primera prueba de escritura
<57c0a8010254:5072696d65726120707275656261206465206573637269747572 61>
```

Ilustración 69. Ejemplo de lo escrito por pantalla para enviar al servidor.

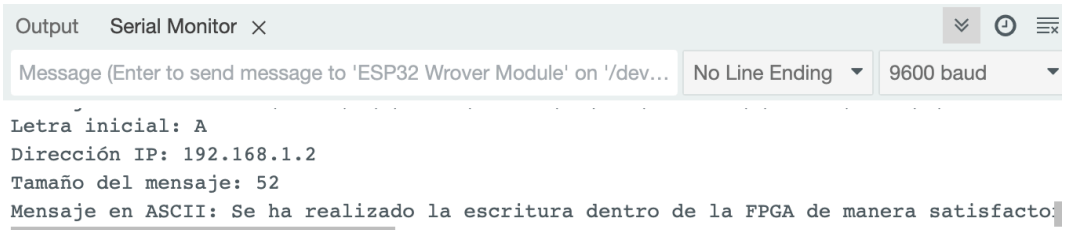


Ilustración 70. Mensaje de respuesta recibido por el cliente del servidor.

AXEXO F – CREACIÓN DE FUNCIONES EN ARDUINO IDE

Se va a proceder a explicar paso a paso lo necesario para crear una librería propia en Arduino IDE.

Primero se debe crear el Sketch que contenga el código que se desea convertir a librería. Como se aprecia en la siguiente imagen:

IPest.cpp.ino

```
#include "Arduino.h"
#include <WiFi.h>

#define Baudios 115200;

IPest::IPest(const char* ssid, const char* password, IPAddress local_IP, IPAddress gateway, IPAddress subnet, IPAddress primaryDNS)
{
  _ssid = ssid;
  _password = password;
  _local_IP = local_IP;
  _gateway = gateway;
  _subnet = subnet;
  _primaryDNS = primaryDNS;
}

void IPest() {
  Serial.begin(Baudios);

 if (!WiFi.config(_local_IP, _gateway, _subnet, _primaryDNS)) {
    Serial.println("Error al configurar la dirección IP estática");
  }

  WiFi.begin(_ssid, _password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }

  Serial.println("Conectado a WiFi");
  Serial.print("Dirección IP: ");
  Serial.println(WiFi.localIP());
}

```

Ilustración 71. Creación librería arduino.

Se deben crear dos archivos para la librería: un archivo de encabezado, con la extensión .h, y un archivo de código fuente, con la extensión .cpp. En el archivo de encabezado se debe incluir las definiciones para la librería, incluyendo una lista de todo lo que está en su interior. En el archivo de código fuente se debe incluir el código real de la librería [34]. Todo se muestra en las imágenes a continuación:

```
/*

    IPest.h - Library for setting a static IP address.
    Created by Javier C.S., July 17, 2023.
*/

#ifndef IPest_h
#define IPest_h

#include "Arduino.h"

class IPestatica
{
  public:
    void IPest(const char* ssid, const char* password);

  private:

    const char* _ssid;
    const char* _password;
    IPAddress _local_IP;
    IPAddress _gateway;
    IPAddress _subnet;
    IPAddress _primaryDNS;
}
```

Ilustración 72. Creación de librería en arduino.

Se debe mover las funciones y variables del sketch creado a los archivos de la librería, asegurándose que se hayan encapsulado correctamente en una clase y que puedan ser accesibles desde fuera de la misma. Es decir, que éstos sean públicos. Hay que asegurarse de incluir un archivo #include en el archivo de encabezado para tener acceso a los tipos y constantes estándar de Arduino [34].

Una vez creados los archivos de la librería se deben mover a la carpeta "libraries" dentro del sketchbook para que estén disponibles.

IPest	hoy, 14:41	--	Carpeta
IPest.cpp	hoy, 14:28	766 bytes	Texto
IPest.h	hoy, 14:39	982 bytes	Texto

Ilustración 73. Archivo librerías arduino.

También hay una alternativa y es comprimir los archivos de la librería como se aprecia a continuación y añadirlos según se indica en el tutorial de la página oficial: En la barra de menú seleccionar Sketch> Include Library > Add .ZIP Library [35].

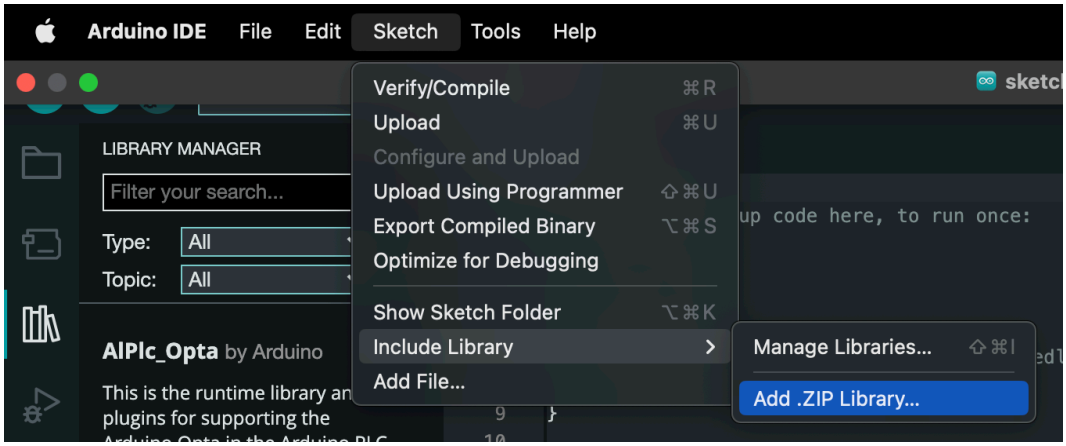


Ilustración 74. Ejemplo adicción librería.

En el caso tratado no se emplea esta función del ARDUINO IDE, pero cabe destacar su posible uso para posibles mejoras y optimización de código.

ANEXO G: SISTEMA HEXADECIMAL

El sistema hexadecimal es un sistema numérico de base 16. Utiliza dieciséis símbolos distintos para representar números. Estos símbolos son 0-9 y A-F, donde A representa 10, B representa 11, hasta F que representa 15.

Ejemplos rápidos: 0004 en hexadecimal sería el 4, pero 0010 en hexadecimal sería el 16, F.

En el ejemplo tratado el mayor número que se puede formar es FFFF que es 65535 en decimal. Porque como cada dígito puede ser desde 0-9 + A-F y cada dígito representa 4 bits. 2^4 posibles valores de 4 bits es 16, ya que cada bit puede tomar valor 0-1. En total se tienen 2 bytes 2^16- 1 es el mayor número que se puede representar.

Si en el ejemplo tenemos una F, tendríamos los 4 bits 1111, si fuese una B, 1011…

ANEXO H: TABLA ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Ilustración 75. Tabla ascii utilizada. [36]