ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

PROGRAMACIÓN DE MÓDULOS ENTRENADORES LÓGICOS CON CONEXIÓN INALÁMBRICA PARA EL LABORATORIO DE TECNOLOGÍA ELÉCTRICA Y ELECTRÓNICA ÁREA DE SISTEMAS DIGITALES

TRABAJO PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ELECTRÓNICA Y TELECOMUNICACIONES

PABLO JOSÉ BAHAMONDE CAILLAMARA

pablo.bahamonde@epn.edu.ec

DIRECTOR: ING. FERNANDO BECERRA, MSc.

fernando.becerrac@epn.edu.ec

CODIRECTOR: ING. FABIO GONZÁLEZ, MSc.

fabio.gonzalez@epn.edu.ec

Quito, enero de 2021

DECLARACIÓN

Yo, Pablo José Bahamonde Caillamara, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Pablo José Bahamonde Caillamara

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por Pablo José Bahamonde Caillamara, bajo nuestra supervisión.

Ing. Fernando Becerra, MSc.	**ING. Fabio González, MSc.**
DIRECTOR DE PROYECTO	**CODIRECTOR DE PROYECTO**

AGRADECIMIENTOS

A mi madre Gladys Bahamonde por su apoyo en todos estos años, en los cuales, con amor, cariño y sobre todo con paciencia me ha enseñado que con esfuerzo y dedicación puedo lograr alcanzar todos los objetivos que me proponga en la vida.

A mis abuelos José Luis Bahamonde y María Elina Caillamara a los cuales considero como mis padres, por su apoyo incondicional que nos brindaron a mí y a mi madre, gracias por ese inmenso cariño.

De manera especial quiero agradecer a mi abuelo José Luis Bahamonde, él supo actuar como el padre que no tuve, siendo una gran influencia en mi vida. Aunque ya no se encuentre siempre será la persona que más admire y por la cual me motiva a seguir adelante.

A la Escuela de Formación de Tecnólogos por haberme brindado los conocimientos y la oportunidad de desarrollar este trabajo de titulación para uno de sus laboratorios.

Un agradecimiento especial para la Ing. Viviana Párraga y el Ing. Fernando Becerra por su apoyo en el desarrollo de este proyecto.

Un agradecimiento especial al Ing. Fabio González por el apoyo y los consejos brindados para la mejora de este proyecto.

DEDICATORIA

El presente proyecto va dedicado a mi madre, por su apoyo, comprensión y consejos que me supo brindar.

A mi abuela, que siempre me ha brindado su amor y apoyo.

A mi abuelo, que siempre será uno de los motores que me guía a seguir adelante y es a quien más extraño.

ÍNDICE DE CONTENIDOS

DECLARACIÓN i

CERTIFICACIÓN ii

AGRADECIMIENTOS iii

DEDICATORIA iv

ÍNDICE DE CONTENIDOS v

ÍNDICE DE FIGURAS vii

RESUMEN 1

ABTRACT 2

1 INTRODUCCIÓN 3

1.1 Planteamiento del problema 3

1.2 Justificación 3

1.3 Objetivo General 4

1.4 Objetivo Especifico 4

2 METODOLOGÌA 5

2.1 Arduino 5

2.2 *Software* de Arduino 6

2.3 Librerías de Arduino 8

2.4 *Hardware* de Arduino 9

2.5 *Hardware* Nextion 10

2.6 *Software* Nextion 10

2.7 Comunicación inalámbrica WiFi 16

2.8 Módulo WiFi 18

3 RESULTADOS Y DISCUSIÓN 20

3.1 Programación de módulos entrenadores lógicos 20

3.2 Configuración de módulo WiFi esp8266 como AP 39

3.3 Configuración página de registro 44
3.4 Instalación del programa en un módulo de prueba. 53
3.5 Pruebas de funcionamiento. 59
4 CONCLUSIONES Y RECOMENDACIONES 59
4.1 Conclusiones 59
4.2 Recomendaciones 60
5 BIBLIOGRAFÍA 62
6 ANEXOS 65
6.1 Anexo A 65
6.2 Anexo B Diagrama de flujo 66
6.3 Anexo C Código Arduino Mega 67
6.4 Anexo D Código Módulo esp8266 81

ÍNDICE DE FIGURAS

Figura 2.1: Página oficial de Arduino 5

Figura 2.2*:* Ventana Arduino IDE 6

Figura 2.3: Monitor serial 7

Figura 2.4: Menús de Arduino 7

Figura 2.5: Arduino IDE: Incluir librerías 8

Figura 2.6: Gestor de librerías de Arduino IDE 8

Figura 2.7: Variantes de modelos de las placas Arduino. [8] 9

Figura 2.8: Pantalla Nextion. [10] 10

Figura 2.9: Componentes de *Nextion Editor* 10

Figura 2.10: Tabla de atributos de botón 11

Figura 2.11: Ejemplo de uso del botón en *Nextion Editor* 11

Figura 2.12: Páginas de la interfaz gráfica de los módulos entrenadores [11] 12

Figura 2.13: Página 0 información de registro. 13

Figura 2.14: Página 1 pantalla principal. 13

Figura 2.15: Página 2 Comprobador de *display* de 7 segmentos 14

Figura 2.16: Página 3 Comprobador de compuertas. 14

Figura 2.17: Página 4 Interfaz gráfica del voltímetro 15

Figura 2.18: Página 5 Interfaz gráfica del generador de frecuencia. 15

Figura 2.19: Página 6 Interfaz gráfica del Comprobador de transistores. 16

Figura 2.20: Página 7 Interfaz modo *debug* 16

Figura 2.21: Distribución de pines del módulo esp8266. [15] 20

Figura 3.1: Diagrama de flujo primera sección 21

Figura 3.2: Selección de placa Arduino 21

Figura 3.3: Ejemplo de separación de datos 22

Figura 3.4: Ejemplo de evento *push* y *pop* 24

Figura 3.5: Declaración de nombres de pines 24

Figura 3.6: Declaración de variables de la pantalla táctil 25

Figura 3.7: Declaración de variables en Arduino 25

Figura 3.8: Eventos táctiles 26

Figura 3.9: Diagrama de flujo segunda sección 27

Figura 3.10: Declaración de las variables de funcionamiento 28

Figura 3.11: Separación de datos 29

Figura 3.12: Uso del comando *toCharArray ()* 29

Figura 3.13: Cambio de datos de tipo texto en la pantalla Nextion. 29

Figura 3.14: Sentencia if para verificación de datos 31

Figura 3.15: Ejemplo de acceso a módulo entrenador. 31

Figura 3.16: Eventos táctiles 32

Figura 3.17: Actualización de los valores de voltímetro, frecuencímetro, reloj y estado lógico 33

Figura 3.18: Ejemplo del uso de la Librería *FreqCount.h* 33

Figura 3.19: Obtención estados lógicos. 34

Figura 3.20: Obtención del voltímetro. 34

Figura 3.21: Pantalla Nextion valores siempre en pantalla 34

Figura 3.22: Comprobación de LEDs y *displays* 35

Figura 3.23: Módulo entrenador, comprobador de LEDs 35

Figura 3.24: Código para la señal de reloj 36

Figura 3.25: Módulo entrenador, señal de reloj 36

Figura 3.26: Generar señal PWM 37

Figura 3.27: Detección de compuertas lógicas 74XX 37

Figura 3.28: Bucle para *and*, *nand, or* y *xor* 38

Figura 3.29: Compuerta *NOT*.....38

Figura 3.30: Compuertas *NOR* 39

Figura 3.31: Arduino IDE menú Archivo 39

Figura 3.32: Menú de preferencias de Arduino IDE 40

Figura 3.33: Arduino IDE menú Herramientas.....40

Figura 3.34: Arduino IDE, Gestor de tarjetas.....41

Figura 3.35: Arduino IDE menú Herramientas > Placas 41

Figura 3.36: Arduino IDE menú Herramientas > *Upload Speed* 42

Figura 3.37: Instalación del controlador finalizada 43

Figura 3.38: Arduino IDE ejemplos de esp8266. 43

Figura 3.39: Ubicación de la carpeta de *tools*. 45

Figura 3.40: Arduino IDE menú *ESP8266 Sketch Data Upload* 45

Figura 3.41: Instalación de librería *ESPAsyncWebServer*.....46

Figura 3.42: Instalación de librería *ESPAsyncTCP* 46

Figura 3.43: Librerías incluidas esp8862.....47

Figura 3.44: *Sketches* necesarios para el servidor.....47

Figura 3.45: Carpeta contenedora del *sketch* del esp8862 47

Figura 3.46: Configuración de SSID y *password*. 47

Figura 3.47: Configuración de dirección IP local para el módulo esp8266 48

Figura 3.48: Configuración de módulo esp8266 como AP.....49

Figura 3.49: Configuración de servidor *web*.....49

Figura 3.50: Página de registro.....50

Figura 3.51: Envío de datos por comunicación serial a Arduino.....50

Figura 3.52: Contenido carpeta data.....51

Figura 3.53: Arduino IDE, *Sketch Data Upload* 52

Figura 3.54: Sistema de archivos de SPIFFS 52

Figura 3.55: Código fuente de la página HTML registro 53

Figura 3.56: Arduino conectado a la computadora 53

Figura 3.57: Módulo entrenador lógico armado 54

Figura 3.58: Módulo entrenador, Etapa de encendido 54

Figura 3.59: Módulos entrenadores, registro 55

Figura 3.60: Módulo entrenador, registro exitoso 55

Figura 3.61: Módulo entrenador, registro fallido 56

Figura 3.62: Comprobador de LEDs y *displays* 56

Figura 3.63: Comprobador de compuertas 57

Figura 3.64: Comprobador de transistores 58

Figura 3.65: Voltímetro 58

Figura 3.66: Pantalla principal del módulo lógico 59

ÍNDICE DE TABLAS

Tabla 2.1: Especificaciones placa Arduino Mega 2560. [9] 9

Tabla 2.2: Características de 802.11 [12] 17

Tabla 2.3: Características básicas de la banda de 2.4 GHz del módulo esp8266 17

Tabla 2.4: Canales de la banda 2.4 GHz. [13] 18

Tabla 2.5: Características del módulo 8266. [14] 19

Tabla 3.1: Lista de comandos de la librería *Timerone.h.* [16] 22

Tabla 3.2: Comandos para la declaración de variables de la librería *Nextion.h* [17] ... 23

Tabla 3.3: Tipos de variables de Arduino. [19] 26

Tabla 3.4: Descripción de funciones de los pines. [11] 27

Tabla 3.5: Variables necesarias asociadas a la pantalla Nextion. [11] 30

Tabla 3.6: Características de los botones en Nextion. [11] 32

Tabla 3.7: Modos de funcionamiento del módulo esp8266. [15] 44

RESUMEN

En este documento se desarrolla la programación de módulos entrenadores lógicos con conexión inalámbrica para el laboratorio de tecnología eléctrica y electrónica área de sistemas digitales de la Escuela de Formación de Tecnólogos.

En la primera sección se describe como se desarrollaron las practicas del laboratorio de sistemas digitales, y se señalan los elementos de un circuito análogo o digital está compuesto, indicando los elementos necesarios que debe incluir los módulos entrenadores para verificar el correcto funcionamiento de un circuito.

En la segunda sección incluye información básica sobre el *software* y *hardware* de Arduino, las pantallas Nextion y sobre el uso librerías en Arduino.

En la tercera sección describe las librerías usadas para la programación de los módulos entrenadores al igual presenta los comandos usados en este programa, se describe el funcionamiento del código de los módulos entrenadores.

En la Cuarta sección se encuentran las conclusiones y recomendaciones.

En la Quinta sección se encuentra la bibliografía usada en el documento.

En la Sexta sección se encuentra los anexos.

ABTRACT

This document develops the programming of logic training modules with wireless connection for the electrical and electronic technology laboratory and digital systems area of the School of Technology Training.

The first section describes how the practices of the digital systems laboratory were developed, and the elements are indicated an analog or digital circuit is composed, indicating the necessary elements that the training modules must include to verify the correct operation of a circuit.

The second section includes basic information about Arduino software and hardware, Nextion screens and about the use of Arduino libraries.

The third section describes the libraries used for programming the training modules, as well as the commands used in this program, and describes the operation of the training modules' code.

The fourth section contains conclusions and recommendations.

The fifth section contains the bibliography used in the document.

The sixth section contains the annexes.

1 INTRODUCCIÓN

1.1 Planteamiento del problema

En las prácticas de laboratorio de sistemas digitales, el estudiante debe demostrar que se cumple la teoría impartida en clase. Para esto a los estudiantes se les proporciona una guía por cada práctica en la cual se detalla, el tema, objetivos, trabajo preparatorio, circuito a implementar, equipo disponible en el laboratorio, procedimiento práctico y cuestionario [1]. El circuito desarrollado debe cumplir con el trabajo preparatorio planteado en la guía de laboratorio, por ejemplo: “Realizar la tabla y cálculos correspondientes para implementar la función F= Σ m (0,1,2,5,7,8,9,13,14,15) mediante un multiplexor de 8 a 1 (74151)” [1] y modificar el circuito según el instructor disponga.

En el desarrollo de las prácticas algunas veces se presentan dificultades que impiden que sea implementada con éxito, tales como: mala implementación del circuito, uso de dispositivos electrónicos dañados o incorrectos, entre otros. Estos inconvenientes provocan que las prácticas no se finalicen, dificultando la identificación de errores en el circuito. Si algún estudiante no logra completar la práctica dentro del horario establecido, este recibe una sanción en la calificación de dicha práctica.

Otro factor importante que obstaculiza la presentación de una práctica es la dificultad en identificar un elemento electrónico, como son las compuertas lógicas o los transistores, ya que la manipulación de estos elementos provoca que el encapsulado se deteriore y se borre el código del elemento. Todos estos factores dificultan la ejecución exitosa de las prácticas del componente práctico de la materia Electrónica Analógica y Digital de la carrera de Tecnología Superior en Redes y Telecomunicaciones, Pensum 2017 citado como caso de estudio.

1.2 Justificación

Un circuito analógico o digital debe estar compuesto por 5 elementos fundamentales que a continuación se menciona: generadores eléctricos son los que proporcionan energía al circuito, conductores que permiten circular la corriente, receptores que trasforman la energía eléctrica en otro tipo de energía, elementos de maniobra y control que permiten abrir o cerrar el circuito y elementos de protección que protegen al circuito de sobrecargas de tensión [2].

Las prácticas de laboratorio involucran el uso de compuertas lógicas, multiplexores, *flip-flops* y contadores [2]. Según la práctica a realizar se puede utilizar uno o más circuitos

integrados de acuerdo con el diseño del estudiante, como es el caso de la práctica de Sumadores en Binario del laboratorio de electrónica analógica y digital [3]. Dicho circuito llega a utilizar más de un CI, y cuando alguno de estos dispositivos llegan a fallar es difícil identificarlo, provocando que el estudiante desperdicie tiempo antes de poder solucionarlo.

Los elementos necesarios para verificar el correcto funcionamiento de un circuito digital o analógico incluyen:

- Receptores como LEDs o *displays*.
- Elementos de maniobra y control como compuertas lógicas o transistores.
- Elementos de protección como detectores de cortocircuito.
- Elementos de comprobación como detectores de estados lógicos en el caso de no contar con elementos receptores [2]
- Medidores como el voltímetro que permite medir la diferencia de potencial eléctrico entre 2 puntos de un circuito y un frecuencímetro que permite medir la frecuencia, estos equipos ayudan a encontrar fallos presentes en un circuito o en algún elemento de este.

El presente proyecto presenta la programación de módulos entrenadores lógicos conformados con fuentes de voltaje DC, generador de señal de reloj, frecuencímetro, voltímetro, LED de corto circuito y comprobadores de transistores TBJ, LEDs, *displays* de 7 segmentos ánodo y cátodo común, compuertas lógicas 74xx de doble entrada y estados lógicos, los cuales permitirán a los estudiantes desarrollar las prácticas del componente práctico de la materia de Electrónica Analógica y Digital, mejorando el tiempo y detectando más rápido los fallos que el circuito pueda llegar a tener. Los módulos lógicos incluirán una conexión inalámbrica 802.11 que permitirá guardar un registro de los estudiantes que usen los módulos.

1.3 Objetivo General

Programar módulos entrenadores lógicos con conexión inalámbrica para el Laboratorio de Tecnología Eléctrica y Electrónica área de Sistemas Digitales.

1.4 Objetivo Especifico

- Desarrollar un programa para cada función del módulo entrenador lógico y su conexión general.

- Desarrollar un programa que permita el envío y recepción de datos a través de una red inalámbrica con 802.11.
- Instalar los programas diseñados con su interfaz de conexión.
- Realizar pruebas de eficiencia de los programas desarrollados e integrados.

2 METODOLOGÌA

2.1 Arduino

Arduino nació como un proyecto desarrollado por estudiantes del Instituto de Diseño Interactivo de Ivrea, Italia, con el fin de dar a los estudiantes una alternativa económica para el uso de *software* de programación y *hardware* electrónico. [4] Yúbal define Arduino como [4]: “una plataforma de creación de electrónica de código abierto, la cual está basada en *hardware* y *software* libre, flexible y fácil de utilizar para los creadores y desarrolladores”.

Actualmente Arduino como compañía continúa creciendo, ofreciendo oportunidades al público de aprender más sobre Arduino con las posibilidades de obtener un certificado de Arduino, de igual manera presenta de manera continua y abierta a todo el público las actualizaciones sobre las librerías, entre otras opciones, estas noticias son actualizadas y se las puede encontrar en la página oficial de Arduino como se observa en la figura 2.1.

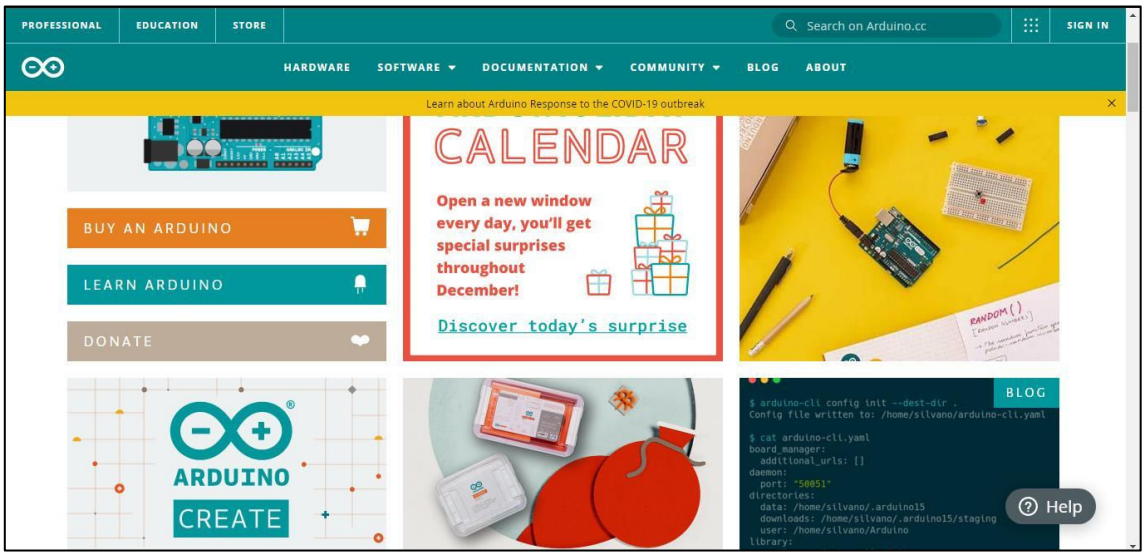


Figura 2.1: Página oficial de Arduino

2.2 *Software* de Arduino

El *software* de Arduino o Arduino IDE (*Integrated Development Environment*), ofrece a los usuarios un ambiente multiplataforma que les permite escribir código a través de un editor de texto fácil de utilizar, que permite realizar funciones de edición texto. En la figura 2.2 se puede observar las opciones que Arduino IDE ofrece para trabajar en un *sketch*. [5]

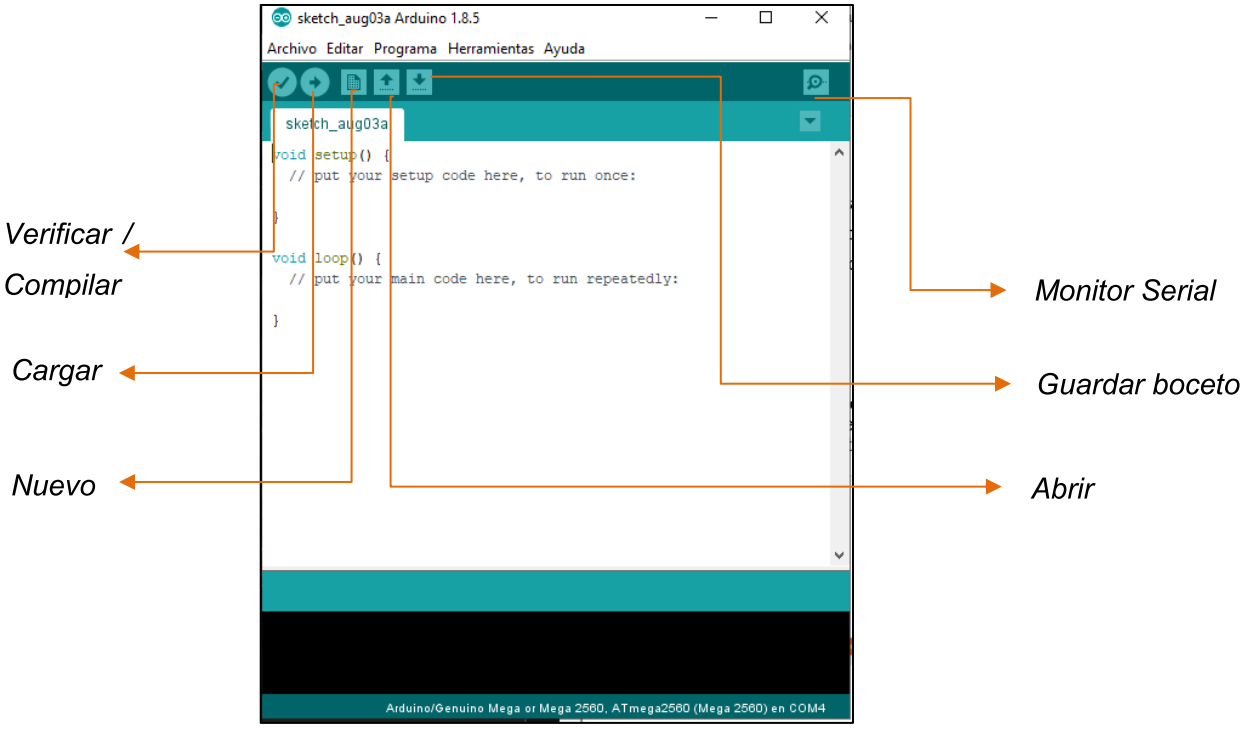


Figura 2.2*:* Ventana Arduino IDE

- Verificar / compilar: Comprueba si el documento tiene errores al compilarlo; informará el uso de memoria para el código y las variables en el área de la consola.
- Carga: Carga el archivo binario en la placa configurada a través del puerto configurado.
- Nuevo: Abre una nueva ventana de Arduino IDE para desarrollar.
- Guardar Boceto: Guarda el documento con el nombre actual o uno diferente.
- Abrir: Presenta un menú de todos los documentos.
- Monitor Serial: Abre el monitor serial.

Arduino IDE cuenta con un monitor serial que muestra los datos que son enviados al puerto serial y también permite enviar datos por el puerto serial. Para acceder al monitor

serial es necesario conectar una placa de Arduino. La ventana del monitor serial se la puede observar en la figura 2.3.

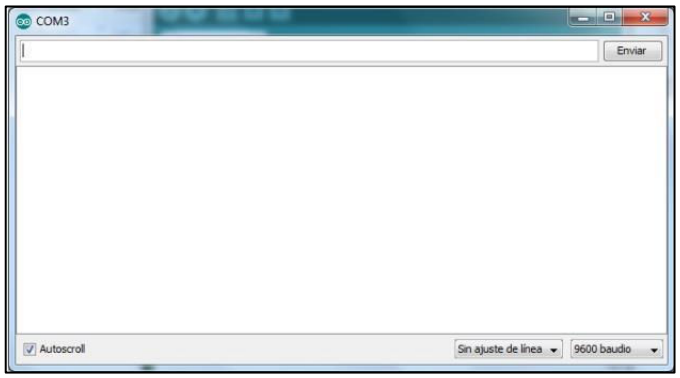


Figura 2.3: Monitor serial

Arduino posee 5 menús como se observa en la figura 2.4, se pueden encontrar más menús los cuales están disponibles según el trabajo que se esté realizando.

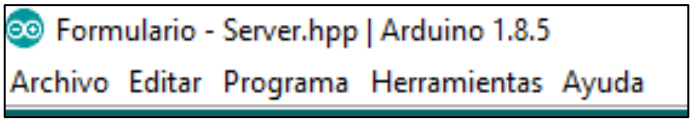


Figura 2.4: Menús de Arduino

En el menú de Archivo se encuentran las siguientes opciones: [6]

- Nuevo: Permite crear un nuevo *sketch* en otra ventana con la estructura mínima para realizar un *sketch*.
- Ejemplos: Muestra los ejemplos proporcionados por Arduino o por alguna librería.
- Cerrar: Cierra la ventana de Arduino en la que se está trabajando.
- Guardar como: Permite guardar el *sketch* con un nombre o dirección diferente.
- Preferencias: Abre la ventana de preferencias.
- Salir: Cierra todas las ventanas abiertas de Arduino IDE.

En el menú de Programa se encuentran muchas opciones que se las puede encontrar en la ventana principal de Arduino IDE, como se observa en la figura 2.2, y a continuación se las describe: [6]

- Subir usando programador: Sobrescribirá el cargador de arranque en la placa utilizando toda la capacidad de la memoria *flash*.
- Mostrar carpeta del programa: Abre la ubicación donde esté guardado el *sketch*.
- Incluir Librería: Agrega una librería al *sketch* insertando la instrucción #include.
- Añadir Fichero: Agrega un archivo de origen al *sketch*, que aparecerá en una pestaña.

2.3 Librerías de Arduino

Las librerías son un conjunto de códigos realizados por terceros que facilitan la escritura y comprensión de un *sketch*, igualmente se puede encontrar librerías para el manejo de diferentes dispositivos como sensores, pantallas, módulos. Para utilizar una librería en Arduino se la debe instalar previamente en Arduino IDE, para ello en la figura 2.5 se puede observar la opción de Gestor de librerías, para ello se debe ingresar a los siguientes menús Programa > Incluir Librería > Gestionar Librerías. [7]

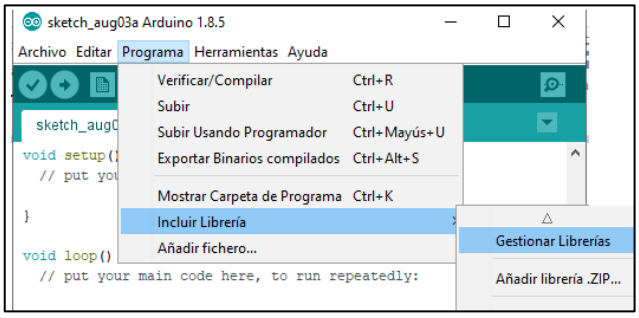


Figura 2.5: Arduino IDE: Incluir librerías

Se abrirá la ventana del Gestor de Librerías como se observa en la figura 2.6, donde se puede filtrar las librerías por tipo y tema, también en la parte superior derecha se encuentra una barra de búsqueda que permite filtrar diferentes librerías en base a una o más palabras clave.

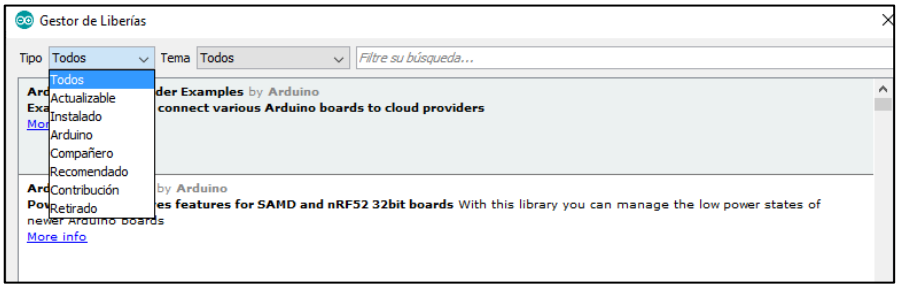


Figura 2.6: Gestor de librerías de Arduino IDE

2.4 *Hardware* de Arduino

Una placa electrónica de Arduino cuenta con un microcontrolador reprogramable y una serie de pines, que pueden ser configurados como entradas o salidas lógicas u analógicas, según sea necesario. [8] Existen diferentes modelos de placas de Arduino como se observa en la figura 2.7.

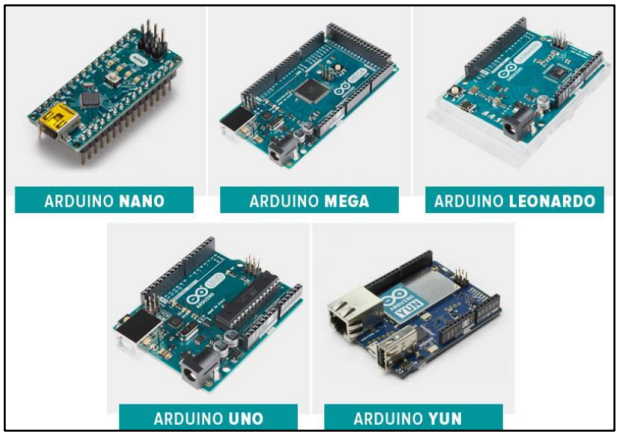


Figura 2.7: Variantes de modelos de las placas Arduino. [8]

Los módulos entrenadores utilizan la placa Arduino Mega 2560, cuyas especificaciones se visualizan en la tabla 2.1. [9]

Tabla 2.1: Especificaciones placa Arduino Mega 2560. [9]

Microcontrolador	Atmega2560
Voltaje operativo	5 V
Voltaje de entrada	7-12 V
Voltaje de entrada(límites)	6-20 V
Pines digitales de I/O	54
Pines analógicos de entrada	16
Corriente DC por cada PIN I/O	40 mA
Memoria *Flash*	256 KB (8 KB usados por el *bootloader*)
SRAM	8 KB
EEPROM	4 KB
Velocidad del reloj	16 MHz

2.5 *Hardware* Nextion

Se puede observar en la figura 2.8 un ejemplo de la estructura interna y externa de la pantalla Nextion. Las pantallas Nextion se comunican con un microcontrolador a través de TTL serial, cuentan con un procesador integrado y una pantalla táctil con memoria que junto con el *software Nextion Editor* permite crear, personalizar una interfaz de usuario táctil, programar de eventos. [10]

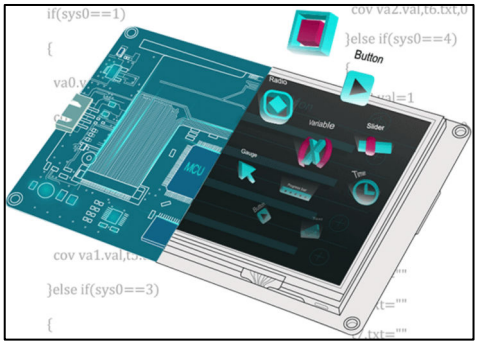


Figura 2.8: Pantalla Nextion. [10]

2.6 *Software* Nextion

El *software Nextion Editor* permite crear una interfaz de usuario táctil, cuenta con componentes fáciles de manejar, la programación de eventos táctiles y una interfaz gráfica de usuario personalizada. El uso del *software* permite crear de manera rápida y sencilla una interfaz gráfica con componentes de tipo gráficos, textos, botones, etc. Lo anteriormente descrito se observa en la figura 2.9. [10]

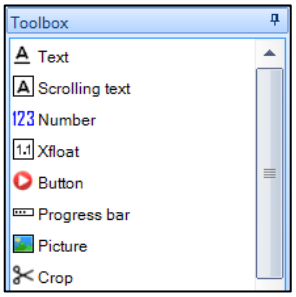


Figura 2.9: Componentes de *Nextion Editor*

Nextion Editor a través de un sistema de *drag-and-drop* permite colocar los componentes con mayor facilidad, cada componente cuenta con una tabla de atributos de los cuales la ID del competente y el nombre de este son los necesarios para lograr una interacción entre una pantalla y un microcontrolador externo. En la figura 2.10 se

observa la tabla de atributos de un componente tipo botón en el cual se puede editar el tipo de fuente del texto del botón, posición del botón en la pantalla, posición de texto dentro del botón, entre otras características. [10]

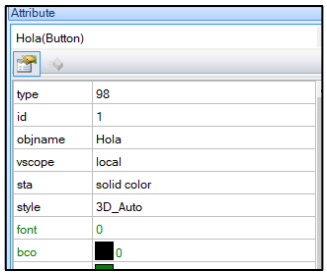


Figura 2.10: Tabla de atributos de botón

Los componentes de tipo botón pueden generar dos tipos de eventos, uno al pulsar el botón y el otro al soltarlo, como ejemplo en la figura 2.11 se configuró al botón “hola” con un evento de pulso que editará el texto del componente “t0” cambiándolo por “holamundo”. Siendo este un ejemplo sencillo del manejo de una pantalla Nextion, para un mejor uso de las pantallas Nextion proporciona un *set* de instrucciones el cual está categorizado e incluye los comandos que pueden ser usados en las pantallas.

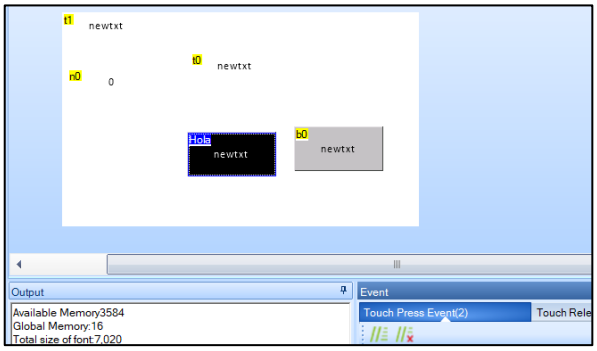


Figura 2.11: Ejemplo de uso del botón en *Nextion Editor*

La interfaz de usuario de los módulos entrenadores se la puede encontrar en el trabajo de titulación *“Diseño de Módulos Entrenadores Lógicos con Conexión Inalámbrica a Dispositivos Móviles para el Laboratorio de Tecnología Eléctrica y Electrónica, Área de Sistemas Digitales”*. En dicho trabajo de titulación se encuentran especificadas las páginas usadas para la interfaz gráfica de usuario de los módulos entrenadores y los botones necesarios para el desplazamiento entre las mismas, las distintas páginas usadas para la interfaz de usuario se las visualiza en la figura 2.12. [11]

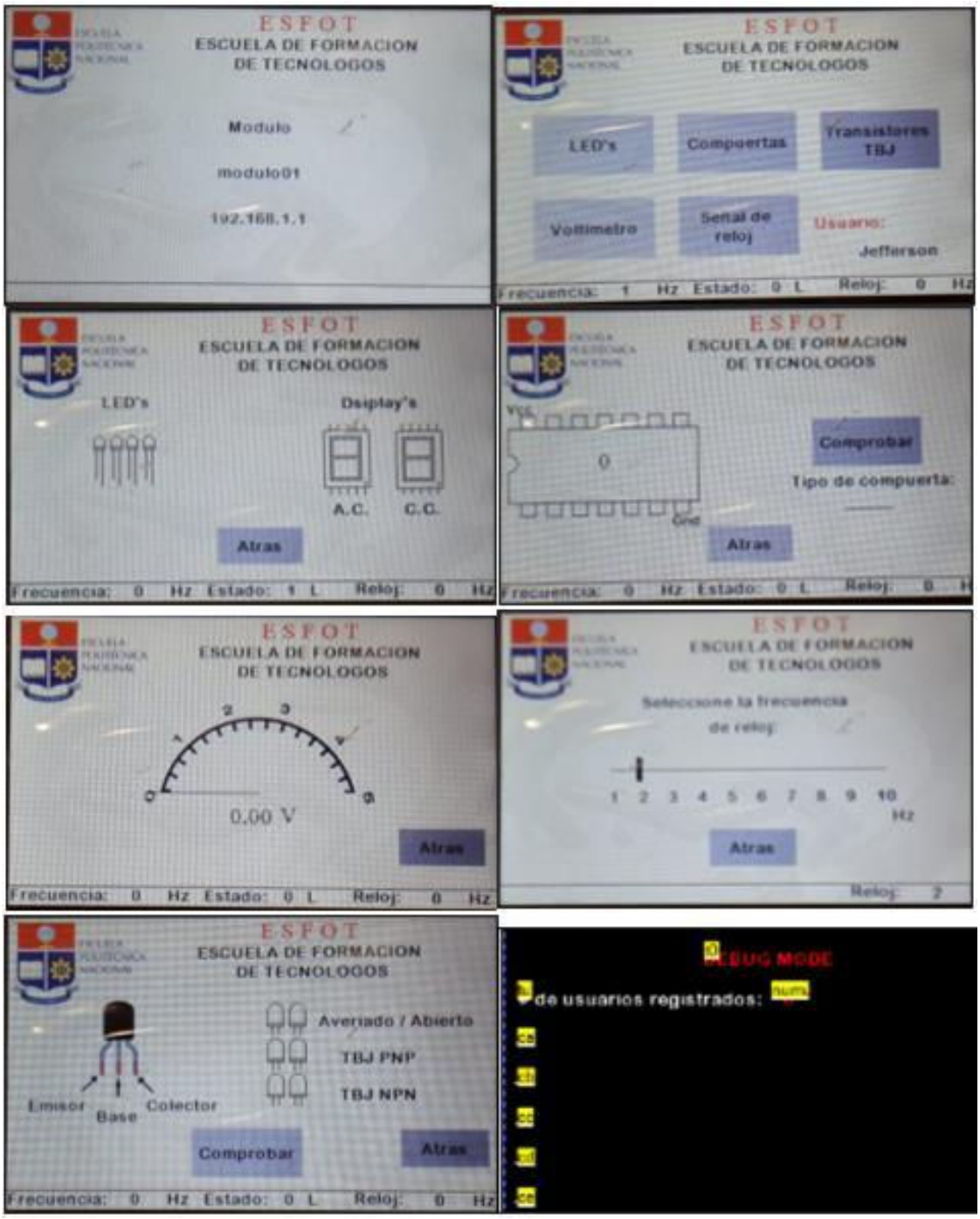


Figura 2.12: Páginas de la interfaz gráfica de los módulos entrenadores [11]

En la figura 2.13 se puede observar las variables que componen estas páginas, en las variables ssid, pass e ip, se mostrarán los datos del nombre de la red, contraseña de red y la dirección IP que son obtenidos del módulo *WiFi*, cuando el estudiante ingrese sus datos en estas variables se remplazarán por el nombre, apellido y número de cédula. Si una de estas variables se encuentra vacías en la pantalla se mostrará el texto “ERROR: Datos vacíos” en la variable “err”.

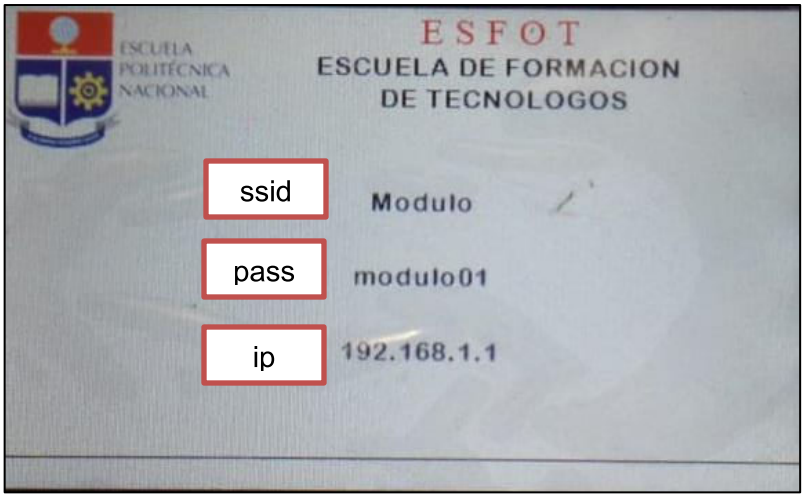


Figura 2.13: Página 0 información de registro.

En la figura 2.14 se muestran los submenús de las diferentes características del módulo, alado del botón "Señal de Reloj" se muestra el nombre del estudiante, Los botones "*btnled*" y *"btnvlt*" activarán las funciones de comprobar LEDs y el voltímetro.

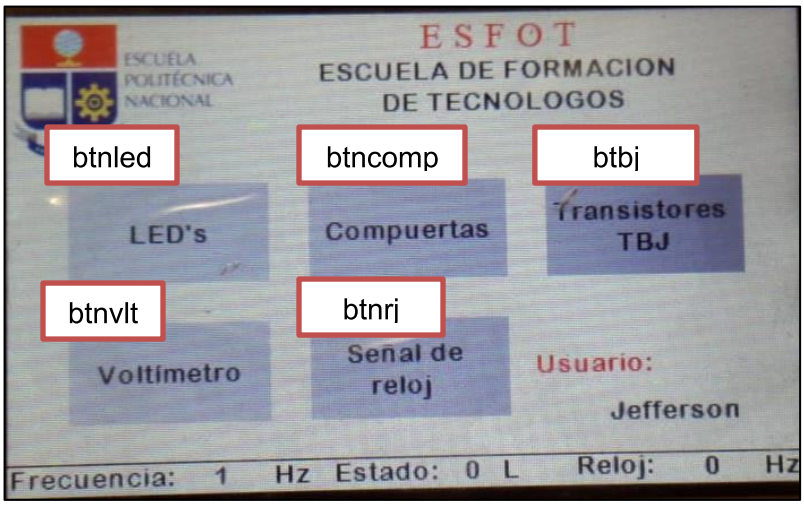


Figura 2.14: Página 1 pantalla principal.

En la figura 2.15 se muestra la polarización que deben tener los LEDs y la posición en la que se deben conectar los *displays* de 7 segmentos, con el botón "*b0*", cuyo nombre en la pantalla es "Atrás", retorna a la página 1.

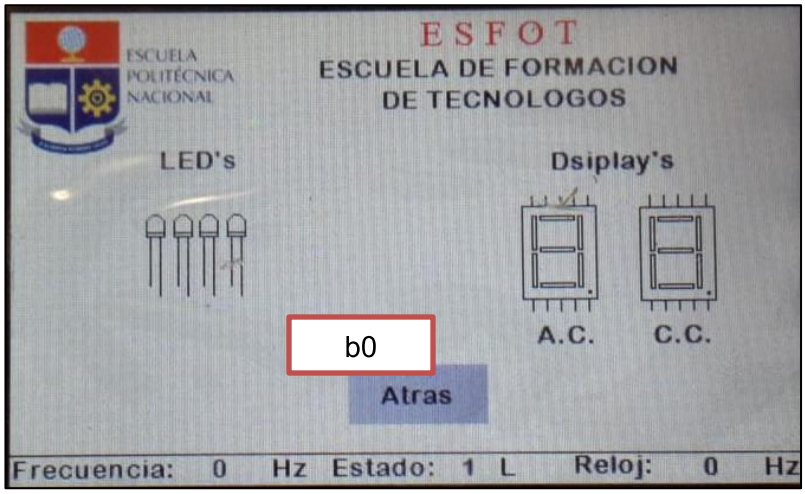


Figura 2.15: Página 2 Comprobador de *display* de 7 segmentos

En la figura 2.16 se muestra la interfaz del comprobador de compuertas, para comprobar una compuerta se pulsa el botón "*btncom"* con el nombre "Comprobar", el número de la compuerta se verá en la variable "*num"* y el nombre de la compuerta se observará en la variable "comp".

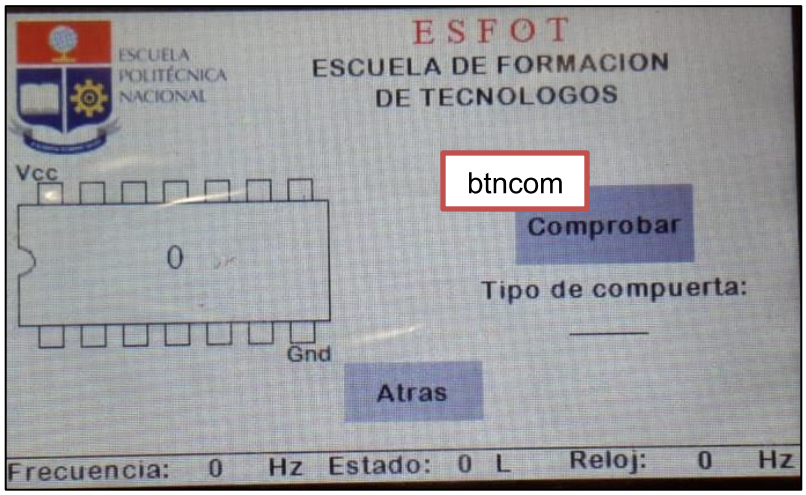


Figura 2.16: Página 3 Comprobador de compuertas.

En la figura 2.17 se muestra la interfaz gráfica del voltímetro donde en la parte inferior se mostrará el valor medido por el voltímetro.

Figura 2.17: Página 4 Interfaz gráfica del voltímetro

En la figura 2.18 se muestra la interfaz gráfica del generador de frecuencia, que con una *slider* permite cambiar el valor del generador de frecuencia.

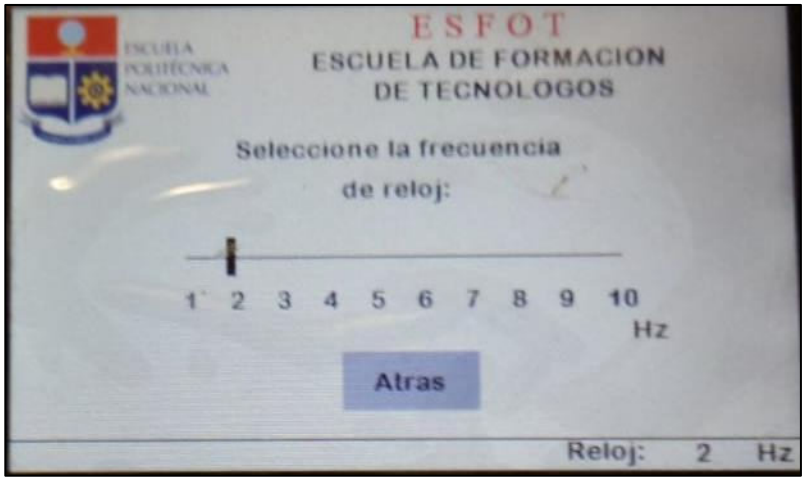


Figura 2.18: Página 5 Interfaz gráfica del generador de frecuencia.

En la figura 2.19 se muestra la interfaz gráfica del comprobador de transistores, para comprobar un transistor se debe pulsar el botón "*btntbj*" con el nombre "Comprobar", que inicia en la pantalla una animación que permitirá identificar el tipo de transistor.

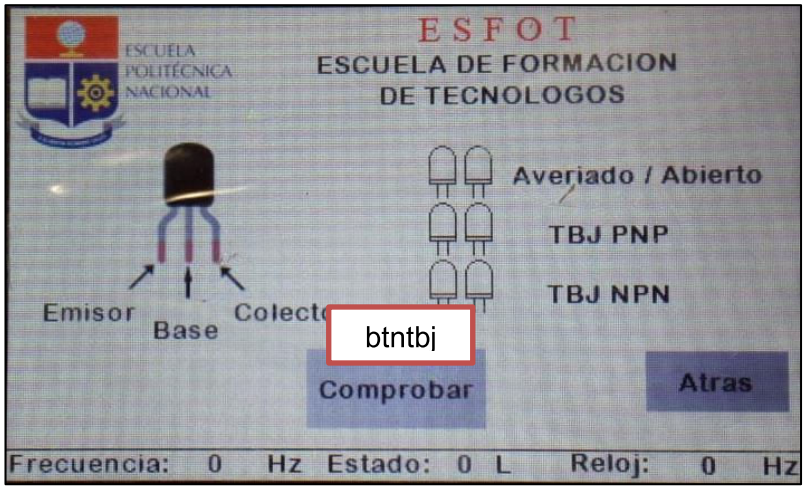


Figura 2.19: Página 6 Interfaz gráfica del Comprobador de transistores.

En la figura 2.20 se muestra la interfaz del modo *debug*, en esta interfaz se mostrarán los últimos 5 usuarios que han ingresado al módulo, en la variable *"num"* se mostrará el número total de usuarios conectados.

Figura 2.20: Página 7 Interfaz modo *debug*

Para usar las pantallas Nextion con un microcontrolador externo este debe utilizar la librería Nextion.h. Los componentes que posean una interacción entre la pantalla Nextion y el Arduino Mega, deberán tener asignados los mismos componentes en cada dispositivo, estos componentes deben tener tanto el nombre como la id del componente iguales en cada dispositivo para poder interactuar entre sí.

2.7 Comunicación inalámbrica WiFi

El estándar definido para redes inalámbricas que trabajan en las bandas de frecuencia de 2.4 GHz y 5 GHz, es el estándar IEEE 802.11 que permite interconectar dispositivos inalámbricos.

Una red inalámbrica es aquella que utiliza como medio de trasmisión el espectro radioeléctrico, usando ondas electromagnéticas de radio o microondas, permitiendo la conexión entre diferentes dispositivos e incluso diferentes tecnologías.

Para la comunicación inalámbrica entre los módulos entrenadores y un dispositivo móvil se usó un módulo esp8266 que trabaja con IEEE 802.11 que maneja los estándares 802.11b, 802.11g y 802.11n cuyas características se las observa en la tabla 2.2.

Tabla 2.2: Características de 802.11 [12]

ESTÁNDAR	VELOCIDAD DE TRANSMISIÓN	BANDAS DE OPERACIÓN
802.11b	11 Mbps	2.4 GHz
802.11g	54 Mbps	2.4 GHz
802.11n	Hasta 600 Mbps	2.4 GHz y 5 GHz

La banda de 2.4 GHz es la banda más saturada, ya que la mayoría de los dispositivos móviles tienen la capacidad de conectarse a esta banda y está sujeta a interferencias.

En la tabla 2.3 se describen las características que posee la banda de 2.4 GHz. En esta banda existen un total de 13 canales para tener una mejor tolerancia a interferencia. Por asuntos de regulación de la asignación del espectro radioeléctrico, en Ecuador en la banda de 2.4 GHz solo están habilitados 11 canales, pero los dispositivos usados en este proyecto trabajan con los 13 canales.

Tabla 2.3: Características básicas de la banda de 2.4 GHz del módulo esp8266

Banda	2.4 GHz
Canales	13 canales
Velocidad de transmisión	50 o 60 Mbps
Estándar	IEEE 802.11b, 802.11g, 802.11n (B, G y N)

Dentro de la banda de 2.4 GHz se divide en 13 canales separados 5 MHz entre sí, con un ancho de 22 MHz, lo que provoca que existan solapamientos que reducen la calidad de la señal, como se observa en la tabla 2.4. [13]

Tabla 2.4: Canales de la banda 2.4 GHz. [13]

Canales	Frecuencia Central
Canal 1	2.412 GHz
Canal 2	2.417 GHz
Canal 3	2.422 GHz
Canal 4	2.427 GHz
Canal 5	2.432 GHz
Canal 6	2.437 GHz
Canal 7	2.442 GHz
Canal 8	2.447 GHz
Canal 9	2.452 GHz
Canal 10	2.457GHz
Canal 11	2.462 GHz
Canal 12	2.467 GHz
Canal 13	2.472 GHz

2.8 Módulo WiFi

El módulo esp8266 sirve como un puente entre un microcontrolador e Internet o un dispositivo móvil, cuyas características se las puede observar en la tabla 2.5. [14]

Tabla 2.5: Características del módulo 8266. [14]

WiFi	**Estándar**	802.11 b/g/n
	Certificación	*Wi-Fi Alliance*
	Rango de frecuencia	2.4 GHz - 2.5 GHz
	Memoria *flash*	1 MB
	Pila de Protocolos	TCP/IP
	CPU	32 bits de baja potencia
Hardware	**Procesador CPU**	Tensilica L106 de 32 bits
	Interfaz Periférica	Control remoto UART / SDIO / SPI / I2C / I2S / IR GPIO / ADC / PWM / LED Luz y botón
	Voltaje de operación	3,3 V
	Temperatura de funcionamiento	-40 °C a 100 °C
Software	**Modo WiFi**	Estación / SoftAP / SoftAP + estación
	Seguridad	WPA / WPA2
	Cifrado	WEP / TKIP / AES
	Actualización de *firmware*	UART Descarga / OTA (a través de la red)
	Protocolos de red	IPv4 / TCP / UDP / HTTP
	Configuración de usuario	*AT Instruction Set, Cloud Server*, aplicación Android / iOS

La distribución de pines del módulo esp8266 se observa en la figura 2.21 seguido por una descripción de los pines.

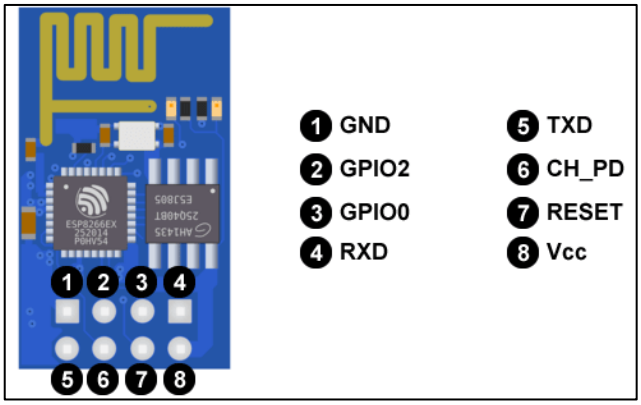


Figura 2.21: Distribución de pines del módulo esp8266. [15]

Descripción de pines

- GND – tierra.
- TX – transmisión serial.
- RX – recepción serial.
- VCC – 3.3 VDC.
- CH_PD – El modo de arranque - debe ser 1 para habilitar WiFi.
- RESET (RST) – Tierra para restablecer.
- GPIO0 – Entrada/salida de propósito general.
- GPIO2 – Entrada/salida de propósito general.

3 RESULTADOS Y DISCUSIÓN

3.1 Programación de módulos entrenadores lógicos

Para identificar los múltiples procesos que los módulos entrenadores deben ejecutar se realizó un diagrama de flujo, en la figura 3.1 se puede observar parte del diagrama de flujo que se encuentra en el **Anexo B,** donde se observan los procesos que se ejecutan al iniciar un módulo entrenador.

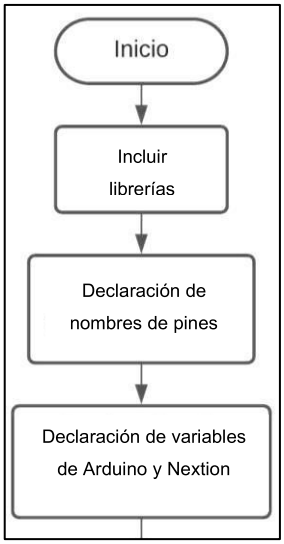


Figura 3.1: Diagrama de flujo primera sección

Antes de realizar un código se debe especificar el modelo de la placa que se usará, para ello en la figura 3.2 se observa cómo se debe seleccionar el modelo de placa Arduino con la que se quiere trabajar, para ello se debe dirigir a la pestaña de Herramientas > Placa donde se desplegará una lista con los diferentes modelos de placas de Arduino que existen, en la cual se selecciona la placa Arduino Mega 2560.

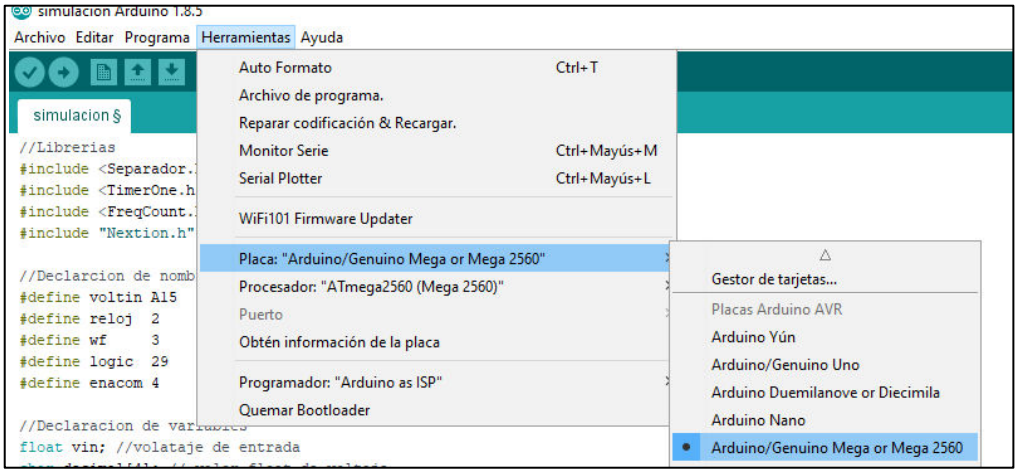


Figura 3.2: Selección de placa Arduino

Para facilitar la comprensión del código se usaron 4 librerías, la librería de *Separador.h* que permite separar los datos recibidos por el puerto serial. Los datos recibidos son guardados en una variable de tipo *string* llamada “datos”, para separar en elementos el *string* de datos se debe especificar un caracter que marque una separación entre elementos, este caracter debe estar entre comillas simples (‘’) como se observa en el ejemplo de la figura 3.3.

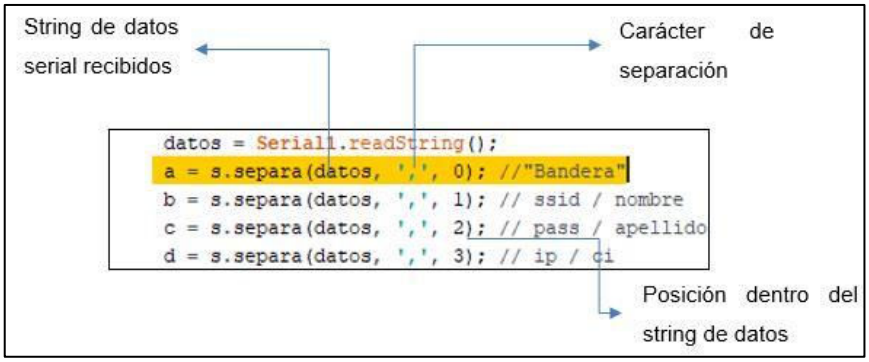


Figura 3.3: Ejemplo de separación de datos

La librería *Timerone.h* que facilita el uso del *timer 1* del microcontrolador ATmega2560, permitiendo trabajar de una manera más sencilla, esta librería se usó para generar una señal de reloj, a continuación, en la tabla 3.1 se puede observar los comandos para usar la librería del *timer1*. [16]

Tabla 3.1: Lista de comandos de la librería *Timerone.h.* [16]

COMANDO	DESCRIPCIÓN
Timer1.initialize	Permite iniciar el *TimerOne*.
Timer1.serPeriod	Asigna un periodo de trabajo para el *TimerOne* entre 1 y 8388480 microsegundos.
Timer1.pwm	Permite generar una PWM basado en tres argumentos, primero el pin por donde la señal será leída, segundo el ciclo de la PWM que debe ir entre 0 a 1023 y tercer argumento opcional que es el periodo de trabajo.
Timer1.attachInterrupt	Permite llamar al controlador de interrupciones.
Timer1.setPwmDuty	Permite establecer una nueva PWM sin tener que configurar de nuevo el pin de salida.
Timer1.detachInterrupt	Permite desactivar las interrupciones en cualquier momento.
Timer1.disablePwm	Permite desactivar la PWM en el pin de salida que se había definido previamente.
Timer1.read	Permite leer el tiempo trascurrido en microsegundos desde el último *rollover.*

La librería *Nextion.h* permite interactuar con los eventos que genera la pantalla táctil, para ello se debe crear dos elementos iguales en la pantalla Nextion y en el Arduino

mega, estos elementos se relacionan por la ID de la página y la ID del componente. No todos los componentes que posea la pantalla Nextion deben tener su homólogo en el Arduino mega, solo aquellos componentes que provoquen una interacción entre la pantalla y el Arduino serán los que deben ser definidos en los dos dispositivos.

La librería *Nextion.h* incluye nuevas variables que deben ser declaradas en Arduino, estas variables se encuentran en la tabla 3.2.

Tabla 3.2: Comandos para la declaración de variables de la librería *Nextion.h* [17]

COMANDO	ESTRUCTURA	DESCRIPCIÓN
NexDSButton	NexDSButton Nombre en Arduino = NexButton (#página, ID botón, "nombre en pantalla")	Crear una variable que se comporta como un botón y lo relaciona con un botón declarado en el *software* de la pantalla Nextion.
NexPage	NexPage Nombre de la página en Arduino = NexPage (#página, Id de página, "Nombre de la página en la pantalla")	Crea una variable y la relaciona con una página declarada en la pantalla Nextion.
NexNumber	NexNumber Nombre en Arduino = NexNumber (#página, ID del Número, "Nombre en la Pantalla")	Crea una variable en formato numérico que puede leer el valor de un potenciómetro o enviar el valor y lo relaciona con su variable en la pantalla.
NexText	NexText Nombre en Arduino = NexText (# de página, Id del Texto, "Nombre del texto en la pantalla")	Crea una variable para texto en Arduino y lo relaciona con su variable texto en la pantalla.

Los botones pueden generar dos tipos de eventos, el evento *push* o presionar que se produce al presionar el botón y el evento *pop* que se produce al liberar el botón, estos eventos se detectan mediante línea de código, cada evento puede ser detectado ejecutar una acción como se pueden observar en la figura 3.4. [18]

```
#include "Nextion.h"

//Standard Button
NexButton b0 = NexButton(3 , 1 , "b0");
NexButton b1 = NexButton(3 , 2 , "b1");

NexTouch *components[] = {
  &b0,
  &b1,
  NULL
};

void setup() {
  nexInit();

  dbSerial.println("Debug OK");
  b0.attachPush( pushCallback, &b0 );
  b1.attachPop( popCallback, &b1 );

}

void loop() {

  nexLoop( components );
}

void pushCallback(void *ptr)
{
  dbSerial.println("Push ");
}

void popCallback(void *ptr)
{
  dbSerial.println("Pop ");
}
```

Figura 3.4: Ejemplo de evento *push* y *pop*

En la figura 3.5 se define un nombre para un pin del Arduino, donde se usa el comando *#define* seguido del nombre que se le quiere asignar al pin y el número del pin del Arduino al cual se le relaciona con el pin.

```
//Declaración de nombres de pines
#define voltin A15
#define reloj  2
#define wf     3
#define logic  29
#define enacom 4
```

Figura 3.5: Declaración de nombres de pines

En la figura 3.6 se observa las variables de la librería *Nextion.h* necesarias para la manipulación de la pantalla táctil.

```
//Declaracion de variables pantalla Nextion
NexPage     pagel    = NexPage    (1, 0, "pagel");      //Pagina 1
NexDSButton btnled   = NexDSButton(1, 14, "btnled");    // Btn LED's
NexDSButton btncom   = NexDSButton(3, 16, "btncom");    // Btn compuertas
NexDSButton btnvlt   = NexDSButton(1, 13, "btnvlt");    // Btn voltimetro
NexNumber   rj       = NexNumber  (1, 10, "n0");        // Numero del reloj
NexNumber   frec     = NexNumber  (1, 12, "n2");        // Numero de la frecuencia
NexNumber   logc     = NexNumber  (1, 11, "n1");        // Estado logico
NexNumber   num      = NexNumber  (3, 17, "num");       // Numero compuerta
NexText     tvlt     = NexText    (4, 13, "t2");        // Texto voltaje
NexText     tcom     = NexText    (3, 5,  "comp");      // Texto compuertas
NexText     tb       = NexText    (0, 1,  "ssid");      // Texto ssid / nombre
NexText     tc       = NexText    (0, 4,  "pass");      // Texto pass / apellido
NexText     td       = NexText    (0, 5,  "ip");        // Texto ip  / ci
NexText     err      = NexText    (0, 6,  "err");       // Texto error de datos vacios
NexText     tnomb    = NexText    (1, 15, "nomb");      // Texto nombre

NexDSButton mas      = NexDSButton(5, 11,  "mas");       // Frecuncia mas
NexDSButton men      = NexDSButton(5, 12,  "men");       // Frecuncia maenos
```

Figura 3.6: Declaración de variables de la pantalla táctil

Las variables definidas anteriormente en la figura 3.6 generan un cambio de estado ya sea en el Arduino o en la pantalla, cada variable genera un cambio según el tipo de variable, las variables de tipo botón provocan un evento, en caso de las variables de tipo numérico y de texto estas cambiarán el texto o el valor numérico según se necesite.

En la figura 3.7 se observa las variables de Arduino que se usaron en el código de los módulos entrenadores.

```
//Declaracion de variables
float vin; //volataje de entrada
char decimal[4]; // valor float de voltaje
char chb[10]; // ssid / nombre
char chc[10]; // pass / apellido
char chd[14]; // ip / ci
String datos, a, b, c, d; //trama  | "Bandera" | ssid / nombre  |  pass / apellido | ip / ci
Separador s; //separador
unsigned long count = 0, us = 1000000, Fr = 1; // frecuencimetro | frecuencia en us | frecuencia reloj
uint32_t estadovlt, estadoled, estadocom, stmas, stmen; // estados botones pantalla
bool l; // estado logico
int out1, out2, out3, out4, out5, out6, out7, out8; // valores compuertas
byte y[3][2] = {
  {0, 0}, //out1, out4, out6
  {0, 1}, //out2, out5, out7
  {1, 1}, //out3,       out8
};
```

Figura 3.7: Declaración de variables en Arduino

Cada tipo de variable en Arduino tiene características propias, estas características se pueden observar en la tabla 3.3 donde se describen los diferentes tipos de variables que fueron necesarias para programar los módulos entrenadores. [19]

Tabla 3.3: Tipos de variables de Arduino. [19]

TIPO DE VARIABLE	DESCRIPCIÓN
float	Formato de dato de tipo coma flotante se aplica a números decimales, tienen una resolución de 32 bits con un rango de 3.4028235E+38 a -3.4028235E+38.
char	Formato de dato de 1 byte de memoria usando codificación ASCII, para caracteres simples se usa comillas simples y para múltiples caracteres se usa comillas dobles.
string	Formato de datos que permite guardar una cadena de caracteres.
unsigned long	Formato de datos extendido que almacena números positivos de hasta 4 bytes.
uint32_t	Formato de datos numéricos de 4 bytes.
bool	Formato de datos que contiene 2 valores.
int	Formatos de datos enteros de 2 bytes.
byte	Formato de datos que representa a una variable de 8 bits.

En la figura 3.8 se observa los eventos táctiles que la pantalla posee, cada botón está ligado a una función que tiene el módulo entrenador, donde se encuentran las funciones tales como un comprobador de LEDs y *displays*, compuertas, transistores TBJ, voltímetro y señal de reloj.

Figura 3.8: Eventos táctiles

En el diagrama de flujo de la figura 3.9 se muestran los procesos que siguen los módulos para funcionar correctamente.

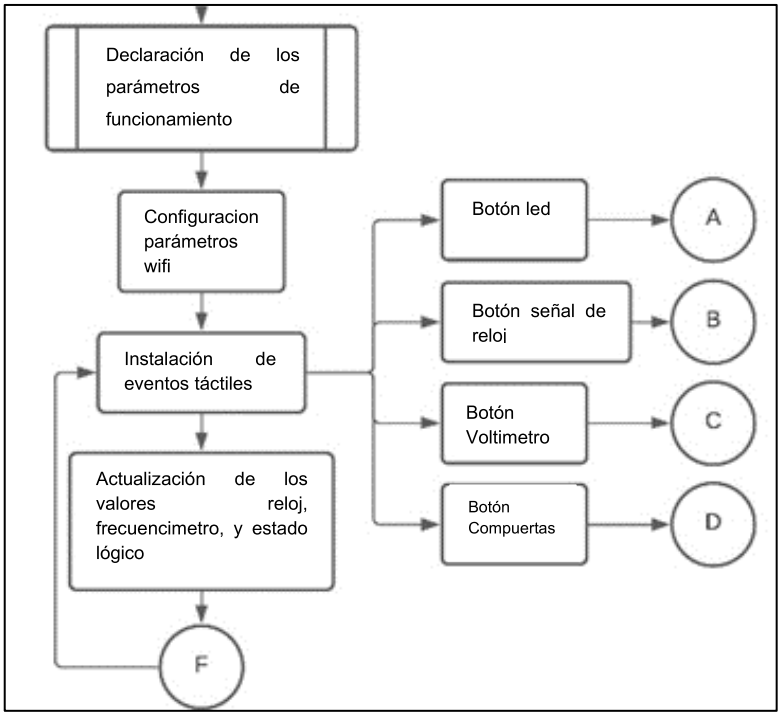


Figura 3.9: Diagrama de flujo segunda sección

Las variables para que los módulos se inicialicen de manera correcta, se muestran en la figura 3.10 indicando que se inicie la pantalla Nextion y mostrando la página 1 de la pantalla, para los puertos de Arduino se declaran las entradas y salidas de los puertos A, C, F y K, se configura el modo de trabajo de los pines voltin, reloj, wf, logic y enacom. En la tabla 3.4 se observa una descripción de las funciones que llevan a cabo los pines.

Tabla 3.4: Descripción de funciones de los pines. [11]

Numero de Pin	Nombre	I/O	Función
A15	voltin	*INPUT*	Permite la lectura de voltaje en el pin.
2	reloj	*OUTPUT*	Permite la salida de la señal de reloj.
3	wf	*OUTPUT*	Pin que habilita o deshabilita el módulo esp8266.
29	logic	*INPUT*	Permite la lectura de estados lógicos.
4	enacom	*OUTPUT*	Pin que habilita o deshabilita el comprobador de compuertas.

Para realizar la trasmisión de datos se declara la velocidad de comunicación de los pines RX y TX en 9600 baudios para la comunicación WiFi. Mediante el comando *Serial.begin*(9600), esta velocidad debe ser la misma tanto en el Arduino como en el módulo WiFi para que la conexión sea exitosa. Mediante el comando *FreqCount.begin*(1000) se establece que al inicio del frecuencímetro se configura el *timer1* para trabajar en microsegundos y se activa la interrupción de este, como se observa en la figura 3.10.

```
//Iinicializacion de pantalla y botones
nexInit();
page1.show(); //DEBUG

//Inicializacion y declaracion de puertos
DDRA  =  B01111111;
//PORTA =  PORTA | B01111111; // INICIA LEDD Apagados
DDRC  =  B11111111;

//DECLARAR PINES DE COMPUERTAS COMO ENTRADAS DIGITALES PARA EVITAR REINICIOS /////////
DDRF = B00000000;
DDRK = B0000;

pinMode(voltin, INPUT);
pinMode(reloj, OUTPUT);
pinMode(wf,    OUTPUT);
pinMode(logic, INPUT);
pinMode(enacom, OUTPUT);

//Inicializacion de comunicacion wifi
Serial1.begin(9600);

//Inicializacion del frecuencimetro
FreqCount.begin(1000);

//Inicializacion de TimerOne
Timer1.initialize(1000000); // (unidad a trabajar en u.sec)
Timer1.attachInterrupt(subida);  //(funcion de ISR)
```

Figura 3.10: Declaración de las variables de funcionamiento

En la figura 3.11 para lograr la comunicación inalámbrica con el módulo WiFi esp8266, que permite la recepción de datos a través de una red inalámbrica 802.11, primero se habilita el módulo WiFi enviando 1 lógico al “wf” o pin 3. Mientras el puerto serial1 se encuentre habilitado, este recibirá los datos del módulo WiFi guardándoles en la variable datos, los primeros datos recibidos son el nombre de la red, contraseña y dirección IP del módulo, estos datos son guardados en un solo *string* de datos por lo que es necesario separarlos usando la librería S*eparador.h.*

```
void coneccionwf() {

  //delay(100);
  digitalWrite(wf, HIGH);
  if (Serial1.available() > 0) {
    while (Serial1.available() > 0) {
      datos = Serial1.readString();
      a = s.separa(datos, ',', 0); //"Bandera"
      b = s.separa(datos, ',', 1); // ssid / nombre
      c = s.separa(datos, ',', 2); // pass / apellido
      d = s.separa(datos, ',', 3); // ip / ci
```

Figura 3.11: Separación de datos

Se leen los datos recibidos en el puerto serial y se guardan en la variable llamada datos, luego se separan los datos recibidos en 4 secciones, la sección “a” guarda los datos de bandera, la sección “b” guarda los datos de ssid/nombre, la sección “c” guarda los datos de *password*/apellido y la sección “d” guarda los datos *ip*/C. I, luego que los datos son separados se elimina el contenido de la variable datos, como se muestra en la figura 3.11.

Para poder enviar los datos a la pantalla Nextion, se usa el comando *toCharArray (),* que copia los datos de una variable del tipo *string* a una variable de tipo char se debe especificar el tamaño de la variable, como se observa en a la figura 3.12.

```
b.toCharArray(chb, 10);
c.toCharArray(chc, 10);
d.toCharArray(chd, 14);
```

Figura 3.12: Uso del comando *toCharArray ()*

Para enviar los datos a la pantalla Nextion se usa el comando *setText()*. El primer valor corresponde a la variable de texto en Nextion, y el texto que se enviará deberá estar dentro de paréntesis, como se observa en la figura 3.13

```
tb.setText(chb);
tc.setText(chc);
td.setText(chd);
```

Figura 3.13: Cambio de datos de tipo texto en la pantalla Nextion.

En la tabla 3.5 se puede observar las variables por las que pasan los datos de ssid/nombre, password/apellido e ip/C.I, para que sean proyectados en la pantalla Nextion.

Tabla 3.5: Variables necesarias asociadas a la pantalla Nextion. [11]

Variable *String*	Variable char	Variable Nextion	Longitud	Función
b	chb	tb	10	Guarda en texto el nombre del estudiante y el nombre del módulo.
c	chc	tc	10	Guarda en texto el apellido del estudiante y la contraseña del módulo.
d	chd	td	14	Guarda en texto el número de cédula del estudiante y la contraseña de ingreso al módulo.

Con una sentencia *if* si la variable a de tipo *string*, que es la bandera para diferenciar los datos del esp8266es, es igual a “datos”, este procederá a comprobar si los datos ingresados están correctos.

Para verificar los datos se usa una sentencia *if,* si los datos de los *strings* b, c y d se encuentran vacíos, se usará la variable “err” para enviar a la pantalla el texto “ERROR Datos vacíos”.

Si los datos entregados no están vacíos, no se envía ningún texto a la variable “err”, el módulo entrenador iniciará de forma normal imprimiendo en pantalla el mensaje de “Módulo Configurado Bienvenido” por 2 segundos, luego indica a la pantalla Nextion que muestre la página 1, y en el componente de texto “*tnomb”* de la pantalla de Nextion escribirá el nombre del estudiante que está guardado en la variable “chb”, esto se observa en la figura 3.14.

```
if ( a == "datos") {
  if(b =="" || c =="" || d ==""){
    err.setText("ERROR Datos vacios");
    return;
  }
  else
  err.setText("");
  delay(3000);
  tb.setText("Modulo");
  tc.setText("Configurado");
  td.setText("Bienvenido");
  delay(2000);
  page1.show();
  tnomb.setText(chb);
  Serial1.end();
  digitalWrite(wf, LOW);
}
```

Figura 3.14: Sentencia if para verificación de datos

En la figura 3.15 se puede observar la pantalla del módulo que indica el nombre de la red, clave de acceso y la dirección IP de la página de registro, estos parámetros de acceso a la red son configurados en el módulo WiFi esp8266, mediante comunicación serial los datos de nombre, apellido y cédula de identidad son enviados hacia el Arduino mega y son almacenados.

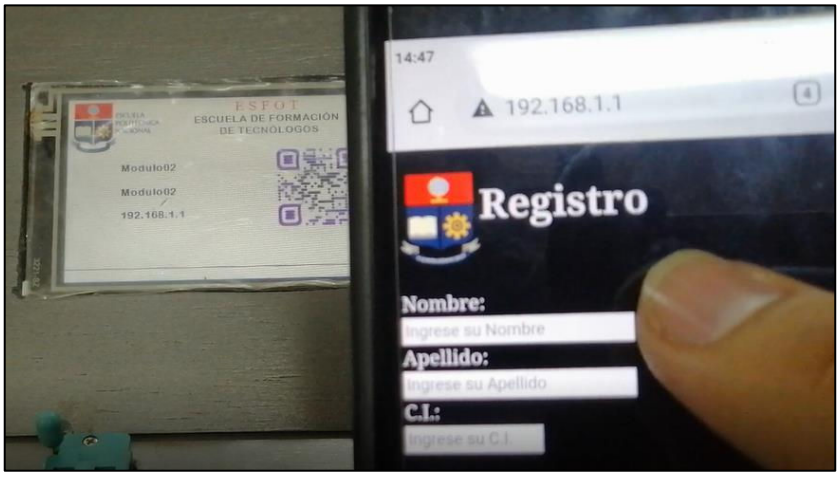


Figura 3.15: Ejemplo de acceso a módulo entrenador.

En la figura 3.16 se muestra una lista de los eventos táctiles, sin embargo, para que el Arduino logre identificar dichos eventos se debe usar "*nexLoop(nex_listen_list)*", cada evento táctil debe tener una variable que se relacione en Arduino y en la pantalla Nextion.

```
//lista de eventos tactiles
NexTouch *nex_listen_list[] =
{
  &btncom,
  &btnled,
  &btnvlt,
  &mas,
  &men,
  NULL
```

Figura 3.16: Eventos táctiles

Para la correcta interacción de los botones de la pantalla Nextion con el Arduino, estos deben tener bien identificada la ID de la página donde se encuentra el botón, ID del botón y el nombre en Nextion. En la tabla 3.6 se puede observar la ID de página y de los elementos que son necesarios para que el Arduino reconozca si es un botón, un cuadro de texto o numérico cuyo valor se pueda modificar de la pantalla Nextion.

Tabla 3.6: Características de los botones en Nextion. [11]

Tipo	Nombre en Arduino	Nombre en Nextion	ID de página	ID de elemento
Botón	btnled	btnled	1	14
	btocom	btocom	3	16
	btovlt	btovlt	1	13
	mas	mas	5	11
	men	men	5	12
Número	rj	n0	1	10
	frec	n2	1	12
	logc	n1	1	11
	num	num	3	17
Texto	tvlt	t2	4	13
	tcom	comp	3	5
	tb	ssip	0	1
	tc	pass	0	4
	td	ip	0	5
	err	err	0	6
	tnomb	nomb	1	15

En la figura 3.17 se observa que mediante líneas de código se forma un bucle, lo que permite una actualización continua de los valores del voltímetro, frecuencímetro, reloj y el estado lógico.

```
btnvlt.getValue(&estadovlt);

if (estadovlt == 1) {
  //Voltimetro
  voltimetro();

  //Frecuencimetro
  frecuencimetro();

  //Reloj
  rj.setValue(Fr);

  //Estado Logico
  estadoL();
}
else {
  //Frecuencimetro
  frecuencimetro();

  //Reloj
  setfr();

  //Estado Logico
  estadoL();

  //LED's
  leds();

  //Compuertas
  comp();
}

delay(25);
```

Figura 3.17: Actualización de los valores de voltímetro, frecuencímetro, reloj y estado lógico

Para realizar el frecuencímetro se ocupó la librería *FreqCount.h*, esta librería permite realizar una medición de frecuencia en un pin del Arduino, se puede observar en la figura 3.18 cómo se realiza la medición de la frecuencia, para ello se tiene que iniciar la librería con el comando *FreqCount.begin()* iniciando con un valor de 1000. Cuando exista un nuevo valor de medida disponible se usa el comando *if(FreqCount.available())* que devuelve un verdadero, el comando *count* = *FreqCount()* realiza una lectura de un pin, que por defecto es el pin 5, y lo guarda en la variable *count*. [20]

```
void frecuencimetro() { //Frecuencimetro
  if (FreqCount.available()) {
    count = FreqCount.read();
    frec.setValue(count);
  }
```

Figura 3.18: Ejemplo del uso de la Librería *FreqCount.h*

En la figura 3.19 se observa que la variable *bool* "l" lee el valor lógico del pin 29 cuyo nombre es "l*ogic*". Con el comando *setValue* establece el valor de la variable "*logic*" de la pantalla Nextion para que sea igual a la variable *bool* "l".

```
void estadoL() { //Estado Logico
  l = digitalRead(logic);
  logc.setValue(l);
}
```

Figura 3.19: Obtención estados lógicos.

En la figura 3.20 se observa que la variable “vin” será igual al valor analógico obtenido en el pin A15 multiplicado por cinco y dividido para 1023. Luego con el comando *dtostrf ()*, permite la conversión de un número a una cadena, la estructura *dtostrf* es la siguiente: (número a convertir, tamaño del número en caracteres, número de decimales, lugar donde se almacena). Con el comando *set.text* es envía a la variable “tvlt” de Nextion el texto que se encuentra en la variable “decimal” del Arduino.

```
void voltimetro() { //Voltimetro
  vin = ((analogRead(voltin) * 5.0) / 1023);
  dtostrf(vin, 5, 2, decimal);
  tvlt.setText(decimal);
}
```

Figura 3.20: Obtención del voltímetro.

Los datos del estado lógico, frecuencia y señal de reloj siempre serán visibles, estos datos se encontrarán en la parte inferior de las diferentes páginas de la pantalla, como se observa en la figura 3.21.

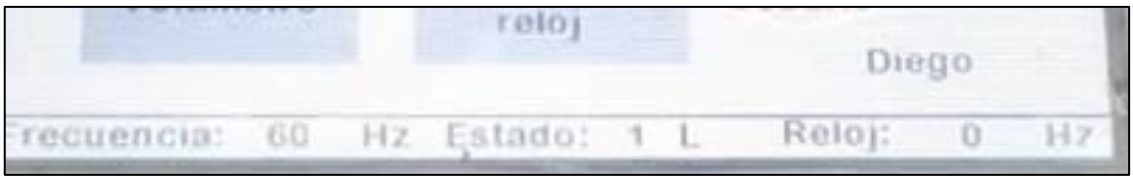


Figura 3.21: Pantalla Nextion valores siempre en pantalla

Al pulsar el botón de LEDs que se observa en la figura 3.18, se ejecuta el código de la figura 3.22, con el uso de la sentencia *If* se logra encender los 8 LEDs, en caso de ser verdadero entra en un lazo *for* que envía 0 lógico del pin 22 al 38 de manera ascendente, si es falso entra en un lazo for envía 1 lógico del pin 29 al 45, los cuales son usados para verificar que no exista algún LED quemado en los *displays* de ánodo y cátodo común.

```
void leds() { //Encendido de puertos A y C (LED's)

  btnled.getValue(&estadoled);

  if (estadoled == 1) {
    for (int i = 0; i < 8; i++) {
      digitalWrite(22 + i, LOW);
      digitalWrite(30 + i, LOW); ///HIGH    //LEDS INVETIDOS
      delay(25);
      estadoled = 0;
    }
  }
  else {
    for (int i = 0; i < 8; i++) {
      digitalWrite(29 - i, HIGH);
      digitalWrite(37 - i, HIGH); /////LOW
      delay(25);
    }
  }
}
```

Figura 3.22: Comprobación de LEDs y *displays*

En la figura 3.23 se observa que en la pantalla Nextion muestra la polaridad de los LEDs para ser conectados al módulo y también indica que el *display* del lado izquierdo es para ánodo común y el derecho para cátodo común.

Figura 3.23: Módulo entrenador, comprobador de LEDs

El módulo entrenador permite reemplazar los *displays* y LEDs cuando se encuentren encendidos, lo cual también se conoce como realizar un cambio en caliente, de igual manera si uno de los LEDs que conforma el *display* se encuentra averiado esto se podrá observar claramente.

Los botones de *stmas* y *stmen* permiten cambiar los parámetros de la señal de reloj aumentándola o disminuyéndola en función de la variable llamada “Fr”, el valor generado es usado para realizar una onda PWM, esto se puede observar en la figura 3.24.

```
void setfr() {

  mas.getValue(&stmas);
  if (stmas == 1) {
    Fr = Fr + 1;
    mas.setValue(0);
  }
  men.getValue(&stmen);
  if (stmen == 1) {
    Fr = Fr - 1;
    men.setValue(0);
    if (Fr == 0) {
      Fr = 1;
    }
  }
  rj.setValue(Fr);
}
```

Figura 3.24: Código para la señal de reloj

En la figura 3.25 se puede observar que mediante una barra *slide,* se modifica el valor de la señal de reloj, para observar los cambios de la frecuencia se usa un multímetro que posea la función de frecuencimetro.

Figura 3.25: Módulo entrenador, señal de reloj

Con la ayuda del *timer1* se crea una señal PWM, con una interrupción se define el ancho de la señal PWM, esto se puede observar en la figura 3.26 la cual contiene el código necesario.

```
void subida() { //Reloj Rise

  digitalWrite(reloj, HIGH);
  Timer1.attachInterrupt(bajada, (us / (2 * Fr))); //Interrupcion (ISR, Ancho de pulso)
}

void bajada() {//Reloj Down
  digitalWrite(reloj, LOW);
  Timer1.attachInterrupt(subida, (us / Fr) - (us / (2 * Fr))); //Interrupcion (ISR, Periodo - Ancho de pulso)
}
```

Figura 3.26: Generar señal PWM

En la figura 3.27 se observa cómo identificar las compuertas lógicas de la familia 74XX, se debe tener en cuenta que una compuerta lógica genera una tabla de estados, estas tablas de estados se las puede observar en el **ANEXO A**. Para lograr identificar las compuertas se usa la sentencia *if* y el operador and "*&&*" como se ve en la figura 3.13, que detecta que una compuerta es *and* si se cumple que *out1* == 0 y *out2* == 0 y *out3* == 4, en este caso imprime en la pantalla el nombre de la compuerta *and* y el número de esta, siendo en este caso 7408.

```
void comp() { //Compuertas

  btncom.getValue(&estadocom);

  if (estadocom == 1) {

    digitalWrite(enacom, LOW);
    uno();
    dos();
    tres();
    DDRF = B00000000;
    DDRK = B0000;
    digitalWrite(enacom, HIGH);
    btncom.setValue(0);

    if (out1 == 0 && out2 == 0 && out3 == 4) {
      tcom.setText("AND");
      num.setValue(7408);
    }
    else if (out1 == 4 && out2 == 4 && out3 == 0) {
      tcom.setText("NAND");
      num.setValue(7400);
    }
    else if (out1 == 0 && out2 == 4 && out3 == 4) {
      tcom.setText("OR");
      num.setValue(7432);
```

Figura 3.27: Detección de compuertas lógicas 74XX

Para las compuertas *and, nand, or* y *xor* en la figura 3.28 se muestra cómo se generan los estados lógicos para las cuatro compuertas que tiene un circuito integrado, gracias a la sentencia *for* que permite generar las tablas de estados por combinación de 00, 01, 10 y 11. Luego se procede a leer el valor entregado por la compuerta.

```
  DDRF = B11011011;
  DDRK = B0110;

  for (int i = 0; i < 2; i++) { /////Combinancion 0 0
    digitalWrite(A0 + i,   y[0][i]);
    digitalWrite(A3 + i,   y[0][i]);
    digitalWrite(A6 + i,   y[0][i]);
    digitalWrite(A9 + i,   y[0][i]);
  }
  out1 = digitalRead(A2) + digitalRead(A5) +  digitalRead(A8) + digitalRead(A11);
  ////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinancion 0 1 = 1 0
    digitalWrite(A0 + i,   y[1][i]);
    digitalWrite(A3 + i,   y[1][i]);
    digitalWrite(A6 + i,   y[1][i]);
    digitalWrite(A9 + i,   y[1][i]);
  }
  out2 = digitalRead(A2) + digitalRead(A5) +  digitalRead(A8) + digitalRead(A11);
  ////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinancion  1 1
    digitalWrite(A0 + i,   y[2][i]);
    digitalWrite(A3 + i,   y[2][i]);
    digitalWrite(A6 + i,   y[2][i]);
    digitalWrite(A9 + i,   y[2][i]);
  }
  out3 = digitalRead(A2) + digitalRead(A5) +  digitalRead(A8) + digitalRead(A11);
  delay(100);
}
```

Figura 3.28: Bucle para *and*, *nand, or* y *xor*

Dentro de un circuito integrado se encuentran 6 compuertas *not*, debido a que estas compuertas invierten el valor lógico que reciben en la entrada, en la figura 3.29 usando la sentencia *for* se generan los estados para cada compuerta del circuito integrado, luego se procede a leer el valor entregado por la compuerta.

```
void dos() { ///// Compuertas NOT

  DDRF = B01010101;
  DDRK = B0101;

  for (int i = 0; i < 11; i = i + 2) { /////Combinancion 0
    digitalWrite(A0 + i, LOW);
  }
  out4 = digitalRead(A1) + digitalRead(A3) +  digitalRead(A5) + digitalRead(A7) + digitalRead(A9) + digitalRead(A11);
  ////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 11; i = i + 2) { /////Combinancion 1
    digitalWrite(A0 + i, HIGH);
  }
  out5 = digitalRead(A1) + digitalRead(A3) +  digitalRead(A5) + digitalRead(A7) + digitalRead(A9) + digitalRead(A11);
  delay(100);
}
```

Figura 3.29: Compuerta *NOT*

Para la compuerta *nor* el procedimiento para generar los estados es similar al de las compuertas *nand*, debido a que la distribución interna del circuito integrado de una compuerta *nor* es diferente al de las demás compuertas se utilizó diferentes líneas de código para generar la tabla de estados, como se observa en la figura 3.30.

```
void tres() { ///// Compuertas NOR

  DDRF = B10110110;
  DDRK = B1101;

  for (int i = 0; i < 2; i++) { /////Combinancion 0 0
    digitalWrite(A1 + i,   y[0][i]);
    digitalWrite(A4 + i,   y[0][i]);
    digitalWrite(A7 + i,   y[0][i]);
    digitalWrite(A10 + i,  y[0][i]);
  }
  out6 = digitalRead(A0) + digitalRead(A3) +  digitalRead(A6) + digitalRead(A9);
  ////////////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinancion 0 1 = 1 0
    digitalWrite(A1 + i,   y[1][i]);
    digitalWrite(A4 + i,   y[1][i]);
    digitalWrite(A7 + i,   y[1][i]);
    digitalWrite(A10 + i,  y[1][i]);
  }
  out7 = digitalRead(A0) + digitalRead(A3) +  digitalRead(A6) + digitalRead(A9);
  ////////////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinancion 1 1
    digitalWrite(A1 + i,   y[2][i]);
    digitalWrite(A4 + i,   y[2][i]);
    digitalWrite(A7 + i,   y[2][i]);
    digitalWrite(A10 + i,  y[2][i]);
  }
  out8 = digitalRead(A0) + digitalRead(A3) +  digitalRead(A6) + digitalRead(A9);
  delay(100);
```

Figura 3.30: Compuertas *NOR*

3.2 Configuración de módulo WiFi esp8266 como AP

El módulo esp8266 funciona como si fuera otra placa de Arduino por lo que es necesario programarlo individualmente, para poder realizar la programación del módulo primero se debe instalar el *plugin* del esp8266, este *plugin* permitirá que en Arduino IDE se puede seleccionar al módulo esp8266 como placa programable.

Para ello primero se ingresa a Arduino IDE, se dirige al menú Archivo > Preferencias como se indica en la figura 3.31.

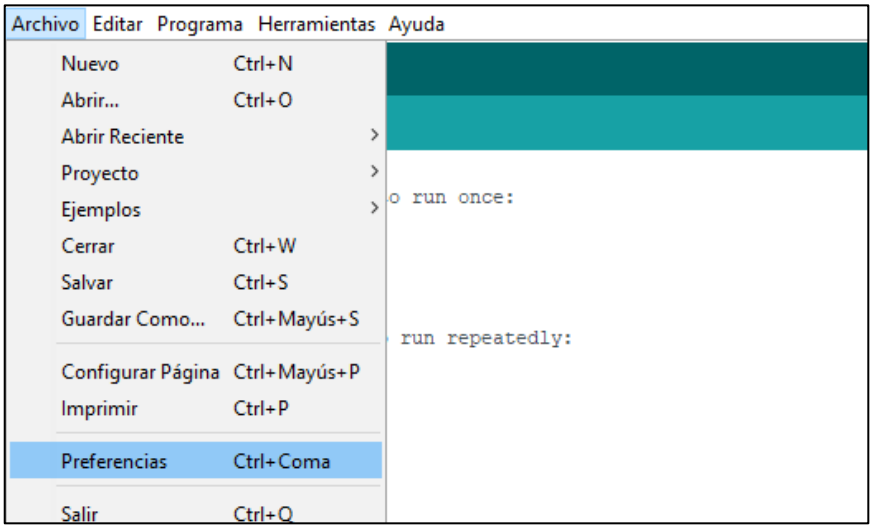


Figura 3.31: Arduino IDE menú Archivo

Se desplegará la ventana del menú de preferencias, donde en la parte inferior de la ventana, al lado del Gestor de URL´s Adicionales de Tarjetas, se encontrará un cuadro de texto donde se pega la siguiente dirección *"http://arduino.esp8266.com/stable/package_esp8266com_index. json*", este paso preconfigura Arduino para poder trabajar con el módulo esp8266 como si fuese otra placa de Arduino, como se muestra en la figura 3.32.

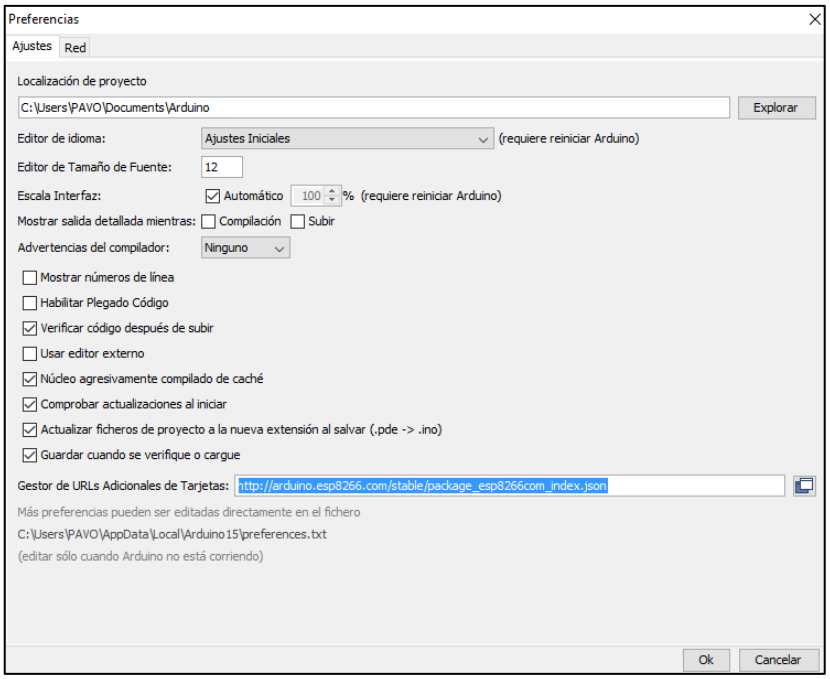


Figura 3.32: Menú de preferencias de Arduino IDE

Posteriormente se ingresa en la pestaña de Herramientas > Placas > Gestor de Tarjetas como se observa en la figura 3.33.

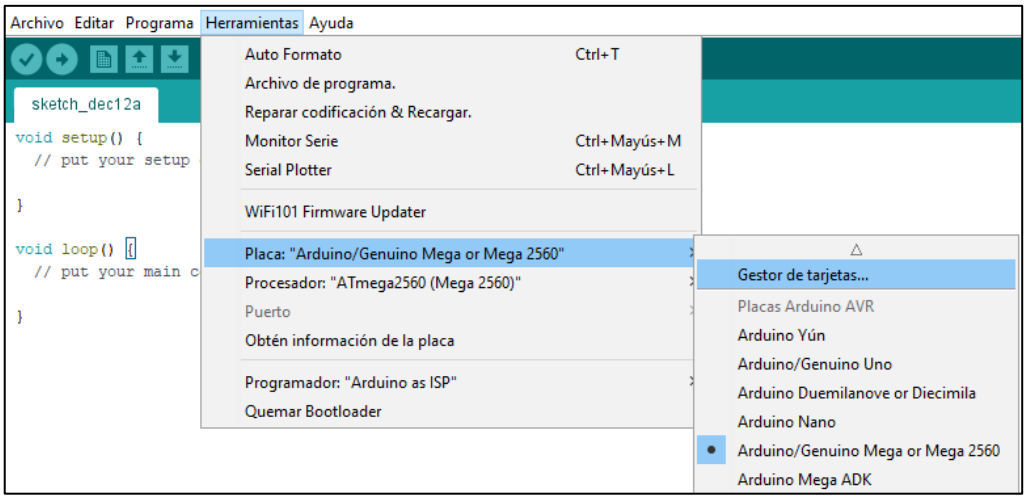


Figura 3.33: Arduino IDE menú Herramientas

Una vez abierta la ventana del gestor de tarjetas este procederá a realizar una actualización, una vez terminada la actualización en la barra de búsqueda se procede a digitar el nombre del módulo que se requiere, siendo en este caso el módulo esp8266. Se procede a seleccionar el paquete de instalación con el nombre de "*ESP8266 Community"* y la versión del *plugin* es la 2.3.0. En la figura 3.34 se observa el paquete de instalación del *plugin* del módulo esp8266 y se procede a instalar. [21]

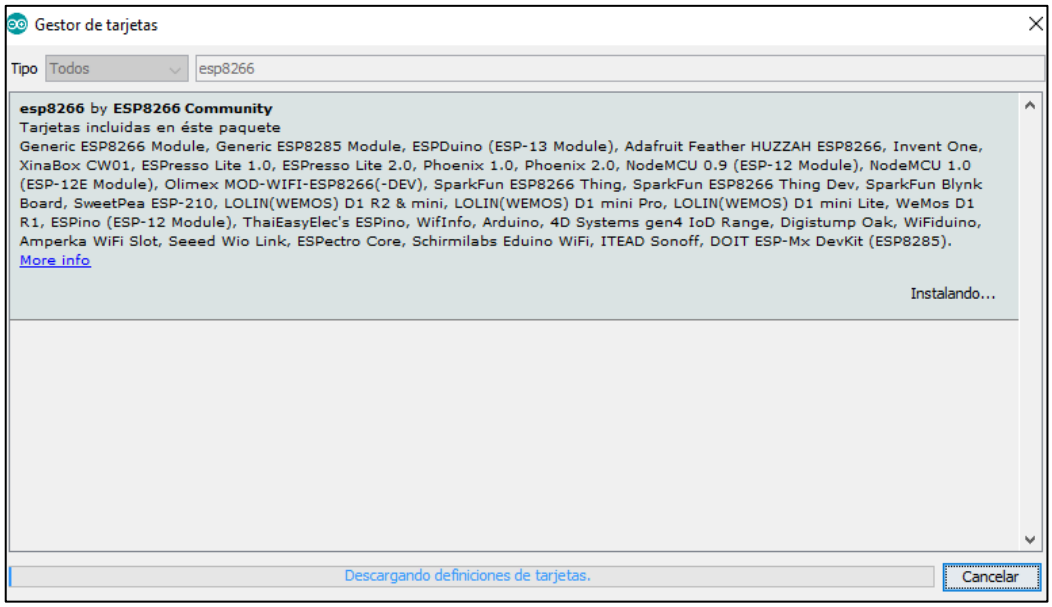


Figura 3.34: Arduino IDE, Gestor de tarjetas

Una vez instalado, se selecciona la placa "*NodeMCU 1.0 (ESP-12E Module)*" en el menú Herramientas > Placa > *NodeMCU 1.0 (ESP-12E Module), c*omo se observa en la figura 3.35.

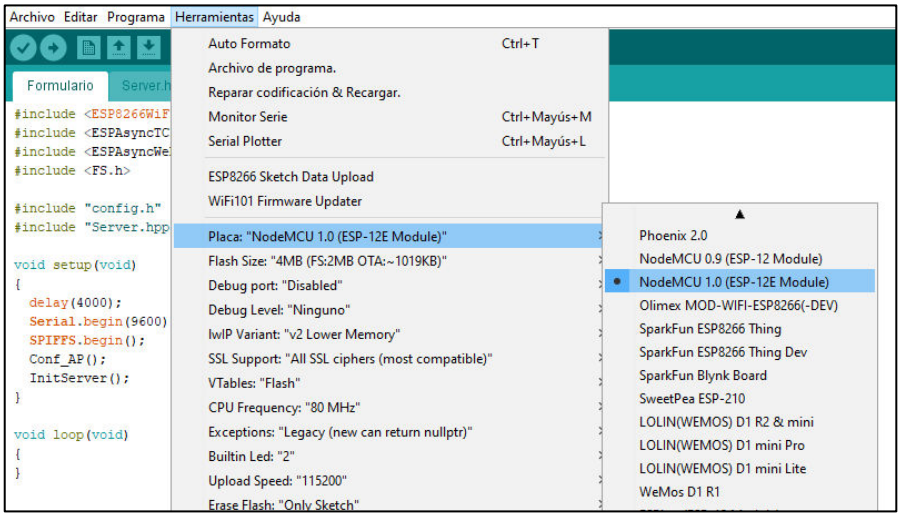


Figura 3.35: Arduino IDE menú Herramientas > Placas

Posteriormente se procede a cambiar la velocidad de subida, esta velocidad debe ser la misma que se configuró en el programa principal que se realizó para el Arduino Mega, esta velocidad es de 9600 bps, para el módulo esp8266 se la selecciona en el menú > *Upload Speed* > 9600, como se observa en la figura 3.36.

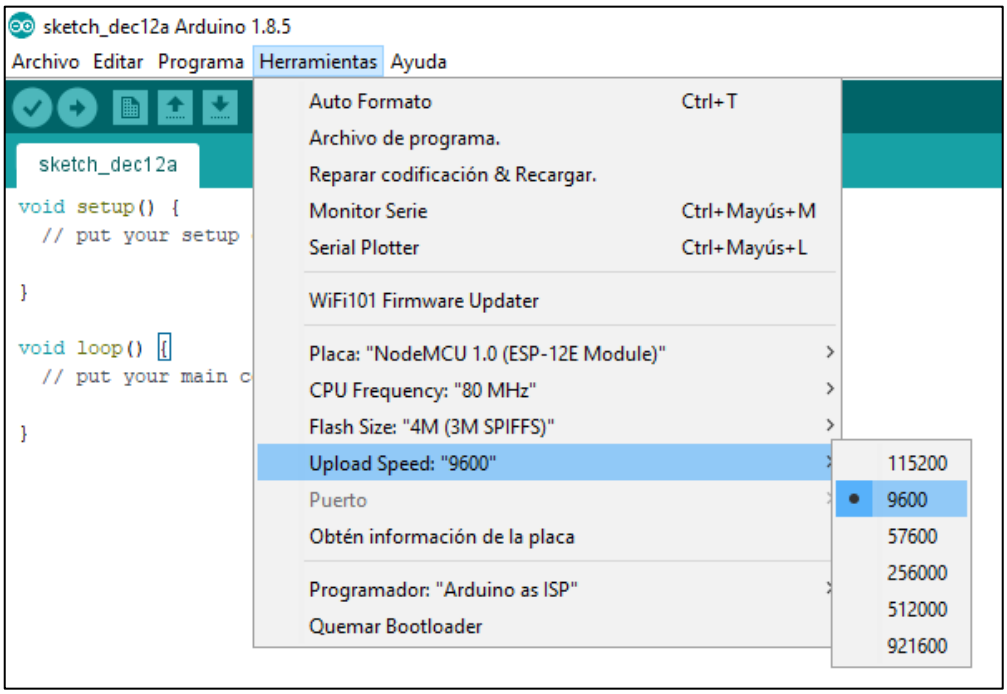


Figura 3.36: Arduino IDE menú Herramientas > *Upload Speed*

Para que la computadora y la placa de Arduino reconozcan al módulo esp8266 se debe instalar *drivers* adicionales, el controlador a instalar es genérico de *silicon 210X*, que permite la conversión serial USB del módulo WiFi con el computador.

Al iniciar la instalación del controlador se abrirá la ventana del Asistente para la instalación de controladores de dispositivos, se pulsa el botón siguiente para que inicie la instalación.

Una vez terminada la instalación del controlador se mostrará en una ventana el nombre del controlador y el estado de este, cuando el estado se encuentre en listo para su uso, como se observa en la figura 3.37, se cerrará la ventana del asistente de instalación.

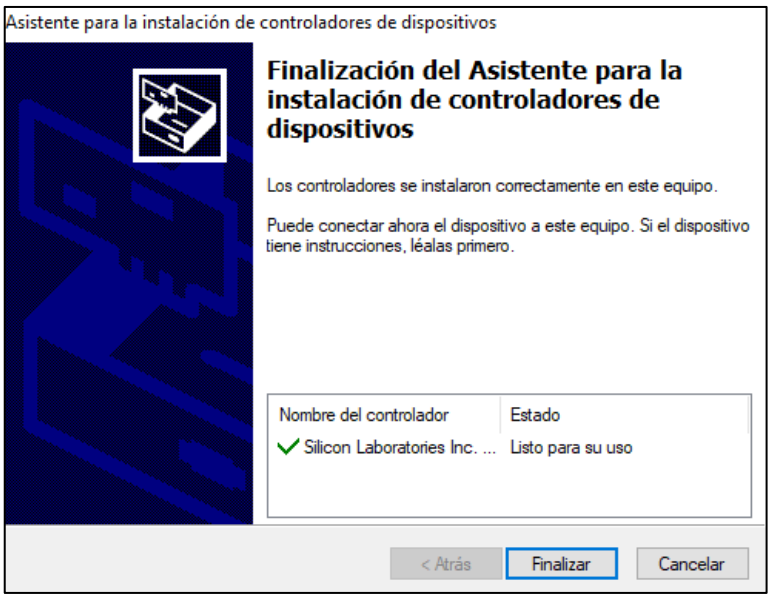


Figura 3.37: Instalación del controlador finalizada

Una vez terminada la instalación se habilitarán ejemplos usando el módulo esp8266, estos ejemplos se encontrarán en el menú Archivos > ejemplos, como se observa en la figura 3.38.

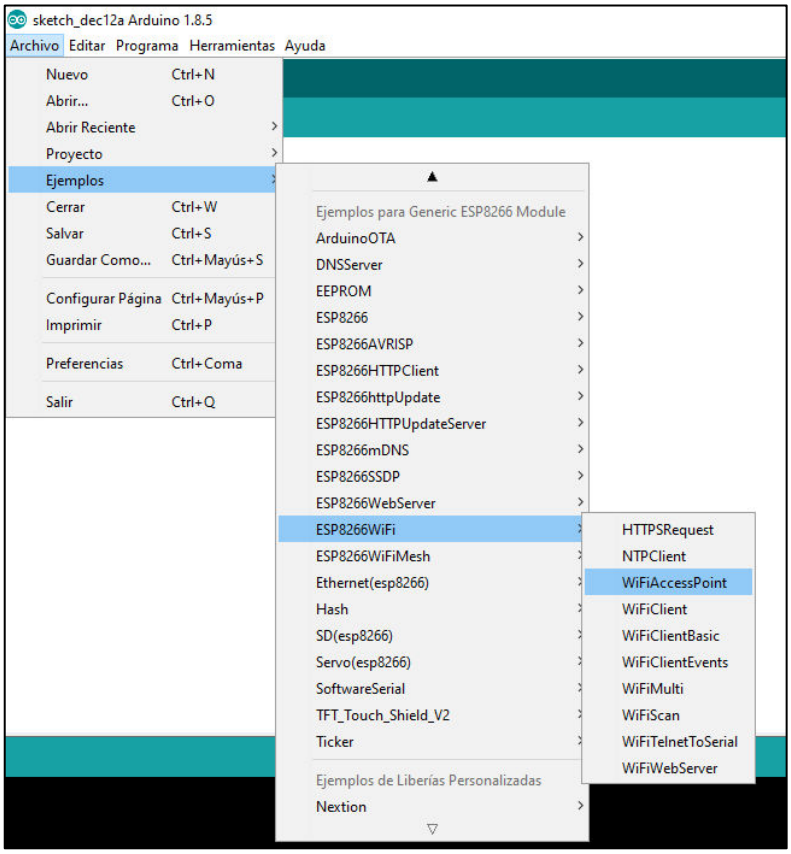


Figura 3.38: Arduino IDE ejemplos de esp8266.

El módulo esp8266 posee tres modos de funcionamiento explicados en la tabla 3.7.

Tabla 3.7: Modos de funcionamiento del módulo esp8266. [15]

Modo de Funcionamiento	Descripción
Station Mode (STA)	En este modo el módulo se conectará a una red WiFi-generada por un *Access Point,* donde en el módulo se debe configurar el nombre de la red y la clave de la red a la que se quiere conectar.
Access Point (AP)	En este modo el módulo actúa como un *Access Point* proporcionando una red WiFi a otras estaciones móviles. A este modo se lo denomina *Soft Access Point* porque no cuenta con una interfaz alámbrica que permita la conexión a Internet.
Soft AP + Station	Este modo está configurado para trabajar tanto en modo *station* como en modo *access point*.

La función que tomará esp8266 dentro de los módulos entrenadores es de un *Access Point*, el cual permitirá a los estudiantes ingresar a la red WiFi que genera el módulo, con un dispositivo que posea acceso a la red WiFi y podrán ingresar su nombre, apellido y número de cédula.

Para configurar el módulo esp8266 se debe crear un *sketch* único para cada entrenador, donde el nombre de la red y la contraseña de este irán variando por cada módulo.

Para el uso del módulo esp8266 se instalaron los *plugins* y controladores necesarios para que pueda ser configurado en Arduino IDE, caso contrario sería necesario configurar el módulo con comandos AT. [22]

3.3 Configuración página de registro

Para configurar el módulo esp8866 para que trabaje como un servidor y permita mostrar la información que contiene en el código, esta debe estar configurada como un *Access Point* y como servidor *web* que permita responder una petición realizada por un usuario conectado al módulo esp8862. [23]

Para ello es necesario tener dos *plugins* instalados, el primero que permita reconocer al módulo esp8862 como una placa de Arduino y el segundo que permita acceder a la

memoria *flash* de un microcontrolador como si fuera un sistema de archivos de un ordenador normal, a este sistema se lo conoce como *SPIFFS*. [24]

Usar el módulo esp8266 junto con *SPIFFS* resulta útil para: [25]

- Crear archivos de configuración.
- Guardar datos de forma permanente.
- Guardar archivos HTML y CSS para construir un servidor.
- Guardar imágenes, iconos, etc.

El uso de *SPIFFS* permite que el código HTML no esté dentro del mismo código, permitiendo crear un archivo HTML externo y guardarlo dentro de la memoria *flash* del módulo esp8266.

Para la instalación del *Filesystem Uploader* primero se descargó el archivo *ESP8266FS-0.0.5.zip,* el cual debe ser descomprimido en la carpeta *C:\Program Files (x86)\Arduino\tools*, como se muestra en la figura 3.39.

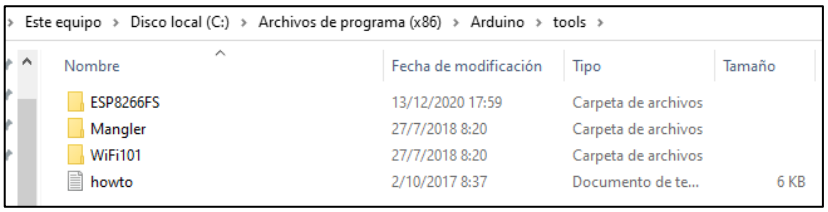


Figura 3.39: Ubicación de la carpeta de *tools*.

Posteriormente es necesario reiniciar Arduino IDE, luego se abre un archivo nuevo o el que se estaba trabajando, se debe tener seleccionado como placa al módulo esp8266, si se instaló correctamente el *plugin*, en el menú de herramientas aparecerá una nueva opción cuyo nombre es *ESP8266 Sketch Data Upload*, como se puede observar en la figura 3.40.

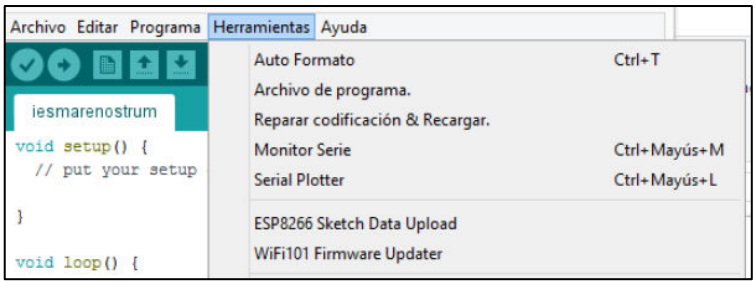


Figura 3.40: Arduino IDE menú *ESP8266 Sketch Data Upload*

El módulo esp8266 debe actuar como un servidor *web* para ello es necesario que este logre atender a varios clientes de forma simultánea, es necesario que el módulo esp8266 trabaje como un servidor asincrónico, para ello es necesario usar la librería *ESPAsyncWebServer.h*. [26]

Para agregar la librería *ESPAsyncWebServer.h,* se ingresa al menú Programa > Incluir librería > Gestionar librería, en la ventana emergente del Gestor de Librerías se busca en la barra de búsqueda la librería *ESPAsyncWebServer,* como se observa en la figura 3.41.

Figura 3.41: Instalación de librería *ESPAsyncWebServer*

Otra librería que se necesita para que el módulo esp8266 funcione como servidor *web* es la librería *ESPAsyncTCP.h,* para incluir esta librería se ingresa al menú Programa > Incluir librería > Gestionar librería, en la ventana emergente del Gestor de Librerías buscamos en la barra de búsqueda la librería *ESPAsyncTCP,* como se observa en la figura 3.42.

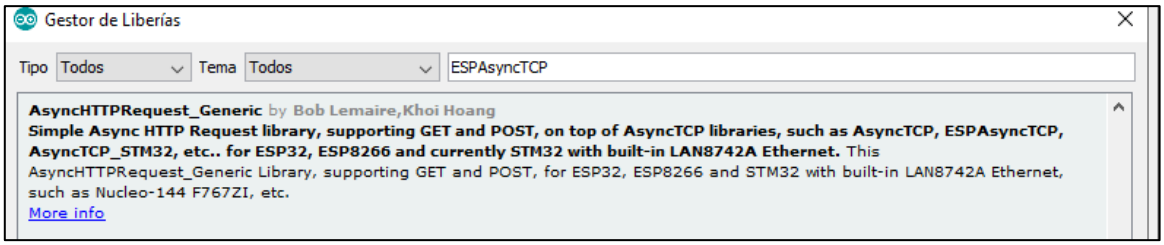


Figura 3.42: Instalación de librería *ESPAsyncTCP*

Con los *plugins* y librerías previamente instaladas se realiza un código que permita el manejo del esp8862 como un servidor *web*, en la figura 3.43 se observa cómo se incluyen las librerías *ESP8266WiFi.h, ESPAsyncTCP.h, ESPAsyncWebServer.h, FS.h.*

```
Formulario | Server.hpp | config.h

#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
```

Figura 3.43: Librerías incluidas esp8862

En la figura 3.44 se usó el comando *#include* que permite incluir bibliotecas externas al *sketch* si se usa las “<>”, para usar un *sketch* local se usa las comillas dobles, para que el *sketch* pueda ser utilizado, estos deben estar en la misma carpeta que el programa principal, como se observa en la figura 3.45

```
#include "config.h"
#include "Server.hpp"
```

Figura 3.44: *Sketches* necesarios para el servidor

Figura 3.45: Carpeta contenedora del *sketch* del esp8862

En la figura 3.45 también se observan los archivos que se configuraron en el servidor *web* con el nombre de *config.h* y el archivo de *server.hpp* donde se configura la página de registro que será presentada al estudiante.

Para la configuración del módulo esp8862 se usó la librería *ESP8266WiFi.h* la cual permite configurar el módulo WiFi en modo *station* y aumentar la seguridad de red configurando los parámetros *SSID* y *Password*, como se muestra en la figura 3.46. Los parámetros de SSID y *Password* son únicos para cada módulo.

```
char* ssid     = "Modulo12"; // "Modulo12"
char* password = "Modulo12"; // "modulo12"
char* hostname = "Modulo12";
```

Figura 3.46: Configuración de SSID y *password*.

Es necesario definir los parámetros de dirección IP, *gateway* y *subnet* para el esp8266, esto se lo realiza con el comando *IPaddress* seguido por dirección IP, *gateway* o *subnet* según sea la dirección IP que se desea configurar y entre paréntesis se coloca la dirección IP que se desea como se observa en la figura 3.47.

```
//configuracion de ip´s modulo esp8266
IPAddress ip(192, 168, 1, 1);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
```

Figura 3.47: Configuración de dirección IP local para el módulo esp8266

Una vez configuradas las direcciones IP del módulo esp8266, es necesario crear el punto de acceso, para ello es necesario definir la velocidad del puerto serial con 9600 baudios que es la misma que se configuró en el Arduino Mega2560. Para crear el punto de acceso se usa el comando *WiFi.softAP(ssid, password, channel, hidden, max_connection)* donde: [15]

- *Ssid*: Cadena de caracteres que contiene el nombre de la SSID con un máximo de 32 caracteres
- *Password*: Cadena de caracteres opcionales que contiene la contraseña, para que la red esté protegida con WPA2-PSK la contraseña debe ser de al menos 8 caracteres de longitud.
- *Channel*: Parámetro opcional que define el canal de WiFi que va a ser usado, va de 1 a 13 y por defecto se usa el canal 1.
- *Hidden*: Parámetro opcional que oculta el nombre de la SSID.
- *max_connection*: Define el número máximo de conexiones simultáneas que soportará el esp8266, va de 1 a 8, por defecto permite 4 conexiones simultáneas.

Para usar el módulo esp8266 como un AP se debe configurar las direcciones IP, *gateway* y *subnet*, las direcciones IP configuradas para el módulo esp8266 se las puede observar en la figura 3.47.

Para configurar las direcciones IP del módulo esp8266 se debe usar el comando *Wifi.softAPConfig(ip, gateway, subnet)* que configura la dirección IP, el *gateway* y la máscara de la red.

Con el comando *Wifi.softAP(ssid, password, Número del canal) se configura el nombre de la red, la contraseña y el número del canal usado,* estas variables fueron definidas en la figura 3.47 lo cual permite editar de manera rápida el nombre de la red y la contraseña sin necesidad de alterar el código de gran manera, como se observa en la figura 3.48.

```
void Conf_AP()
{
  WiFi.mode(WIFI_AP);
  WiFi.softAPConfig(ip, gateway, subnet);
  WiFi.softAP(ssid, password, 13); // (ssid, password, channel, hidden, max_connection)

  Serial.print(",");
  Serial.print(ssid); //nombre de la red
  Serial.print(",");
  Serial.print(password); //contraseña
  Serial.print(",");
  Serial.print(WiFi.softAPIP());//ip a conectarse
}
```

Figura 3.48: Configuración de módulo esp8266 como AP

A través de la comunicación serial, el nombre de la red, contraseña y la dirección IP del módulo esp8266 que esté usando el módulo entrenador serán enviados al Arduino, para que estos puedan ser presentados al estudiante a través de la pantalla Nextion.

Para la configuración del servidor *web* es necesario el uso de la librería *ESPAsyncWebServer.h.* En la figura 3.50 se usa el comando *AsyncWebServer server (80)*, se asigna el puerto 80 por defecto al esp8266, se usa el puerto 80 para la navegación *web* de forma no segura HTTP.

```
AsyncWebServer server(80);

void handleFormText(AsyncWebServerRequest *request)
{
  String nomb = request->arg("nomb");
  String apel = request->arg("apel");
  String ci = request->arg("ci");
```

Figura 3.49: Configuración de servidor *web*

En la figura 3.49 se crea tres variables del tipo *string* las cuales esperan a recibir los datos de nombre, apellido y cédula de identidad que provienen de la página de registro que se observa en la figura 3.50, estos datos serán ingresados por el estudiante y serán recibidos por el módulo esp8266.

Figura 3.50: Página de registro

En la figura 3.51, cuando los datos del estudiante son recibidos se los envía por comunicación serial al Arduino, los datos enviados serán agrupados, para poder diferenciarlos de los datos *ssid, password* y dirección IP del módulo. Los datos del estudiante serán enviados junto con una bandera "User," seguido por el nombre, apellido y número de cédula que el estudiante ingresó en la página de registro, se usa el caracter "," para identificar los datos en el Arduino y poder separarlos.

```
void handleFormText(AsyncWebServerRequest *request)
{
  String nomb = request->arg("nomb");
  String apel = request->arg("apel");
  String ci = request->arg("ci");

  Serial.print("User,");
  Serial.print(nomb + ",");
  Serial.print(apel + ",");
  Serial.print(ci);

  if (nomb == "" || apel == "" || ci == "") {
    request->redirect("/");
  }
  else if((nomb == "admin" || apel == "admin" || ci == "1726018136")) {
    request->redirect("/devs.html");
  }
  else {
    WiFi.mode(WIFI_OFF); //apaga wifi despues de enviar todos los datos
  }
}
```

Figura 3.51: Envío de datos por comunicación serial a Arduino

Cuando no se encuentren los datos, los datos vacíos se enviarán hacia el Arduino y volverá a iniciar el código esperando a recibir los datos completos del estudiante, si se ingresan los datos de administrador se mostrará en la pantalla del dispositivo móvil una página con el nombre de los involucrados en el diseño, programación e implementación de los módulos digitales. Cuando todos los datos enviados son correctos ese procederá a apagar el módulo WiFi.

Para mostrar la página del servidor es necesario tener dos archivos, los cuales son el código de la página en HTML y los estilos de texto, estos archivos deben estar dentro de una carpeta nombrada “data” y esta carpeta debe estar en la misma ubicación que el archivo del *sketch* de Arduino del módulo esp8266, como se observa en la figura 3.52. En la carpeta data también deben estar incluidos los archivos multimedia que necesiten mostrar las páginas HTLM. [25]

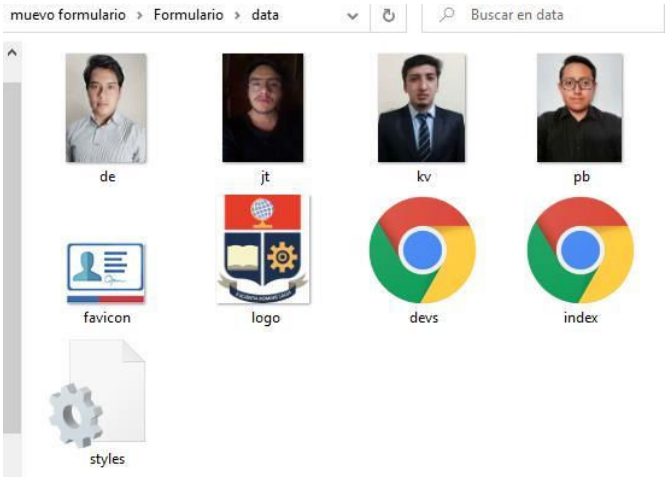


Figura 3.52: Contenido carpeta data

Con los archivos HTML y los *styles* guardados en la carpeta data, en el programa de Arduino IDE en el cual se instaló un *plugin* que permite cargar la carpeta data en la memoria *flash* del módulo esp8266, con el *sketch* del módulo esp8862 abierto, en el menú de Herramientas se encontrará la opción *ESP8266 Sketch Data Upload* para cargar la carpeta data a la memoria *flash* del Arduino, como se observa en la figura 3.53.

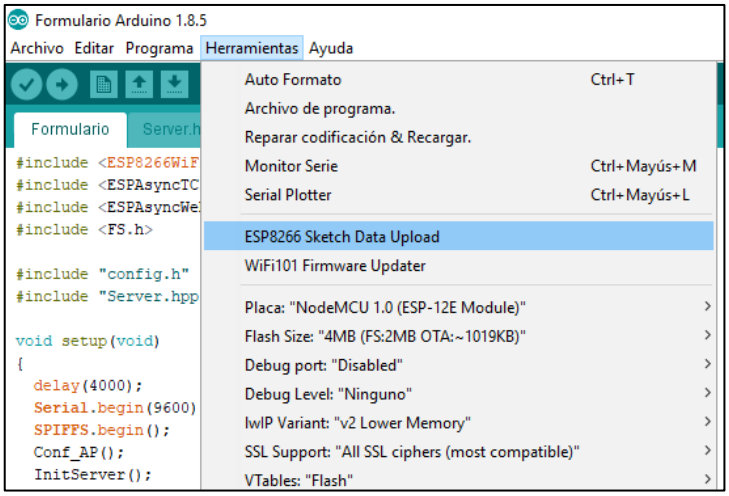


Figura 3.53: Arduino IDE, *Sketch Data Upload*

En la figura 3.54 a través del sistema de archivos de SPIFFS se cargará la página de registro como archivo HTML que será mostrado en la pantalla del dispositivo móvil, el comando "*handleFormText*" permite imprimir el texto recibido por puerto serial y redireccionar al estudiante a la página index.html. Con el comando "*server. on("/devs", HTTP_POST, handleFormText); "* redirecciona al estudiante a la página devs.html. Cuando termina la conexión muestra en texto plano el mensaje de "Página no encontrada".

```
void InitServer()
{
  server.serveStatic("/", SPIFFS, "/").setDefaultFile("index.html");

  server.on("/SetText", HTTP_POST, handleFormText);

  server.on("/devs", HTTP_POST, handleFormText);

  server.onNotFound([](AsyncWebServerRequest * request) {
    request->send(404, "text/plain", "Pagina no encontrada");
  });

  server.begin();
}
```

Figura 3.54: Sistema de archivos de SPIFFS

Las páginas de *index.html* y *devs.html*, fueron programadas en HTML, en la figura 3.55 se muestra que para poder obtener los datos que el estudiante ingrese se usará *<input type="text" name="nomb" value=" " placeholder="Ingrese su Nombre">
* que genera un cuadro de texto definido como entrada el cual guardará el nombre ingresado en una variable "nomb", dentro del cuadro de texto muestra el texto de "Ingrese su Nombre", esto se lo realiza para el apellido y el número de cédula también.

```html
<html class="no-js" lang="es">
   <head>
      <meta charset="utf-8">
      <meta http-equiv="x-ua-compatible" content="ie=edge">
      <link rel="icon" type="image/png" href="favicon.png">
      <title>REGISTRO</title>
      <meta name="Registro modulos logicos" content="">
      <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=yes">
   </head>
   <body style="background-color:black">
      <h1 style="color: white"> <img src="logo.png" width="65" height="75" alt="Logo EPN" align="middle">Registro</h1>
        <form action="/SetText" method="post">
            <font style="color: white">Nombre:</font><br>
            <input type="text" name="nomb" value="" placeholder="Ingrese su Nombre"><br>
            <font style="color: white">Apellido:</font><br>
            <input type="text" name="apel" value="" placeholder="Ingrese su Apellido"><br>
            <font style="color: white">C.I.:</font><br>
            <input type="number" name="ci" value="" placeholder="Ingrese su C.I." min="0100000009" max="2460000009"><br><br>
        <button type="submit" value="Submit">Enviar</button>
      </form>
    </body>
</html>
```

Figura 3.55: Código fuente de la página HTML registro

3.4 Instalación del programa en un módulo de prueba.

Con el código de los módulos terminado se procede a cargar el *sketch* en la placa de Arduino Mega 2560, para ello se usa el *software* de Arduino IDE y la placa de Arduino Mega debe estar conectada a la computadora por un puerto USB de la computadora. Para cargar un archivo primero se debe identificar en qué puerto está conectado el Arduino, para cargar el programa se pulsa el botón de “subir” para copilar el código en la placa de Arduino, si el código no presenta ningún error de programación se cargará exitosamente en la placa de Arduino, como se observa en la figura 3.56.

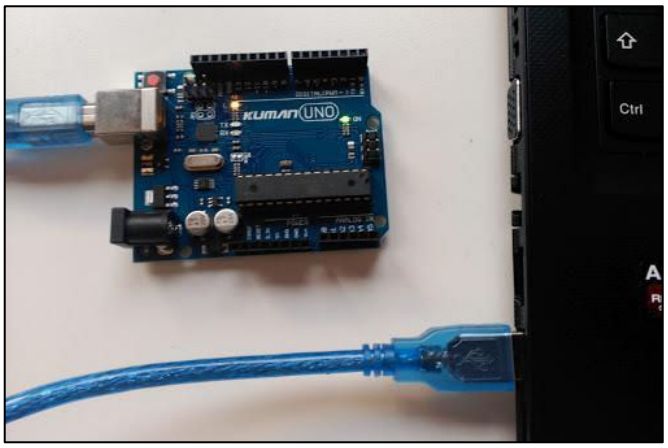


Figura 3.56: Arduino conectado a la computadora

Para comprobar que todas las funciones del programa funcionen correctamente se debe colocar un Arduino mega con el programa cargado dentro de un módulo entrenador que se encuentre armado, como el de la figura 3.57.

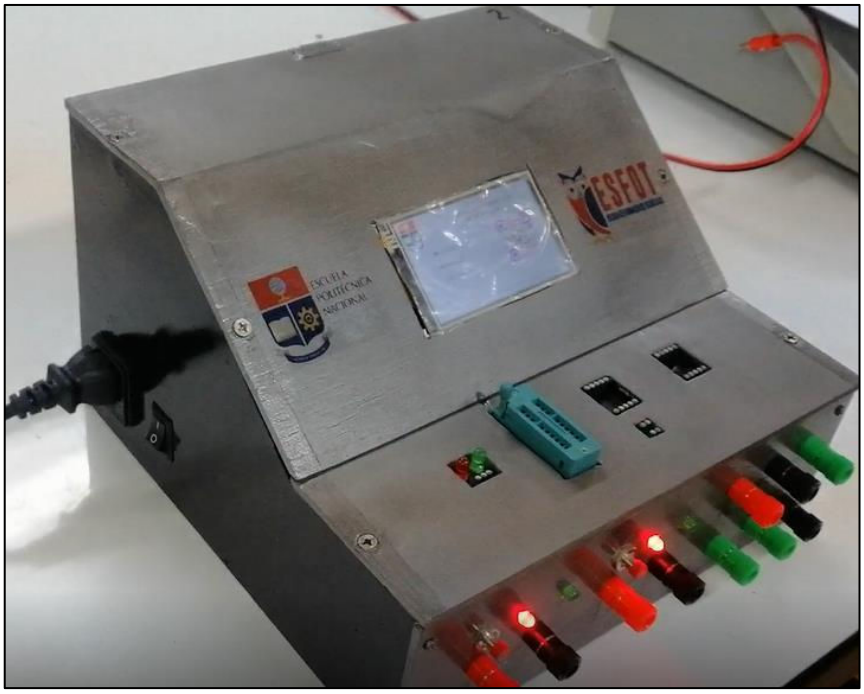


Figura 3.57: Módulo entrenador lógico armado

Al encender el módulo la pantalla Nextion muestra el nombre de la red, la contraseña y la dirección IP, en la posición que se encuentren estos datos serán remplazados por los valores del nombre de la red, la contraseña y la dirección IP que están configurados en cada módulo esp8266. Estos datos se les puede observar en la figura 3.58, cabe aclarar que tanto el nombre como la contraseña de cada módulo es diferente, solo la dirección IP para el registro es la misma para todos los módulos.

Figura 3.58: Módulo entrenador, Etapa de encendido

Para realizar el registro en el módulo es necesario el uso de un dispositivo móvil el cual permita el acceso a una red WiFi, una vez conectado a la red del módulo, se abre el navegador del dispositivo móvil y se ingresa la dirección IP 192.168.1.1 que fue

mostrada en la pantalla Nextion, la cual llevará a la página de registro donde se podrá ingresar los datos del estudiante como se muestra en la figura 3.59, se procede a llenar los datos.

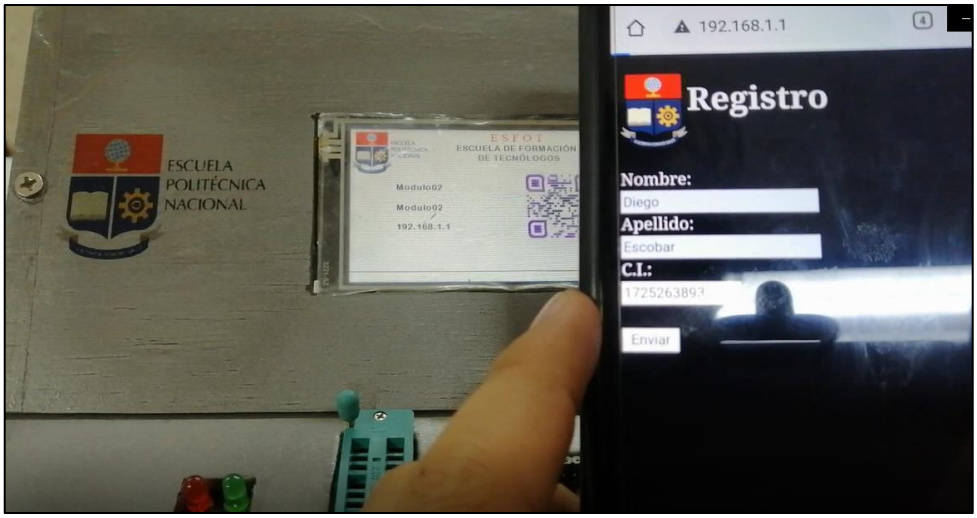


Figura 3.59:Módulos entrenadores, registro

Un registro exitoso se logra al haber ingresado todos los datos de nombre, apellido y número de cédula, al ser exitoso se muestra en la pantalla los datos que el estudiante ingresó en la página de registro como se observa en la figura 3.60 y posteriormente muestra la página 1 de la pantalla Nextion.

Figura 3.60: Módulo entrenador, registro exitoso

Un registro fallido sucede cuando uno o más de los datos se encuentren vacíos, como se muestra en la figura 3.61, esto no evita que el estudiante pueda volver a ingresar los

datos en la página de registro, si se ingresan los datos correctamente el módulo se iniciará normalmente.

Figura 3.61: Módulo entrenador, registro fallido

En la figura 3.62 se puede observar que la pantalla indica el modo correcto en que se debe conectar los LEDs, se deben conectar con el lado negativo a la izquierda y para los *displays* el ánodo común se encuentra a la izquierda y el de cátodo común se encuentra a la derecha. Al colocar un *display* mal o en el lugar incorrecto no afectará ni provocará ningún daño hacia el módulo o el *display*.

Figura 3.62: Comprobador de LEDs y *displays*

Para usar el comprobador de compuertas lógicas de la familia 74XX se debe:

- Colocar el encapsulado con la muesca apuntando hacia la pantalla.
- Asegurarlo de manera firme al zócalo ZIF bajando la palanca que este posee como se visualiza en la figura 3.63.
- Presionar el botón de comprobar que se encuentra en la pantalla Nextion.

En la pantalla se mostrará la numeración de la compuerta lógica, si no se encuentra dañada, si la compuerta presenta algún daño en la pantalla Nextion se mostrará el mensaje de “Averiada”, en caso de que la compuerta esté mal asegurada se mostrará el mismo mensaje de “Averiada”.

Figura 3.63: Comprobador de compuertas

El comprobador de transistores se desarrolló mediante un circuito integrado 555, por la parte de programación se habilita el funcionamiento al entrar a la opción de transistores TBJ, al pulsar el botón comprobar, el circuito del 555 comenzará el funcionamiento, en la pantalla se mostrará la combinación de LEDs encendidos y el tipo de transistor al que pertenece como se ve en la figura 3.64.

Figura 3.64: Comprobador de transistores

El voltímetro del módulo permite leer voltajes de cero a cinco voltios, que se ve reflejado en la pantalla mediante un gráfico de aguja y el valor de voltaje medido será visto en la parte inferior del gráfico de la aguja, los valores de voltaje estarán representados en números decimales, esto se puede observar en la figura 3.65.

Figura 3.65: Voltímetro

En la parte inferior de la pantalla se va a lograr visualizar los valores de frecuencia, estado lógico y señal de reloj, estos valores se lograrán actualizar si se produce algún cambio. Estos datos se pueden observar en la figura 3.66.

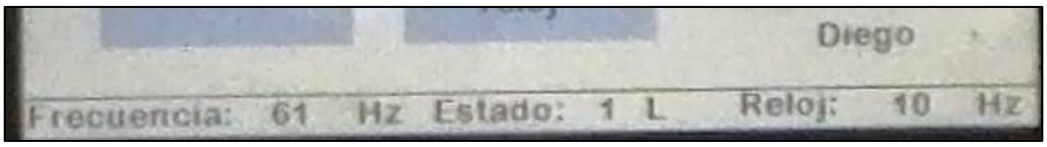


Figura 3.66: Pantalla principal del módulo lógico

3.5 Pruebas de funcionamiento.

El modulo entrenador se lo mantuvo encendido en periodos de dos horas en un día, simulando un día normal de clases, donde no se observo que el módulo presente algún efecto negativo ni por el software o el hardware. De igual manera se simulo un corte repentino de la red eléctrica, donde se desconecto el cable de poder del módulo sin apagarlo previamente.

Se pudo observar que el modulo no presenta los datos de nombre de la red, contraseña y dirección ip, cuando a este se lo reinicia repetidamente un periodo corto de tiempo, para que el modulo presente estos datos esta ves se lo debe apagar y esperar dos minutos antes de volver a encenderlo.

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Cuando se escribe un *sketch* este es compilado por el *software* de Arduino IDE, si existe algún tipo de error, este será detectado por el *software*, pero cabe aclarar que, si el programa de compilación no detecta ningún error, esto no significa que el código esté funcional al cien por ciento, para ello hay que realizar diversas pruebas que lleven al estrés al código, permitiendo detectar cualquier posible fallo.

Tener buenas prácticas al realizar un sketch, como comentar de manera clara lo que realiza el código, tener bien identificada las diferentes partes del sketch facilita la comprensión del mismo, aunque estas buenas prácticas no son obligatorias en todos los casos, son de gran ayuda para que otras personas pueden comprender el funcionamiento del código y si es necesario modificarlo según su conveniencia.

En el desarrollo de este proyecto se observó que las características que este código ofrece son de gran ayuda para los estudiantes, porque reúne en un módulo elementos que resultan de gran utilidad al momento de realizar una práctica de laboratorio.

Se evidenció que existe una gran variedad de librerías y ejemplos que son útiles al momento de realizar un código de Arduino, muchas de las librerías se encuentran en inglés, por ello para comprender de mejor manera cómo funciona la librería es necesario contar con un nivel de inglés adecuado.

Las placas de Arduino al igual que el *software*, son libres de modificarlos o agregar nuevos módulos al *hardware*, en el caso de los códigos y librerías que son de código abierto permiten que varios desarrolladores independientes creen nuevas o modifiquen las librerías existentes, logrando tener un repertorio gigante de opciones para el momento de desarrollar un código.

Se puede concluir que al escribir un código o *sketch* para Arduino o para algún otro microcontrolador, según el lenguaje de programación usado el código puede reducir o aumentar su complejidad.

Por tanto, con las diferentes funciones que fueron incluidas en el código presente en este documento, resultan de gran ayuda para los estudiantes en el Laboratorio de Tecnología Eléctrica y Electrónica, área de Sistemas Digitales, tanto para los estudiantes con experiencia en el uso circuitos electrónicos como para los que no la tienen.

4.2 Recomendaciones

Con el crecimiento de Arduino y la facilidad de programación, diversas marcas de equipos tecnológicos ofrecen sus propias librerías para interactuar con sus equipos, también es posible que un elemento como un sensor cuente con librerías no oficiales que facilitan la obtención de datos de este. Esto permite que su uso educativo como profesional sea mas fácil, pero si no se logra comprender el funcionamiento de estos puede perjudicar y dificultar el uso del mismo.

Cuando se realiza la prueba de un código, se lo simula por primera vez y este funciona correctamente, esto no significa que el código no pueda contener errores, por lo cual es necesario probar exhaustivamente todas las funciones del código. Esto es más evidente cuando el código contiene una gran cantidad de líneas de código.

El pasar de un ambiente virtual al físico presenta diferentes complicaciones, tales como los factores ambientales, las características propias de los materiales usados e incluso se debe tener en cuenta la negligencia humana que puede afectar el resultado final. Todos estas factores siempre estarán presentes, los cuales deben ser identificados para

poder corregirlos o en el pero de los casos reducir el impacto que tendrán en el resultado final.

5 BIBLIOGRAFÍA

[1] I. V. P. Ing. Luis Tituaña, «Tópico 7: Multiplexores,» Laboratorio de Sistemas Digitales, 2017.

[2] T. L. Floyd, Fundamentos de sistemas, Madrid: PEARSON EDUCACIÓN S.A, 2006.

[3] I. V. P. Ing. Luis Tituaña, «Tópico 6: Sumadores,» Laboratorio de Sistemas Digitales, 2017.

[4] Y. FM, «xataka.com,» 3 agosto 2018. [En línea]. Available: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno. [Último acceso: 17 junio 2020].

[5] «https://www.arduino.cc/,» 7 septiembre 2015. [En línea]. Available: https://www.arduino.cc/en/Guide/Environment#. [Último acceso: 17 junio 2020].

[6] «Arduuino.cc,» 5 febrero 2018. [En línea]. Available: https://www.arduino.cc/en/Guide/Environment. [Último acceso: 12 dicicembre 2020].

[7] «https://www.arduino.cc,» 7 febrero 2017. [En línea]. Available: https://www.arduino.cc/en/Guide/Libraries. [Último acceso: 17 junio 2020].

[8] «https://arduino.cl,» [En línea]. Available: https://arduino.cl/que-es-arduino/. [Último acceso: 17 junio 2020].

[9] C. Veloso, «ETOOLS,» 19 06 2018. [En línea]. Available: https://www.electrontools.com/Home/WP/arduino-mega-2560-caracteristicas/. [Último acceso: 20 10 2020].

[10] «nextion.tech,» [En línea]. Available: https://nextion.tech/. [Último acceso: 1 agosto 2020].

[11] J. T. Kevin Viteri, «DISEÑO DE MODULOS ENTRENADORES LOGICOS CON CONEXIÓN INALAMBRICA A DISPOSITIVOS MOVILES PARA EL

LABORATOTIO DE TECNOLOGIA DE ELECTRICA Y ELECTROMECANICA, AREA DE SISITEMAS DIGITALES,» Quito, 2020.

[12] J. J. Y. Torres, «Diseño de una red Wi-Fi para la E.S.I,» [En línea]. Available: http://bibing.us.es/proyectos/abreproy/11138/fichero/memoria%252FCap%C3%ADtulo+3.pdf+. [Último acceso: 12 12 2020].

[13] A. Gonzales, «T3cnologic,» 14 mayo 2015. [En línea]. Available: http://www.t3cnologic.com/blog/2015/05/14/elegir-el-mejor-canal-en-el-router-para-configurar-tu-wifi/. [Último acceso: 12 diciembre 2020].

[14] «N.A,» [En línea]. Available: https://rambal.com/wi-fi/544-modulo-wifi-esp8266.html. [Último acceso: 12 12 2020].

[15] L. d. V. Hernández, «programarfacil.com,» [En línea]. Available: https://programarfacil.com/podcast/esp8266-wifi-coste-arduino/. [Último acceso: 5 febrero 2021].

[16] «creatividadcodificada.com,» 5 diciembre 2019. [En línea]. Available: https://creatividadcodificada.com/arduino/libreria-timerone-en-arduino/. [Último acceso: 1 agosto 2020].

[17] E. Zhou, «https://www.itead.cc/,» 30 julio 2015. [En línea]. Available: https://www.itead.cc/blog/nextion-tutorial-based-on-nextion-arduino-library. [Último acceso: 5 febrero 2021].

[18] «https://zaragozamakerspace.com/,» [En línea]. Available: https://zaragozamakerspace.com/index.php/lessons/mvc-nextion-arduino-communication/. [Último acceso: 1 agosto 2020].

[19] «wordpress.com,» 16 marzo 2015. [En línea]. Available: https://aprendiendoarduino.wordpress.com/2015/03/26/tipos-de-datos/. [Último acceso: 19 agosto 2020].

[20] CarlosVolt, «http://rogerbit.com/,» 24 abril 2018. [En línea]. Available: http://rogerbit.com/wprb/2018/04/frecuencimetro-con-arduino-uno-y-display-oled-sh1106/. [Último acceso: 1 agosto 2020].

[21] J. CORPORATION, «youtube,» 31 marzo 2020. [En línea]. [Último acceso: 12 diciembre 2020].

[22] I. Grokhotkov, «ESP8266 Arduino Core,» 2017. [En línea]. Available: https://esp8266-arduino-spanish.readthedocs.io/es/latest/esp8266wifi/readme.html. [Último acceso: 12 diciembre 2020].

[23] J. Parejo, «https://www.tecnologia-informatica.es/,» [En línea]. Available: https://www.tecnologia-informatica.es/Servidor-web-con-nodemcu-esp8266/. [Último acceso: 12 diciembre 2020].

[24] «www.tecnologia-informatica.es,» [En línea]. Available: https://www.tecnologia-informatica.es/Instalar-el-filesystem-uploader-para-esp8266/. [Último acceso: 12 diciembre 2020].

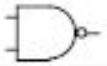
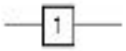
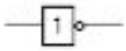
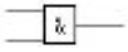
[25] «tecnologia-informatica.es,» [En línea]. Available: https://www.tecnologia-informatica.es/Instalar-el-filesystem-uploader-para-esp8266/. [Último acceso: 12 diceimbre 2020].

[26] «https://www.luisllamas.es/,» 4 septiembre 2019. [En línea]. Available: https://www.luisllamas.es/como-hacer-un-servidor-asincrono-en-esp8266/. [Último acceso: 12 diciembre 2020].

6 ANEXOS

6.1 Anexo A

RESUMEN DE COMPUERTAS LÓGICAS

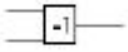
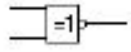
YES	NOT	AND	NAND
$S = A$	$S = \overline{A}$	$S = A \times B$	$S = \overline{A \times B}$
			
			
La salida es "1" cuando la entrada es "1"	La salida es "0" cuando la entrada es "1"	La salida es "1" cuando TODAS las entradas son "1"	La salida es "0" cuando TODAS las entradas son "1"

A	S
0	0
1	1

A	S
0	1
1	0

B	A	S
0	0	0
0	1	0
1	0	0
1	1	1

B	A	S
0	0	1
0	1	1
1	0	1
1	1	0

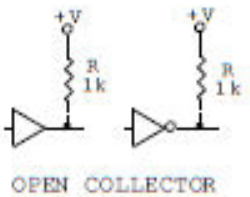
OR	NOR	XOR	XNOR
$S = A + B$	$S = \overline{A + B}$	$S = A \oplus B$ $S = (A \times \overline{B}) + (\overline{A} \times B)$	$S = \overline{A \oplus B}$ $S = (A \times B) + (\overline{A} \times \overline{B})$
			
			
La salida es "1" cuando ALGUNA entrada es "1"	La salida es "0" cuando ALGUNA entrada es "1"	La salida es "1" cuando son CONTRARIAS las entradas	La salida es "0" cuando son CONTRARIAS las entradas

B	A	S
0	0	0
0	1	1
1	0	1
1	1	1

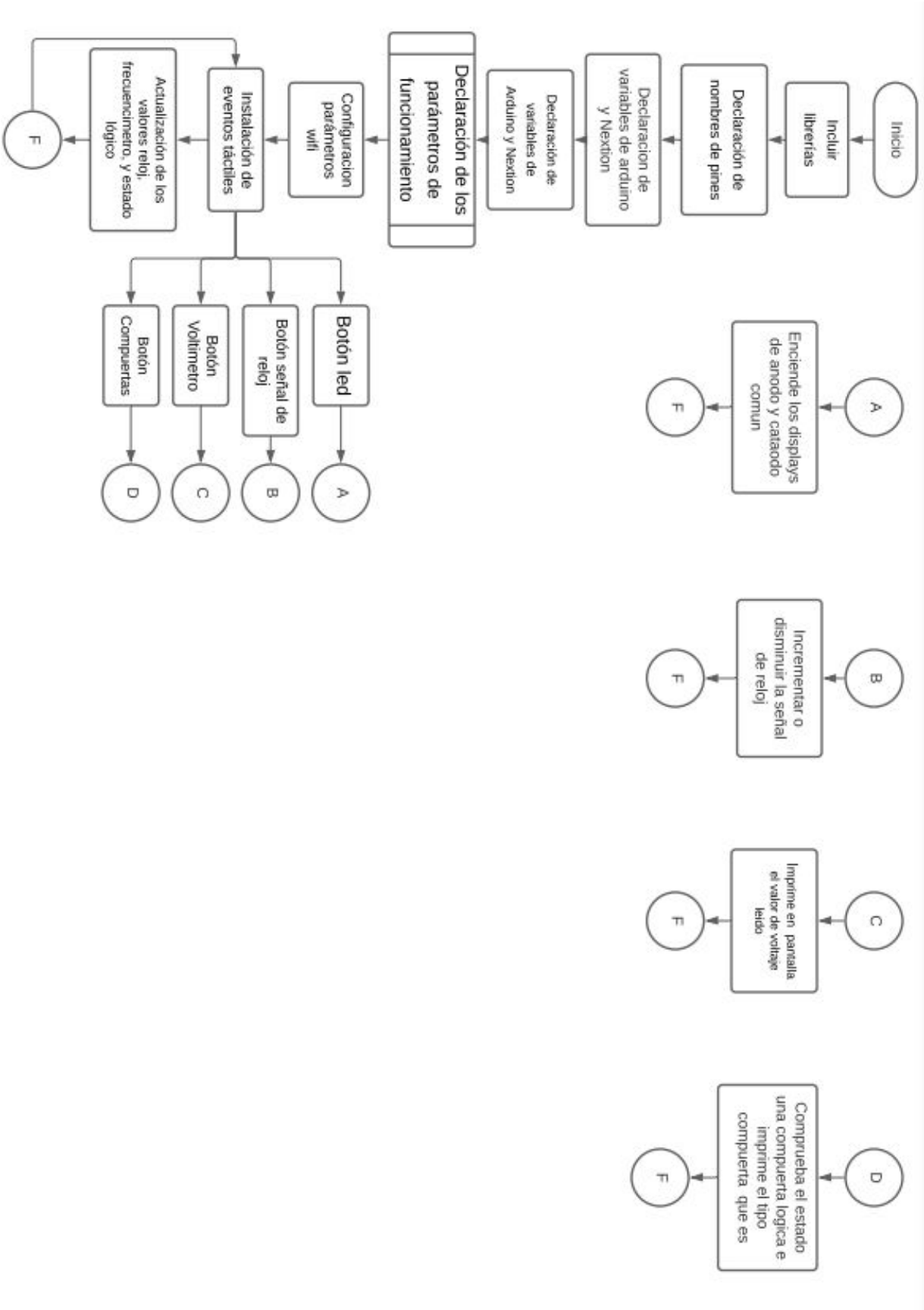
B	A	S
0	0	1
0	1	0
1	0	0
1	1	0

B	A	S
0	0	0
0	1	1
1	0	1
1	1	0

B	A	S
0	0	1
0	1	0
1	0	0
1	1	1



6.2 Anexo B Diagrama de flujo



6.3 Anexo C Código Arduino Mega

```
//Librerías

#include <Separador.h>    //Separar tramas

#include <TimerOne.h>     //Señal de reloj

#include <FreqCount.h>    //Frecuencímetro

#include "Nextion.h"      //Pantalla


//Declaración de nombres de pines

#define voltin A15

#define reloj  2

#define wf     3

#define logic  29

#define enacom 4


//Declaración de variables

float vin; //voltaje de entrada

char decimal[4]; // valor float de voltaje

char chb[10]; // ssid / nombre

char chc[10]; // pass / apellido

char chd[14]; // ip / ci

String datos, a, b, c, d; //trama | "Bandera" | ssid / nombre | pass / apellido | ip / ci

Separador s; //separador

unsigned long count = 0, us = 1000000, Fr = 1; // frecuencímetro | frecuencia en microsegundos | frecuencia reloj

uint32_t estadovlt, estadoled, estadocom, stmas, stmen; // estados botones pantalla
```

```
bool l; // estado lógico
int out1, out2, out3, out4, out5, out6, out7, out8; // valores compuertas
byte y[3][2] = {
  {0, 0}, //out1, out4, out6
  {0, 1}, //out2, out5, out7
  {1, 1}, //out3,        ,out8
};
//Declaración de variables pantalla Nextion
NexPage     page1    = NexPage (1, 0, "page1");    //Página 1
NexDSButton btnled   = NexDSButton(1, 14, "btnled");    // Btn LED's
NexDSButton btncom   = NexDSButton(3, 16, "btncom");    // Btn compuertas
NexDSButton btnvlt   = NexDSButton(1, 13, "btnvlt");    // Btn voltímetro
NexNumber   rj       = NexNumber  (1, 10, "n0");        // Numero del reloj
NexNumber   frec     = NexNumber  (1, 12, "n2");        // Numero de la frecuencia
NexNumber   logc     = NexNumber  (1, 11, "n1");        // Estado lógico
NexNumber   num      = NexNumber  (3, 17, "num");       // Numero compuerta
NexText     tvlt     = NexText    (4, 13, "t2");        // Texto voltaje
NexText     tcom     = NexText    (3, 5,  "comp");      // Texto compuertas
NexText     tb       = NexText    (0, 1,  "ssid");      // Texto ssid / nombre
NexText     tc       = NexText    (0, 4,  "pass");      // Texto pass / apellido
NexText     td       = NexText    (0, 5,  "ip");        // Texto ip  / ci
NexText     err      = NexText    (0, 6,  "err");       // Texto error de datos vacíos
NexText     tnomb    = NexText    (1, 15, "nomb");      // Texto nombre

NexDSButton más      = NexDSButton(5, 11,  "mas");       // Frecuencia mas
```

```
NexDSButton men      = NexDSButton(5, 12,  "men");       // Frecuencia menos


//lista de eventos táctiles
NexTouch *nex_listen_list[] =
{
  &btncom,
  &btnled,
  &btnvlt,
  &mas,
  &men,
  NULL
};
void setup() {
  //Inicialización de pantalla y botones
  nexInit();
  page1.show(); //DEBUG
  //Inicialización y declaración de puertos
  DDRA  =  B01111111;
  //PORTA =  PORTA | B01111111; // INICIA LED Apagados
  DDRC  =  B11111111;
  //DECLARAR PINES DE COMPUERTAS COMO ENTRADAS DIGITALES PARA EVITAR REINICIOS ////////////////////////////////////////////////////
  DDRF = B00000000;
  DDRK = B0000;
  pinMode(voltin, INPUT);
  pinMode(reloj, OUTPUT);
```

```
  pinMode(wf,    OUTPUT);
  pinMode(logic, INPUT);
  pinMode(enacom, OUTPUT);
  //Inicialización de comunicación wifi
  Serial1.begin(9600);
  //Inicialización del frecuencímetro
  FreqCount.begin(1000);
  //Inicialización de TimerOne
  Timer1.initialize(1000000); // (unidad a trabajar en microsegundo)
  Timer1.attachInterrupt(subida);  //(función de ISR)
}
void loop() {
  //Inicia configuración de registro
  coneccionwf();
    //Inicialización de eventos de la lista
  nexLoop(nex_listen_list);
  btnvlt.getValue(&estadovlt);
  if (estadovlt == 1) {
    //Voltimetro
    voltímetro();
    //Frecuencímetro
    frecuencimetro();
    //Reloj
    rj.setValue(Fr);
    //Estado Logico
```

```
    estadoL();
  }
  else {
    //Frecuencímetro
    frecuencimetro();
    //Reloj
    setfr();
    //Estado Lógico
    estadoL();
    //LED's
    leds();
    //Compuertas
    comp();
  }
  delay(25);
}
void leds() { //Encendido de puertos A y C (LED's)
  btnled.getValue(&estadoled);
  if (estadoled == 1) {
    for (int i = 0; i < 8; i++) {
      digitalWrite(22 + i, LOW);
      digitalWrite(30 + i, LOW); ///HIGH    //LEDS INVETIDOS
      delay(25);
      estadoled = 0;
    }
```

```
  }
  else {
    for (int i = 0; i < 8; i++) {
      digitalWrite(29 - i, HIGH);
      digitalWrite(37 - i, HIGH); /////LOW
      delay(25);
    }
  }
}

void setfr() {

  mas.getValue(&stmas);
  if (stmas == 1) {
    Fr = Fr + 1;
    mas.setValue(0);
  }
  men.getValue(&stmen);
  if (stmen == 1) {
    Fr = Fr - 1;
    men.setValue(0);
    if (Fr == 0) {
      Fr = 1;
    }
  }
```

```
  rj.setValue(Fr);
}
void subida() { //Reloj Rise
  digitalWrite(reloj, HIGH);
  Timer1.attachInterrupt(bajada, (us / (2 * Fr))); //Interrupción (ISR, Ancho de pulso)
}
void bajada() {//Reloj Down
  digitalWrite(reloj, LOW);
  Timer1.attachInterrupt(subida, (us / Fr) - (us / (2 * Fr))); //Interrupcion (ISR, Periodo -
Ancho de pulso)
}
void frecuencimetro() { //Frecuencímetro
  if (FreqCount.available()) {
    count = FreqCount.read();
    frec.setValue(count);
  }
}
void estadoL() { //Estado Lógico
  l = digitalRead(logic);
  logc.setValue(l);
}
void voltímetro() { //Voltímetro
  vin = ((analogRead(voltin) * 5.0) / 1023);
  dtostrf(vin, 5, 2, decimal);
  tvlt.setText(decimal);
}
```

```
void comp() { //Compuertas
  btncom.getValue(&estadocom);
  if (estadocom == 1) {
    digitalWrite(enacom, LOW);
    uno();
    dos();
    tres();
    DDRF = B00000000;
    DDRK = B0000;
    digitalWrite(enacom, HIGH);
    btncom.setValue(0);
    if (out1 == 0 && out2 == 0 && out3 == 4) {
      tcom.setText("AND");
      num.setValue(7408);
    }
    else if (out1 == 4 && out2 == 4 && out3 == 0) {
      tcom.setText("NAND");
      num.setValue(7400);
    }
    else if (out1 == 0 && out2 == 4 && out3 == 4) {
      tcom.setText("OR");
      num.setValue(7432);
    }
    else if (out1 == 0 && out2 == 4 && out3 == 0) {
      tcom.setText("XOR");
```

```
      num.setValue(7486);
    }
    else if (out4 == 6 && out5 == 0) {
      tcom.setText("NOT");
      num.setValue(7404);
    }
    else if (out6 == 4 && out7 == 0 && out8 == 0) {
      tcom.setText("NOR");
      num.setValue(7402);
    }
    else {
      tcom.setText("Averiada");
      num.setValue(0);
    }
  }
}
void uno() { /////Compuertas AND, NAND, OR, XOR
  DDRF = B11011011;
  DDRK = B0110;
  for (int i = 0; i < 2; i++) { /////Combinación 0 0
    digitalWrite(A0 + i,   y[0][i]);
    digitalWrite(A3 + i,   y[0][i]);
    digitalWrite(A6 + i,   y[0][i]);
    digitalWrite(A9 + i,   y[0][i]);
  }
```

```
  out1 = digitalRead(A2) + digitalRead(A5) +  digitalRead(A8) + digitalRead(A11);
  ////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinación 0 1 = 1 0
    digitalWrite(A0 + i,   y[1][i]);
    digitalWrite(A3 + i,   y[1][i]);
    digitalWrite(A6 + i,   y[1][i]);
    digitalWrite(A9 + i,   y[1][i]);
  }
  out2 = digitalRead(A2) + digitalRead(A5) +  digitalRead(A8) + digitalRead(A11);
  ////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinación  1 1
    digitalWrite(A0 + i,   y[2][i]);
    digitalWrite(A3 + i,   y[2][i]);
    digitalWrite(A6 + i,   y[2][i]);
    digitalWrite(A9 + i,   y[2][i]);
  }
  out3 = digitalRead(A2) + digitalRead(A5) +  digitalRead(A8) + digitalRead(A11);
  delay(100);
}

void dos() { ///// Compuertas NOT

  DDRF = B01010101;
  DDRK = B0101;
```

```
  for (int i = 0; i < 11; i = i + 2) { /////Combinación 0
    digitalWrite(A0 + i, LOW);
  }
  out4 = digitalRead(A1) + digitalRead(A3) +  digitalRead(A5) + digitalRead(A7) + digitalRead(A9) + digitalRead(A11);
  ////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 11; i = i + 2) { /////Combinación 1
    digitalWrite(A0 + i, HIGH);
  }
  out5 = digitalRead(A1) + digitalRead(A3) +  digitalRead(A5) + digitalRead(A7) + digitalRead(A9) + digitalRead(A11);
  delay(100);
}

void tres() { ///// Compuertas NOR

  DDRF = B10110110;
  DDRK = B1101;

  for (int i = 0; i < 2; i++) { /////Combinación 0 0
    digitalWrite(A1 + i,   y[0][i]);
    digitalWrite(A4 + i,   y[0][i]);
    digitalWrite(A7 + i,   y[0][i]);
    digitalWrite(A10 + i,  y[0][i]);
  }
  out6 = digitalRead(A0) + digitalRead(A3) +  digitalRead(A6) + digitalRead(A9);
```

```
  ////////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinación 0 1 = 1 0
    digitalWrite(A1 + i,   y[1][i]);
    digitalWrite(A4 + i,   y[1][i]);
    digitalWrite(A7 + i,   y[1][i]);
    digitalWrite(A10 + i,  y[1][i]);
  }
  out7 = digitalRead(A0) + digitalRead(A3) +  digitalRead(A6) + digitalRead(A9);
  ////////////////////////////////////////////////////////////////////////////////////////////////////
  for (int i = 0; i < 2; i++) { /////Combinación 1 1
    digitalWrite(A1 + i,   y[2][i]);
    digitalWrite(A4 + i,   y[2][i]);
    digitalWrite(A7 + i,   y[2][i]);
    digitalWrite(A10 + i,  y[2][i]);
  }
  out8 = digitalRead(A0) + digitalRead(A3) +  digitalRead(A6) + digitalRead(A9);
  delay(100);
}

void coneccionwf() {

  delay(100);
  digitalWrite(wf, HIGH);
  if (Serial1.available() > 0) {
    while (Serial1.available() > 0) {
```

```
datos = Serial1.readString();
a = s.separa(datos, ',', 0); //"Bandera"
b = s.separa(datos, ',', 1); // ssid / nombre
c = s.separa(datos, ',', 2); // pass / apellido
d = s.separa(datos, ',', 3); // ip / ci

datos = "";

b.toCharArray(chb, 10);
c.toCharArray(chc, 10);
d.toCharArray(chd, 14);
tb.setText(chb);
tc.setText(chc);
td.setText(chd);

if ( a == "datos") {
  if(b =="" || c =="" || d ==""){
    err.setText("ERROR Datos vacíos");
    return;
  }
  else
  err.setText("");
  delay(3000);
  tb.setText("Módulo");
  tc.setText("Configurado");
```

```
        td.setText("Bienvenido");
        delay(2000);
        page1.show();
        tnomb.setText(chb);
        Serial1.end();
        digitalWrite(wf, LOW);
      }
    }
  }
}
```

6.4 Anexo D Código Módulo esp8266

Formulario

```
#include <ESP8266WiFi.h>                  //Incluir librería ESP8266WiFi.h

#include <ESPAsyncTCP.h>                  //ESPAsyncTCP.h

#include <ESPAsyncWebServer.h>            //Incluir ESPAsyncWebServer.h

#include <FS.h>                           //Incluir librería FS.h


#include "config.h"                       //Incluir archivo config.h

#include "Server.hpp"                      //Incluir archivo server,hpp

void setup(void)

{

  delay(4000);                            //Retraso de 4 segundos

  Serial.begin(9600);                     //Iniciar el puerto serial en 9600 baudios

  SPIFFS.begin();                          //Inicia SPIFFS

  Conf_AP();                              //Configuración AP

  InitServer();                           //

}

void loop(void)

{

}
```

Server.hpp

```
AsyncWebServer server(80);        //Asignación puerto 80


void handleFormText(AsyncWebServerRequest *request)

{
```

```
  String nomb = request->arg("nomb"); //Variable string nomb recibe información de nomb
  String apel = request->arg("apel"); //Variable apel nomb recibe información de apel
  String ci = request->arg("ci");     //Variable ci nomb recibe información de ci

  Serial.print("User,");              //Envía User, por el puerto serial
  Serial.print(nomb + ",");           //Envía variable nomb, por el puerto serial
  Serial.print(apel + ",");           //Envía variable apel, por el puerto serial
  Serial.print(ci);                   //Envía variable ci, por el puerto serial

  if (nomb == "" || apel == "" || ci == "") { //Sentencia if para datos vacíos
    request->redirect("/");
  }
  else if((nomb == "admin" || apel == "admin" || ci == "1726018136")) { //Sentencia if para identificar los datos de administración
    request->redirect("/devs.html");          //Carga la página devs.tml
  }
  else {
    WiFi.mode(WIFI_OFF); //apaga WiFi después de enviar todos los datos
  }
}
void InitServer()
{
  server.serveStatic("/", SPIFFS, "/").setDefaultFile("index.html");
  server.on("/SetText", HTTP_POST, handleFormText);
  server.on("/devs", HTTP_POST, handleFormText);
```

```
  server.onNotFound([](AsyncWebServerRequest * request) {
    request->send(404, "text/plain", "Página no encontrada");
  });
  server.begin();
```

Config.h

```
char* ssid     = "Modulo12";  // Nombre de la red
char* password = "Modulo12";  // Contraseña de la red
char* hostname = "Modulo12";  // Nombre del host

IPAddress ip(192, 168, 1, 1); //variables del formato IP
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

void Conf_AP()
{
  WiFi.mode(WIFI_AP);   //Configura el módulo esp8266 como AP
  WiFi.softAPConfig(ip, gateway, subnet);// configura las direcciones IP del módulo
  WiFi.softAP(ssid, password, 13); // (ssid, password, channel, hidden, max_connection)
  Serial.print(",");  //separador
  Serial.print(ssid); //nombre de la red
  Serial.print(",");  //separador
  Serial.print(password); //contraseña
  Serial.print(",");    //separador
  Serial.print(WiFi.softAPIP());//muestra la dirección IP a conectarse
}
```