UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Tecnología Electrónica

Área de Conocimiento de Tecnología Electrónica

TRABAJO FIN DE GRADO

Desarrollo de un tablero de instrumentos digital inalámbrico para automoción

Grado en Ingeniería Electrónica Industrial

Autor: Juan Molina Moreno

Tutor: Gabriel Valencia Miranda

MÁLAGA, 4 de septiembre de 2.023

RESUMEN

En el siguiente trabajo, se presenta en su totalidad el proceso de desarrollo de un tablero de instrumentos digital e inalámbrico para automoción, abarcando tanto su fundamentación teórica como la ejecución práctica.

Este proyecto se enfoca en el diseño e implementación de un sistema que permite al usuario final la visualización de datos del vehículo, provenientes de la centralita, y que se muestran a través de una pantalla táctil, haciendo que el usuario pueda interactuar con el sistema.

Este sistema se realiza a partir de una herramienta de diagnóstico como el dispositivo ELM327, una placa Arduino MEGA 2560 con un módulo bluetooth como es el HC-05 y una pantalla táctil LCD.

En primer lugar, se detallan los fundamentos teóricos en los que se basa la adquisición de datos de la centralita (ECU) y la comunicación serie a través de la tecnología bluetooth.

Posteriormente, se comentan tanto el hardware como el software utilizado, incluyendo la programación desarrollada en la placa Arduino Mega 2560, necesaria para que se lleve a cabo la petición y adquisición de los datos, su identificación y almacenamiento, para posteriormente ser enviados a la pantalla.

Como parte final, se presentan las conclusiones derivadas tras la completa implementación del sistema en su forma física, junto con un análisis de los resultados obtenidos. Este apartado representa el cierre del proceso ofreciendo una visión integral de la ejecución.

Palabras clave: diagnóstico, vehículo, adquisición de datos, monitorización, bluetooth,

ABSTRACT

In the following work, the development process of a digital and wireless dashboard for automotive is presented in its entirety, covering both its theoretical foundation and practical implementation.

This project focuses on the design and implementation of a system that allows the end user to visualize vehicle data, coming from the ECU, and displayed through a touch screen, allowing the user to interact with the system.

This system is made from a diagnostic tool such as the ELM327 device, an Arduino MEGA 2560 board with a bluetooth module such as the HC-05 and an LCD touch screen.

First of all, the theoretical foundations on which the data acquisition from the ECU and the serial communication through bluetooth technology are based are detailed.

Subsequently, both the hardware and the software used are discussed, including the programming developed on the Arduino Mega 2560 board, necessary for the request and acquisition of data, its identification and storage, to be subsequently sent to the display.

As a final part, the conclusions derived after the complete implementation of the system in its physical form are presented, together with an analysis of the results obtained. This section represents the closure of the process offering a comprehensive view of the implementation.

ÍNDICE GENERAL

Tabla de contenido

ÍNDICE DE FIGURAS

Capítulo 1. Introducción

Los vehículos comerciales han evolucionado a lo largo del tiempo de manera exponencial, pasando de ser un simple medio de transporte a ser complejas máquinas que, actualmente, han desembocado en un sistema de máquinas interconectadas entre sí y que gracias a la electrónica se logra un funcionamiento integrado y coordinado, con el fin de crear vehículos más eficientes aumentando a su vez el rendimiento del motor y ofrecer una experiencia de conducción mucho más confortable y segura.

Actualmente, los avances tecnológicos en los vehículos modernos provocan que estos tengan que generar y procesar simultáneamente una ingente cantidad de datos en tiempo real. Todos estos datos abarcan desde datos de control o diagnóstico hasta una amplia gama de aspectos como pueden ser sistemas de asistencia a la conducción, navegación, frenado, dirección, seguridad y entretenimiento. Una gran parte de estos datos se utilizan para establecer una conexión comunicativa entre el usuario y el vehículo a través del tablero de instrumentos.

Objetivos

En los vehículos de competición, se requiere una mayor cantidad de información en el tablero de instrumentos en comparación con los vehículos comerciales, en los que los valores más relevantes son: la velocidad, la revolución, nivel de combustible, temperatura del motor. En el caso de los vehículos de competición se deben tener en cuenta muchos más aspectos como pueden ser la temperatura y la presión de aceite, presión de combustible y posición de pedal del acelerador.

Al considerar la transformación de un vehículo de serie en un vehículo de competición o destinado a correr en circuito de manera amateur, surge la necesidad de acceder a dichos parámetros sin requerir a la instalación de dispositivos cableados.

Por ello, como objetivo principal se tiene el desarrollo e implementación de un tablero de instrumentos digital inalámbrico basado en la integración del dispositivo de diagnóstico ELM327, proporcionando al usuario una herramienta eficiente y versátil para monitorear en tiempo real los parámetros clave de su vehículo.

A continuación, se desarrollan los distintos objetivos específicos necesarios:

Diseño del sistema

Para la realización del diseño del sistema, primero se llevará a cabo un estudio o análisis de los requisitos funcionales y estéticos que necesita un tablero de instrumentos digital. Una se tiene esto, es necesario definir la arquitectura de dicho sistema y como interaccionan las distintas partes que lo componen, como son el ELM327, el Arduino y la pantalla LCD.

Conexión y comunicación

Un punto clave en el desarrollo del sistema es establecer la conexión Bluetooth entre el dispositivo ELM327 y el Arduino utilizando para ello un módulo Bluetooth conectado a uno de los puertos serie del Arduino, en este caso el módulo bluetooth es un HC-05.

Captura y procesamiento de datos

Implementar el código necesario para solicitar e inmediatamente capturar los datos en tiempo real enviados por el ELM327, desarrollando también las secciones del código necesarias para almacenar dichos datos en las variables que correspondan, facilitando así su identificación y poder mostrar al usuario una información coherente.

Interfaz de usuario

Diseñar una interfaz de usuario intuitiva que muestre los datos procesados en la pantalla. Teniendo en cuenta la disposición de los distintos elementos que la componen, con la intención de proporcionar de forma clara y comprensible la información al usuario en el desempeño de la conducción.

Personalización

Incorporar alternativas de personalización, dando la capacidad de decidir al usuario entre recibir datos básicos del vehículo o, en cambio, recibir una mayor cantidad de información más detallada y específica sobre el vehículo.

Descripción del sistema implementado

El sistema implementado se compone principalmente de 3 partes, siendo la primera de ellas el dispositivo de diagnóstico ELM327. Por otro lado, se tiene el Arduino Mega 2560, con un módulo HC-05 que le permite la comunicación bluetooth y, por último, la pantalla táctil LCD Nextion, que tendrá la función de interfaz de usuario, mostrando los datos en tiempo real enviados por el ELM327.

La interfaz de usuario permite al usuario seleccionar la manera en la que se muestran los datos a través de la pantalla, pudiendo elegir entre 3 modos distintos de funcionamiento. Los modos de visualización o tipo de tableros son los siguientes: clásico, deportivo y Racing.

Tablero clásico: muestra ECT(Engine Coolant Temperature),RPM (Revoluciones Por Minuto), velocidad del vehículo, AAT (Ambient Air Temperature) y nivel de combustible.

Este tipo de tablero de instrumentos muestra la información en tiempo real de la manera que la suele ofrecer un vehículo de serie. A través de dos diales de aguja muestra, por un lado, las revoluciones por minuto a las que se encuentra el motor y, por otro lado, la velocidad del vehículo. También permite saber tanto la temperatura del aire ambiente como la temperatura del refrigerante del vehículo. Por último, muestra la capacidad a la que se encuentra el bidón de combustible.

Tablero deportivo: muestra ECT(Engine Coolant Temperature),RPM (Revoluciones Por Minuto), velocidad del vehículo, MAF (Mass Air Flow) y nivel de combustible.

El tablero deportivo, en comparación con el tablero clásico, desde el punto de vista de la información que proporciona, solo varía en un aspecto y, es que, en vez de mostrar la temperatura del aire ambiente, se muestra el flujo de masa de aire. Donde realmente se encuentra la diferencia es en la manera en la que se muestra la información, ya que las revoluciones en este tablero se muestran en una barra de progreso, al igual que la masa de flujo de aire y, los demás datos como son la temperatura del refrigerante, la velocidad del vehículo y el flujo de masa de aire se representan de forma numérica.

Tablero racing: muestra la misma información que el tablero deportivo añadiendo la presión barométrica absoluta.

La gran diferencia se hace notoria en cuanto a como se muestra la información, toda la información se muestra de manera numérica, excepto las revoluciones del motor que se muestran de dos maneras, de forma numérica y a través de un indicador de aguja.

Una vez el usuario haya iniciado el sistema mediante el botón correspondiente, se mostrará por defecto el modo clásico, pudiendo el usuario, a través de la pantalla táctil acceder a un menú en el que se puede seleccionar el tipo de tablero que desee.

En la *figura 1 se muestra un diagrama tipo grafcet que describe el comportamiento del sistema.*

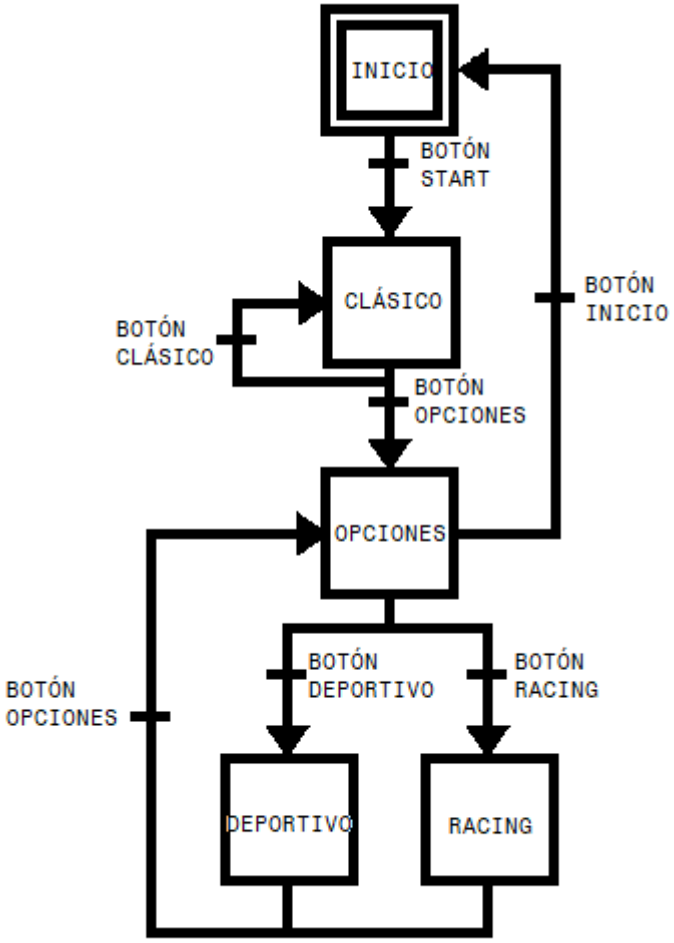


Figura 1. Diagrama tipo grafcet del sistema.

Fundamentos teóricos

Evolución de la electrónica del automóvil

Con el transcurrir de los años, la electrónica ha experimentado una notable evolución en la industria de la automoción, reconfigurando la forma en que los vehículos interactúan con el entorno y la manera en que las personas nos relacionamos con ellos. Esta evolución de la electrónica del automóvil puede resumirse en las siguientes etapas:

Décadas de 1960 y 1970 – Primeros indicios

En esta época, la electrónica en los automóviles tenía un uso muy básico que se limitaba al sistema de encendido y de luces. La electrónica aún no tenía un papel dominante en el funcionamiento de los vehículos.

Década de 1980 – Control de emisiones y computadoras a Bordo

Una vez ya en los años 80, se introducen en los vehículos las primeras ECUs, con el fin de gestionar y controlar las emisiones de los sistemas de escape. Las ECUs eran capaces de ajustar parámetros del motor en tiempo real, mejorando el rendimiento y la eficiencia del vehículo.

Década de 1990 - Electrónica avanzada y sistemas de seguridad

La década de los 90 trajo un aumento muy significativo en cuanto a la electrónica avanzada, introduciendo en los vehículos diferentes sistemas de seguridad. Ejemplos de estos sistemas son el sistema de frenos antibloqueo (ABS) y el sistema de control de tracción (ESP). Además de estos sistemas, también se implementaron sistemas de diagnóstico como el OBD-II (On-Board Diagnostics) el cual permite detectar y registrar fallos tanto del motor como de otros sistemas que componen el vehículo.

Década de 2000 – Conectividad y comunicación

Una vez entrado el siglo XXI, la conectividad con el vehículo pasó a ser algo muy importante. Los sistemas de navegación GPS, bluetooth y la integración de dispositivos móviles pasaron de ser algo exclusivo para transformarse en elementos habituales en los vehículos.

Década de 2010 y actualidad – vehículos autónomos y electrificación

En la última década, debido a los problemas de contaminación en grandes ciudades, se ha logrado un avance notorio en cuanto a los vehículos de propulsión eléctrica y los híbridos. Igualmente, los vehículos autónomos han experimentado un significativo desarrollo, gracias a los avances en la tecnología de sensores y en la capacidad de toma de decisiones en tiempo real por parte de la electrónica del automóvil.

En resumen, la electrónica ha evolucionado desde lo más básico, hasta convertirse a día de hoy en una red compleja de sensores, actuadores y sistemas de gestión, reconfigurando por completo la experiencia de la conducción.

Introducción a los sistemas OBD

El sistema OBD (On-Board Diagnostics) tiene sus raíces en una necesidad: la regulación de la contaminación causada por los gases de escape de los vehículos de combustión.

En la década de los 60 y 70, a medida que la industria automotriz emergía y los vehículos empezaban a ser algo cotidiano e imprescindible, aumentaba la preocupación por el efecto perjudicial de los gases de escape en el ámbito de la salud pública.

Fue por este motivo que, la Agencia de Protección Ambiental de los Estados Unidos (EPA), implantó regulaciones más estrictas con el objetivo de poder controlar las emisiones de los vehículos. En el año 1988, la EPA exigió a los fabricantes de automóviles que equiparan sus vehículos con sistemas de diagnóstico a bordo con la finalidad de poder identificar anomalías en los sistemas encargados de controlar las emisiones. Esta medida estableció los cimientos del sistema OBD que se conoce en la actualidad.

Conforme se van incorporando con el tiempo sensores y sistemas electrónicos de control en los vehículos, se van transformando en máquinas que son capaces de autorregularse y autoevaluarse. Que los vehículos puedan autoevaluarse y puedan identificar de donde proviene al fallo permite a los técnicos dar un servicio más rápido, sencillo y preciso.

En conclusión, el sistema OBD nació como una respuesta ante la preocupación por la contaminación causada por los gases de escape de los motores de combustión, teniendo una evolución que ha desembocado en la estandarización del sistema OBD-II (OBD de segunda generación) que proporciona una manera de poder diagnosticar vehículos de manera sencilla y concreta.

Sistemas OBD-II

El sistema OBD-II, que hace referencia a la segunda generación, es una estandarización universal del sistema OBD y que es fundamental en el diagnóstico a

bordo de los vehículos para la detección de problemas en los distintos sistemas que componen al vehículo.

Este sistema de diagnóstico es estándar, y para ello también lo es el conector o puerto de comunicación. El conector estándar de la segunda generación de OBD es el conector J1962 o DLC, que cuenta con 16 pines, dispuestos en dos filas de ocho. En la *figura 2,* se puede ver la disposición de los pines y la finalidad de cada uno de ellos.

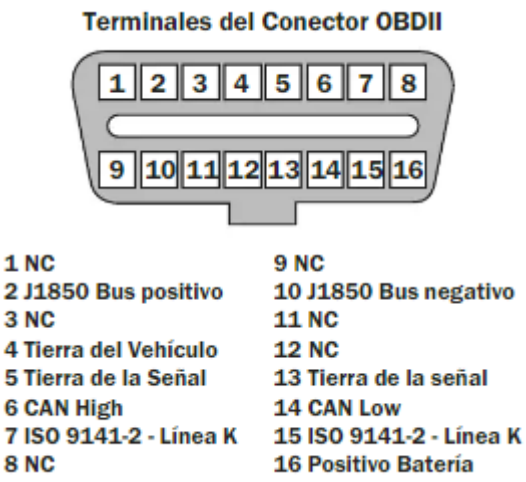


Figura 2. Terminales del conector DLC (J1962).

Fuente [11].

Las localizaciones habituales de este conector en los vehículos suelen ser: en la parte de debajo del volante, en la parte inferior de la consola central o en algunos casos en la parte de la consola que se encuentra detrás del freno de mano.

Por otro lado, el sistema OBD opera mediante “modos”, cada uno de los diferentes modos otorgan al usuario diagnósticos específicos en diferentes aspectos del vehículo. A continuación, se muestran los diferentes modos que existen, así como una breve explicación de su función. [11]

MODO 01 – Datos en tiempo real:

Permite el acceso a ciertos datos del vehículo en tiempo real, brindando información útil para el diagnóstico y el monitoreo del rendimiento del vehículo.

MODO 02 – Acceso a datos congelados:

Este modo captura y almacena ciertos datos en el momento en el que se detecta una falla en el vehículo, pudiendo ver las condiciones en las que se encontraba el vehículo en dicho instante.

MODO 03 – Códigos DTCs pendientes almacenados:

Los códigos pendientes son aquellos códigos que indican problemas, pero aún no han llegado a valores que hagan activar el testigo de fallo en el tablero. Esto posibilita que se pueda abordar el problema de manera preventiva, antes de que el fallo pase a mayores.

MODO 04 – Borrado de DTCs y datos congelados:

Con este modo, después de haber reparado cualquier anomalía en el vehículo que haya activado alguno de los testigos de fallo, se puede reiniciar la memoria haciendo que, si todo está correcto, se apague el testigo causante del fallo.

MODO 05 – Resultado del test sensores de oxígeno:

Proporciona información en cuanto a los resultados de las pruebas de oxígeno que se encuentra en los gases de escape, con la finalidad de evaluar el rendimiento de la combustión y el funcionamiento del catalizador en la reducción de emisiones contaminantes.

MODO 06 – Resultado de pruebas, supervisión de otros componentes/sistemas:

Arroja información detallada de los resultados de las pruebas de diagnósticos de sistemas y componentes específicos, ayudando a identificar incidencias potenciales que no activan ninguno de los testigos de advertencia del tablero de instrumentos.

MODO 07 – Códigos de diagnóstico de fallo pendientes (último ciclo de conducción):

Muestra los resultados de los fallos o DTCs detectados en el último ciclo de conducción. Registra los DTCs solo del último ciclo de conducción, pudiendo observar si en el ciclo de conducción anterior se encontraba el fallo o no.

MODO 08 – Control del funcionamiento del componente/sistema de a bordo:

Se trata de un modo que permite realizar pruebas en tiempo específico y ajustes, haciendo que sea un modo muy útil para los técnicos del sector pudiendo calibrar sensores, controlar actuadores y activar o desactivar alguno de los componentes como pueden ser la bomba de combustible, la EGR (Exhaust Gas Recirculation) o la válvula de ralentí.

MODO 09 – Solicitar información sobre el vehículo:

Ofrece información detallada del vehículo como datos de identificación y configuración. Estos datos pueden ser de utilidad para identificar de forma precisa el vehículo y obtener los requisitos de mantenimiento apropiados.

MODO 0A – Códigos de diagnóstico de problemas permanentes (DTC borrados):

Son códigos de fallo que, aunque en el momento el testigo de fallo que le corresponde no se encuentre activado, han tenido un valor importante y han provocado el encendido MIL en algún momento, quedando registrado y pudiendo verlo a través de este modo.

En cuanto al presente proyecto, el modo de operación que se utiliza es el modo 01, ya que este modo permite mostrar datos del vehículo en tiempo real. Los datos más comunes que se pueden mostrar se muestran en la *tabla 1*, que van desde el 00 al 3F.

Tabla 1. Algunos códigos PID del modo 01. Fuente [5].

PID (HEX)	Descripción	PID (HEX)	Descripción	PID (HEX)	Descripción
00	PIDs admitidos (00 a 20)	16	Sensor de oxígeno 3	2D	Falla EGR
01	Estado monitores diagnóstico, desde borrado DTCs	17	Sensor de oxígeno 4	2E	Purga evaporativa comandada
02	Almacena códigos DTCs de un evento	18	Sensor de oxígeno 5	2F	Nivel del tanque de combustible
03	Estado del sistema de combustible	19	Sensor de oxígeno 6	30	Cantidad de calentamientos desde que se borraron fallas

04	Carga del motor	1A	Sensor de oxígeno 7	31	Distancia recorrida desde que se borraron las fallas
05	Temperatura del refrigerante	1B	Sensor de oxígeno 8	32	Presión de vapor del sistema evaporativo
06	Ajuste combustible-corto plazo- banco 1	1C	Estándar OBD del vehículo	33	Presión barométrica absoluta
07	Ajuste combustible-largo plazo-banco 1	1D	Sensores de oxígeno banco 4	34	Sensor de oxígeno 1
08	Ajuste combustible-corto plazo-banco 2	1E	Estado de las entradas auxiliares	35	Sensor de oxígeno 2
09	Ajuste combustible-largo plazo-banco 2	1F	Tiempo desde que se puso en marcha el motor	36	Sensor de oxígeno 3
0A	Presión del combustible	20	PIDs admitidos (21-40)	37	Sensor de oxígeno 4
0B	Presión absoluta del colector de admisión	21	Distancia recorrida con luz MIL encendida	38	Sensor de oxígeno 5
0C	RPM del motor	22	Presión del tren de combustible, relativa al colector de vacío	39	Sensor de oxígeno 6
0D	Velocidad del vehículo	23	Presión del medidor del tren de combustible	3A	Sensor de oxígeno 7
0E	Avance del tiempo	24	Sensor de oxígeno 1	3B	Sensor de oxígeno 8
0F	Temperatura aire del colector de admisión	25	Sensor de oxígeno 2	3C	Temperatura catalizador-banco 1- sensor 1
10	Velocidad del flujo de aire (MAF)	26	Sensor de oxígeno 3	3D	Temperatura catalizador-banco 1- sensor 2
11	Posición del acelerador	27	Sensor de oxígeno 4	3E	Temperatura catalizador-banco 2- sensor 1
12	Estado del aire secundario	28	Sensor de oxígeno 5	3F	Temperatura catalizador-banco 2- sensor 2
13	Presencia de sensores de oxígeno	29	Sensor de oxígeno 6		
14	Sensor de oxígeno 1	2B	Sensor de oxígeno 8		
15	Sensor de oxígeno 2	2C	EGR comandado		

Para este proyecto no se utilizan todos estos datos, en la *tabla 2* se muestran los datos que sí se han utilizado, mostrando sus rangos de valores, la fórmula que se debe aplicar para obtener el valor correcto y su correspondiente unidad.

Tabla 2. Códigos PID utilizados en el proyecto.

PID	Descripción	Valor min	Valor max	Fórmula	Unidad
05	Temperatura del refrigerante	-40	215	A - 40	ºC
0C	RPM del motor	0	16.383,75	(256A+B)/4	rpm
0D	Velocidad del vehículo	0	255	A	km/h
10	Velocidad del flujo de aire (MAF)	0	655,35	(256A+B)/100	g/s
2F	Nivel del tanque de combustible	0	100	A*255/100	%
33	Presión barométrica absoluta	0	255	A	kPa
46	Temperatura aire del ambiente	-40	215	A-40	ºC

Protocolos OBD-II

El sistema de OBD-II consta de numerosos protocolos para asegurar una correcta comunicación entre el vehículo y la herramienta de diagnóstico. Estos protocolos varían según la región y el fabricante. Dichos protocolos son los citados a continuación:

SAE J1850 PWM (41.6 Kbaud)

SAE J1850 PWM (41.6 Kbaud)

ISO 9141-2 (5 baud inicio, 10.4 Kbaud)

ISO 14230-4 KWPP (5 baud inicio, 10.4 Kbaud)

ISO 14230-4 KWPP (inicio rápido, 10.4 Kbaud)

ISO 15764-4 CAN (11 bit ID, 500 Kbaud)

ISO 15764-4 CAN (29 bit ID, 500 Kbaud)

ISO 15764-4 CAN (11 bit ID, 250 Kbaud)

ISO 15764-4 CAN (29 bit ID, 250 Kbaud)

SAE J1939 CAN (29 biit ID, 250 Kbaud)

Los 10 protocolos citados anteriormente, son variaciones de 3 tipos de protocolos, PWM, KWPP y CAN, los cuales se explican de forma breve a continuación.

El protocolo PWM (Pulse Width Modulation) se basa en una técnica llamada modulación por ancho de pulso, que consiste en modular el ciclo de trabajo de la señal de onda cuadrada para transmitir datos desde la ECU del vehículo y la herramienta de diagnóstico. Según el ancho de pulso de la señal, se representa un valor u otro.

Este protocolo ya no es muy común, debido a que la tasa de transmisión es más baja en comparación a otros protocolos más modernos.

El protocolo KWPP (Key Word Protocol) se basa en una comunicación serie donde la herramienta de diagnóstico envía solicitudes y recibe respuestas desde la ECU del vehículo. La velocidad de transmisión depende del fabricante o modelo del vehículo.

Este protocolo ha sido reemplazado de manera gradual por el protocolo CAN, que es el protocolo que puede verse en los vehículos más modernos, debido a una mayor velocidad de transmisión.

El protocolo CAN (Controller Area Network) es uno de los más utilizados en los vehículos modernos. Este protocolo se basa en la comunicación de dos hilos, el CAN-H (CAN - HIGH) y CAN-L (CAN - LOW), permitiendo una comunicación robusta y fiable aún en entornos ruidosos.

Este protocolo cuenta con detección de colisiones, e incluso permite una arquitectura de red, pudiendo múltiples redes CAN en un mismo vehículo, conectando diferentes sistemas y módulos, e incluso pueden ser líneas CAN de alta velocidad (HS-CAN) o de baja velocidad (LS-CAN).

Debido a su fiabilidad y a su alta velocidad de comunicación, el protocolo CAN es el protocolo por excelencia en los vehículos modernos.

Comunicación serie Bluetooth

El bluetooth es una tecnología de comunicación serie inalámbrica de corto alcance. Desde su creación hasta el día de hoy, el bluetooth se ha ido incorporando en nuestras vidas, desempeñando un papel crucial en cuanto a conectividad.

Tal es así, que se encuentra en numerosos dispositivos cotidianos, estando muy presente en dispositivos como auriculares, relojes, altavoces,

teclados, ratones, smartphones e incluso se encuentra muy presente en los vehículos, permitiendo enlazar los teléfonos móviles con los equipos multimedia.

Los inicios del bluetooth se remontan a los años 1990, cuando la compañía Ericsson buscaba una manera de reemplazar los cables que conectaban dispositivos móviles. [8]

Su funcionamiento se basa en una red de dispositivos maestros y esclavos, los cuales se conectan entre mediante conexiones punto a punto o punto a multipunto.

Los dispositivos maestros pueden comunicarse con varios dispositivos esclavos siempre que se encuentren en su rango de alcance. Antes de establecer la comunicación, se ha de iniciar un proceso de emparejamiento entre los dispositivos a conectar, intercambiando entre ellos claves de seguridad garantizando la privacidad de la transmisión.

La tecnología bluetooth, con el fin de proporcionar robustez en la comunicación y evitar interferencias en entornos ruidosos, utiliza la técnica de salto de frecuencia, en la cual, los dispositivos cambian de frecuencia de forma muy rápida, dentro del espectro permitidos, haciendo que las interferencias no afecten a la comunicación.

En conclusión, el bluetooth es una pieza fundamental en la comunicación inalámbrica contemporánea. Su evolución desde sus inicios hasta su adopción en multitud de dispositivos demuestra la importancia en la conectividad digital de la actualidad.

Capítulo 2. Hardware empleado

En el siguiente capítulo se detalla la parte de hardware que se ha utilizado para llevar a cabo el desarrollo y la implementación del tablero de instrumentos digital e inalámbrico.

Selección de componentes

En este apartado se citan y se explican los componentes escogidos, las características funcionales de estos y el motivo de su elección para el presente proyecto.

Microcontrolador

El microcontrolador es una parte esencial en el proyecto, siendo el encargado de interpretar, almacenar los datos obtenidos y paralelamente enviarlos para su posterior monitoreo. Este dispositivo debe cumplir las siguientes funciones:

- Realizar la petición de datos al dispositivo ELM327, a través de una comunicación serie mediante un puerto serie que lo enlace al módulo bluetooth.
- Recepcionar los datos solicitados a través de la comunicación serie anteriormente mencionada.
- Almacenar los datos recibidos y extraer la información útil de estos.
- Enviar dicha información útil a la pantalla mediante una comunicación serie.
- Responder a la información que envía la pantalla a través del puerto serie.
-

Para realizar las funciones que se han citado anteriormente, se ha decidido usar el microcontrolador Arduino MEGA2560 (*figura 3*), la cual está basada en el microcontrolador ATMEGA 2560, permitiendo la interacción con múltiples dispositivos externos.

En este caso, se necesita de 2 puertos de comunicación serie físicos, uno para la comunicación serie con el dispositivo de diagnóstico y el otro para la comunicación serie con la pantalla LCD.

Figura 3. Arduino Mega 2560 versión DIP.

Las características de la placa Arduino MEGA2560 [2] son las siguientes:

- Microcontrolador: ATmega2560.
- Voltaje de entrada recomendado: 7-12 V, voltaje de operación 5 V.
- Voltajes de entrada mínimo y máximo: 6-20 V.
- Dispone de 54 pines digitales, 16 pines analógicos, 4 puertos seriales (UART) y 2 puertos I2C (SDA y SCL).
- 256 kB de memoria flash.
- 8kB de memoria SRAM.
- 4Kb de memoria EEPROM.
- Frecuencia de funcionamiento: 16 MHz.
- Dimensiones de 101.52 mm x 53.3 mm y 37 gramos de peso.

Módulo bluetooth

Para establecer la comunicación inalámbrica se necesita de un módulo bluetooth, en este caso se ha decidido escoger el módulo HC-05 (*figura 4*). La razón de escoger este es debido a que no solo se reciben datos a través de dicho módulo, por lo que se tiene la necesidad de escoger un módulo bluetooth que permita configurarlo como maestro.

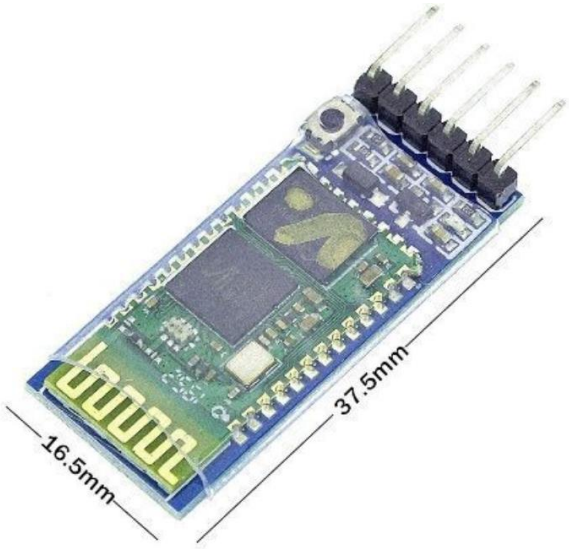


Figura 4. Módulo bluetooth HC-05.

La función del módulo HC-05 en este proyecto es la de establecer una comunicación inalámbrica entre el dispositivo de diagnóstico y la placa Arduino Mega2560.

Las características funcionales principales del HC-05 [3] son:

- Voltaje de operación: 3.6 V - 6 V.
- Consumo corriente: 50 mA.
- Frecuencia: banda ISM 2.4 GHz.
- Modulación GFSK.
- Potencia de transmisión: 4 dBm, clase 2.
- Sensibilidad: -84 dBm
- Alcance: 10 metros.
- Velocidad de transmisión: 1200 bps hasta 1.3 Mbps.
- Baudrate por defecto: 38400,8,1,n.
- Dimensiones de 37 x 16 mm y 3,6 gramos de peso.

Elm327

La elección del dispositivo de diagnóstico ELM327 (*figura 5*) para el presente proyecto se debe a dos razones de importancia, la primera es que se necesita un dispositivo OBD-II que permita una conexión inalámbrica por bluetooth y, la segunda, es su bajo coste frente a sus características y versatilidad, ofreciendo una base sólida y funcional que está ampliamente disponible y es asequible en el mercado.

Figura 5. Herramienta de diagnóstico bluetooth ELM327.

Las especificaciones técnicas de este dispositivo [4] son:

- Voltaje de operación: 12 V a través del puerto OBD-II.
- Versión del firmware: 2.1.
- Bluetooth 2.0.
- Frecuencia: banda ISM de 2,4 GHz.
- Velocidad de comunicación: desde 9600 bps hasta 115200 bps.
- Protocolos compatibles: ISO 155765-4 (CAN), ISO9141-2, J1850 VPW, J1850 PWM.

Pantalla táctil LCD

La parte visible del proyecto es la interfaz de usuario, haciendo que sea una parte importante. Para ello, se ha decidido usar una pantalla LCD táctil de tipo capacitivo de la marca Nextion, concretamente el modelo NX8048P070-011C (*figura 6*). La razón principal por la cual se ha escogido esta pantalla es debido a que la marca Nextion tiene su propio software de edición (Nextion Editor) que permite la personalización total de la interfaz, pudiendo adaptar la apariencia y la disposición de los elementos según las necesidades del usuario.

Además, hay que incluir que la pantalla es una pantalla táctil de tipo capacitivo, dando la posibilidad de que el usuario pueda interactuar con el sistema de una manera mucho más amplia, y abriendo un mundo de posibilidades a la hora de crear la interfaz. En resumen, la pantalla ofrece una combinación de facilidad de uso, personalización, eficiencia y funcionalidad táctil haciendo que sea idónea para el proyecto.

Figura 6. Pantalla táctil LCD Nextion NX8048T070.

Las características técnicas [6] más importantes de la pantalla:

- Resolución: 800 x 480 pixel, 7 pulgadas.
- Pantalla táctil de tipo capacitivo.
- Baudrate típico: 9600 bps.
- Baudrate (límites): 2400 – 921600 bps.
- Voltaje de operación: 4.75 V – 6.5 V.
- Voltaje de alimentación recomendada: 5 V.
- Memoria flash de 120 MB.
- Memoria EEPROM 1024 B.
- Memoria RAM 512 kB.

En la *figura 7,* se muestran algunas de las partes y componentes más relevantes que se encuentran en el reverso de la pantalla.

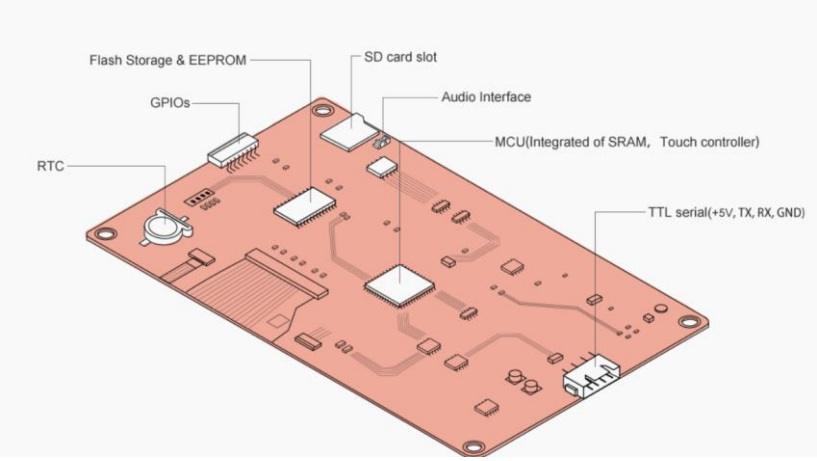


Figura 7. Componentes relevantes en la cara posterior de la pantalla Nextion.

Alimentación del sistema.

La forma de alimentar el sistema es a través del cable de la placa Arduino, que va de USB tipo B a USB tipo A, como se puede ver en la *figura 8.*

Figura 8. Cable USB tipo A-B.

Este cable va conectado a la toma de mechero del vehículo mediante un adaptador para USB tipo A, pudiendo conectar el cable a la toma de 12 V del vehículo. En la *figura 9* se muestra el adaptador.

Figura 9. Adaptador de mechero para USB tipo B.

Este adaptador permite una entrada de tensión de 12 V a 24 V, entregando a la salida 5 V a 1 A.

Simulador ECU

Este dispositivo no es una parte del tablero digital como tal, pero tiene una función muy relevante en el proyecto. Para poder desarrollar el código de Arduino e ir testeándolo a medida que se va avanzando, no se ha utilizado un coche real, dado a la incomodidad que supone tener que conectarse a un vehículo cada vez que se necesite comprobar alguna parte del código.

Para evitar esto, se ha utilizado una placa emuladora de ECU, más concretamente la placa CAN BUS ECU Simulator v2.0 de la marca OZEN ELEKTRONIK (*figura 10*).

Además, la placa emuladora permite modificar valores como la temperatura del refrigerante, la velocidad del vehículo o las revoluciones del motor a través de unos potenciómetros, pudiendo llegar a valores que serían inconcebibles alcanzar en condiciones normales con un vehículo real.

Figura 10. Placa emuladora CAN BUS ECU simulator v2.0 de la marca OZEN ELEKTRONIK.

Diagrama de bloques

En la *figura 11* se tiene el diagrama de bloque que muestra la comunicación serie, cableada o no, que se lleva a cabo en el proyecto de tablero de instrumentos digital e inalámbrico. El elemento "central" es la placa Arduino, que hace de intermediario y traductor entre el dispositivo de diagnóstico ELM327 y la pantalla LCD Nextion.

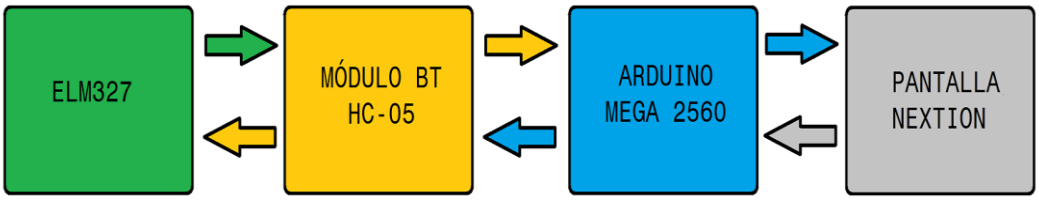


Figura 11. Diagrama de bloques de la comunicación serie.

El bloque naranja pertenece al módulo bluetooth que va conectado de manera cableada al Arduino Mega 2560. Estos dos bloques, el naranja y el azul, podrían considerarse uno. Luego se tienen los elementos que se encuentran en los extremos, en este caso el bloque verde (ELM327) y el bloque gris (Pantalla Nextion), el bloque verde es el encargado de comunicarse con la centralita del vehículo, y el gris el encargado de comunicarse con el usuario final del dispositivo.

Conexión módulo bluetooth – Arduino

En la *figura 12* se tiene el esquema de conexión entre el módulo bluetooth HC-05 y la placa Arduino. Los cables de color rojo y negro son los cables de alimentación, que van conectados por un lado a las patillas 5V y GND del Arduino mega y van a las patillas VCC y GND del dispositivo HC-05 respectivamente.

Por otro lado, se tienen los cables de comunicación, estos cables deben ir cruzados, es decir, el transmisor de un dispositivo con el receptor del otro. Siguiendo lo dicho, el cable naranja se conecta a la patilla Rx1 (receptor) del Arduino y a la patilla TxD (transmisor) del módulo bluetooth. El cable amarillo se conecta al Arduino por Tx1 (transmisor) y al HC-05 por RxD (receptor).

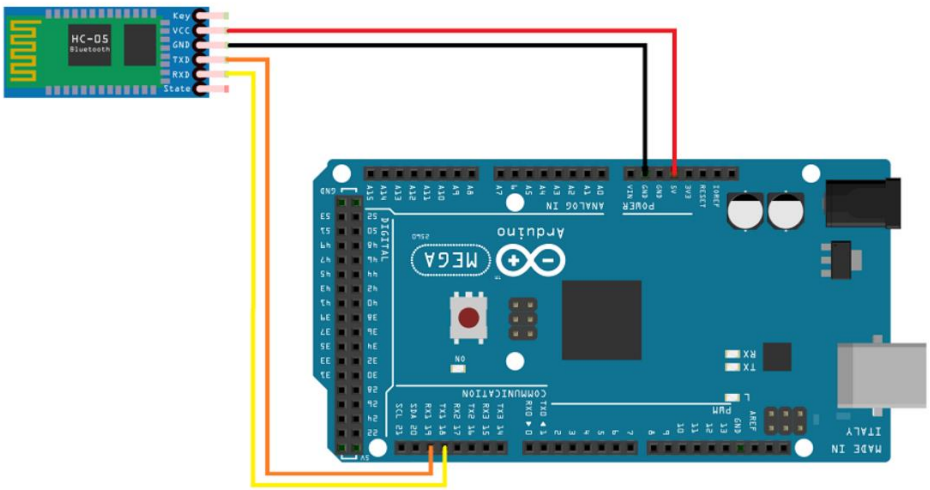


Figura 12. Esquema de conexión entre Arduino y HC-05.

Conexión pantalla táctil LCD – Arduino

En el esquema que se muestra en la *figura 13* se puede ver las conexiones establecidas entre la placa Arduino y la pantalla LCD de Nextion. En este caso las conexiones se hacen de la misma manera que en el esquema anterior.

Por un lado, los cables rojo y negro de alimentación y por el otro lado, los cables que se encargan de la comunicación serie de los dispositivos, que son los cables de color amarillo y azul. El amarillo va desde Tx0 de Arduino hasta la patilla

Rx de la pantalla, mientras que la azul se conecta desde la Rx0 de la placa hasta la patilla Tx de la pantalla.

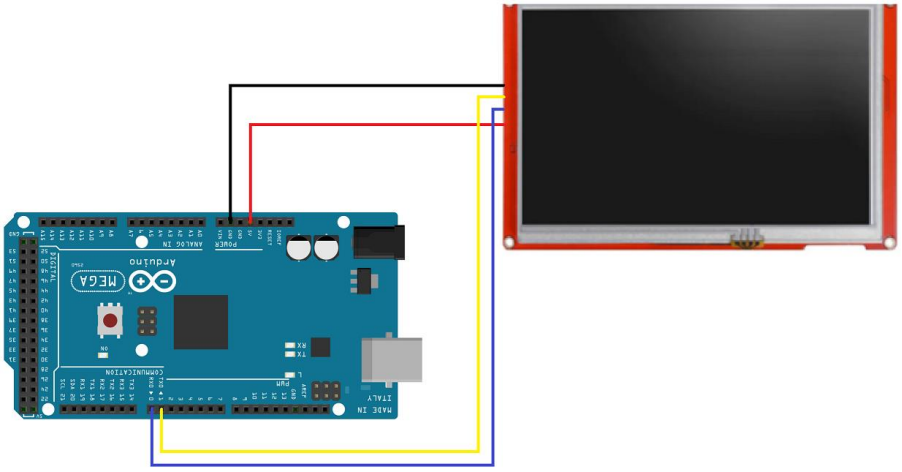


Figura 13. Esquema de conexión entre Arduino y pantalla Nextion.

Esquema de conexión final

En la *figura 14*, se muestra el esquema de conexión del sistema sin tener en cuenta el dispositivo de diagnosis.

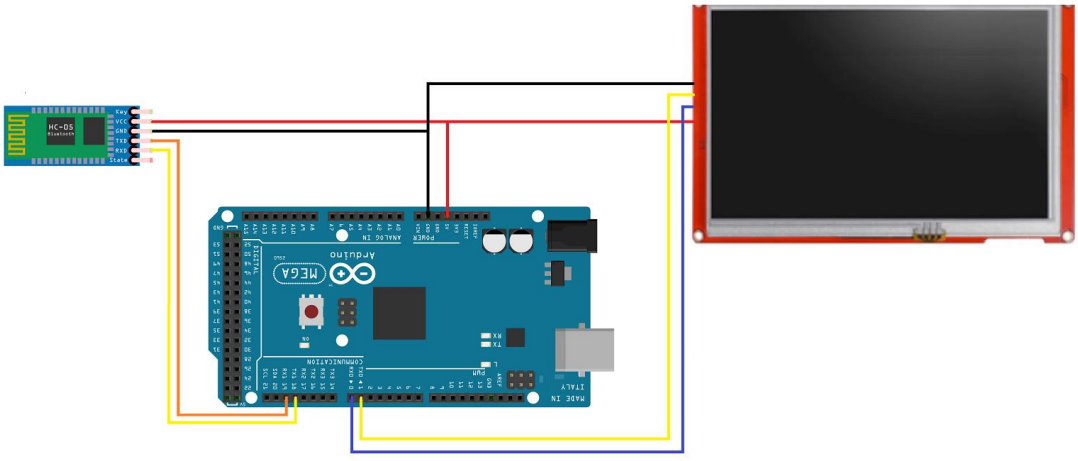


Figura 14. Esquema de conexión de la parte cableada del sistema.

La *figura 15* muestra el sistema completo, teniendo en cuenta el dispositivo ELM327.

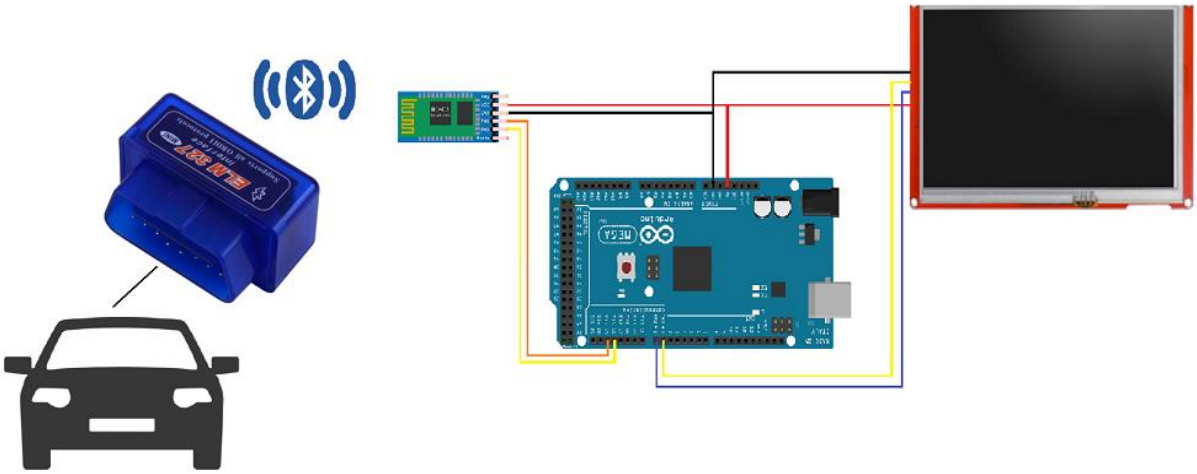


Figura 15. Esquema de conexión del sistema completo.

Capítulo 3. Software empleado

Los diferentes softwares empleados en la realización del proyecto son, el IDE de Arduino para desarrollar el código y llevar a cabo su programación y, por otro lado, el software de edición y creación de interfaces de usuario Nextion Editor.

Entorno de programación de Arduino

El funcionamiento de la placa Arduino se debe a través del programa externo que se carga en ella para luego ser ejecutado por el microcontrolador.

El programa o entorno de programación encargado de esto se llama Arduino IDE (Integrated Development Enviroment), o entorno de desarrollo integrado traducido al español.

El usuario desarrolla el código en el programa, una vez que se ha finalizado, se compila y, si el programa no retorna ningún tipo de error en el código se procede a cargar el programa en la placa Arduino. Una vez se tiene el código subido a la placa, ésta lo ejecutará de forma automática siempre que reciba la alimentación necesaria.

Programación Arduino

En este apartado y los siguientes subapartados se mostrarán las diferentes partes que componen el código que se ha desarrollado para el proyecto de tablero de instrumentos digital e inalámbrico.

De forma general, en el código se pueden distinguir diferentes partes como pueden ser el bucle principal que siempre se ejecuta y las diferentes funciones que lo componen. Dentro de las funciones se pueden distinguir dos bloques, las funciones de petición y adquisición de datos, y las funciones de monitorización de dichos datos.

Configuración módulo bluetooth

Inicialmente, se ha de configurar el módulo bluetooth con la finalidad de enlazar el módulo HC-05 con el dispositivo de diagnóstico ELM327. Además de enlazar los dispositivos, se configura la velocidad de comunicación y se establece al módulo como maestro.

La configuración del módulo bluetooth HC-05 se realiza a través de los comandos AT. Estos comandos no son más que unas instrucciones que son utilizadas en las comunicaciones serie, con el propósito de controlar y configurar los módulos de comunicación. Mediante estos comandos se puede obtener información sobre la configuración además de modificarla.

Para poder utilizar los comandos AT, primero se debe poner el módulo en modo configuración, manteniendo el pulsador que trae integrado durante los primeros 5 segundos una vez es alimentado el módulo. A su vez la patilla EN (enable) debe encontrarse a nivel alto.

Con el código de la *figura 16*, realizamos una comunicación serie entre el puerto serie del Arduino y el módulo HC-05. Esto se hace con el fin de introducir los comandos AT a través del monitor serie.

File Edit Sketch Tools Help

comandosAT §

```
#include <SoftwareSerial.h>

SoftwareSerial HC05(10,11);

void setup() {
  Serial.begin(9600);
  HC05.begin(9600);
//CONFIGURACIÓN DEL MODULO BT HC-05
//POR EL PIN 8 (PATILLA ENABLE)
  pinMode(8,OUTPUT);
  digitalWrite(8,HIGH);


  Serial.println("INICIO DE COMANDO AT");

}

void loop() {

  if(HC05.available())          //lee HC05 y lo envia al arduino
    Serial.write(HC05.read());

  if(Serial.available())
    HC05.write(Serial.read());  //lee arduino y envía a HC05
}
```

Figura 16. Código Arduino para configuración HC-05 (comandos AT).

Una vez se ha cargado el programa a la placa Arduino, se procede a configurar el dispositivo HC-05 escribiendo los comandos AT correspondientes a través del puerto COM del IDE de Arduino. En este caso, para la configuración que se desea obtener es, en este orden, los siguientes comandos:

AT

Enviar: AT

Respuesta: OK

Se comprueba la comunicación entre Arduino y HC-05.

AT+ORGL

Enviar: AT+ORGL

Respuesta: OK

Se establece la configuración de fábrica, en algunas versiones la velocidad es de 38400 baudios y en otros 9600.

AT+UART=<Baud>, <Bitparada>, <Paridad>

Enviar: AT+UART=38400,0,0

Respuesta: OK

Con este comando se configura la velocidad de comunicación, aunque de fábrica se establece esta velocidad, se asegura que el dispositivo funcionará con esta velocidad.

AT+ROLE

Enviar: AT+ROLE=1

Respuesta: OK

Por defecto, el dispositivo viene configurado como esclavo. Para este proyecto no es válido, puesto que el dispositivo va a comunicarse en ambas direcciones, es decir, el dispositivo HC-05 envía y recibe datos. Por esto, el dispositivo se debe configurar como maestro.

AT+CMODE

Enviar: AT+CMODE=0

Respuesta: OK

Enviando AT+CMODE=0 se indica al dispositivo que se va a realizar la conexión a una dirección especifica.

AT+BIND

Enviar: AT+BIND=6611E,32,F39F70

Respuesta: OK

Con el comando AT+BIND= <dirección>, y una vez se ha configurado el modo de conexión en 0 (CMODE=0), se indica al dispositivo bluetooth la dirección de dispositivo esclavo, en este caso el ELM327, que tiene como dirección MAC: 66:1E:32:F3:9F:70.

AT+PAIR

Enviar: AT+PAIR=6611E,32,F39F70

Respuesta: OK

Tras enviar el comando AT+PAIR= <dirección> que, en este caso la dirección del dispositivo a enlazar es 66:1E:32:F3:9F:70, el módulo HC-05 buscará establecer el emparejamiento con el dispositivo ELM327, una vez que se establece el emparejamiento entre dispositivos, se devolverá como respuesta un “OK”.

AT+LINK

Enviar: AT+LINK=6611E,32,F39F70

Respuesta: OK

Posteriormente al comando AT+PAIR, es necesario enviar el comando AT+LINK=<dirección>, con la dirección MAC del dispositivo ELM327. Este comando establece la conexión entre ambos dispositivos. [7]

Definición de variables

En la *figura 17* se muestra el fragmento del código que se encarga de definir las variables globales del programa.

```
1  //PROGRAMA TABLERO INSTRUMENTOS DIGITAL E INALÁMBRICO
2
3  // Variables para almacenar los datos del vehículo.
4  int velocidad;
5  int revoluciones;
6  int temperatura;
7  int flujoAire;
8  int fuel;
9  int tambiente;
10 int presion;
11
12 //valores mapeados para enviar a la pantalla.
13 int RPMmap; //valor de revoluciones mapeado para aguja/dial.
14 int SPEEDmap; // valor de velocidad mapeado para aguja/dial.
15 int RPMbarra; //valor de revoluciones mapeado para barra de progreso.
16
17
18 boolean START=false; //  true= inicia la comunicación, false= detiene la comunicación.
19 boolean clasico=false; // true= indica que la interfaz se encuentra en el modo clásico.
20 boolean deportivo=false; // true= indica que la interfaz se encuentra en el modo deportivo.
21 boolean racing=false; //true= indica que la interfaz se encuentra en el modo racing.
22
23
24 int datoIN; //dato entrante enviado por la pantalla.
25
```

Figura 17. Variables del programa de Arduino.

De la línea 4 a la línea 10 inclusive se tienen las variables de tipo entero que almacenan los valores que se obtienen del vehículo. En estas variables se guarda el valor de los parámetros obtenidos de la centralita del vehículo, los cuales se obtienen extrayéndolos de la trama de bytes, realizando la conversión de hexadecimal a decimal y, además aplicando la fórmula que corresponda para cada caso.

En las líneas 13, 14 y 15 se encuentra las variables de tipo entero "*RPMmap*", "*SPEEDmap*" y "*RPMbarra*". Estas variables tienen como finalidad almacenar el valor a representar una vez se ha mapeado.

De las líneas 18 a 21 se tienen 4 variables booleanas, la primera variable es "*START*", la cual indicará que el sistema está iniciado con su valor TRUE y que se encuentra detenido cuando su valor es FALSE. Las 3 variables restantes indican el modo de interfaz en el que se encuentra el tablero, siendo el valor TRUE indicativo de que dicho modo se encuentra activo.

Por último, en la línea 24 se encuentra la variable de tipo entero "datoIN", que se encarga de almacenar los datos de entrada que tienen como origen la pantalla LCD. Es decir, si la pantalla transmite un dato hacia la placa Arduino, éste dato se almacena en dicha variable hasta que sea reemplazado por otro dato entrante.

Void setup

A la hora de iniciar el código del proyecto en el entorno de programación, vienen por defector dos bucles, el primero de ellos es el bucle de configuración o "void setup". Este bucle tiene como finalidad establecer una configuración al iniciar el programa, dado que el código que haya en el interior solo se ejecutará una vez al inicio. En la *figura 18* se muestra el código utilizado en este bucle de configuración.

```
26 void setup() {
27   // Configuración de los puertos serie para comunicarse con el HC05.
28
29    Serial.begin(9600); //inicia puerto serie que comunica la pantalla con el arduino.
30    Serial1.begin(38400);// inicia el puerto serie que comunica el HC05 con el arduino.
31
32   Serial1.write("ATSP0\r\n"); // configura el ELM327 con protocolo automático
33   delay(100); // retraso de 100 ms
34   Serial1.write("ATST14\r\n"); // configura el tiempo de espera de respuesta.
35   delay(100); // retraso de 100 ms
36
37 }
```

Figura 18. función de configuración del Arduino.

Como se puede observar, éste se extiende desde la línea 26 hasta la línea 37. Inicialmente, en las líneas 29 y 30 se inician los puertos serie que se utilizarán a lo largo del proyecto para comunicar los dispositivos externos.

El puerto serie 0, el utilizado para la comunicación con la pantalla LCD, se le ha establecido una velocidad de comunicación de 9600 Baudios, mientras que al puerto serie 1, el que se utiliza para la comunicación con el dispositivo de diagnóstico, se ha configurado con una velocidad de 38400 Baudios.

A continuación, se transmite al ELM327 por el puerto serie 1, un comando AT, con el cual se establece el modo de búsqueda de protocolo automático, estableciendo el protocolo de comunicación necesario en cada caso. El comando usado para ello es ATSP0 (AT Set Protocol 0), haciendo referencia el 0 al protocolo automático, como se muestra en la *figura 19*. Acto seguido se ejecuta un retraso de 100 ms, dando tiempo a que se establezca la comunicación y la configuración de manera exitosa.

Protocol	Description
0	Automatic
1	SAE J1850 PWM (41.6 Kbaud)
2	SAE J1850 VPW (10.4 Kbaud)
3	ISO 9141-2 (5 baud init)
4	ISO 14230-4 KWP (5 baud init)
5	ISO 14230-4 KWP (fast init)
6	ISO 15765-4 CAN (11 bit ID, 500 Kbaud)
7	ISO 15765-4 CAN (29 bit ID, 500 Kbaud)
8	ISO 15765-4 CAN (11 bit ID, 250 Kbaud)
9	ISO 15765-4 CAN (29 bit ID, 250 Kbaud)
A	SAE J1939 CAN (29 bit ID, 250* Kbaud)
B	User1 CAN (11* bit ID, 125* Kbaud)
C	User2 CAN (11* bit ID, 50* Kbaud)

*user adjustable

Figura 19. Protocolos de comunicación OBD-II en el dispositivo ELM327.

De manera similar, en la línea de programación 34, se envía el comando ATST14 (AT Set Timeout hh), pudiendo establecer el tiempo que el dispositivo espera a la respuesta. Si pasado este tiempo no se obtiene ninguna respuesta por parte del dispositivo de diagnóstico, la respuesta que se recibe es “NO DATA”.

El valor escogido para el presente proyecto es el valor hexadecimal 14, que da un tiempo de espera de 80 ms, dado que por cada valor hexadecimal se añaden 4 ms. Por último, para dar tiempo a que se establezca la configuración, se añade una instrucción de retardo de 100 ms en la línea 35.

Bucle principal

En el void loop o bucle principal, se desarrolla la parte del código que se va a ejecutar en bucle de manera infinita. En la *figura 20* se muestra el fragmento encargado de la selección del tipo de tablero, es decir, su finalidad es saber que tipo de tablero ha escogido el usuario a través de la pantalla táctil, para así poder aplicar la parte del código correspondiente a cada tipo de tablero.

```
38
39 void loop() { //programa principal
40
41 if (Serial.available()){  //si el puerto serie 1 (pantalla) tiene datos en el buffer, hace lo siguiente:
42  datoIN=Serial.read();} //guarda el valor del buffer de entrada en la variable datoIN.
43
44  if(datoIN=='s'){
45        START=true;
46        clasico=true;}  // si el dato de entrada es "s", comienza la comunicación y accede al modo clásico.
47
48  if (datoIN=='i'){
49        START=false;
50        clasico=false;
51        deportivo=false;
52        racing=false;} //si el dato es "i", para la comunicación y sale del modo en el que se encuentre.
53
54   if(datoIN=='c'){
55        deportivo=false;
56        racing=false;
57        clasico=true;} //si dato es "c", entra en modo clásico.
58
59   if(datoIN=='d'){
60        clasico=false;
61        racing=false;
62        deportivo=true;}//si dato es "d", entra en modo deportivo.
63
64   if(datoIN=='r'){
65        clasico=false;
66        deportivo=false;
67        racing=true;}//si dato es "r", entra en modo racing.
68
```

Figura 20. Bucle principal del programa de Arduino (parte 1).

Como se puede observar, las líneas 41 y 42 del código permiten saber si el usuario ha seleccionado alguno de los botones de la interfaz y almacenar el valor que corresponde con cada botón en la variable "*datoIN*".

Posteriormente, tendremos 5 funciones "*if*", cada una encargada de aplicar el código que corresponda según el botón pulsado. En las líneas 44, 45 y 46 se tiene el código correspondiente al botón "*START*", con el cual el usuario pone en funcionamiento el tablero si la variable "*datoIN*" es la letra "s", haciendo TRUE las variables booleanas "*START*" y "*clasico*", indicando que se ha iniciado y poniendo por defecto el tipo de tablero clásico.

Desde la línea 48 a la línea 52 se tiene el código que hace lo contrario, si el tablero se encuentra funcionando y el usuario pulsa el botón inicio, la pantalla transmitirá la letra "i", provocando que el sistema finalice, y por esto, en las líneas 49, 50, 51 y 52 se ponen en FALSE las variables booleanas "*START*", "*clasico*", "*deportivo*"y "*racing*".

Si el dato proveniente de la pantalla es la letra "c", significa que el usuario ha pulsado el botón de tipo de tablero clásico, por lo que dentro del comando condicional *if*, que se encuentra en la línea 54, la variable booleana "clasico" pasará a TRUE, y las "deportivo" y "racing" se forzarán a tener un valor FALSE, tal y como se ve en las líneas 55, 56 y 57.

Sin embargo, si el dato enviado por la pantalla es la letra "d", se ejecuta la secuencia de instrucciones de las líneas 60, 61 y 62, las cuales se encuentran dentro del comando condicional *if* que se encuentra en la línea 59. La variable "*deportivo*" se le asignará el estado TRUE, en cambio, las variables "*clasico*" y "*racing*" pasarán a FALSE.

El último comando condicional *if*, funciona de la misma manera que los dos anteriores, estos van de la línea 64 a 67, poniendo en TRUE a la variable booleana "racing", y poniendo en FALSE a las variables "clásico" y "deportivo".

En las *figuras 21*, *22* y *23* se puede apreciar otro comando condicional *if* en la línea 71, el cual se extiende hasta la línea 134. A su vez, dentro de este condicional el código se divide en otros 3 condicionales, ejecutando cada uno de ellos el código correspondiente para cada tipo de tablero.

```
69
70  //si se ha iniciado la documentación y no hay datos en el buffer de entrada de la pantalla:
71 if(START==true && Serial.available()==0){
72       if(clasico==true){  // si está en modo clásico:
73                     lecturaRPM();  //lee revoluciones
74                         printRPM_CLASICO();       // envía a la pantalla los valores para el modo clásico
75                     lecturaSPEED();               //lee velocidad
76                         printSPEED_CLASICO();     // envía a la pantalla los valores para el modo clásico
77                     lecturaECT();                 // lee temperatura del refrigerante
78                         printECT_CLASICO();       // envía a la pantalla los valores para el modo clásico
79                     lecturaRPM();                 //lee revoluciones
80                         printRPM_CLASICO();       // envía a la pantalla los valores para el modo clásico
81                     lecturaSPEED();               //lee velocidad
82                         printSPEED_CLASICO();     // envía a la pantalla los valores para el modo clásico
83                     lecturaAAT();                 //lee temperatura del aire del ambiente
84                         printAAT_CLASICO();       // envía a la pantalla los valores para el modo clásico
85                     lecturaRPM();                 //lee revoluciones
86                         printRPM_CLASICO();       // envía a la pantalla los valores para el modo clásico
87                     lecturaSPEED();               //lee velocidad
88                         printSPEED_CLASICO();     // envía a la pantalla los valores para el modo clásico
89                     lecturaFUEL();                //lee nivel de combustible
90                         printFUEL_CLASICO();      // envía a la pantalla los valores para el modo clásico
91                             }
```

Figura 21. Bucle principal del programa de Arduino (parte 2).

```
92
93      if(deportivo==true){    // si está en modo deportivo:
94                  lecturaECT();                   //lee temperatura del refrigerante
95                      printECT_DEPORTIVO();       // envía a la pantalla los valores para el modo deportivo
96                  lecturaRPM();                   //lee revoluciones
97                      printRPM_DEPORTIVO();       // envía a la pantalla los valores para el modo deportivo
98                  lecturaSPEED();                 // lee velocidad
99                      printSPEED_DEPORTIVO();     // envía a la pantalla los valores para el modo deportivo
100                 lecturaFUEL();                  // lee nivel de combustible
101                     printFUEL_DEPORTIVO();      // envía a la pantalla los valores para el modo deportivo
102                 lecturaRPM();                   // lee revoluciones
103                     printRPM_DEPORTIVO();       // envía a la pantalla los valores para el modo deportivo
104                 lecturaSPEED();                 // lee velocidad
105                     printSPEED_DEPORTIVO();     // envía a la pantalla los valores para el modo deportivo
106                 lecturaMAF();                   // lee flujo de masa de aire
107                     printMAF_DEPORTIVO();       // envía a la pantalla los valores para el modo deportivo
108
109                         }
110
```

Figura 22. Bucle principal del programa de Arduino (parte 3).

```
111
112     if(racing==true){
113                 lecturaECT();                   // lee temperatura del refrigerante
114                     printECT_RACING();          // envía a la pantalla los valores para el modo racing
115                 lecturaRPM();                   // lee revoluciones
116                     printRPM_RACING();          // envía a la pantalla los valores para el modo racing
117                 lecturaSPEED();                 // lee velocidad
118                     printSPEED_RACING();        // envía a la pantalla los valores para el modo racing
119                 lecturaFUEL();                  // lee nivel de combustible
120                     printFUEL_RACING();         // envía a la pantalla los valores para el modo racing
121                 lecturaRPM();                   // lee revoluciones
122                     printRPM_RACING();          // lee revoluciones
123                 lecturaSPEED();                 // lee velocidad
124                     printSPEED_RACING();        // envía a la pantalla los valores para el modo racing
125                 lecturaMAF();                   // lee flujo de masa de aire
126                     printMAF_RACING();          // envía a la pantalla los valores para el modo racing
127                 lecturaRPM();                   // lee revoluciones
128                     printRPM_RACING();          // envía a la pantalla los valores para el modo racing
129                 lecturaSPEED();                 // lee velocidad
130                     printSPEED_RACING();        // envía a la pantalla los valores para el modo racing
131                 lecturaBARO();                  // lee presión barométrica absoluta
132                     printBARO_RACING();         // envía a la pantalla los valores para el modo racing
133                             }
134 }
```

Figura 23. Bucle principal del programa de Arduino (parte 4).

El primer *if* corresponde al tipo de tablero clásico y tiene como condición que la variable booleana “clasico” tenga un estado TRUE. Las instrucciones correspondientes al tablero clásico son las que se ven de la línea 73 a la línea 90. Se trata de hacer peticiones al dispositivo ELM327 de la información deseada e inmediatamente después se envía la respuesta a la pantalla LCD.

En el modo clásico se establece el siguiente orden: revoluciones, velocidad, temperatura refrigerante, revoluciones, velocidad, temperatura aire ambiente, revoluciones, velocidad y combustible. Como se puede apreciar, las revoluciones y la velocidad se repiten de manera intercalada con los demás datos, esto se hace debido a que, el valor de las revoluciones y la velocidad varían de manera más rápida en el vehículo, por lo que debe tener un tiempo de refresco menor en comparación a los demás.

En el segundo *if*, se tiene como condición que la variable “deportivo” se encuentre en un valor TRUE. La secuencia de llamada a funciones se desarrolla desde la línea 94 hasta la 107. Es similar al código del tablero clásico, se intercalan las peticiones de revoluciones y velocidad.

Por último, en el tercer *if*, la condición es un valor verdadero de la variable “racing”. Esta secuencia de funciones comienza en la línea 113 y finaliza en la línea 133 y funciona de manera equivalente a los dos comandos condicionales anteriores.

Funciones peticiones PID

En el siguiente subapartado a comentar las funciones creadas para peticiones de parámetros del sistema OBD-II.

En conjunto para este proyecto, se han realizado 7 funciones diferentes que tienen como finalidad realizar las peticiones de datos a la ECU a través del dispositivo de diagnóstico ELM327. Las funciones antes mencionadas siguen una estructura similar.

En primer lugar, se envía a través del puerto serie 1 (correspondiente al dispositivo bluetooth HC-05) el valor PID que corresponda en cada ocasión. Una vez se envía la petición, se almacena en una variable tipo *String* la respuesta que se obtiene del dispositivo de diagnóstico ELM327.

Inmediatamente después, se comprueba la longitud del mensaje, debido a que no siempre se tiene una respuesta homogénea y puede variar la longitud de dicho mensaje, provocando que la información que es de utilidad se encuentre en una posición diferente.

En este caso, el simulador de ECU que se utiliza para el desarrollo del sistema utiliza un protocolo CAN, que tiene una trama de mensaje de 8 bytes, con un identificador que puede ser de 11 o 29 bits. De estos 8 bytes de dato no siempre se utilizan todos, depende del tamaño de la información solicitada.

Ya conocida la longitud del mensaje, podemos ubicar el fragmento del mensaje que se necesita y se puede extraer del string, almacenando este fragmento en una variable de tipo entero. [1]

Las funciones son las siguientes que se muestra en la:

lecturaECT(): esta función se muestra en la *figura 24* y se extiende desde la línea 138 hasta la línea 155.

```
138 void lecturaECT(){
139   Serial1.write("0105\r\n");//------ECT
140     delay(100);
141 
142     String ECT_IN = Serial1.readStringUntil(62);//lee y guarda en string
143     int longECT = ECT_IN.length(); //guarda el valor de la longitud
144 
145     if(longECT==17){//si la longitud es 17
146     temperatura = strtol(ECT_IN.substring(12,14).c_str(), NULL, 16)-40 ;}
147     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13) y restar 40
148     if(longECT==18){//si la longitud es 18
149     temperatura = strtol(ECT_IN.substring(13,15).c_str(), NULL, 16)-40 ;}
150     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 13 y 14) y restar 40
151     if(longECT==19){//si la longitud es 19
152     temperatura = strtol(ECT_IN.substring(14,16).c_str(), NULL, 16)-40 ;}
153     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 14 y 15) y restar 40
154 
155               }
```

Figura 24. Función lecturaECT.

Se puede observar que mediante el puerto serie 1 se envía “0105\r\n” siendo el 0105 el PID que corresponde al parámetro Engine Coolant Temperature (ECT), o temperatura del refrigerante motor traducido al español. La respuesta obtenida se almacena en la variable de tipo *String* “*ECT_IN*”, la longitud del mensaje pasa a la variable *int* **“***longECT***”** y, según esta longitud, se extrae el valor útil del resto del mensaje.

Si la longitud del mensaje es de 17, el valor de temperatura se encuentra en las posiciones 12 y 13 del *String*. Si la longitud es de 18 el valor está en las posiciones 13 y 14 y, si la longitud fuese en algún caso 19, se encontrarían en las posiciones 14 y 15 del *String* “*ECT_IN*”. Estos valores que son los que se necesitan y que en ocasiones cambian de posición, se almacenan en la variable “*temperatura*” siempre habiendo restando antes el valor 40, para convertirlo al valor real de temperatura.

lecturaRPM(): En este caso, el HC-05 envía “010C\r\n” que hace referencia al PID de las revoluciones del motor (RPM) y, la respuesta que se obtiene se guarda en la variable de tipo string “*RPM_IN*”. Se muestra en la *figura 25.*

```
159 void lecturaRPM(){
160   Serial1.write("010C\r\n");//------RPM
161     delay(100);//retraso de 100 ms
162 
163     String RPM_IN = Serial1.readStringUntil(62);//lee y guarda en string
164     int longRPM = RPM_IN.length();//guarda el valor de la longitud
165 
166 
167      if(longRPM==20){//si la longitud es 20
168      int rev1B = strtol(RPM_IN.substring(12,14).c_str(), NULL, 16)*64;//Convertir hexa a decimal posición 12 y 13,por 64
169      int rev2B = strtol(RPM_IN.substring(15,17).c_str(), NULL, 16)/4 ;//Convertir hexa a decimal posición 15 y 16,entre 4
170      revoluciones= rev1B+rev2B ;}//sumamos los 2 bytes
171 
172       if(longRPM==21){//si la longitud es 21
173      int rev1B = strtol(RPM_IN.substring(13,15).c_str(), NULL, 16)*64;//Convertir hexa a decimal posición 13 y 14,por 64
174      int rev2B = strtol(RPM_IN.substring(16,18).c_str(), NULL, 16)/4 ;//Convertir hexa a decimal posición 16 y 17,entre 4
175      revoluciones= rev1B+rev2B ; //sumamos los 2 bytes
176      }
177 
178   }
```

Figura 25. Función lecturaRPM.

La longitud en este caso se almacena en la variable int "longRPM" y su función es la misma en todas las funciones, pero en este caso, el valor ocupa 2 bytes, por lo que el valor de las revoluciones en el string ocupa 4 posiciones de este. El primer byte se almacena en la variable "*rev1B*" y el segundo byte en "*rev2B*". El valor que se busca se guarda en la variable de tipo entero "*revoluciones*".

lecturaSPEED(): En el caso de lectura de la velocidad funciona similar al de la temperatura, cambiando, evidentemente las variables donde se almacenan los distintos datos. Para conocer la velocidad del vehículo se debe enviar "010D\r\n", la respuesta se almacena en el *string* "*SPEED_IN*", la longitud en "*longSPEED*", y el valor final ya extraído de la trama en "*velocidad*". Todo esto se muestra en la *figura 26.*

```
181 void lecturaSPEED(){
182   Serial1.write("010D\r\n");//------SPEED
183    delay(100);//retraso de 100 ms
184 
185     String SPEED_IN = Serial1.readStringUntil(62);//lee y guarda en string
186     int longSPEED= SPEED_IN.length();//guarda el valor de la longitud
187 
188     if(longSPEED==17){//si la longitud es 17
189     velocidad = strtol(SPEED_IN.substring(12,14).c_str(), NULL, 16);}
190     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13) y resta 40
191     if(longSPEED==18){//si la longitud es 18
192     velocidad = strtol(SPEED_IN.substring(13,15).c_str(), NULL, 16);}
193     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 13 y 14) y resta 40
194     if(longSPEED==19){//si la longitud es 19
195     velocidad = strtol(SPEED_IN.substring(14,16).c_str(), NULL, 16);}
196     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 14 y 15) y resta 40
197 
198   }
```

Figura 26.Función lecturaSPEED.

lecturaMAF(): La lectura del flujo de masa de aire (MAF) se comporta como el anterior. Se envía el PID "0110\r\n", guarda la respuesta en el *string* "*MAF_IN*" y la longitud de la trama en "*longMAF*". El valor del primer byte se guarda en maf1B y el segundo byte en maf2B, una vez aplicada la fórmula correspondiente a cada byte, y el valor final, que no es más que la suma de ambos bytes, se almacena en "*flujoAire*". Esta función está comprendida entre las líneas 197 y 214 tal y como se ve en la *figura 27*.

```
200 void lecturaMAF(){
201   Serial1.write("0110\r\n");//------MAF
202    delay(100);//retraso de 100 ms
203
204     String MAF_IN = Serial1.readStringUntil(62);//lee y guarda en string
205     int longMAF= MAF_IN.length();//guarda el valor de la longitud
206
207     if(longMAF==20){//si la longitud es 20
208
209      int maf1B = strtol(MAF_IN.substring(12,14).c_str(), NULL, 16)*256/100;//Convertir hexa a dec posición 12 y 13, por 256/100
210      int maf2B = strtol(MAF_IN.substring(15,17).c_str(), NULL, 16)/100 ;//Convertir hexa a dec posición 15 y 16,entre 100
211      flujoAire= maf1B+maf2B ;} //suma de ambos bytes
212
213     if(longMAF==21){//si la longitud es 21
214
215      int maf1B = strtol(MAF_IN.substring(13,15).c_str(), NULL, 16)*256/100;//Convertir hexa a dec posición 13 y 14,por 256/100
216      int maf2B = strtol(MAF_IN.substring(16,18).c_str(), NULL, 16)/100 ;//Convertir hexa a dec posición 16 y 17,entre 100
217      flujoAire= maf1B+maf2B ;}//suma de ambos bytes
218  }
```

Figura 27.Función lecturaMAF.

lecturaFUEL(): En la función de lectura del nivel de combustible, la petición se realiza mediante el PID "012F", el *string* para la respuesta es el "FUEL_IN", la longitud se retiene en "longFUEL". El valor final una vez aplicada su fórmula se almacena en "*fuel*". La función va de la línea 216 a la línea 234 como se muestra en la *figura 28.*

```
220 void lecturaFUEL(){
221
222   Serial1.write("012F\r\n");//------FUEL
223    delay(100);//retraso de 100 ms
224     String FUEL_IN = Serial1.readStringUntil(62);//lee y guarda en string
225     int longFUEL = FUEL_IN.length();//guarda el valor de la longitud
226
227     if(longFUEL==17){//si la longitud es 17
228     fuel = strtol(FUEL_IN.substring(12,14).c_str(), NULL, 16)*100/255;
229     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13) y multiplica 100/255
230     }
231
232     if(longFUEL==18){//si la longitud es 18
233     fuel = strtol(FUEL_IN.substring(13,15).c_str(), NULL, 16)*100/255;
234     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 13 y 14) y multiplica 100/255
235     }
236
237      if(longFUEL==19){//si la longitud es 19
238     fuel = strtol(FUEL_IN.substring(14,16).c_str(), NULL, 16)*100/255;
239     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 14 y 15) y multiplica 100/255
240     }
241  }
```

Figura 28.Función lecturaFUEL.

lecturaBARO(): La función tiene un tamaño que va desde la línea 237 a la línea 252 y se encarga de la petición y lectura de la presión barométrica absoluta. El PID que se ha de enviar es "0133", la respuesta se introduce en el string "BARO_IN", se comprueba su longitud a través de la variable "longBARO" y el valor final se almacena en la variable de tipo entero "presion". La *figura 29* muestra esta función.

```
244 void lecturaBARO(){
245   Serial1.write("0133\r\n");//------ presion absoluta barometrica ABP
246     delay(100);//retraso de 100 ms
247
248
249      String BARO_IN = Serial1.readStringUntil(62);//lee y guarda en string
250       int longBARO = BARO_IN.length();//guarda el valor de la longitud
251
252      if(longBARO==17){//si la longitud es 17
253      presion = strtol(BARO_IN.substring(12,14).c_str(), NULL, 16);}
254      //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13)
255
256      if(longBARO==18){//si la longitud es 18
257      presion = strtol(BARO_IN.substring(13,15).c_str(), NULL, 16);}
258      //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 13 y 14)
259
260      if(longBARO==19){//si la longitud es 19
261      presion = strtol(BARO_IN.substring(14,16).c_str(), NULL, 16);}
262      //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 14 y 15)
263
264
265          }
```

Figura 29.Función lecturaBARO.

lecturaAAT(): Esta función, que se muestra en la *figura 30*, es la encargada de conocer el valor de la temperatura aire del ambiente y se encuentra de la línea 258 a la línea 274. Su PID es "0146", el mensaje de respuesta se introduce en el string "*AAT_IN*", la longitud en "*longAAT*", y el valor final buscado en "*tambiente*".

```
268 void lecturaAAT(){
269  Serial1.write("0146\r\n");//------ AAT TEMPERATURA AIRE AMBIENTE
270    delay(100); //retraso de 100 ms
271
272
273     String AAT_IN = Serial1.readStringUntil(62);//lee y guarda en string
274     int longAAT = AAT_IN.length();//guarda el valor de la longitud
275
276     if(longAAT==17){//si la longitud es 17
277     tambiente = strtol(AAT_IN.substring(12,14).c_str(), NULL, 16)-40;}
278     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13) y resta 40
279    if(longAAT==18){//si la longitud es 18
280     tambiente = strtol(AAT_IN.substring(13,15).c_str(), NULL, 16)-40;}
281     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13) y resta 40
282     if(longAAT==19){//si la longitud es 19
283     tambiente = strtol(AAT_IN.substring(14,16).c_str(), NULL, 16)-40;}
284     //convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion 12 y 13) y resta 40
285        }
```

Figura 30.Función lecturaAAT.

Funciones para el envío de datos a la pantalla

En este subapartado se mostrarán las diferentes funciones que se encargan de enviar los datos obtenidos a la pantalla LCD a través del puerto serie 0. Se puede dividir en 3 secciones, una para cada tipo de tablero, debido a que un mismo dato, se envía o se muestra en pantalla de forma diferente, dependiendo del tablero seleccionado por el usuario.

Tablero clásico

printECT_CLASICO():

En la *figura 31* se muestra la función encargada de enviar el valor de la temperatura del refrigerante a la pantalla cuando se está en modo clásico.

```
290 void printECT_CLASICO(){
291   if(temperatura>-40){//si la temperatura es mayor que -40
292 
293             Serial.print("n0.val=");   //envía (nombre del objeto).val=
294             Serial.print(temperatura); //valor del parámetro
295                 Serial.write(0xFF);
296                 Serial.write(0xFF);
297                 Serial.write(0xFF);  //final de la trama
298     }}
```

Figura 31. Función printECT_CLASICO.

printRPM_CLASICO():

En la *figura 32* se muestra la función encargada de enviar el valor de las revoluciones del motor a la pantalla cuando se está en modo clásico.

```
300 void printRPM_CLASICO(){
301   RPMmap = map(revoluciones,0,9000,0,270);
302   if(RPMmap>=0) {
303 
304             Serial.print("z0.val=");  //envía (nombre del objeto).val=
305             Serial.print(RPMmap);     //valor del parámetro
306                 Serial.write(0xFF);
307                 Serial.write(0xFF);
308                 Serial.write(0xFF);   //final de la trama
309                                     }
310       }
```

Figura 32. Función printRPM_CLASICO.

printSPEED_CLASICO():

En la *figura 33* se muestra la función encargada de enviar el valor de la velocidad del vehículo a la pantalla cuando se está en modo clásico.

```
313 void printSPEED_CLASICO(){
314
315   if(velocidad>=0){
316
317               Serial.print("z1.val=");   //envía (nombre del objeto).val=
318               Serial.print(velocidad);   //valor del parámetro
319                   Serial.write(0xFF);
320                   Serial.write(0xFF);
321                   Serial.write(0xFF);    //final de la trama
322    }                        }
323
```

Figura 33. Función printSPEED_CLASICO.

printFUEL_CLASICO():

En la *figura 34* se muestra la función encargada de enviar el valor del nivel de combustible a la pantalla cuando se está en modo clásico.

```
325 void printFUEL_CLASICO(){
326               Serial.print("j0.val=");   //envía (nombre del objeto).val=
327               Serial.print(fuel);        //valor del parámetro
328                    Serial.write(0xFF);
329                    Serial.write(0xFF);
330                    Serial.write(0xFF);    //final de la trama
331     }
332
```

Figura 34. Función printFUEL_CLASICO.

printAAT_CLASICO():

En la *figura 35* se muestra la función encargada de enviar el valor de la temperatura del ambiente a la pantalla cuando se está en modo clásico.

```
334 void printAAT_CLASICO(){
335    Serial.print("n1.val=");              //envía (nombre del objeto).val=
336                Serial.print(tambiente);  //valor del parámetro
337                    Serial.write(0xFF);
338                    Serial.write(0xFF);
339                     Serial.write(0xFF); //final de la trama
340      }
341
```

Figura 35. Función printAAT_CLASICO.

printECT_DEPORTIVO():

En la *figura 36* se muestra la función encargada de enviar el valor de la temperatura del refrigerante a la pantalla cuando se está en modo deportivo.

```
344 void printECT_DEPORTIVO(){
345
346   Serial.print("n0.val="); //envía (nombre del objeto).val=
347               Serial.print(temperatura); //valor del parámetro
348                    Serial.write(0xFF);
349                    Serial.write(0xFF);
350                    Serial.write(0xFF); //final de la trama
351    }
```

Figura 36. Función printECT_DEPORTIVO.

printRPM_DEPORTIVO():

En la *figura 37* se muestra la función encargada de enviar el valor de las revoluciones a la pantalla cuando se está en modo deportivo.

```
352 void printRPM_DEPORTIVO(){
353   RPMbarra = map(revoluciones,0,16360,0,100);
354   Serial.print("j2.val="); //envía (nombre del objeto).val=
355               Serial.print(RPMbarra); //valor del parámetro
356                    Serial.write(0xFF);
357                    Serial.write(0xFF);
358                    Serial.write(0xFF);//final de la trama
359     }
```

Figura 37. Función printRPM_DEPORTIVO.

printSPEED_DEPORTIVO():

En la *figura 38* se muestra la función encargada de enviar el valor de la velocidad del vehículo a la pantalla cuando se está en modo deportivo.

```
360 void printSPEED_DEPORTIVO(){
361   Serial.print("n1.val="); //envía (nombre del objeto).val=
362               Serial.print(velocidad); //valor del parámetro
363                   Serial.write(0xFF);
364                   Serial.write(0xFF);
365                   Serial.write(0xFF);//final de la trama
366    }
```

Figura 38. Función printSPEED_DEPORTIVO.

printFUEL_DEPORTIVO():

En la *figura 39* se muestra la función encargada de enviar el valor del nivel de combustible a la pantalla cuando se está en modo deportivo.

```
367 void printFUEL_DEPORTIVO(){
368   Serial.print("j0.val="); //envía (nombre del objeto).val=
369               Serial.print(fuel); //valor del parámetro
370                   Serial.write(0xFF);
371                   Serial.write(0xFF);
372                   Serial.write(0xFF);//final de la trama
373    }
```

Figura 39. Función printFUEL_DEPORTIVO.

printMAF_DEPORTIVO():

En la *figura 40* se muestra la función encargada de enviar el valor del flujo de masa de aire a la pantalla cuando se está en modo deportivo.

```
374 void printMAF_DEPORTIVO(){
375
376   Serial.print("n2.val="); //envía (nombre del objeto).val=
377               Serial.print(flujoAire); //valor del parámetro
378                   Serial.write(0xFF);
379                   Serial.write(0xFF);
380                   Serial.write(0xFF);//final de la trama
381      }
```

Figura 40. Función printMAF_DEPORTIVO.

printECT_RACING():

En la *figura 41* se muestra la función encargada de enviar el valor de la temperatura del refrigerante a la pantalla cuando se está en modo racing.

```
384 void printECT_RACING(){
385   Serial.print("n0.val="); //envía (nombre del objeto).val=
386               Serial.print(temperatura); //valor del parámetro
387                     Serial.write(0xFF);
388                     Serial.write(0xFF);
389                     Serial.write(0xFF); //final de la trama
390   }
```

Figura 41. printECT_RACING.

printRPM_RACING():

En la *figura 42* se muestra la función encargada de enviar el valor de las revoluciones del motor a la pantalla cuando se está en modo racing.

```
391 void printRPM_RACING(){
392     RPMmap = map(revoluciones,0,9000,0,270);
393
394   Serial.print("n2.val="); //envía (nombre del objeto).val=
395               Serial.print(revoluciones); //valor del parámetro
396                     Serial.write(0xFF);
397                     Serial.write(0xFF);
398                     Serial.write(0xFF);//final de la trama
399
400   Serial.print("z0.val="); //envía (nombre del objeto).val=
401               Serial.print(RPMmap); //valor del parámetro
402                     Serial.write(0xFF);
403                     Serial.write(0xFF);
404                     Serial.write(0xFF);//final de la trama
405    }
```

Figura 42. printRPM_RACING.

printSPEED_RACING():

En la *figura 43* se muestra la función encargada de enviar el valor de la velocidad del vehículo a la pantalla cuando se está en modo racing

```
406 void printSPEED_RACING(){
407   Serial.print("n1.val="); //envía (nombre del objeto).val=
408               Serial.print(velocidad); //valor del parámetro
409                   Serial.write(0xFF);
410                   Serial.write(0xFF);
411                   Serial.write(0xFF);//final de la trama
412   }
```

Figura 43. printSPEED_RACING.

printFUEL_RACING():

En la *figura 44* se muestra la función encargada de enviar el valor del nivel de combustible a la pantalla cuando se está en modo racing.

```
413 void printFUEL_RACING(){
414   Serial.print("n3.val="); //envía (nombre del objeto).val=
415               Serial.print(fuel); //valor del parámetro
416                   Serial.write(0xFF);
417                   Serial.write(0xFF);
418                   Serial.write(0xFF);//final de la trama
419   }
```

Figura 44. printFUEL_RACING.

printMAF_RACING():

En la *figura 45* se muestra la función encargada de enviar el valor del flujo de masa de aire a la pantalla cuando se está en modo racing.

```
420 void printMAF_RACING(){
421   Serial.print("n4.val="); //envía (nombre del objeto).val=
422               Serial.print(flujoAire); //valor del parámetro
423                   Serial.write(0xFF);
424                   Serial.write(0xFF);
425                   Serial.write(0xFF);//final de la trama
426   }
```

Figura 45. printMAF_RACING.

printBARO_RACING():

En la *figura 46* se muestra la función encargada de enviar el valor de la presión barométrica absoluta a la pantalla cuando se está en modo racing.

```
427 void printBARO_RACING(){
428   Serial.print("n5.val="); //envía (nombre del objeto).val=
429                 Serial.print(presion); //valor del parámetro
430                     Serial.write(0xFF);
431                     Serial.write(0xFF);
432                     Serial.write(0xFF);//final de la trama
433                     }
```

Figura 46. printBARO_RACING

Nextion Editor (software desarrollo interfaz)

Nextion Editor es un sotfware específico que proporciona la marca Nextion para la creación y el diseño de las interfaces gráficas. Una vez completado el proyecto por parte del desarrollador o usuario, solo basta con compilar el proyecto y cargarlo a una pantalla Nextion para su funcionamiento.

Elementos básicos

En este subapartado, se muestra y se explican de manera breve los elementos básicos de diseño que ofrece el software de diseño de interfaz Nextion Editor y que se utilizarán en la creación de la interfaz del proyecto del tablero de instrumentos. En la *figura 47,* se puede observar la ventana en la que el programa muestra los diferentes elementos o herramientas que ofrece a la hora de diseñar la interfaz de usuario.

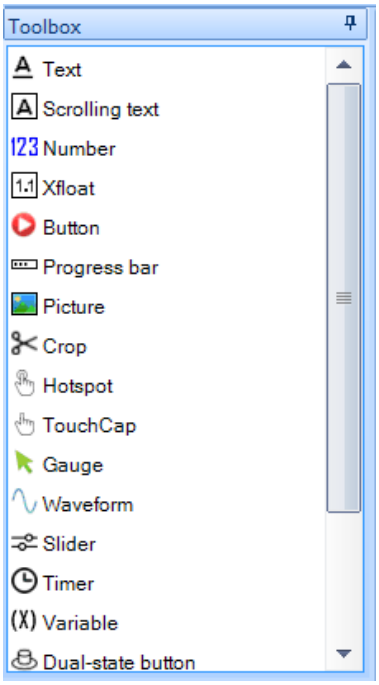


Figura 47. Ventana de elementos de Nextion Editor.

Los elementos usados en el presente proyecto son:

TEXTO (Text): Permite poner en pantalla información textual, como etiquetas o instrucciones. Pueden personalizarse en muchos aspectos como la fuente, el tamaño, el color y alineación.

NÚMERO (Number): elemento esencial para poder mostrar valores numéricos en tiempo real de manera clara. Al igual que con el texto, se pueden personalizar bastantes aspectos.

FLOTANTE (Xfloat): es un elemento imprescindible si, además de necesitar mostrar un valor numérico, éste tiene decimales. Su personalización es similar a los elementos anteriormente mencionados.

BOTÓN (Button): este elemento es un elemento interactivo, a través del cual los usuarios pueden activar o desactivar acciones específicas. Tiene una amplia gama de personalización, pudiendo ser configurados con colores, imágenes o incluso siendo transparentes de manera parcial o total.

En este elemento es primordial la programación por parte del desarrollador, dado que le permiten la programación de acciones o eventos, pudiendo pasar de página o enviando mensajes al microcontrolador. Además, el desarrollador puede elegir si dichas acciones se llevan a cabo nada más pulsar el botón, o en cambio, activarse solo cuando el usuario decida soltarlo.

BARRA DE PROGRESO (Progress bar): Este elemento es uno de los mas visuales, pudiendo mostrar el valor o porcentaje de algo de manera muy intuitiva y sin necesidad de elementos numéricos.

AGUJA (Gauge): como finalidad principal es la de mostrar magnitudes que puedan ser representadas en una escala. Se suele usar para valores analógicos, como en este proyecto, como la velocidad, las revoluciones del motor. Al igual que los elementos anteriores permite una gran personalización, de colores, tamaños, ángulos, límites y apariencia.

Creación de interacciones y comportamientos

En este subapartado se comenta la capacidad de Nextion Editor en cuanto a genera interacciones dinámicas en la interfaz de usuario mediante la posibilidad de asignar acciones o eventos específicos a diferentes elementos como pueden ser botones.

En este proyecto las interacciones que se usan se han asignado a botones, haciendo que cuando se suelta el botón, se cambie de página y, en la mayoría de los botones configurados, justo antes de cambiar de página se envía un mensaje al Arduino para poder interaccionar también con este dispositivo a través de la función táctil de la pantalla. [9]

Paginas utilizadas

Para la interfaz de usuario de este proyecto se han creado 5 páginas diferentes, las cuales están numeradas del 0 al 4, tal y como se muestra en la *figura 48*.

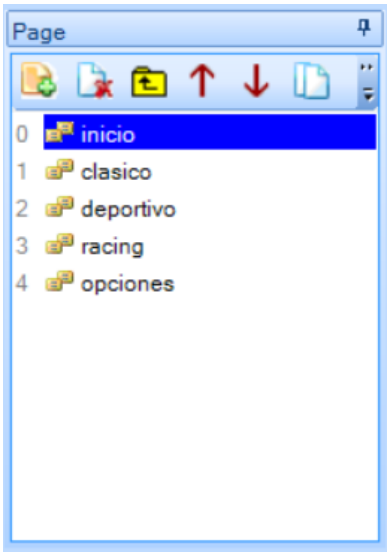


Figura 48. Ventana de páginas de Nextion Editor.

A continuación, se comentan las distintas páginas existentes en el proyecto, indicando los diferentes elementos que las componen y, además, mostrando las acciones o eventos programados cuando los haya.

En la *figura 49* se muestra la página INICIO.

Figura 49. Disposición de elementos usados en la página de inicio.

La página INICIO, consta de tres elementos, 2 elementos de texto (t0 y t1) y un botón (b0). A parte de estos tres elementos, tiene una imagen de fondo, haciendo la página de inicio más agradable a la vista del usuario e indicando un poco de manera visual la finalidad del dispositivo.

t0: texto indicando de que trata el proyecto.

t1: texto guía, indica al usuario que para comenzar el funcionamiento se debe pulsar el botón b0.

b0: botón con fondo y texto “START”,al que se le ha asignado un evento mediante código. El evento envía un mensaje a la placa Arduino haciendo que este comience la adquisición de datos del vehículo e, inmediatamente pasa de la página “inicio” a la página “clasico”.

En la *figura 50* se muestra la ventana en la que se asignan los eventos, a lo largo del proyecto los eventos se han configurado para que se realicen a la hora de soltar el botón.

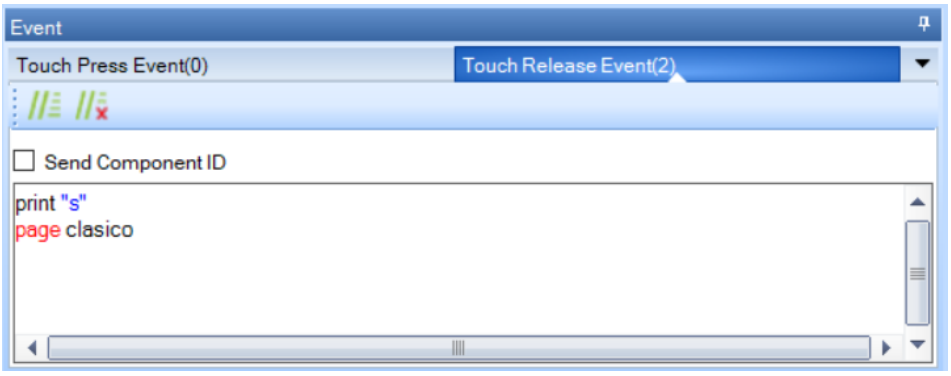


Figura 50. Ventana de eventos asociados al botón "START".

Para la página "clasico", se han utilizado los elementos que se muestran en la *figura 51*.

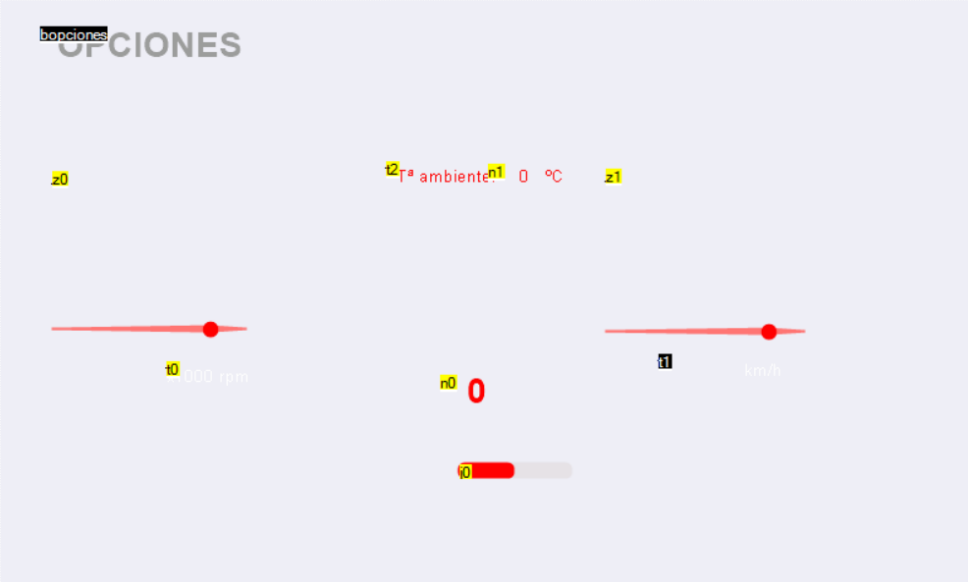


Figura 51. Disposición de elementos usados en la página "clásico".

En el tipo de tablero clásico se tiene un botón (bopciones) el cual está presente en los tres tipos de tablero, tres elementos de textos (t0, t1 y t2), dos indicadores de aguja (z0 y z1), dos indicadores numéricos (n0 y n1) y por último un indicador de barra (j0).

A continuación, se explica brevemente el uso que se ha dado a cada uno de estos elementos.

bopciones: en el botón se encuentra escrita la palabra “opciones”, esto es debido a que cuando el usuario suelte este botón, la interfaz cambia a la página opciones.

t0: este elemento de texto solo indica la escala en la que están las revoluciones del motor y su unidad.

t1: este texto indica la unidad en la que se mostrará la velocidad.

t2: en este texto pone Tª ambiente, para poder dar un contexto al valor numérico que se muestra en n1.

z0: este indicador de aguja se comportará como un tacómetro, indicando las revoluciones del motor.

z1: mostrará la velocidad del vehículo a través de un indicador de aguja.

n0: se mostrará la temperatura del refrigerante del motor a través de un valor numérico.

n1: a través de un valor numérico se muestra el valor de la temperatura del ambiente.

j0: una manera visual y sencilla de indicar el nivel de combustible es a través de un indicador de barra como en este caso.

De todos estos elementos solo el botón “OPCIONES” tiene un evento asociado mediante código. Este se muestra en la *figura 52.*

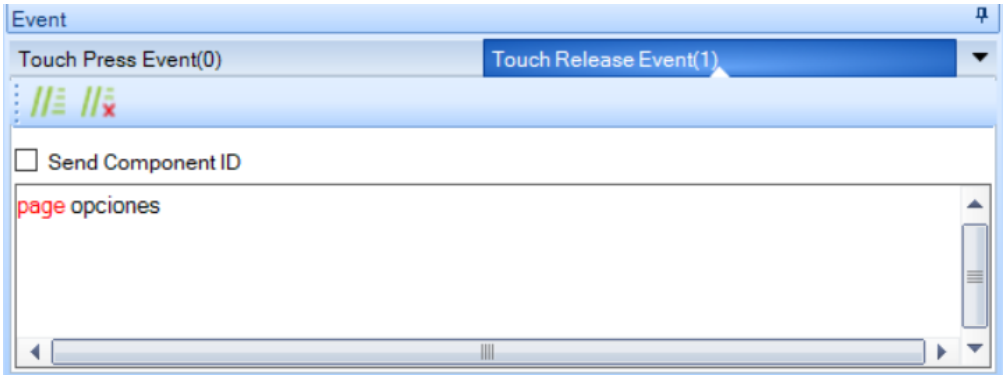


Figura 52. Ventana de eventos asociados al botón opciones.

DEPORTIVO

En el caso del tablero deportivo se ha intentado darle un toque más moderno. Para ello se tienen dos indicadores de barra (j0 y j2), tres indicadores

numéricos (n0, n1 y n2), cinco elementos de texto (t0, t1, t2, t3 y t4) y el botón (bopciones). Estos elementos se pueden ver en la *figura 53*.

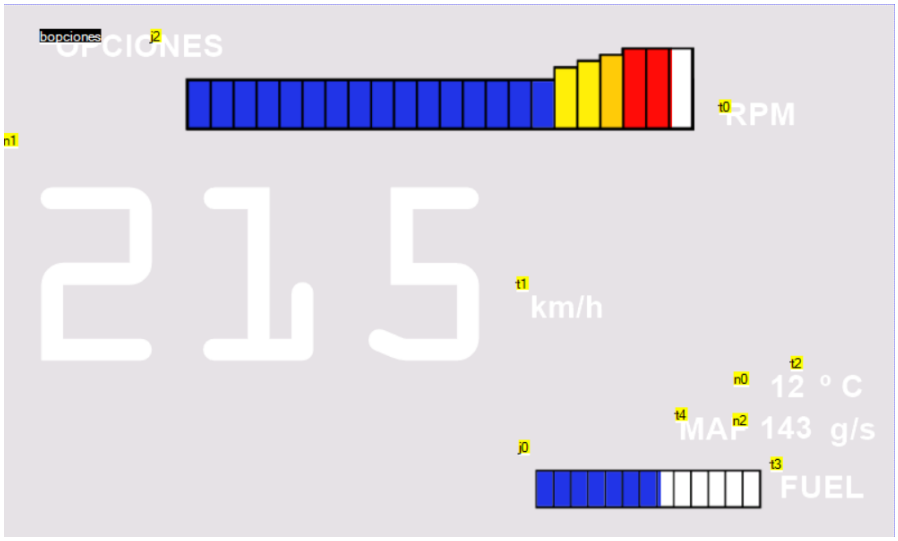


Figura 53.Disposición de elementos usados en la página "deportivo".

A continuación, se citan acompañados de una breve descripción los diferentes elementos de diseño que componen el tablero deportivo.

j0: Barra indicadora personalizada que indicar el nivel de combustible.

j2: Barra indicadora también personalizada para indicar las revoluciones del motor.

n0: indicador numérico utilizado para mostrar el valor de temperatura del refrigerante.

n1: este elemento numérico indica la velocidad a la que va el vehículo.

n2: indicador numérico encargado de mostrar el valor del MAF.

t0: texto “RPM” utilizado para indicar la unidad de las revoluciones del motor.

t1: texto “km/h” para indicar la unidad a la que se muestra la velocidad del vehículo.

t2: texto “ºC” para expresar la unidad a la que se muestra la temperatura del refrigerante.

t3: texto “FUEL” para hacer referencia a que ese indicador de barra representa el nivel de combustible del vehículo.

t4: texto “MAF” para dar significado al valor numérico que sirve para indicar el flujo de masa de aire del vehículo.

bopciones: botón en el que está escrito la palabra “opciones”, indicando que una vez se suelte este botón, la interfaz pasa a la página de opciones.

Los indicadores de barra se han creado para la interfaz de usuario, se han diseñado con un programa de diseño. El software Nextion Editor permite personalizar los indicadores de barra a través de dos imágenes que, a medida del valor a indicar, se ve un porcentaje de una o de otra. En la *figura 54* se muestra el ejemplo de la barra indicadora de las revoluciones del motor (j2).

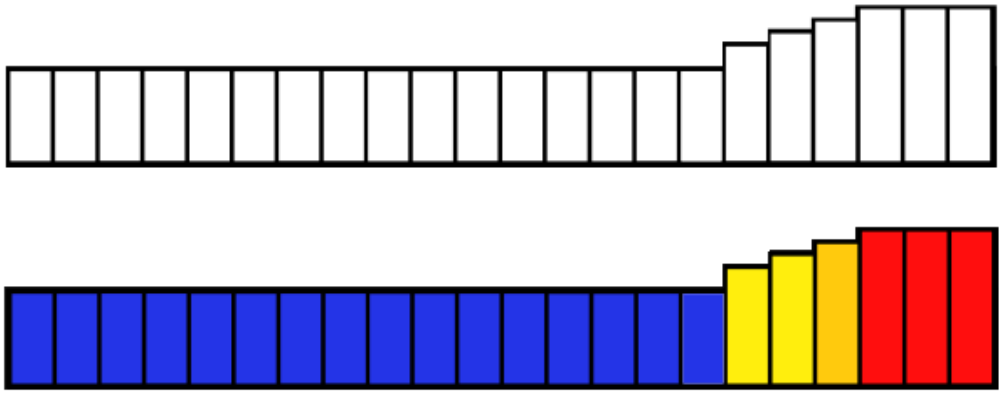


Figura 54. Imágenes creadas barra indicadora de revoluciones.

Estas 2 imágenes creadas, se superponen la una con la otra, si el valor a mostrar es 70 %, se muestra el 70% de la imagen inferior a la izquierda, y el 30% se verá blanco en este caso.

Se tiene lo mismo para la barra indicadore de combustible, como se muestra en la *figura 55*.

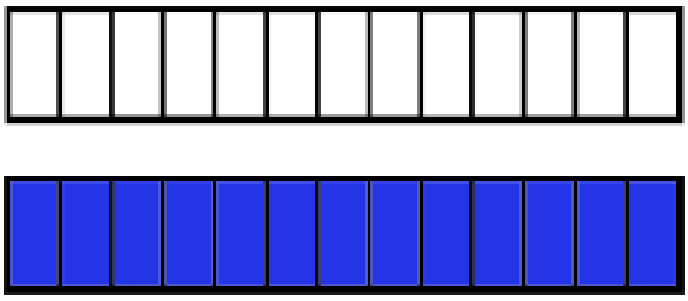


Figura 55. Imágenes creadas para barra indicadora de combustible.

RACING

Para el tipo de tablero racing, se tienen 7 elementos numéricos (n0, n1, n2, n3, n4, n5 y n7), un indicador de aguja (z0), 7 elementos de texto (t0, t1, t3, t4, t5, t6 y t7) y como en los demás tipos de tableros, también se tiene el botón de opciones, que se encuentra en la parte superior izquierda de la pantalla. En la *figura 56*, se muestra la disposición de dichos elementos.

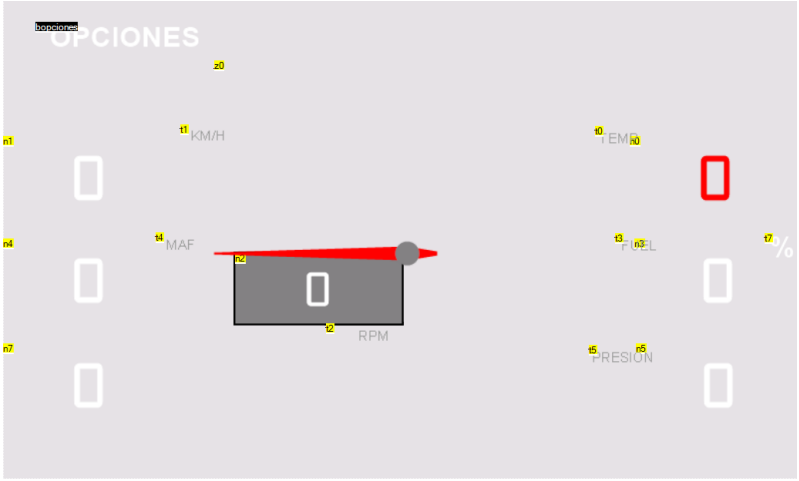


Figura 56.Disposición de elementos usados en la página "racing".

A continuación, se citan la función que se le ha adjudicado a los distintos elementos.

n0: este valor numérico representa la temperatura del líquido refrigerante del motor del vehículo.

n1: muestra la velocidad del vehículo de forma numérica.

n2: el valor de revoluciones por minuto del motor se

n3: indica el porcentaje del nivel de combustible del vehículo.

n4: este elemento numérico muestra el valor del flujo de masa de aire (MAF).

n5: muestra el valor de presión absoluta barométrica

n7: a este elemento no se le ha asignado ninguna función. Solo da un aspecto de simetría.

t0: el texto que muestra este elemento es “TEMP”, indicando que el valor que hay a su derecha es el valor de temperatura del refrigerante del motor.

t1: “KM/H” indica la unidad del valor asociado a este elemento de texto.

t3: en este texto se tiene la palabra “FUEL” indicando que el valor que hay a su lado indica el nivel de combustible.

t4: las letras “MAF” señalan que el valor mostrado es el valor de flujo de masa de aire.

t5: “PRESION” indica que el valor relacionado a este texto es un valor de presión.

t7: este texto contiene “%” indicando que el valor de nivel de combustible es un porcentaje.

z0: este indicador de aguja muestra de manera visual el valor de revoluciones del motor.

bopciones: botón que lleva a la página de opciones de la interfaz.

OPCIONES

Por último, se encuentra la página nombrada como opciones. En esta página se le permite al usuario elegir entre los tres tipos de tableros antes mencionados, o bien, volver al inicio, finalizando la comunicación con el vehículo.

En la *figura 57* se tiene la página de opciones.

Figura 57. Disposición de elementos usados en la página "opciones".

Esta página tiene una interfaz muy sencilla, el texto t0 hace saber que puede escoger el tipo de tablero con los botones b0, b1 y b2. En el margen superior derecho se encuentra el botón b3, para volver al inicio del sistema.

En las *figuras 58, 59, 60 y 61* se muestran las respectivas ventanas de eventos asociados dichos botones.

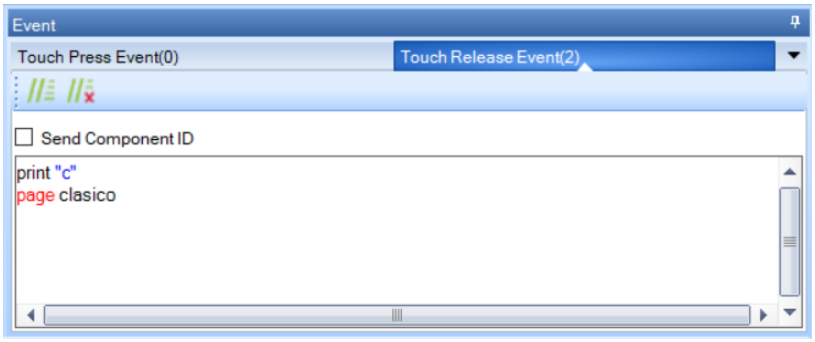


Figura 58. Ventana de eventos asociados al botón b0 de la página "opciones".

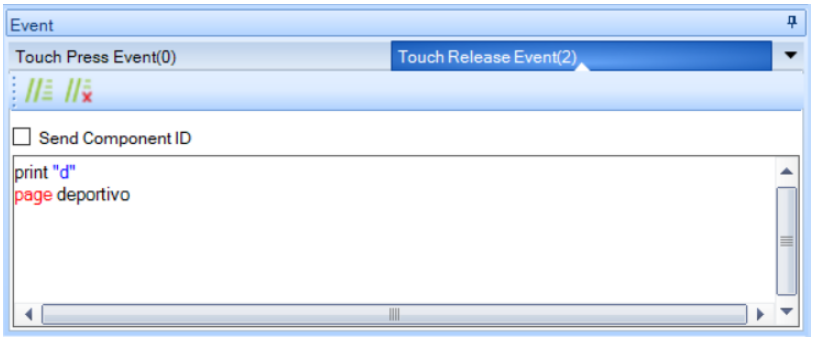


Figura 59. Ventana de eventos asociados al botón b1 de la página "opciones".

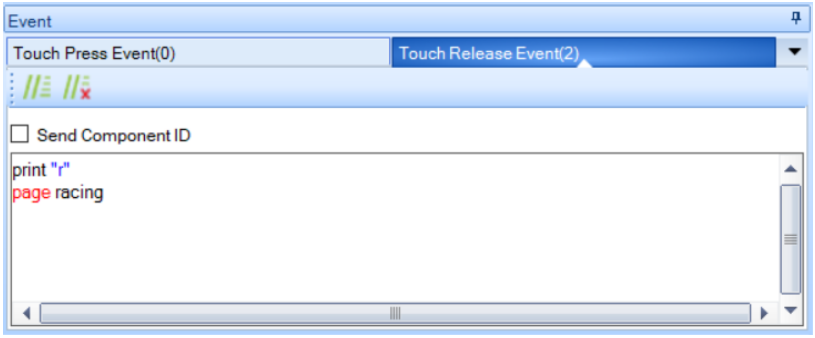


Figura 60. Ventana de eventos asociados al botón b2 de la página "opciones"

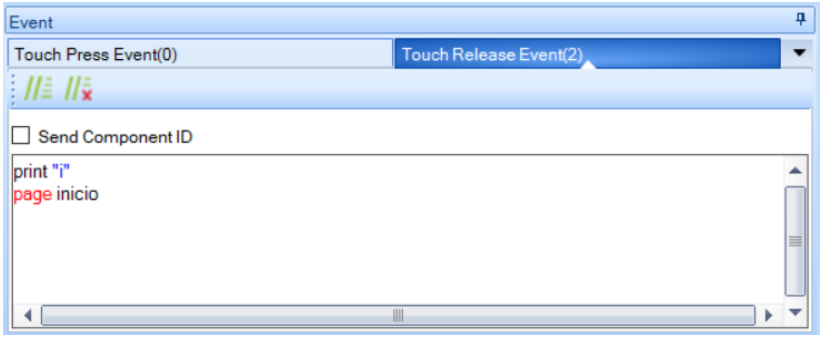


Figura 61. Ventana de eventos asociados al botón b3 de la página "opciones".

Simulación y depuración

En este subapartado se muestra una característica del software Nextion Editor que es de gran ayuda para la persona que se encuentre desarrollando la interfaz para un proyecto.

Esta característica es la de incluir un entorno de simulación, permitiendo a los desarrolladores verificar la funcionalidad y la apariencia sin necesidad de una pantalla física, pudiendo realizar pruebas de interacciones táctiles, como variar elementos deslizantes o pulsar botones, con el fin de comprobar el correcto funcionamiento de los eventos asociados a ellos.

Para en el modo de simulación se debe clicar en el botón “debug”, el cual se encuentra en la barra de herramientas situada en la parte superior de la ventana, tal y como se muestra en la *figura 62*. [9]

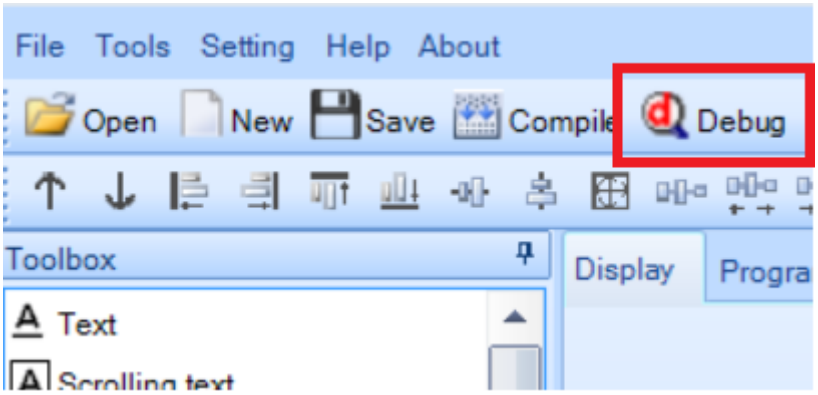


Figura 62. Disposición del botón "Debug" en el software Nextion Editor.

En la *figura 63* se puede observar la ventana de la simulación.

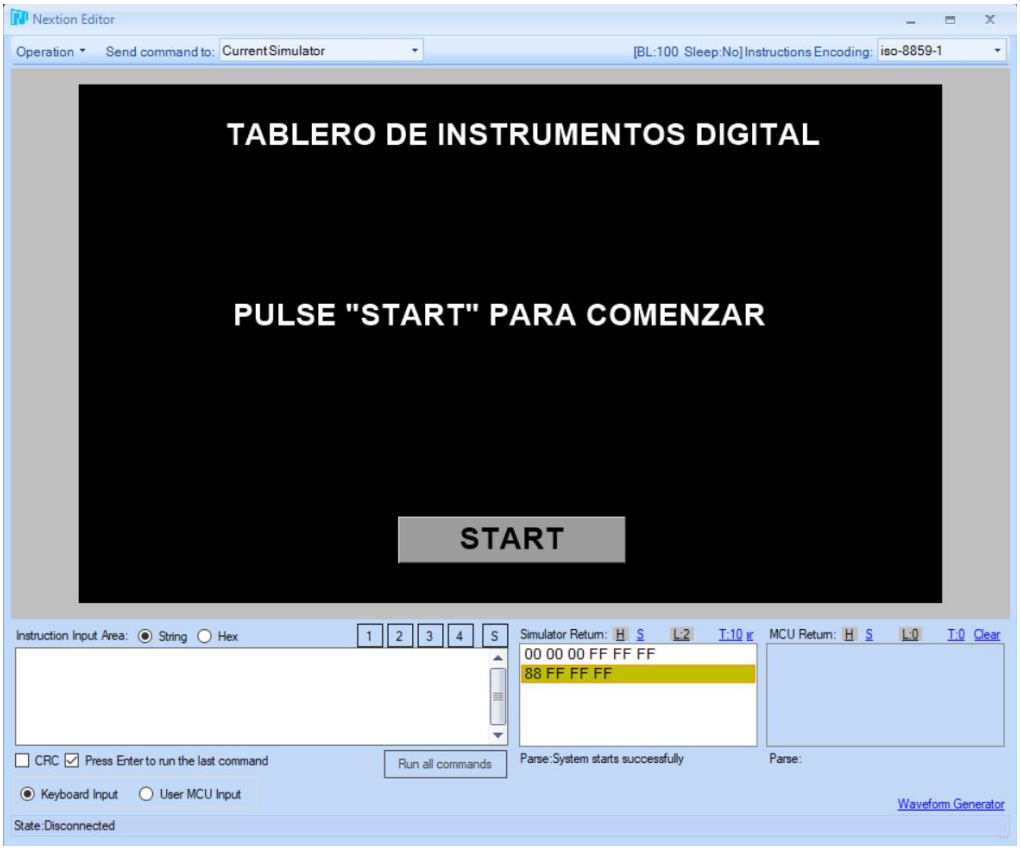


Figura 63. Ventana de simulación del software Nextion Editor.

Para realizar la simulación en conjunto con la placa Arduino se ha de seleccionar la opción “User MCU Input” que se encuentra en la parte inferior izquierda. Una vez seleccionado, se elige el puerto COM en el que se encuentra el Arduino conectado y la velocidad de comunicación, esto se puede observar en la *figura 64*. [9]

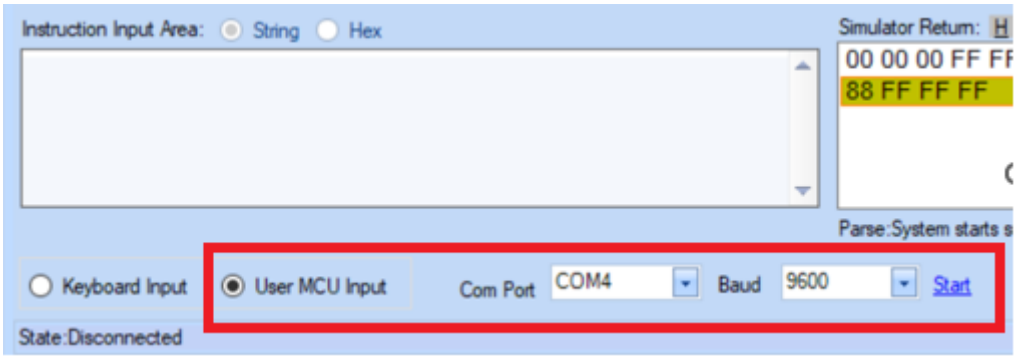


Figura 64. Disposición de la opción "User MCU Input" en la ventana de simulación del software Nextion Editor.

Compilación y transferencia a la pantalla

En el presente subapartado, se comenta el proceso de compilación de los proyectos realizados con el software Nextion Editor y como se transfieren las interfaces creadas a las pantallas Nextion para su posterior funcionamiento.

Una vez finalizada la interfaz, el siguiente paso es compilar el proyecto usando el botón “compile”, que se encuentra junto al botón “debug”. Si el proyecto no tiene errores o configuraciones contradictorias, el software genera los archivos necesarios para su funcionamiento una vez se transfiere a la pantalla física.

Existen varias formas de transferir la interfaz a la pantalla. En el caso de este proyecto se ha usado una tarjeta de memoria SD, formateada al formato FAT32, en la que se pasa el archivo “.tft” e insertar la tarjeta SD en la ranura que tiene la pantalla.

Después de insertar la tarjeta en la pantalla, se debe encender la pantalla, dejar que se complete la carga, apagar la pantalla y extraer la tarjeta. Una vez seguidos estos pasos, la interfaz ya se ha transferido a la pantalla y está lista para su funcionamiento.

InkScape (Diseño de imágenes para la interfaz).

Una parte muy importante en las interfaces de usuario es que queden estéticas, manteniendo una armonía y una congruencia visual. En las páginas de la interfaz de usuario se han creado unas imágenes de fondo con el fin de obtener cierta calidad visual y dar sentido a la disposición de los elementos utilizados para mostrar los datos que se obtienen del vehículo.

Para diseñar las distintas imágenes de fondo se ha usado el software de diseño gráfico Inkscape, el cual proporciona una gran variedad de herramientas de diseño y dibujo.

En el tipo de tablero clásico, se ha diseñado un fondo que complementa los elementos de la interfaz de Nextion Editor. En la *figura 65* se muestra la imagen de fondo del tablero clásico.

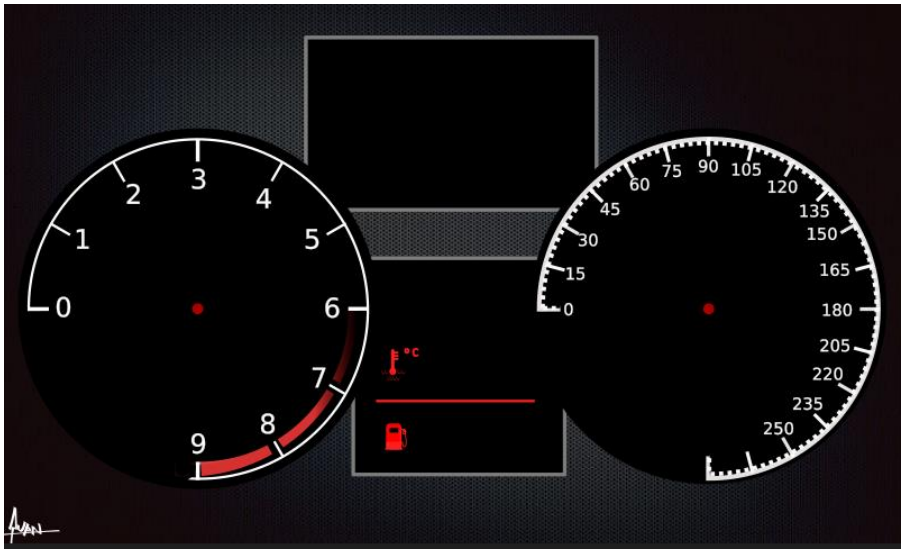


Figura 65. Imagen de fondo del tablero clásico, creada con Inkscape.

Se puede ver como se han diseñado los valores numéricos del tacómetro y del velocímetro, además de los símbolos del combustible y de la temperatura del refrigerante.

Para el tablero deportivo se ha escogido un fondo moderno, al que solo se le han modificado las dimensiones para que coincidan con las dimensiones de la pantalla y la escala de colores. La *figura 66* es el fondo del tablero deportivo.

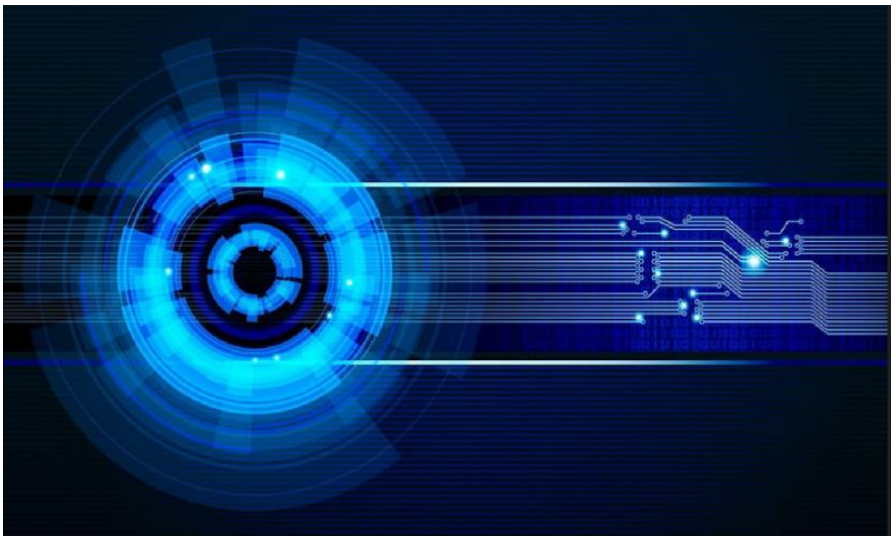


Figura 66. Imagen de fondo del tablero deportivo, modificada con Inkscape.

El fondo del tablero racing se ha diseñado buscando un estilo de competición. En la *figura 67* se muestra el resultado.

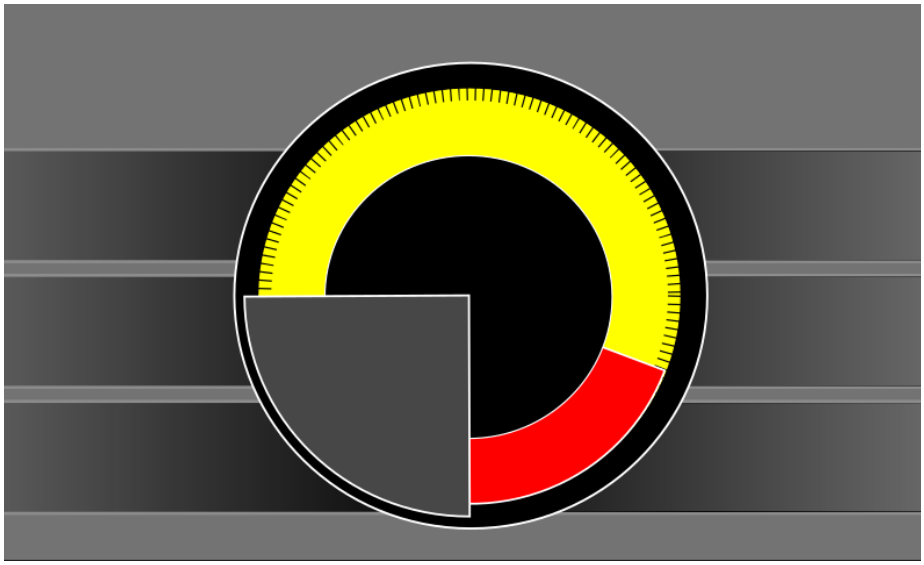


Figura 67. Imagen de fondo del tablero racing, creada con Inkscape.

Capítulo 4. Resultados

En el presente capítulo se muestran los resultados obtenidos una vez se ha desarrollado el tablero de instrumentos digital inalámbrico. Se podrá observar el aspecto físico de los dispositivos, además del aspecto final de la interfaz, una vez se ha importado el archivo a la pantalla.

Aspecto físico del sistema

En este apartado se muestra el aspecto físico del sistema una vez se ha implementado. Para tener una disposición cómoda de los elementos, se ha fabricado una lámina de plástico que va anclada a la parte posterior de la pantalla con la finalidad de poder colocar la placa de Arduino en ella, al igual que el módulo de bluetooth HC-05 y el cableado del sistema, obteniendo así un sistema recogido y organizado.

En la *figura 68* se muestra la parte delantera del sistema.

Figura 68. Aspecto físico del sistema (parte delantera).

En la figura 69 se puede observar como ha quedado la parte posterior de la pantalla, donde se encuentran los demás dispositivos.

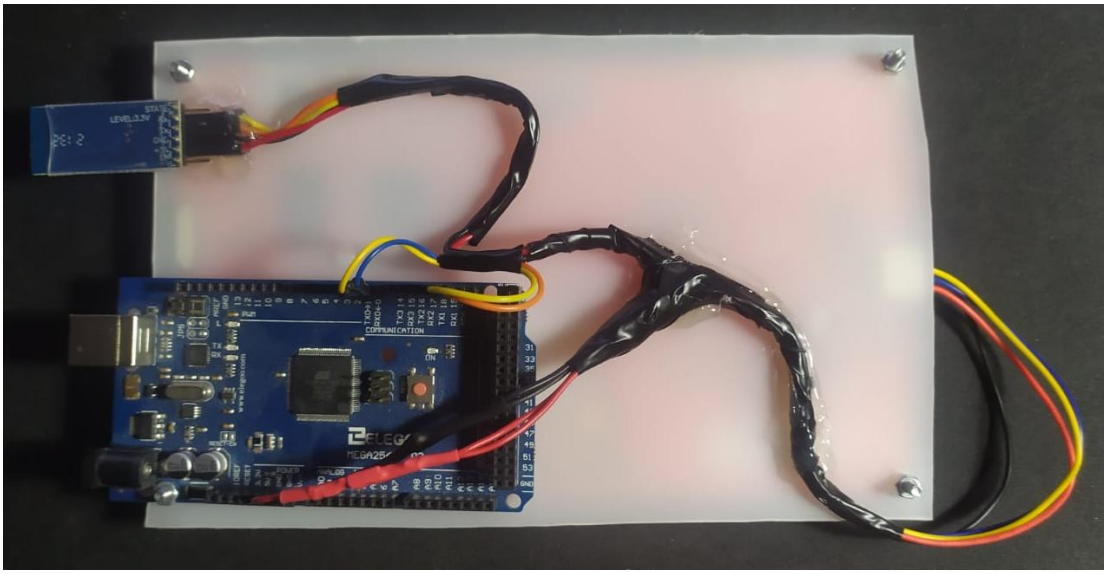


Figura 69. Aspecto físico del sistema (parte trasera)

Interfaz del sistema

En este apartado se enseña el resultado de las diferentes páginas diseñadas para la interfaz del sistema. Se mostrará una comparación visual entre la vista del software Nextion Editor y la interfaz funcionando en la pantalla.

En la *figura 70* y la *figura 71* se muestran la página de inicio en su versión final, una en el simulador y la otra ya cargada en la pantalla.

Figura 70. Página de inicio final (simulación).

Figura 71. Página de inicio final (pantalla física).

Al igual que en las imágenes anteriores, en las *figura 72* y *figura 73* se muestra la comparación de la página de tablero clásico.

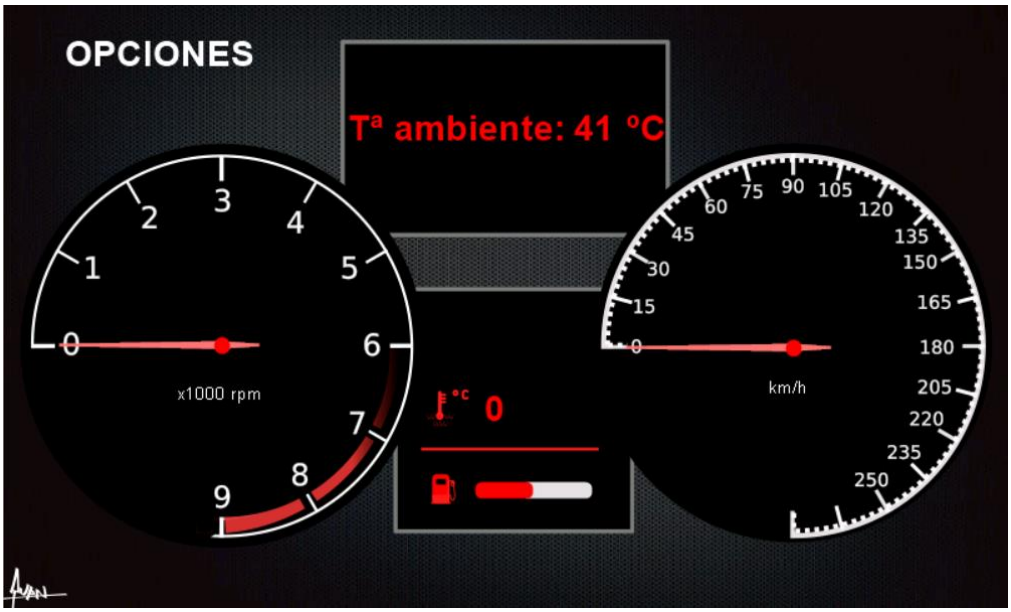


Figura 72. Página de tablero clásico final (simulación).

Figura 73. Página de tablero clásico final (pantalla física).

Siguiendo la misma dinámica, se tienen las *figuras 74* y *75*, que corresponden a la página del tablero de tipo deportivo.

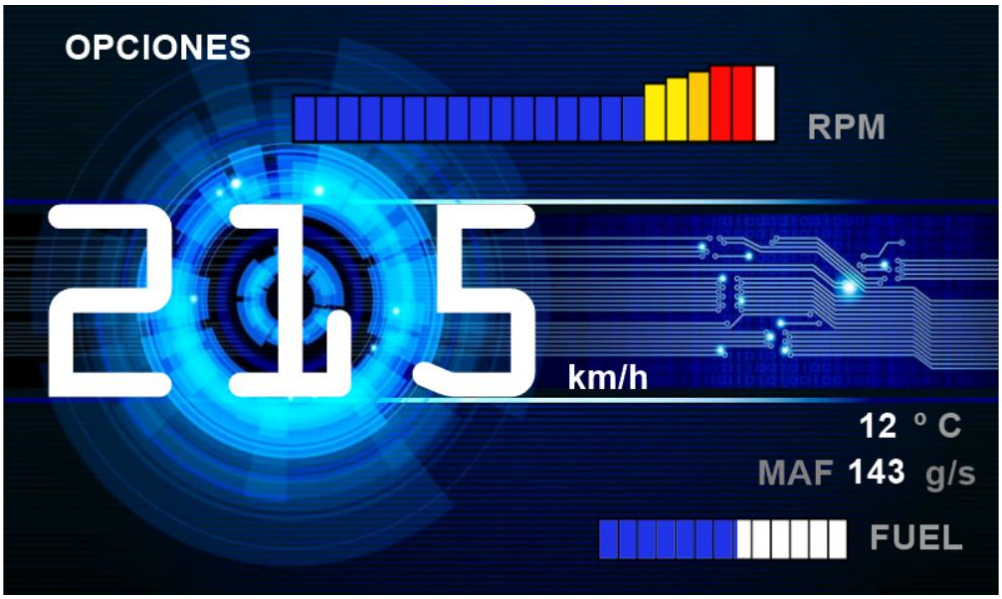


Figura 74. Página de tablero deportivo final (simulación).

Figura 75. Página de tablero deportivo final (pantalla física).

Las *figuras 76* y *77,* muestran la comparación del tercer tipo de tablero, el tipo racing.

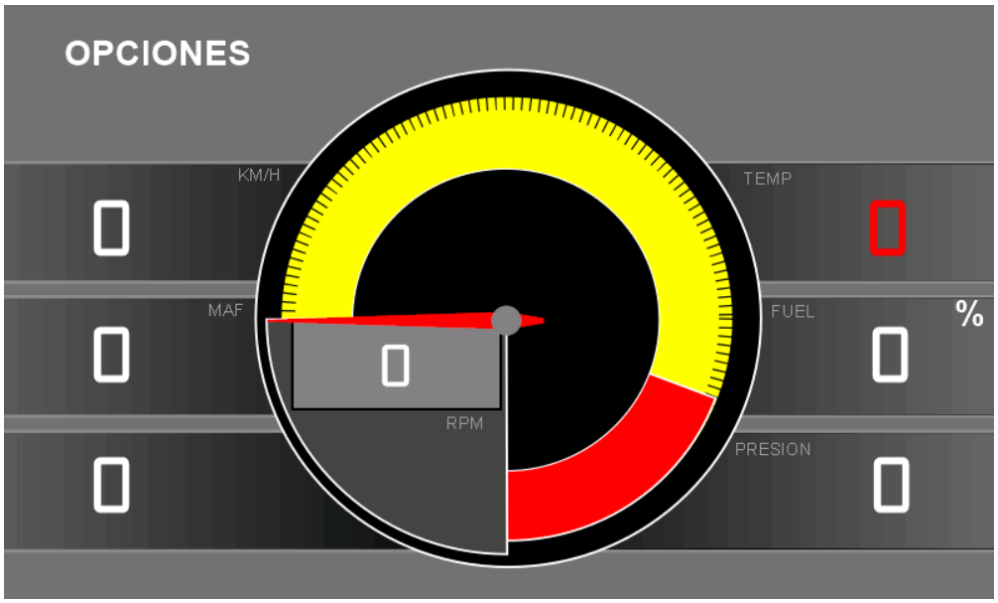


Figura 76. Página de tablero racing (simulación).

Figura 77. Página de tablero racing final (pantalla física).

Por último, la comparación entre el simulador y la pantalla física de la página de opciones del sistema, que se ve en las *figuras 78* y 79.

Figura 78. Página de opciones (simulación).

Figura 79. Página de opciones (pantalla física).

Presupuesto

En la *tabla 3* se puede ver el coste de este proyecto de manera desglosada. Solo se tiene en cuenta los materiales, componentes y dispositivos usados en la implementación, dejando fuera la amortización de herramientas ni la mano de obra.

Tabla 3. Presupuesto del proyecto.

Producto	**Precio unitario (€)**	**Unidades**	**Precio total (€)**
Arduino Mega 2560	16,94	1	19,94
HC-05	3,75	1	3,75
Pantalla táctil Nextion 7”	106,27	1	106,27
Dispositivo diagnóstico ELM327	4,49	1	4,49
Adaptador mechero a USB	1,5	1	1,5
Cableado	-	-	1,6
Tubos termorretráctiles	0,02	2	0,04
Lámina de plástico		1	
	TOTAL	**137,59 €**	

Capítulo 5. Conclusiones

En el presente proyecto se buscaba desarrollar e implementar un tablero de instrumentos que se comunicara con el vehículo de manera inalámbrica a través de la tecnología bluetooth, que permita ver en tiempo real parámetros del vehículo como los muestra un tablero de instrumentos que viene de serie en cualquier vehículo, mostrando información relevante para el conductor del vehículo, como pueden ser las revoluciones del motor, la velocidad del vehículo o la temperatura del líquido refrigerante.

También se planteaba realizar varios tipos de tableros de instrumentos, haciendo que el usuario final pueda seleccionar el que deseé en cada momento. Además de que cada tipo de tablero mostrara la información de una forma diferente, se pretendía que cada tablero se diferencia de los demás en cuanto a la información que se muestra a través de la pantalla.

Esto se ha llevado a cabo con éxito, como se puede ver en los ensayos realizados y los resultados obtenidos, dotando al sistema de las siguientes características:

- Se ha conseguido desarrollar un sistema visual e intuitivo, y que permite al usuario interactuar con el sistema de una manera sencilla y directa. Todo esto debido a la integración de la pantalla táctil en el sistema.

- El usuario final puede escoger entre tres tipos de tableros de instrumentos, permitiendo al usuario un aspecto más clásico o, por el contrario, más deportivo, independientemente de los datos, teniendo en común datos imprescindibles en la conducción como la velocidad, las revoluciones, la temperatura del refrigerante y el nivel de combustible.

- Se ha obtenido un sistema de monitorización de datos estándar, que es compatible con la mayoría de los vehículos del mercado y con el que no es necesario cablear ni desmontar nada en el vehículo para su funcionamiento. Solo es necesario conectar el dispositivo ELM327 al puerto OBD-II del vehículo y dar alimentación al sistema a través de la toma de mechero.

Líneas futuras

Si se analiza el sistema desarrollado, se puede ver que existen muchas posibles mejoras a aplicar usando este proyecto como base, como las siguientes:

- Diseñar más tipos de tableros de instrumentos dando algo más de personalización al usuario final.

- Diseñar un tipo de tablero personalizado, en el que el usuario mediante checkbox, pueda hacer selecciones múltiples de un conjunto de opciones que se muestren en una lista, decidiendo cual quiere que aparezca y cual no.

- Integrar a los tableros de instrumentos los testigos indicadores de fallo (MIL) y, que pulsando el testigo que se haya activado, se le indique a través de una ventana emergente la posible avería o el significado de dicha luz activa.

- Crear un aviso en pantalla cuando la conexión bluetooth se haya establecido, y otro aviso si dicha conexión se pierde.

- Fabricar o adquirir una carcasa que permita su colocación en el vehículo de manera sencilla

Capítulo 6. Referencias

[1]. Diagnosis en Automoción [Diapositivas de PowerPoint]. Escuela de Ingenierías Industriales, Universidad de Málaga. https://eii.cv.uma.es/pluginfile.php/241366/mod_resource/content/5/Presentacion%20Diagnosis%20en%20automoci%C3%B3n.pdf [Último acceso: 24/08/2.023].

[2] Hoja de características de Arduino Mega 2560 rev 3. https://docs.arduino.cc/hardware/mega-2560 . [Último acceso: 02/09/2.023].

[3] Hoja de características de módulo bluetooth HC-05. https://datasheetspdf.com/pdf-file/1418730/ITead/HC-05/1 [Último acceso: 02/09/2.023].

[4] Hoja de características de herramienta de diagnosis bluetooth ELM327. https://www.alldatasheet.com/datasheet-pdf/pdf/197403/ELM/ELM327.html [Último acceso: 02/09/2.023].

[5]Listado de PIDs estándar OBD-II para el modo 01. https://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/104/0/obd2-pids-for-programmers-technical [Último acceso: 28/08/2.023].

[6] Pantalla táctil de tipo capacitivo LCD NX8048P070-011C. https://nextion.tech/datasheets/NX8048P070-011C/ [Último acceso: 02/09/2.023].

[7] Comandos AT para configuración módulo bluetooth HC-05. https://naylampmechatronics.com/blog/24_configuracion-del-modulo-bluetooth-hc-05-usando-comandos-at.html [Último acceso: 02/09/2.023].

[8] Apuntes temario bluetooth. [PDF]. Obtenido de https://biblus.us.es/bibing/proyectos/abreproy/11972/fichero/Cap%C3%ADtulo+2+-+Bluetooth.pdf [Último acceso: 17/08/2.023].

[9]. Cheap Controls [Nombre de usuario en Youtube] (2.020). “Program a button on the nextion for the Arduino with no library tutorial”. [Video]. Youtube. https://www.youtube.com/watch?v=LL4TduSYsZ8&list=PLGOjlGENB8gXVvR9ke2SgF0XTT1dBWmhv [Último acceso: 7/08/2.023].

[10]. Cheap Controls [Nombre de usuario en Youtube] (2.020). “Nextion Display – connected to Arduino in debug mode – Nextion tutorial”. [Video]. Youtube. https://www.youtube.com/watch?v=22pqL1ovKtw&list=PLGOjlGENB8gXVvR9ke2SgF0XTT1dBWmhv&index=29 [Último acceso: 11/08/2.023].

Capítulo 7. Anexos

ELM327

ELM327

OBD to RS232 Interpreter

Description

Almost all new automobiles produced today are required, by law, to provide an interface from which test equipment can obtain diagnostic information. The data transfer on these interfaces follow several standards, none of which are directly compatible with PCs or PDAs. The ELM327 is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and a standard RS232 interface.

The ELM327 builds on improved versions of our proven ELM320, ELM322, and ELM323 interfaces by adding seven CAN protocols to them. The result is an IC that can automatically sense and convert the most common protocols in use today. There are a number of other improvements as well – a high speed RS232 option, battery voltage monitoring, and customizable features through programmable parameters, to name only a few.

The ELM327 requires few external components to make a fully functioning circuit. The following pages discuss the interface details, and show how to use the IC to 'talk' to your vehicle, then concludes with two schematics to get you started.

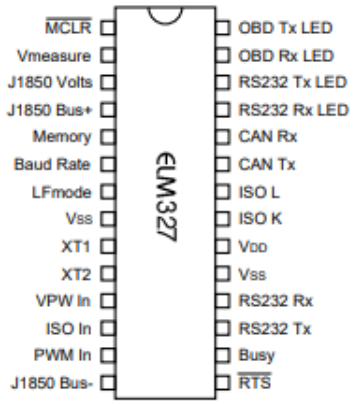
Features

- Supports 12 protocols
- RS232 baud rates to 500Kbps
- Automatically searches for protocols
- Fully configurable with AT commands
- Voltage input for battery monitoring
- Low power CMOS design

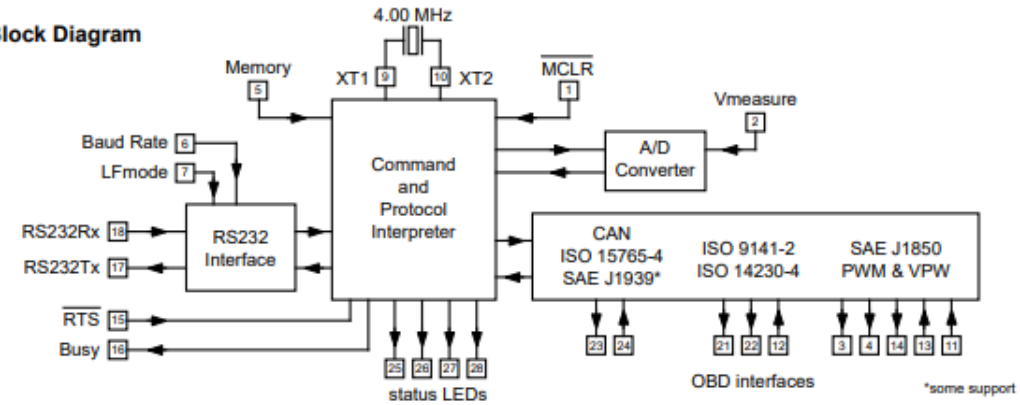
Applications

- Diagnostic trouble code readers
- Automotive scan tools
- Teaching aids

Connection Diagram
PDIP and SOIC
(top view)



Block Diagram



Absolute Maximum Ratings

Storage Temperature..... -65°C to +150°C

Ambient Temperature with Power Applied.....-40°C to +85°C

Voltage on V_{DD} with respect to V_{SS}..... -0.3V to +7.5V

Voltage on any other pin with respect to V_{SS}..... -0.3V to (V_{DD} + 0.3V)

Note:
These values are given as a design guideline only. The ability to operate to these levels is neither inferred nor recommended, and stresses beyond those listed here will likely damage the device.

Electrical Characteristics

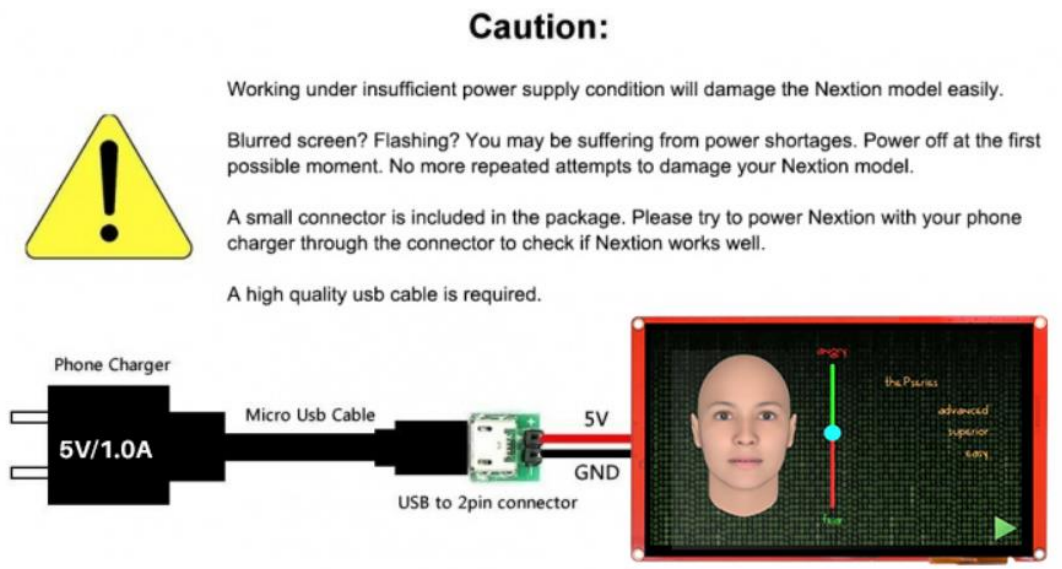
All values are for operation at 25°C and a 5V supply, unless otherwise noted. For further information, refer to note 1 below.

Characteristic	Minimum	Typical	Maximum	Units	Conditions
Supply voltage, V_{DD}	4.5	5.0	5.5	V	
V_{DD} rate of rise	0.05			V/ms	see note 2
Average supply current, I_{DD}		9		mA	see note 3
Input threshold voltage	1.0		1.3	V	all except Schmitt inputs
Schmitt trigger input thresholds rising		2.9	4.0	V	see note 4
Schmitt trigger input thresholds falling	1.0	1.5		V	see note 4
Output low voltage		0.3		V	current (sink) = 10 mA
Output high voltage		4.6		V	current (source) = 10 mA
Brown-out reset voltage	4.11	4.33	4.55	V	
A/D conversion time		7		msec	see note 5

Notes:

1. This integrated circuit is produced with one of Microchip Technology Inc.'s PIC18F2x8x family of devices as the core embedded microcontroller. For further device specifications, and possibly clarification of those given, please refer to the appropriate Microchip documentation (available at http://www.microchip.com/).
2. This spec must be met in order to ensure that a correct power on reset occurs. It is quite easily achieved using most common types of supplies, but may be violated if one uses a slowly varying supply voltage, as may be obtained through direct connection to solar cells or some charge pump circuits.
3. Device only. Does not include any load currents.
4. Pins 1, 11, 12, 13, 15 and 18 (only) have internal Schmitt trigger waveshaping circuitry. All other inputs use standard CMOS (TTL compatible) circuitry.
5. The typical width of the Busy output pulse while the ELM327 interprets the command, measures the voltage, scales it and then transmits the result of a mid-range measurement, with the RS232 rate at 38400 baud.

PANTALLA NEXTION



Nextion Models

Nextion Type	Intelligent Series
Nextion Models	NX8048P070-011C (7.0 inch capacitive touchscreen without enclosure)

Specifications

	Data	Description
Color	65K 65536 colors	16 bit 565, 5R-6G-5B
Layout size	181mm(L)×108mm(W)×9.3mm(H)	NX8048P070-011C
Active Area (A.A.)	164.90mm(L)×100.00mm(W)	
Visual Area (V.A.)	154.08mm(L)×85.92mm(W)	
Resolution	800×480 pixel	Also can be set as 480×800
Touch type	Capacitive	
Touches	> 1 million	
Backlight	LED	
Backlight lifetime (Average)	>30,000 Hours	
Brightness	300nit	0% to 100%, the interval of adjustment is 1%
Weight	310g	

Electronic Characteristics

	Test Conditions	Min	Typical	Max	Unit
Operating Voltage		4.75	5	6.5	V
Operating Current	VCC=+5V, Brightness is 100%	–	430	–	mA
	SLEEP Mode	–	170	–	mA
Power supply recommend: **5V, 1.0A, DC**					

Working Environment & Reliability Parameter

	Test Conditions	Min	Typical	Max	Unit
Working Temperature	5V, Humidity 60%	-20	25	70	°C
Storage Temperature		-30	25	85	°C
Working Humidity	25°C	10%	60%	90%	RH

Interfaces Performance

	Test Conditions	Min	Typical	Max	Unit
Serial Port Baudrate	Standard	2400	9600	921600	bps
Output High Voltage (TXD)	IOH=1mA	3.0	5.0	Vin	V
Output Low Voltage(TXD)	IOL=-1mA		0.1	0.2	V
Input High Voltage(RXD)		3.0	5.0	Vin	V
Input Low Voltage(RXD)		-0.7	0.0	1.3	V
Serial Port Mode	3.3V/5.0V TTL				
Serial Port	4Pin_2.54mm				
USB interface	NO				
SD card socket	Yes (FAT32 format), support maximum 32G Micro SD Card * presence of *.tft file on microSD: socket is exclusive to upgrade Nextion firmware/HMI design * Intelligent Series only: see Instruction Set / Editor Guide for microSD card runtime usage				
Extended IO	8 Digital extended GPIO				
	IO0-IO7 support input, output and component binding event * IO pin / ports are not exclusive, limit current draw to 1mA recommended				
	IO6-IO7 support PWM				
RTC	built-in RTC support (Battery type: CR1220)				

Vin: the input voltage of power supply

Memory Features

Memory Type	Test Conditions	Min	Typical	Max	Unit
FLASH Memory	Store fonts and images			120	MB
User Storage	EEPROM			1024	BYTE
RAM Memory	Store variables			512	KB
Instruction Buffer	Instruction Buffer			4096	BYTE

Audio Features

Speaker	Parameter	Min	Typical	Max	Unit
Power	–	0.5	–	1.5	W
Audio Connector Type: 1.25T-2-2A (1.25mm pitch 2-pin housing)					

HC-05

HC-05

-Bluetooth to Serial Port Module

Overview

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.
Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Specifications

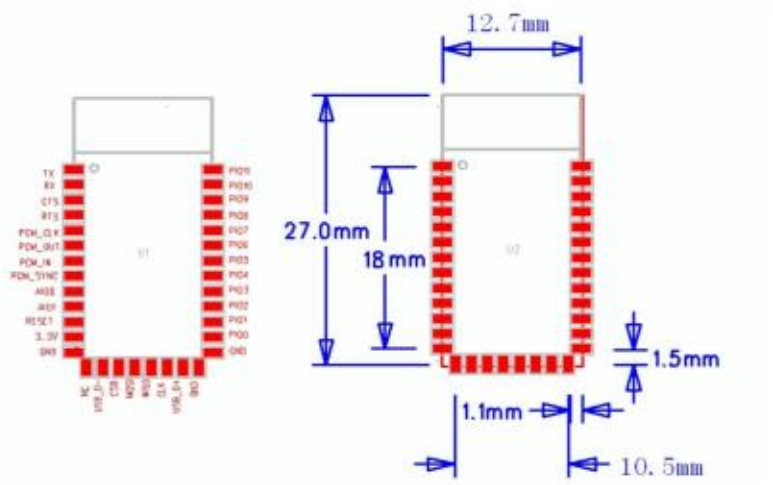
Hardware features

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Software features

- Default Baud rate: 38400, Data bits:8, Stop bit:1,Parity:No parity, Data control: has. Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

Hardware



PIN Name	PIN #	Pad type	Description	Note
GND	13 21 22	VSS	Ground pot	
3.3 VCC	12	3.3V	Integrated 3.3V (+) supply with On-chip linear regulator output within 3.15-3.3V	
AIO0	9	Bi-Directional	Programmable input/output line	
AIO1	10	Bi-Directional	Programmable input/output line	
PIO0	23	Bi-Directional RX EN	Programmable input/output line, control output for LNA(if fitted)	
PIO1	24	Bi-Directional TX EN	Programmable input/output line, control output for PA(if fitted)	
PIO2	25	Bi-Directional	Programmable input/output line	
PIO3	26	Bi-Directional	Programmable input/output line	
PIO4	27	Bi-Directional	Programmable input/output line	
PIO5	28	Bi-Directional	Programmable input/output line	
PIO6	29	Bi-Directional	Programmable input/output line	
PIO7	30	Bi-Directional	Programmable input/output line	
PIO8	31	Bi-Directional	Programmable input/output line	
PIO9	32	Bi-Directional	Programmable input/output line	
PIO10	33	Bi-Directional	Programmable input/output line	
PIO11	34	Bi-Directional	Programmable input/output line	
RESETB	11	CMOS input with weak internal pull-up	Reset if low.input debouncde so must be low for >5MS to cause a reset	
UART_RTS	4	CMOS output, tri-stable with weak internal pull-up	UART request to send, active low	
UART_CTS	3	CMOS input with weak internal pull-down	UART clear to send, active low	
UART_RX	2	CMOS input with weak internal pull-down	UART Data input	
UART_TX	1	CMOS output, Tri-stable with weak internal pull-up	UART Data output	
SPI_MOSI	17	CMOS input with weak internal pull-down	Serial peripheral interface data input	
SPI_CSB	16	CMOS input with weak internal pull-up	Chip select for serial peripheral interface, active low	
SPI_CLK	19	CMOS input with weak internal pull-down	Serial peripheral interface clock	
SPI_MISO	18	CMOS input with weak internal pull-down	Serial peripheral interface data Output	
USB_-	15	Bi-Directional		

USB_+	20	Bi-Directional		
NC	14			
PCM_CLK	5	Bi-Directional	Synchronous PCM data clock	
PCM_OUT	6	CMOS output	Synchronous PCM data output	
PCM_IN	7	CMOS Input	Synchronous PCM data input	
PCM_SYNC	8	Bi-Directional	Synchronous PCM data strobe	

AT command Default:

How to set the mode to server (master):
1. Connect PIO11 to high level.
2. Power on, module into command state.
3. Using baud rate 38400, sent the "AT+ROLE=1\r\n" to module, with "OK\r\n" means setting successes.
4. Connect the PIO11 to low level, repower the module, the module work as server (master).

AT commands: (all end with \r\n)

1. Test command:

Command	Respond	Parameter
AT	OK	-

2. Reset

Command	Respond	Parameter
AT+RESET	OK	-

3. Get firmware version

Command	Respond	Parameter
AT+VERSION?	+VERSION:<Param> OK	Param : firmware version

Example:
AT+VERSION?\r\n
+VERSION:2.0-20100601
OK

4. Restore default

Command	Respond	Parameter
AT+ORGL	OK	-

Default state:

Slave mode, pin code :1234, device name: H-C-2010-06-01 ,Baud 38400bits/s.

5. Get module address

Command	Respond	Parameter
AT+ADDR?	+ADDR:<Param> OK	Param: address of Bluetooth module

Bluetooth address: NAP: UAP : LAP

Example:

AT+ADDR?\r\n

+ADDR:1234:56:abcdef

OK

6. Set/Check module name:

Command	Respond	Parameter
AT+NAME=<Param>	OK	Param: Bluetooth module name (Default :HC-05)
AT+NAME?	+NAME:<Param> OK (/FAIL)	

Example:

AT+NAME=HC-05\r\n set the module name to "HC-05"

OK

AT+NAME=ITeadStudio\r\n

OK

AT+NAME?\r\n

+NAME: ITeadStudio

OK

7. Get the Bluetooth device name:

Command	Respond	Parameter
AT+RNAME?<Param1>	1. +NAME:<Param2> OK 2. FAIL	Param1,Param 2 : the address of Bluetooth device

Example: (Device address 00:02:72:od:22:24，name：ITead)

AT+RNAME? 0002，72，od2224\r\n

+RNAME:ITead

OK

8. Set/Check module mode:

Command	Respond	Parameter
AT+ROLE=<Param>	OK	Param: 0- Slave
AT+ ROLE?	+ROLE:<Param>	

	OK	1-Master 2-Slave-Loop

9. Set/Check device class

Command	Respond	Parameter
AT+CLASS=<Param>	OK	Param: Device Class
AT+ CLASS?	1. +CLASS:<Param> OK 2. FAIL	

10. Set/Check GIAC (General Inquire Access Code)

Command	Respond	Parameter
AT+IAC=<Param>	1.OK 2. FAIL	Param: GIAC (Default : 9e8b33)
AT+IAC	+IAC:<Param> OK	

Example:

AT+IAC=9e8b3f\r\n

OK

AT+IAC?\r\n

+IAC: 9e8b3f

OK

11. Set/Check -- Query access patterns

Command	Respond	Parameter
AT+INQM=<Param>,<Param2>,<Param3>	1.OK 2. FAIL	Param: 0——inquiry_mode_standard 1——inquiry_mode_rssi Param2: Maximum number of Bluetooth devices to respond to Param3: Timeout (1-48 : 1.28s to 61.44s)
AT+ INQM?	+INQM： <Param>,<Param2>,<Param3> OK	

Example:

AT+INQM=1,9,48\r\n

OK

AT+INQM\r\n

+INQM:1, 9, 48

OK

12. Set/Check PIN code:

Command	Respond	Parameter
AT+PSWD=<Param>	OK	Param: PIN code (Default 1234)
AT+ PSWD?	+ PSWD : <Param> OK	

13. Set/Check serial parameter:

Command	Respond	Parameter
AT+UART=<Param>,<Param2>,<Param3>	OK	Param1: Baud Param2: Stop bit Param3: Parity
AT+ UART?	+UART=<Param>,<Param2>,<Param3> OK	

Example:

AT+UART=115200，1,2,\r\n

OK

AT+UART?

+UART:115200,1,2

OK

14. Set/Check connect mode:

Command	Respond	Parameter
AT+CMODE=<Param>	OK	Param: 0 - connect fixed address 1 - connect any address 2 - slave-Loop
AT+ CMODE?	+ CMODE:<Param> OK	

15. Set/Check fixed address:

Command	Respond	Parameter
AT+BIND=<Param>	OK	Param: Fixed address (Default 00:00:00:00:00:00)
AT+ BIND?	+ BIND:<Param> OK	

Example:

AT+BIND=1234，56，abcdef\r\n

OK

AT+BIND?\r\n

+BIND:1234:56:abcdef

OK

16. Set/Check LED I/O

Command	Respond	Parameter
AT+POLAR=<Param1,<Param2>	OK	Param1: 0- PIO8 low drive LED 1- PIO8 high drive LED
AT+ POLAR?	+ POLAR=<Param1>,<Param2> OK	

		Param2: 0- PIO9 low drive LED 1- PIO9 high drive LED

17. Set PIO output

Command	Respond	Parameter
AT+PIO=<Param1>,<Param2>	OK	Param1: PIO number Param2: PIO level 0- low 1- high

Example:

1. PIO10 output high level

AT+PIO=10，1\r\n

OK

18. Set/Check – scan parameter

Command	Respond	Parameter
AT+IPSCAN=<Param1>,<Param2>,<Param3>,<Param4>	OK	Param1: Query time interval Param2：Query duration Param3：Paging interval Param4：Call duration
AT+IPSCAN?	+IPSCAN:<Param1>,<Param2>,<Param3>,<Param4> OK	

Example:

AT+IPSCAN =1234,500,1200,250\r\n

OK

AT+IPSCAN?

+IPSCAN:1234,500,1200,250

19. Set/Check – SHIFF parameter

Command	Respond	Parameter
AT+SNIFF=<Param1>,<Param2>,<Param3>,<Param4>	OK	Param1: Max time Param2: Min time Param3: Retry time Param4: Time out
AT+ SNIFF?	+SNIFF:<Param1>,<Param2>,<Param3>,<Param4> OK	

20. Set/Check security mode

Command	Respond	Parameter
AT+SENM=<Param1>,<Param2>	1. OK 2. FAIL	Param1: 0——sec_mode0+off 1——sec_mode1+non_se
AT+ SENM?	+ SENM:<Param1>,<Param2>	

	OK	cure 2——sec_mode2_service 3——sec_mode3_link 4——sec_mode_unknow n Param2: 0——hci_enc_mode_off 1——hci_enc_mode_pt_t o_pt 2——hci_enc_mode_pt_t o_pt_and_bcast

21. Delete Authenticated Device

Command	Respond	Parameter
AT+PMSAD=<Param>	OK	Param: Authenticated Device Address

Example:
AT+PMSAD =1234,56,abcdef\r\n
OK

22. Delete All Authenticated Device

Command	Respond	Parameter
AT+ RMAAD	OK	-

23. Search Authenticated Device

Command	Respond	Parameter
AT+FSAD=<Param>	1. OK 2. FAIL	Param: Device address

24. Get Authenticated Device Count

Command	Respond	Parameter
AT+ADCN?	+ADCN：<Param> OK	Param: Device Count

25. Most Recently Used Authenticated Device

Command	Respond	Parameter
AT+MRAD?	+ MRAD：<Param> OK	Param: Recently Authenticated Device Address

26. Get the module working state

Command	Respond	Parameter

AT+ STATE?	+ STATE：<Param> OK	Param: "INITIALIZED" "READY" "PAIRABLE" "PAIRED" "INQUIRING" "CONNECTING" "CONNECTED" "DISCONNECTED" "NUKNOW"

27. Initialize the SPP profile lib

Command	Respond	Parameter
AT+INIT	1. OK 2. FAIL	-

28. Inquiry Bluetooth Device

Command	Respond	Parameter
AT+INQ	+INQ: <Param1>, <Param2>, <Param3> .... OK	Param1：Address Param2：Device Class Param3：RSSI Signal strength

Example:

AT+INIT\r\n

OK

AT+IAC=9e8b33\r\n

OK

AT+CLASS=0\r\n

AT+INQM=1,9,48\r\n

At+INQ\r\n

+INQ:2:72:D2224,3E0104,FFBC

+INQ:1234:56:0,1F1F,FFC1

+INQ:1234:56:0,1F1F,FFC0

+INQ:1234:56:0,1F1F,FFC1

+INQ:2:72:D2224,3F0104,FFAD

+INQ:1234:56:0,1F1F,FFBE

+INQ:1234:56:0,1F1F,FFC2

+INQ:1234:56:0,1F1F,FFBE

+INQ:2:72:D2224,3F0104,FFBC

OK

28. Cancel Inquiring Bluetooth Device

Command	Respond	Parameter
AT+ INQC	OK	-

29. Equipment Matching

Command	Respond	Parameter
AT+PAIR=<Param1>,<Param2>	1. OK 2. FAIL	Param1： Device Address Param2： Time out

30. Connect Device

Command	Respond	Parameter
AT+LINK=<Param>	1. OK 2. FAIL	Param： Device Address

Example:

AT+FSAD=1234,56,abcdef\r\n

OK

AT+LINK=1234,56,abcdef\r\n

OK

31. Disconnect

Command	Respond	Parameter
AT+DISC	1. +DISC:SUCCESS OK 2. +DISC:LINK_LOSS OK 3. +DISC:NO_SLC OK 4. +DISC:TIMEOUT OK 5. +DISC:ERROR OK	Param： Device Address

32. Energy-saving mode

Command	Respond	Parameter
AT+ENSNIFF=<Param>	OK	Param： Device Address

33. Exerts Energy-saving mode

Command	Respond	Parameter
AT+ EXSNIFF =<Param>	OK	Param： Device Address

Programa completo Arduino

```
//PROGRAMA TABLERO INSTRUMENTOS DIGITAL E INALÁMBRICO
// Variables para almacenar los datos del vehículo.

int velocidad;
int revoluciones;
int temperatura;
int flujoAire;
int fuel;
int tambiente;
int presion;

//valores mapeados para enviar a la pantalla.

int RPMmap; //valor de revoluciones mapeado para aguja/dial.
int SPEEDmap; // valor de velocidad mapeado para aguja/dial.
int RPMbarra; //valor de revoluciones mapeado para barra de progreso.
boolean START=false; // true= inicia la comunicación, false= detiene la comunicación.
boolean clasico=false; // true= indica que la interfaz se encuentra en el modo clásico.
boolean deportivo=false; // true= indica que la interfaz se encuentra en el modo
deportivo.
boolean racing=false; //true= indica que la interfaz se encuentra en el modo racing.
int datoIN; //dato entrante enviado por la pantalla.


void setup() {
// Configuración de los puertos serie para comunicarse con el HC05.
Serial.begin(9600); //inicia puerto serie que comunica la pantalla con el arduino.
Serial1.begin(38400);// inicia el puerto serie que comunica el HC05 con el arduino.
Serial1.write("ATSP0\r\n"); // configura el ELM327 con protocolo automático
delay(100); // retraso de 100 ms
Serial1.write("ATST14\r\n"); // configura el tiempo de espera de respuesta.
delay(100); // retraso de 100 ms
}


Void loop() { //programa principal
if (Serial.available()){ //si el puerto serie 1 (pantalla) tiene datos en el buffer,
hace lo siguiente:
datoIN=Serial.read();} //guarda el valor del buffer de entrada en la variable datoIN.
if(datoIN=='s'){
START=true;
clasico=true;} // si el dato de entrada es "s", comienza la comunicación y
accede al modo clásico.
if (datoIN=='i'){
START=false;
```

```
clasico=false;
deportivo=false;
racing=false;} //si el dato es "i", para la comunicación y sale del modo en el
que se encuentre.
if(datoIN=='c'){
deportivo=false;
racing=false;
clasico=true;} //si dato es "c", entra en modo clásico.
if(datoIN=='d'){
clasico=false;
racing=false;
deportivo=true;}//si dato es "d", entra en modo deportivo.
if(datoIN=='r'){
clasico=false;
deportivo=false;
racing=true;}//si dato es "r", entra en modo racing.

//si se ha iniciado la documentación y no hay datos en el buffer de entrada de la
pantalla:

if(START==true && Serial.available()==0){
if(clasico==true){ // si está en modo clásico:
lecturaRPM(); //lee revoluciones
printRPM_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaSPEED(); // lee velocidad
printSPEED_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaECT(); // lee temperatura del refrigerante
printECT_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaRPM(); // lee revoluciones
printRPM_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaSPEED(); // lee velocidad
printSPEED_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaAAT(); // lee temperatura del aire del ambiente
printAAT_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaRPM(); // lee revoluciones
printRPM_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaSPEED(); // lee velocidad
printSPEED_CLASICO(); // envía a la pantalla los valores el modo clásico
lecturaFUEL(); // lee nivel de combustible
printFUEL_CLASICO(); // envía a la pantalla los valores el modo clásico
}


if(deportivo==true){ // si está en modo deportivo:
lecturaECT(); // lee temperatura del refrigerante
printECT_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
lecturaRPM(); // lee revoluciones
printRPM_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
lecturaSPEED(); // lee velocidad
printSPEED_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
```

```
lecturaFUEL(); // lee nivel de combustible
printFUEL_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
lecturaRPM(); // lee revoluciones
printRPM_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
lecturaSPEED(); // lee velocidad
printSPEED_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
lecturaMAF(); // lee flujo de masa de aire
printMAF_DEPORTIVO(); // envía a la pantalla los valores el modo deportivo
}


if(racing==true){
lecturaECT(); // lee temperatura del refrigerante
printECT_RACING(); // envía a la pantalla los valores el modo racing
lecturaRPM(); // lee revoluciones
printRPM_RACING(); // envía a la pantalla los valores el modo racing
lecturaSPEED(); // lee velocidad
printSPEED_RACING(); // envía a la pantalla los valores el modo racing
lecturaFUEL(); // lee nivel de combustible
printFUEL_RACING(); // envía a la pantalla los valores el modo racing
lecturaRPM(); // lee revoluciones
printRPM_RACING(); // lee revoluciones
lecturaSPEED(); // lee velocidad
printSPEED_RACING(); // envía a la pantalla los valores el modo racing
lecturaMAF(); // lee flujo de masa de aire
printMAF_RACING(); // envía a la pantalla los valores el modo racing
lecturaRPM(); // lee revoluciones
printRPM_RACING(); // envía a la pantalla los valores el modo racing
lecturaSPEED(); // lee velocidad
printSPEED_RACING(); // envía a la pantalla los valores el modo racing
lecturaBARO(); // lee presión barométrica absoluta
printBARO_RACING(); // envía a la pantalla los valores el modo racing
}
}
}

void lecturaECT(){
Serial1.write("0105\r\n");//------ECT
delay(100);
String ECT_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longECT = ECT_IN.length(); //guarda el valor de la longitud
if(longECT==17){//si la longitud es 17
temperatura = strtol(ECT_IN.substring(12,14).c_str(), NULL, 16)-40 ;}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 12 y 13) y restar 40
if(longECT==18){//si la longitud es 18
temperatura = strtol(ECT_IN.substring(13,15).c_str(), NULL, 16)-40 ;}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 13 y 14) y restar 40
if(longECT==19){//si la longitud es 19
```

```
temperatura = strtol(ECT_IN.substring(14,16).c_str(), NULL, 16)-40 ;}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 14 y 15) y restar 40
}
//-------------------------------------------------------------------------------
void lecturaRPM(){
Serial1.write("010C\r\n");//------RPM
delay(100);//retraso de 100 ms
String RPM_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longRPM = RPM_IN.length();//guarda el valor de la longitud
if(longRPM==20){//si la longitud es 20
int rev1B = strtol(RPM_IN.substring(12,14).c_str(), NULL, 16)*64;//Convertir hexa decimal
posición 12 y 13,por 64
int rev2B = strtol(RPM_IN.substring(15,17).c_str(), NULL, 16)/4 ;//Convertir hexa decimal
posición 15 y 16,entre 4
revoluciones= rev1B+rev2B ;}//sumamos los 2 bytes
if(longRPM==21){//si la longitud es 21
int rev1B = strtol(RPM_IN.substring(13,15).c_str(), NULL, 16)*64;//Convertir hexa
decimal posición 13 y 14,por 64
int rev2B = strtol(RPM_IN.substring(16,18).c_str(), NULL, 16)/4 ;//Convertir hexa decimal
posición 16 y 17,entre 4
revoluciones= rev1B+rev2B ; //sumamos los 2 bytes
}
}
//-------------------------------------------------------------------------------
void lecturaSPEED(){
Serial1.write("010D\r\n");//------SPEED
delay(100);//retraso de 100 ms
String SPEED_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longSPEED= SPEED_IN.length();//guarda el valor de la longitud
if(longSPEED==17){//si la longitud es 17
velocidad = strtol(SPEED_IN.substring(12,14).c_str(), NULL, 16);}
//convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion
12 y 13) y resta 40
if(longSPEED==18){//si la longitud es 18
velocidad = strtol(SPEED_IN.substring(13,15).c_str(), NULL, 16);}
//convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion
13 y 14) y resta 40
if(longSPEED==19){//si la longitud es 19
velocidad = strtol(SPEED_IN.substring(14,16).c_str(), NULL, 16);}
//convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion
14 y 15) y resta 40
}
void lecturaMAF(){
Serial1.write("0110\r\n");//------MAF
delay(100);//retraso de 100 ms
String MAF_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longMAF= MAF_IN.length();//guarda el valor de la longitud
if(longMAF==20){//si la longitud es 20
int maf1B = strtol(MAF_IN.substring(12,14).c_str(), NULL, 16)*256/100;//Convertir
```

```
hexa a dec posición 12 y 13, por 256/100
int maf2B = strtol(MAF_IN.substring(15,17).c_str(), NULL, 16)/100 ;//Convertir hexa
a dec posición 15 y 16,entre 100
flujoAire= maf1B+maf2B ;} //suma de ambos bytes
if(longMAF==21){//si la longitud es 21
int maf1B = strtol(MAF_IN.substring(13,15).c_str(), NULL, 16)*256/100;//Convertir
hexa a dec posición 13 y 14,por 256/100
int maf2B = strtol(MAF_IN.substring(16,18).c_str(), NULL, 16)/100 ;//Convertir hexa
a dec posición 16 y 17,entre 100
flujoAire= maf1B+maf2B ;}//suma de ambos bytes
}
void lecturaFUEL(){
Serial1.write("012F\r\n");//------FUEL
delay(100);//retraso de 100 ms
String FUEL_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longFUEL = FUEL_IN.length();//guarda el valor de la longitud
if(longFUEL==17){//si la longitud es 17
fuel = strtol(FUEL_IN.substring(12,14).c_str(), NULL, 16)*100/255;
//convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion
12 y 13) y multiplica 100/255
}
if(longFUEL==18){//si la longitud es 18
fuel = strtol(FUEL_IN.substring(13,15).c_str(), NULL, 16)*100/255;
//convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion
13 y 14) y multiplica 100/255
}
if(longFUEL==19){//si la longitud es 19
fuel = strtol(FUEL_IN.substring(14,16).c_str(), NULL, 16)*100/255;
//convierte de hexadecimal a decimal un fragmento de la cadena de caracteres(posicion
14 y 15) y multiplica 100/255
}
}
void lecturaBARO(){
Serial1.write("0133\r\n");//------ presion absoluta barometrica ABP
delay(100);//retraso de 100 ms
String BARO_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longBARO = BARO_IN.length();//guarda el valor de la longitud
if(longBARO==17){//si la longitud es 17
presion = strtol(BARO_IN.substring(12,14).c_str(), NULL, 16);}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 12 y 13)
if(longBARO==18){//si la longitud es 18
presion = strtol(BARO_IN.substring(13,15).c_str(), NULL, 16);}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 13 y 14)
if(longBARO==19){//si la longitud es 19
presion = strtol(BARO_IN.substring(14,16).c_str(), NULL, 16);}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 14 y 15)
}
```

```
void lecturaAAT(){
Serial1.write("0146\r\n");//------ AAT TEMPERATURA AIRE AMBIENTE
delay(100); //retraso de 100 ms
String AAT_IN = Serial1.readStringUntil(62);//lee y guarda en string
int longAAT = AAT_IN.length();//guarda el valor de la longitud
if(longAAT==17){//si la longitud es 17
tambiente = strtol(AAT_IN.substring(12,14).c_str(), NULL, 16)-40;}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 12 y 13) y resta 40
if(longAAT==18){//si la longitud es 18
tambiente = strtol(AAT_IN.substring(13,15).c_str(), NULL, 16)-40;}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 12 y 13) y resta 40
if(longAAT==19){//si la longitud es 19
tambiente = strtol(AAT_IN.substring(14,16).c_str(), NULL, 16)-40;}
//convierte de hexadecimal a decimal un fragmento de la cadena de
caracteres(posicion 12 y 13) y resta 40
}

//-------------------------PRINT CLASICO-----------------------

void printECT_CLASICO(){
if(temperatura>-40){//si la temperatura es mayor que -40
Serial.print("n0.val="); //envía (nombre del objeto).val=
Serial.print(temperatura); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
}}

void printRPM_CLASICO(){
RPMmap = map(revoluciones,0,9000,0,270);
if(RPMmap>=0) {
Serial.print("z0.val="); //envía (nombre del objeto).val=
Serial.print(RPMmap); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
}
}


void printSPEED_CLASICO(){
if(velocidad>=0){
Serial.print("z1.val="); //envía (nombre del objeto).val=
Serial.print(velocidad); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
} }
```

```
void printFUEL_CLASICO(){
Serial.print("j0.val="); //envía (nombre del objeto).val=
Serial.print(fuel); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
}


void printAAT_CLASICO(){
Serial.print("n1.val="); //envía (nombre del objeto).val=
Serial.print(tambiente); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
}


//------------------------PRINT DEPORTIVO-----------------------

void printECT_DEPORTIVO(){
Serial.print("n0.val="); //envía (nombre del objeto).val=
Serial.print(temperatura); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
}


void printRPM_DEPORTIVO(){
RPMbarra = map(revoluciones,0,16360,0,100);
Serial.print("j2.val="); //envía (nombre del objeto).val=
Serial.print(RPMbarra); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}


void printSPEED_DEPORTIVO(){
Serial.print("n1.val="); //envía (nombre del objeto).val=
Serial.print(velocidad); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}
```

```
void printFUEL_DEPORTIVO(){
Serial.print("j0.val="); //envía (nombre del objeto).val=
Serial.print(fuel); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}


void printMAF_DEPORTIVO(){
Serial.print("n2.val="); //envía (nombre del objeto).val=
Serial.print(flujoAire); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}


//-------------------------PRINT RACING-----------------------

void printECT_RACING(){
Serial.print("n0.val="); //envía (nombre del objeto).val=
Serial.print(temperatura); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); //final de la trama
}


void printRPM_RACING(){
RPMmap = map(revoluciones,0,9000,0,270);
Serial.print("n2.val="); //envía (nombre del objeto).val=
Serial.print(revoluciones); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama

Serial.print("z0.val="); //envía (nombre del objeto).val=
Serial.print(RPMmap); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}


void printSPEED_RACING(){
Serial.print("n1.val="); //envía (nombre del objeto).val=
Serial.print(velocidad); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
```

```
Serial.write(0xFF);//final de la trama
}


void printFUEL_RACING(){
Serial.print("n3.val="); //envía (nombre del objeto).val=
Serial.print(fuel); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}


void printMAF_RACING(){
Serial.print("n4.val="); //envía (nombre del objeto).val=
Serial.print(flujoAire); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}


void printBARO_RACING(){
Serial.print("n5.val="); //envía (nombre del objeto).val=
Serial.print(presion); //valor del parámetro
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF);//final de la trama
}
```