EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO SUPERIOR DE LIBRES

Organismo Público Descentralizado del Gobierno del Estado de Puebla

"IMPLEMENTACIÓN DE PROTOCOLO DE COMUNICACIÓN SERIAL CAN BUS EN CIRCUITO DE CONTROL DE SISTEMAS ELECTRÓNICOS DEL AUTOMÓVIL; VENTANILLAS, ESPEJOS, LUCES FRONTALES Y TRASERAS"

OPCIÓN I

TESIS PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:
LICENCIATURA EN INGENIERÍA EN SISTEMAS AUTOMOTRICES

PRESENTA:
DANIEL CONTRERAS HERNANDEZ

LIBRES, PUEBLA, MAYO, 2024.

ANEXO XXXIII. FORMATO DE LIBERACIÓN DE PROYECTO PARA LA TITULACIÓN INTEGRAL

Fecha: 09 de Mayo de 2024
Asunto: Liberación de proyecto para la titulación integral

Mtra. Berenice Victoria Corte
Encargada de la Jefatura del Departamento de Estudios Profesionales
P r e s e n t e.

Por este medio informo que ha sido liberado el siguiente proyecto para la titulación integral:

Nombre del estudiante y/o egresado: Contreras Hernandez Daniel
Carrera: Licenciatura en Ingeniería en Sistemas Automotrices
No. de control: 16940331
Nombre del proyecto: Implementación de Protocolo De Comunicación Serial CAN BUS en Circuito de Control de Sistemas Electrónicos del Automóvil; Ventanillas, Espejos, Luces Frontales Y Traseras
Producto: I, Tesis profesional

Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestros egresados.

A T E N T A M E N T E
"POR UNA CULTURA CIENTÍFICA, TECNOLÓGICA Y SUSTENTABLE"

Ing. Humberto Morales Suárez
Jefe de la División de Ingeniería en Sistemas Automotrices

M.C. Román Pérez Saldaña
Asesor

M. I. Angel David Flores Torres
Revisor

S.E.P.
INSTITUTO TECNOLÓGICO SUPERIOR DE LIBRES
DIVISIÓN DE INGENIERÍA EN SISTEMAS AUTOMOTRICES

M. S. C. Elmar Montiel Jiménez
Revisor

M. C. José Tlamani Amador
Revisor

ÍNDICE GENERAL

ÍNDICE DE TABLAS

Página

ÍNDICE DE FIGURAS

Página

CAPÍTULO I
GENERALIDADES

1.1 Introducción

La incorporación de los sistemas electrónicos en automóviles ha aumentado cada vez más, desde la gestión del motor se ha ido ampliando la aplicación del control electrónico y actualmente podemos encontrarlo en todos los sistemas del automóvil: motor, tracción, seguridad, confort y comunicación. En la actualidad la industria automotriz emplea cinco protocolos de comunicación para el control de estos sistemas (MOST, FLEX RAY, CAN, LIN Y VAN).

Dentro de los sistemas que se rigen bajo el protocolo CAN BUS se observa la reducción significativa del número de conexiones, el espacio, la distancia de cableado y el peso del vehículo en comparación de los otros protocolos de comunicación, además, es utilizado para diagnosticar el correcto funcionamiento de las unidades de control que se encuentran integradas en estos sistemas, al igual que demás elementos como sensores y actuadores.

El protocolo CAN BUS (como se menciona anteriormente), es uno de los más utilizados dentro del automóvil, aunque hasta hoy día resulta difícil el acceso a información detallada sobre la estructura de programación por la cual se rige este protocolo y la forma en la que se genera él envió de información en código binario de las señales provenientes de sensores y actuadores integrados en una unidad de control hacía el resto de las unidades interconectadas en la red del BUS.

En este trabajo se propone el diseño y desarrollo de un circuito de control para los sistemas electrónicos del automóvil; ventanillas, espejos, luces frontales y traseras basados en una estructura de comunicación CAN BUS con el objetivo de recabar información bibliográfica y práctica que sirva como medio de apoyo a generaciones posteriores en la Ingeniería en Sistemas Automotrices para fortalecer sus conocimientos e impulsar a futuras investigaciones.

1.2 Planteamiento del problema

En la Ingeniería en Sistemas Automotrices del Tecnológico Nacional de México, Campus Libres se cuenta con material didáctico (tableros prácticos), de diferentes sistemas como son: sistemas de iluminación, info-entretenimiento, frenado, inyección entre otros; los cuales están integrados por Hardware y Software (protocolos de comunicación; LIN, MOST Y CAN BUS) como lo estarían en un vehículo real, dichos tableros tienen por objetivo ayudar a los estudiantes a fortalecer sus conocimientos de manera práctica, estudiando y analizando los componentes que los integran y a la vez permiten simular posibles fallas del vehículo. A pesar de contar con dichas herramientas didácticas, no se cuenta con información que brinde de manera detallada la estructura de los softwares (programas) y como estos pueden transportar información del estado de sensores y actuadores que conforman a los sistemas integrados en cada tablero.

Por lo que en la carrera de Ingeniería en Sistemas Automotrices se tiene la necesidad latente de conocer la base en la cual se fundamenta la implementación de un protocolo de comunicación y la manera en la que se adapta en sistemas específicos del vehículo.

1.3 Objetivos

1.3.1 Objetivo General

Analizar, desarrollar y aplicar el protocolo de comunicación serial CAN BUS mediante la creación de un circuito de control de sistemas electrónicos del automóvil; ventanillas, espejos, luces frontales y traseras integrados en un tablero didáctico.

1.3.2 Objetivos Específicos

- Investigar y comprender el comportamiento del protocolo de comunicación serial CAN BUS.
- Conocer las ventajas que brinda el protocolo de comunicación CAN BUS.
- Desarrollar códigos fuente de programación fundamentado en protocolo CAN BUS.
- Diseñar un circuito de control para los sistemas electrónicos de ventanillas, espejos, luces frontales y traseras del automóvil.
- Elaborar un tablero didáctico que integre los sistemas electrónicos abordados en el proyecto.
- Evaluar el comportamiento del circuito de control integrado en el tablero didáctico.

1.4 Hipótesis

El empleo del protocolo de comunicación CAN BUS dentro de los sistemas electrónicos del automóvil sugiere la reducción de cable eléctrico a integrar, además, de una relevante velocidad de comunicación entre ECU´S interconectadas que se pudieran encontrar al interior de un automóvil.

1.5 Justificación

La Industria Automotriz ha crecido a pasos agigantados en las últimas décadas, dado que día a día se incorporan componentes y softwares cada vez más especializados para el diseño y creación de sistemas complejos dentro del automóvil. Actualmente los protocolos de comunicación principalmente usados en la industria automotriz son (LIN, VAN, MOST, FLEXRAY y CAN BUS) siendo este último el de mayor participación en los sistemas del vehículo.

Dentro de las principales ventajas que brinda el empleo del protocolo CAN BUS

en el automóvil son:

- La comunicación se da entre los módulos integrados lo que permite diagnosticar el funcionamiento de cada uno de ellos y de los sensores debido a su red multimaestra y estandarizada, esto quiere decir que todos los sistemas con sensores y actuadores envían y reciben información para monitorear el comportamiento del vehículo.
- Este sistema reemplaza a los modelos tradicionales de comunicación que utilizaban gran cantidad de cables.
- Otra ventaja es que el protocolo es robusto frente a perturbaciones eléctricas e interferencias electromagnéticas, ideal para aplicaciones críticas donde se demande seguridad, puesto que las tramas CAN (señales) se priorizan por números de identificación, es decir, los datos de máxima prioridad obtienen acceso inmediato al BUS, sin interrumpir otras tramas.

En este trabajo se propone el diseño y desarrollo de un circuito de control para los sistemas electrónicos del automóvil; ventanillas, espejos, luces frontales y traseras, basados en una estructura de protocolo de comunicación CAN BUS con el objetivo de que generaciones posteriores en la Ingeniería en Sistemas Automotrices tengan acceso a esta herramienta para fortalecer sus conocimientos e impulsar a futuras investigaciones.

1.6 Alcance del proyecto

Se pretende conocer y analizar el funcionamiento del protocolo de comunicación serial CAN BUS actualmente implementado en la industria automotriz, al mismo tiempo, comprender la estructura de programación en la que se sustenta e interactúa este protocolo con los circuitos de control encargados de llevar a cabo el correcto funcionamiento de diversos sistemas electrónicos dentro del automóvil, lo anterior con el objetivo principal de crear, desarrollar e implementar un circuito de control para los sistemas específicos de ventanillas, espejos retrovisores, luces frontales y traseras de

un vehículo, para posteriormente representar estos sistemas de manera física en un tablero didáctico.

1.7 Limitaciones

- Difícil acceso a información de circuitos electrónicos de marcas específicas de automóviles.
- Cierta parte de los dispositivos electrónicos a emplear no se encuentran disponibles en la región.
- Acceso limitado a herramientas del ITSL complementarias para el desarrollo y elaboración del presente proyecto.
- Escasa información de dominio público que represente el origen y funcionamiento del protocolo de comunicación CAN BUS.

CAPÍTULO II
MARCO TEÓRICO

2.1 ¿Qué es el CAN BUS?

El CAN BUS (Controller Area Network) por sus siglas en inglés, es un bus automotriz desarrollado por Bosch, que permite que los microcontroladores y dispositivos se comuniquen entre sí dentro de un vehículo sin una computadora host. El CAN BUS es un protocolo basado en mensajes, diseñado específicamente para aplicaciones automotrices, pero ahora también se usa en otras áreas como aeroespacial, automatización industrial y equipos médicos. (AUTOMOTRIZ, 2020)

2.2 Características del sistema CAN BUS

- La información transmitida dentro del sistema se encuentra en código binario.
- El protocolo CAN está orientado a mensajes, esto quiere decir que la información que se envía de un dispositivo a otro primero se descompone en mensajes, los cuales se identifican y analizan por separado para ser enviados por tramas.
- El canal de comunicación CAN es bidireccional (un dispositivo es emisor y receptor de mensajes al mismo tiempo).
- Los mensajes se transmiten por orden de prioridad. (Alarms, 2020)

2.3 Componentes generales de un sistema CAN BUS

Consta de un controlador, un transceptor, dos elementos finales del BUS y dos cables para la transmisión de datos, con excepción de los cables del BUS, todos los componentes están alojados en las unidades de control. (AG, 2022)

- El controlador CAN recibe del microprocesador, en la unidad de control, los datos que han de ser transmitidos. Los acondiciona y los pasa al transceptor CAN. Asimismo, recibe los datos procedentes del transceptor CAN, los

acondiciona asimismo y los pasa al microprocesador en la unidad de control.

- El transceptor CAN es un transmisor y un receptor. Transforma los datos del controlador CAN en señales eléctricas y transmite éstas sobre los cables del CAN BUS. Asimismo, recibe los datos y los transforma para el controlador CAN.
- El elemento final del bus de datos es una resistencia (120 Ohm). Evita que los datos transmitidos sean devueltos en forma de eco de los extremos de los cables y que se falsifiquen los datos.
- Los cables del bus de datos funcionan de forma bidireccional y sirven para la transmisión de los datos. Se denominan con las designaciones CAN HIGH (señales de nivel lógico alto) y CAN LOW (señales de nivel lógico bajo). (AG, 2022). (Ver Figura 2.1)

Para evitar influencias parásitas electromagnéticas y emisiones parásitas, que puedan influir en la transmisión de datos o incluso puedan falsificarlas es necesario retorcer conjuntamente los dos alambres del bus de datos. Es preciso tener en cuenta la distancia o paso de la unión retorcida. (AG, 2022)

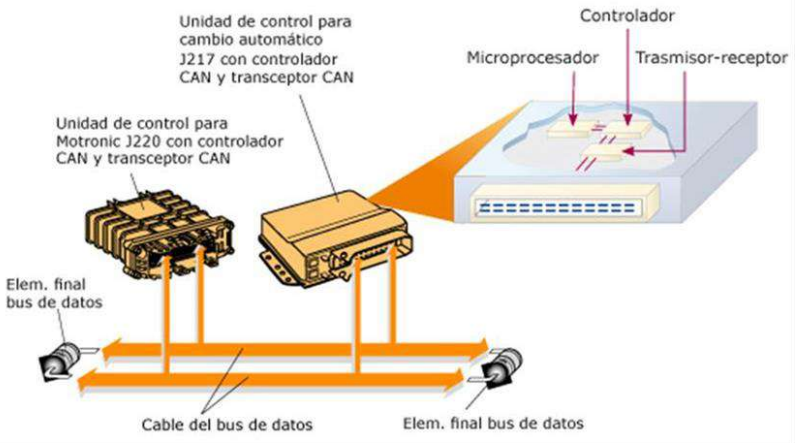


Figura 2.1 Componentes generales de un sistema CAN BUS

2.4 Normalización del CAN BUS

El protocolo CAN esta normalizado bajo el estándar ISO 11898, del cual contiene diversas normas específicas para distintos aspectos del protocolo y diversos tipos de funcionamiento. Por ejemplo, la norma ISO 11898-2 estandariza el protocolo CAN de alta velocidad, pudiendo alcanzar velocidades de hasta 1 MB/s, o la norma ISO 11898-3, que estandariza el protocolo CAN de baja velocidad tolerante a fallos 125 Kbits/s. (Martínez Requena & García Martín, 2017)

En la siguiente tabla se representa la velocidad de transmisión de datos con respecto a la distancia total del BUS. (Ver Tabla 2.1).

Longitud del BUS	Velocidad en bits/s	Tiempo máximo de transmisión*
Hasta 25 m	1 Mbit/s	129 µS
Hasta 100 m	500 Kbit/s	258 µS
Hasta 500 m	125 Kbit/s	1032 µS
Hasta 1000 m	50 Kbit/s	2580 µS
* mensajes de 129 bits de longitud		

Tabla 2.1 Velocidad de transmisión del CAN BUS con respecto a la distancia

2.5 Funcionamiento de protocolo de comunicación CAN BUS

El controlador CAN está conectado a todos los componentes de la red a través de estos dos cables (CAN HIGH y CAN LOW), cada nodo de red tiene un identificador único. Todas las ECU en el bus están efectivamente en paralelo y es por eso que todos los nodos ven todos los datos, todo el tiempo. Un nodo solo responde cuando detecta su propio identificador. Los nodos individuales se pueden eliminar de la red sin afectar a los otros nodos. (AUTOMOTRIZ, 2020)

Cuando el BUS CAN está en modo inactivo, ambas líneas transportan 2.5V, cuando se transmiten bits de datos, la línea CAN H pasa a 3.75V y CAN L a 1.25V, generando un diferencial de 2.5V entre las líneas: cada una de las líneas CAN está referenciada a la otra, no a la tierra del vehículo. Dado que la comunicación se basa en un diferencial de voltaje entre las dos líneas de bus, el bus CAN no es sensible a picos inductivos, campos eléctricos u otros ruidos. (AUTOMOTRIZ, 2020)

En la Figura 2.2 se representa los diferentes valores de voltaje para los cables CAN HIGH y CAN LOW durante una transmisión de datos.

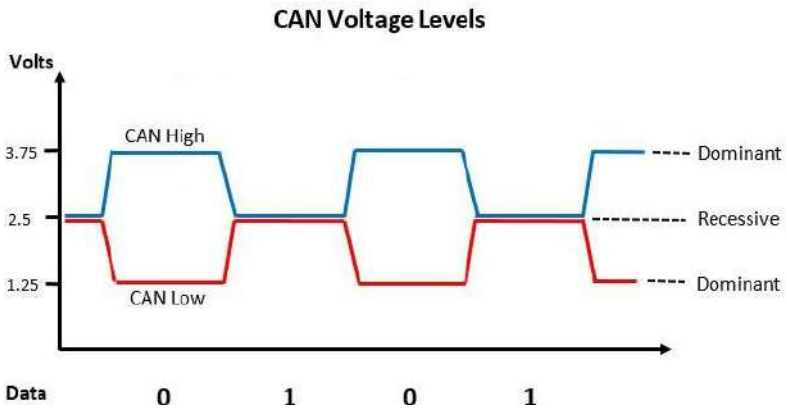


Figura 2.2 Valores de voltaje en los cables CAN HIGH y CAN LOW

2.6 Formato de Trama de CAN BUS

En la Figura 2.3 se representan los distintos campos por los cuales está constituida una trama CAN BUS

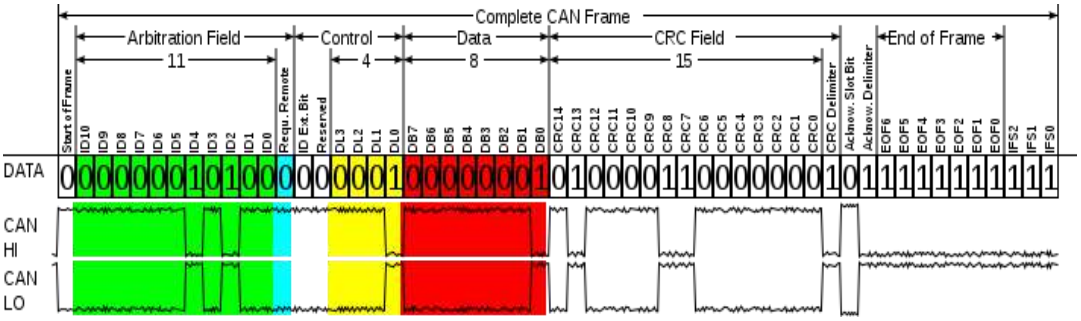


Figura 2.3 Formato de tramas de datos CAN BUS Estándar

- **SOF** (Start of Frame bit): Indica el comienzo del mensaje y permite la sincronización de todos los nodos conectados a la red. Este bit tiene estado dominante (0 lógico).
- **Campo de arbitrio**: Está formado por 12 bits (trama CAN tipo estándar) o 32 bits (trama CAN tipo extendida). Dentro del campo se encuentra el identificador, el cual indica la prioridad del nodo. El nodo con mayor prioridad es aquel que tiene el identificador más bajo (bits recesivos). El bit RTR se utiliza para distinguir entre una trama remota o una trama de datos como se muestra en la Figura 2.4
- **Campo de control**: Formado por 6 bits. El bit IDE indica con un estado dominante que la trama enviada es estándar. El bit RB0, está reservado y se establece en estado dominante por el protocolo CAN. Posteriormente se encuentran cuatro bits, que definen el tamaño del campo de datos del mensaje CAN.
- **Campo de datos:** Puede estar formado por hasta 8 bytes, dependiendo de lo que especifique en el campo de control. En este campo están contenidos los datos del mensaje.
- **Campo de verificación por redundancia cíclica (CRC)**: Este campo de 15 bits, detecta errores en la transmisión del mensaje. Se delimita con un bit final en estado recesivo.
- **Campo de reconocimiento:** El último campo de la trama, está formado por 2 bits. El nodo transmisor manda una trama con el bit de ACK (Acknowledge) en estado recesivo, mientras que los receptores, si han recibido el mensaje correctamente, mandarán un mensaje en estado dominante. Contiene un bit delimitador.
- **El fin de trama o EOF:** consiste en 7 bits recesivos. La trama debe contener al final de esta el espaciado reglamentario entre tramas, compuesto por tres bits recesivos. (García Osés, 2015)

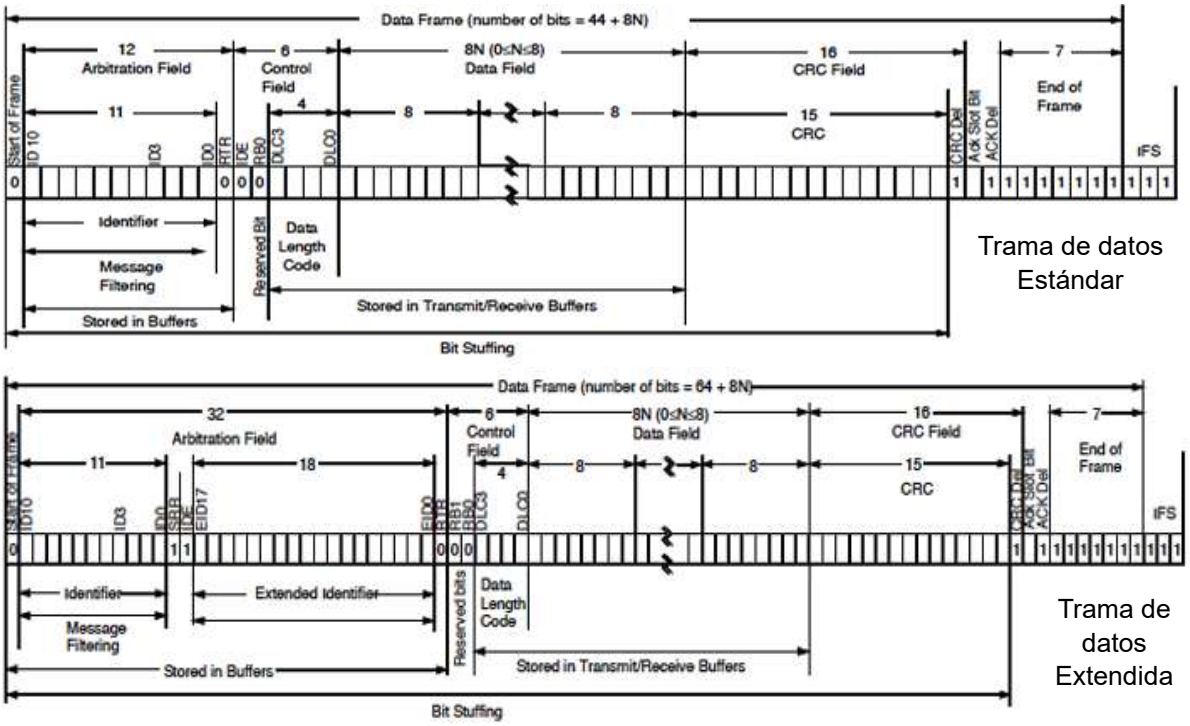


Figura 2.4 Comparativa entre trama de datos CAN BUS estándar y extendida.

2.6.1 Tipos de tramas de CAN BUS

Las tramas CAN BUS se clasifican principalmente en baseal tamaño de su identificador (como se menciona anteriormente) y además en su función como, por ejemplo:

- **Trama de datos:** Como su propio nombre indica, dichas tramas se utilizan para enviar datos a través de la red. Los datos se incluirán en el campo de datos y pueden tener una extensión de 0 a 8 bytes (ver Figura 2.4).
- **Trama remota:** Un nodo tiene la capacidad de solicitar un mensaje de otro nodo usando tramas remotas. El identificador de la trama debe ser el mismo que el del nodo del cual se quiere recibir el mensaje. Además, el campo de datos será 0. Una vez que el nodo receptor reciba el mensaje, éste enviará sus datos (ver Anexo 1).
- **Trama de error:** Se genera al detectar un error en la red por parte de un nodo. Está formada por un campo indicador de error y un campo delimitador (ver Anexo 2).
- **Trama de saturación (Overload frame):** A diferencia de la trama de error, la

trama de saturación sólo se da entre tramas. Es generada al detectar un bit dominante en el espacio entre tramas o al no ser posible el envío de un mensaje por problemas internos (ver Anexo 3). (García Osés, 2015)

2.7 Detección de fallas en el sistema CAN BUS

El CAN es uno de los cinco protocolos incluidos en el estándar OBD-II para el diagnóstico de vehículos. Sin embargo, el escáner OBD-II solo es capaz de identificar problemas básicos, no los fallos más complejos del sistema.

Para diagnosticar el sistema es necesario utilizar un osciloscopio, herramienta que permite conocer la señal de las ondas de la línea CAN BUS, pues en esta transita información digital en código binario, por lo que no solo se debe medir el voltaje. (MotorOK, 2020)

Habiendo aclarado esto, las principales fallas en el sistema CAN BUS según el estándar ISO 11898 son las siguientes:

- CAN H o CAN L interrumpido
- CAN H o CAN L en cortocircuito al voltaje de la batería
- CAN H o CAN L en cortocircuito a tierra
- CAN H y CAN L interrumpidos en la misma ubicación
- CAN L en corto a cable CAN H
- Pérdida de conexión a la red de terminación

2.8 Velocidad y seguridad en la transmisión de información del CAN BUS ante otros protocolos de comunicación

El protocolo que cuenta con mayor velocidad de transición de datos es MOST, con 21.2 Mbps, en segundo lugar, le sigue FlexRay, con una transmisión de datos de 10 Mbps. Los protocolos CAN Y VAN cuentan con velocidades mucho menor de 1 Mbps

y el protocolo con menor velocidad es LIN, con 20 Kbps, como muestra la Figura 2.5 (Sánchez Vela, y otros, 2016)

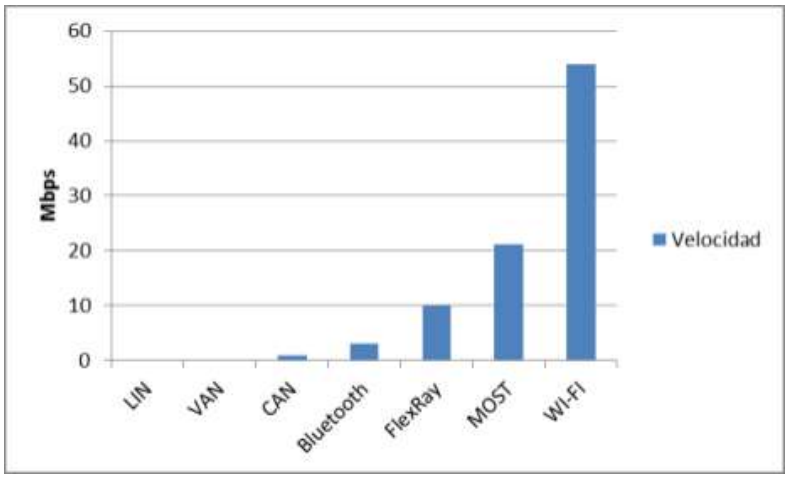


Figura 2.5 Velocidad de transmisión de datos de protocolos de comunicación

Sin embargo, los protocolos de mayor velocidad no son utilizados para sistemas de gestión en el motor o controlar la trasmisión motriz; por no ofrecer suficiente seguridad en la transmisión de datos. Los sistemas con mayor seguridad para trasmitir datos son aquellos que utilizan un cable eléctrico como CAN, VAN, FlexRay o LIN. (Sánchez Vela , y otros, 2016)

Mientras que el protocolo MOST es el más rápido, pero utiliza un medio de transmisión poco seguro (fibra óptica), por esa razón es empleado en sistemas de info-entretenimiento (Radio, pantallas, sonido, controles de volante, etc.) que requieren de una capacidad de trasmisión de datos muy grande, pero en la que no es prioritaria la seguridad de esos datos. (Sánchez Vela, y otros, 2016)

2.9 El CAN BUS en el sistema de confort e iluminación

Dentro de este sistema de confort se transmiten datos de las siguientes funciones:

- Cierre centralizado

- Elevalunas eléctricos
- Iluminación de los mandos
- Retrovisores exteriores regulables y calefactables eléctricamente
- Autodiagnóstico

En el sistema de iluminación podemos abordar las siguientes funciones:

- Luces de posición, luz baja y luz alta
- Luces de matrícula
- Luces antiniebla
- Luces intermitentes
- Luces de freno
- Luz de marcha atrás

2.9.1 Unidad de control del gataway

A ella están conectadas todas las líneas de CAN BUS. Su finalidad es la de comunicar y convertir los mensajes entre las distintas líneas de CAN BUS como puede ser líneas de tracción, info-entretenimiento, aparcamiento asistido, etc.

El gataway también es responsable de volcar la señal de “suspender” y “proseguir” las diferentes líneas con lo que se reduce el consumo eléctrico del vehículo. En la Figura 2.6 se representa la comunicación del gataway con demás unidades de control. (Iglesias Paz, 2019)

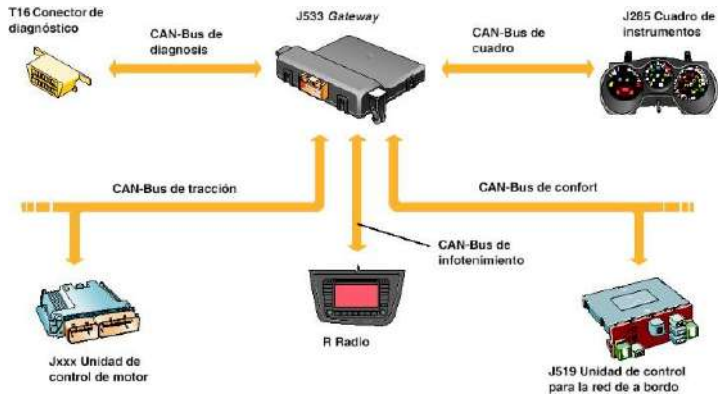


Figura 2.6 Comunicación entre gataway y unidades de control por CAN BUS

2.9.2 Unidad central y unidades de control situadas en puertas

Una unidad de control electrónico (ECU) es un pequeño dispositivo que integra un chip (microcontrolador) programado previamente, responsable de controlar una o varias funciones específicas. Las ECU reciben información de sensores del vehículo y se comunican con los actuadores para ejecutar la tarea deseada.

Las unidades que están pensadas para integrar y ejecutar el protocolo CAN, además de contener el microcontrolador contienen un controlador y transceptor CAN mismos que fueron abordados en el subtema 2.3 (Componentes generales de un sistema CAN BUS). En la Figura 2.7 se ilustra una parte de electrónica dentro de una ECU de confort. (Iglesias Paz, 2019)

Figura 2.7 ECU de sistema de confort

2.9.3 Microcontrolador

Un microcontrolador es un circuito integrado que en su interior contiene una unidad central de procesamiento (CPU), unidades de memoria (RAM y ROM), puertos de entrada y salida y periféricos. Estas partes están interconectadas dentro del

microcontrolador, y en conjunto forman lo que se le conoce como microcomputadora. Se puede decir con toda propiedad que un microcontrolador es una microcomputadora completa encapsulada en un circuito integrado. (Estudio, 2024)

Están enfocados a realizar una tarea específica, reciben señales de sensores, botones o de otros circuitos. Procesan la información y generan señales de activación a circuitos de control para actuadores como relevadores, LEDs, motores, etc. (ver Anexo 4). En la Figura 2.8 se representa un microcontrolador PIC16F877A de la familia Microchip.

Figura 2.8 Microcontrolador PIC16F877A de Microchip

2.9.4 Mando de posición de espejo, pulsadores de elevalunas y palanca de activación de luces

“Los switches interruptores son dispositivos eléctricos que permiten o impiden el flujo de una corriente eléctrica al momento de pulsarlo, su principal función es el encendido y el apagado” (Ltda, 2022)

Estos switches interruptores podemos encontrarlos en distintas posiciones del habitáculo del automóvil con el objetivo de ser manipulados por los usuarios. En la Figura 2.9 se puede observar un mando central de pulsadores para el sistema de elevalunas y espejos eléctricos.

Figura 2.9 Mando central de pulsadores para el Elevalunas y Espejos eléctricos

2.9.5 Motores eléctricos DC de orientación de espejos y accionamiento de elevalunas

Un motor eléctrico de corriente continua es una máquina que transforma energía eléctrica en energía mecánica, funciona mediante la inducción electromagnética. Algunos motores de corriente continua transforman energía mecánica en energía eléctrica funcionando como generador, gracias al principio de reversibilidad. (Valle, 2024)

- **Motor de excitación independiente**: El circuito del devanado inductor se conecta por separado del circuito del devanado inducido a una fuente de alimentación de corriente continua distinta. En la Figura 2.10, se observa el esquema del motor de excitación independiente. (Valle, 2024)

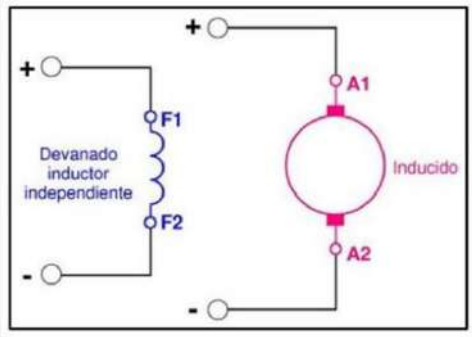


Figura 2.10 Motor de C.C. de excitación independiente

- **Motor shunt o de derivación en paralelo:** Los circuitos del devanado inductor y devanado inducido se conectan en paralelo a una misma fuente de alimentación de corriente continua. En la Figura 2.11, se observa el esquema del motor shunt o derivación en paralelo. (Valle, 2024)

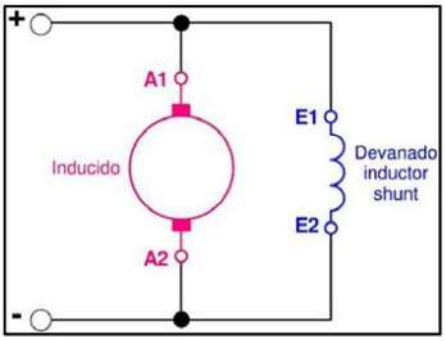


Figura 2.11 Motor de C.C. de derivación en paralelo o Shunt

- **Motor de excitación serie:** Los circuitos del devanado inductor y devanado inducido se conectan en serie a una misma fuente de alimentación de corriente continua. En la Figura 2.12, se observa el esquema del motor de excitación serie. (Valle, 2024)

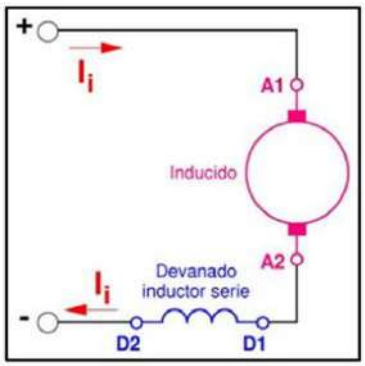


Figura 2.12 Motor de C.C. de excitación serie

- **Motor compuesto o compound**: El motor compuesto o compound tiene dos circuitos de devanado inductor y un circuito de devanado inducido. Un devanado inductor está conectado en serie con el devanado inducido y el otro devanado inductor está conectado en paralelo con el mismo devanado inducido. La conexión del motor compuesto tiene dos variantes: el motor compuesto de corta derivación y el motor compuesto de larga derivación.

En el motor compuesto de corta derivación, el devanado inductor en derivación está directamente en paralelo con el devanado inducido y el otro devanado inductor está conectado en serie con el devanado inducido. En la Figura 2.13, se observa el esquema del motor compuesta de corta derivación.

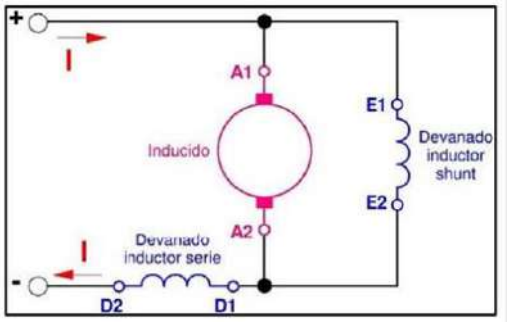


Figura 2.13 Motor de C.C. Compuesta de corta derivación

En el motor compuesto de larga derivación, el devanado inductor en derivación está directamente en paralelo, con el circuito del devanado inducido más el devanado inductor en serie. En la Figura 2.14, se observa el esquema del motor compuesto de larga derivación. (Valle, 2024)

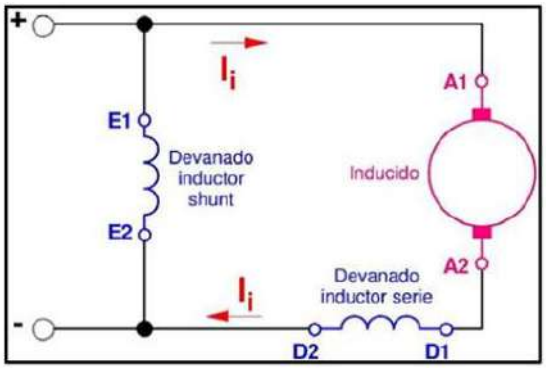


Figura 2.14 Motor de C.C. compuesta de larga derivación

2.9.6 Luces de posición, luces de cruce, carretera, traseras, luces de señalización/intermitentes

Los faros de un vehículo son elementos que tienen por objetivo brindar proyección de luz cuando se maneja en escenarios con poca o nula iluminación. Los ubicados en la parte delantera del vehículo son la primera referencia de visibilidad para el conductor. También se encuentran en la sección trasera de todo automóvil para dar visibilidad a otros conductores sobre la presencia del vehículo en el camino.

Por lo anterior, es de gran importancia mantener en óptimas condiciones los faros de tu vehículo para conducir de manera segura. (Ford, 2024)

En la Figura 2.15 se representa una óptica de un vehículo con las funciones de luces de cruce, carretera y posición.

Figura 2.15 Óptica de vehículo con funciones de luces de cruce, carretera y posición

2.9.7 Sensor de lluvia y luz

En un principio, el sensor de lluvia iba integrado en el pie del espejo retrovisor exterior, aunque hoy en día va colocado en la zona interior de la luna delantera, fuera del campo de visión. El sensor detecta, dentro de su campo de acción, si llueve, y transmite esta información a la electrónica de control de los limpiaparabrisas. De esta manera, la frecuencia de lavado en los intervalos de barrido de las escobillas se ajusta automáticamente a la intensidad de la lluvia. Por ello, el conductor ya no necesita intervenir manualmente. (HELLA, 2024)

El sensor combinado de luz y lluvia aúna dos funciones en un solo módulo.

- Detección de la lluvia y activación automática de los limpiaparabrisas
- Detección de la claridad del ambiente y conexión/desconexión automática de la iluminación del automóvil

En la Figura 2.16 representa un sensor de lluvia y luz de la marca Hella, en la cual se visualizan partes importantes que lo integran.

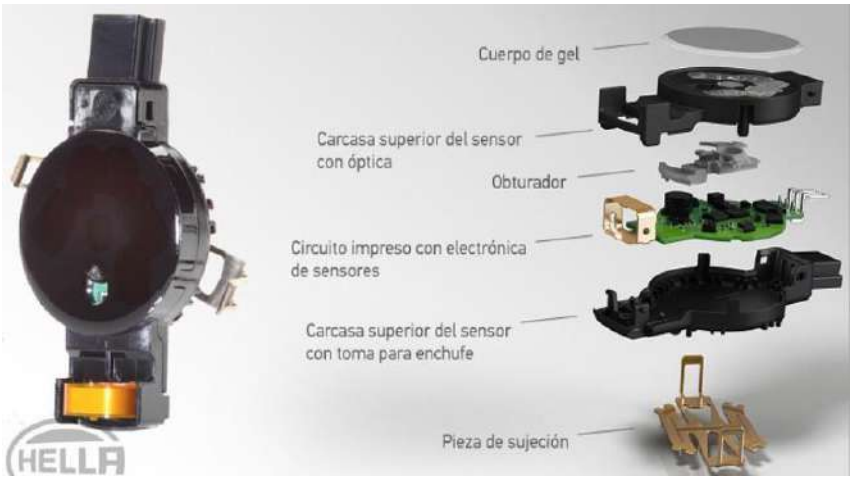


Figura 2.16 Sensor de lluvia y Luz de la marca Hella

Otros dispositivos eléctricos implementados dentro de los sistemas de iluminación y confort son los siguientes:

2.9.8 Relevadores

Los relevadores tienen una bobina conectada a una corriente, cuando esta se activa por la aplicación de 5, 12 0 24V según las especificaciones del fabricante se produce un campo electromagnético, el cual provoca que el contacto del relé que se encuentra normalmente abierto se cierre y de esta forma, permite el paso de la corriente por un circuito para ejercer cierta acción, como arrancar un motor o una bombilla (ver Figura 2.17).

Cuando se deja de proveer corriente a la bobina, el campo electromagnético se retira yel contacto del relé se vuelve a abrir, dejando sin corriente al circuito eléctrico que iba al motor o al objeto en cuestión (SDI, 2022)

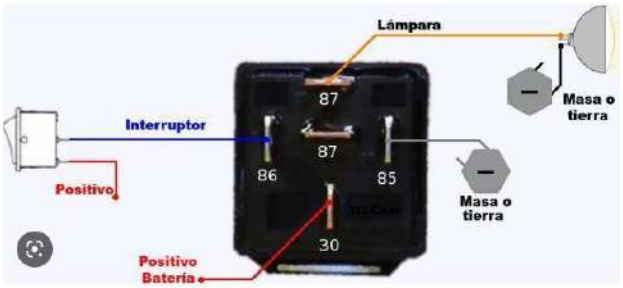


Figura 2.17 Conexión de Relevador tradicional de 5 pines

2.9.9 Fusibles de Protección

Son pequeñas piezas que protegen los sistemas eléctricos del coche. Su función básica es fundirse (de ahí el nombre) cuando existe cualquier sobretensión, para evitar que cualquiera de los sistemas vitales pueda verse dañado. (Basco, 2019)

En la Figura 2.18 se muestra una clasificación de los fusibles según su capacidad decorriente.

Figura 2.18 Capacidad de corriente para fusibles

CAPÍTULO III
METODOLOGÍA

3.1 Modelo Waterfall

Según Laoyan (2021) “El modelo en cascada o modelo Waterfall propuesto por Winston W. Royce en el año 1970. Es una metodología para gestión de proyectos que se divide en distintas fases. Cada fase comienza recién cuando ha terminado la anterior.” En la Figura 3.1 se muestran las diferentes etapas y el orden a seguir para esta metodología.

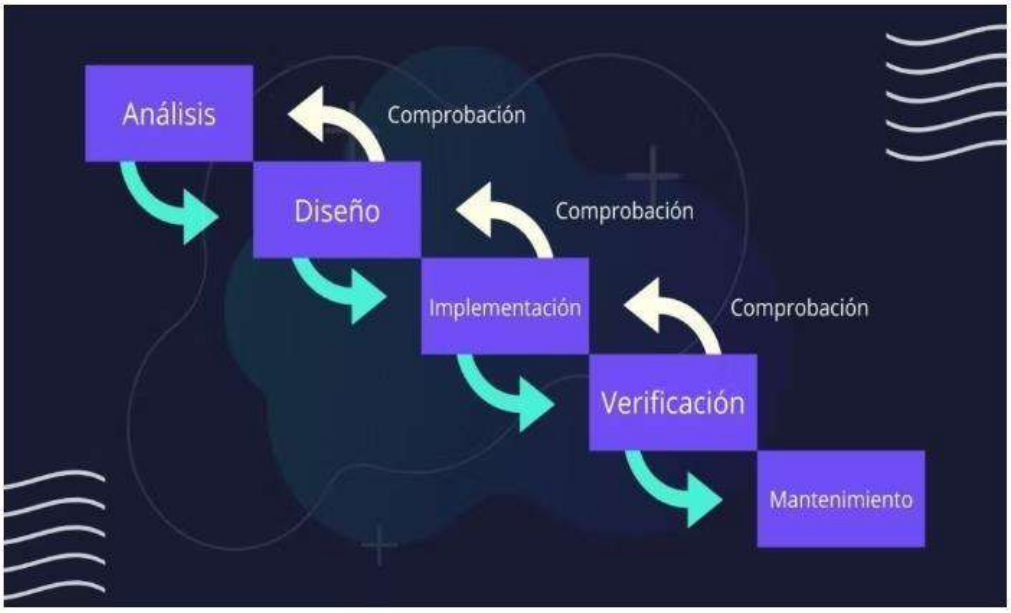


Figura 3.1 Etapas de la Metodología Waterfall

Para la realización de este proyecto que conlleva el diseño, desarrollo e implementación de un circuito de control de sistemas electrónicos del automóvil (ventanillas, espejos, luces frontales y traseras) bajo una estructura fundamentada en el protocolo de comunicación CAN BUS se optó por emplear la metodología anteriormente descrita ya es que es fácil de desarrollar y se basa en el análisis, diseño, implementación, verificación y mantenimiento de este.

3.2 Etapa 1 Análisis

Es el proceso de planificación inicial en el que se reúne toda la información posible para garantizar el éxito del proyecto, con la finalidad de determinar cuáles son sus necesidades y objetivos, posterior a ello, reunir todos los requisitos que se deben cumplir para la siguiente etapa.

En el Instituto Tecnológico Superior de Libres Puebla, en la Ingeniería en Sistemas Automotrices se requiere conocer de manera detallada la estructura de los códigos de programación de las unidades de control (ECU) integradas en los sistemas de confort e iluminación dentro de los tableros didácticos con los que cuenta el laboratorio del Instituto así como la manera en la que estas ECU´S interactúan entre si bajo el protocolo de comunicación serial CAN BUS llevando a cabo la transmisión de datos provenientes de sensores y actuadores cuando se requiere realizar una acción específica dentro de los sistemas anteriormente mencionados. Esto con la finalidad de elaborar un código fuente de programación el cual se integre a un circuito de control (Prototipo) diseñado para los sistemas electrónicos de ventanillas, espejos, luces frontales y traseras.

El conocimiento de algunos sistemas eléctricos tradicionales existentes en vehículos dentro del mercado brinda un panorama general de los componentes base y sus posibles conexiones a emplearse para que al momento de desarrollar sistemas electrónicos estos funcionen de la mejor manera. Contemplando las necesidades de este proyecto es conveniente analizar los sistemas a integrar de manera independiente, esto con el objetivo de evitar posibles fallas y generar una idea clara de los componentes inmersos en cada sistema, quedando seccionados de la siguiente manera:

- Bloque I: Sistema Eléctrico de Ventanillas y Espejos
- Bloque II: Sistema Eléctrico de Luces Frontales y Traseras.

3.2.1 Bloque I: Sistema eléctrico de ventanillas y espejos

Para este bloque se toman de referencia los sistemas eléctricos de un vehículo Mercury Gran Marquiz modelo 2000 dado que los componentes que integran a éstos

son accesibles y de bajo costo en la Ciudad de Libres Puebla.

Para el desarrollo del sistema electrónico de espejos se optó por tomar el sistema original con el que ya cuenta el vehículo (ver Anexo 5) y a partir de este, llevar a cabo la elaboración y programación de las ECU´s a integrar.

En cuanto al sistema de ventanillas para fines prácticos del prototipo solo es necesario representar la ventanilla del piloto y copiloto, es por ello, que se toman partes específicas del diagrama eléctrico mismas que se representan a continuación (ver Figura 3.2).

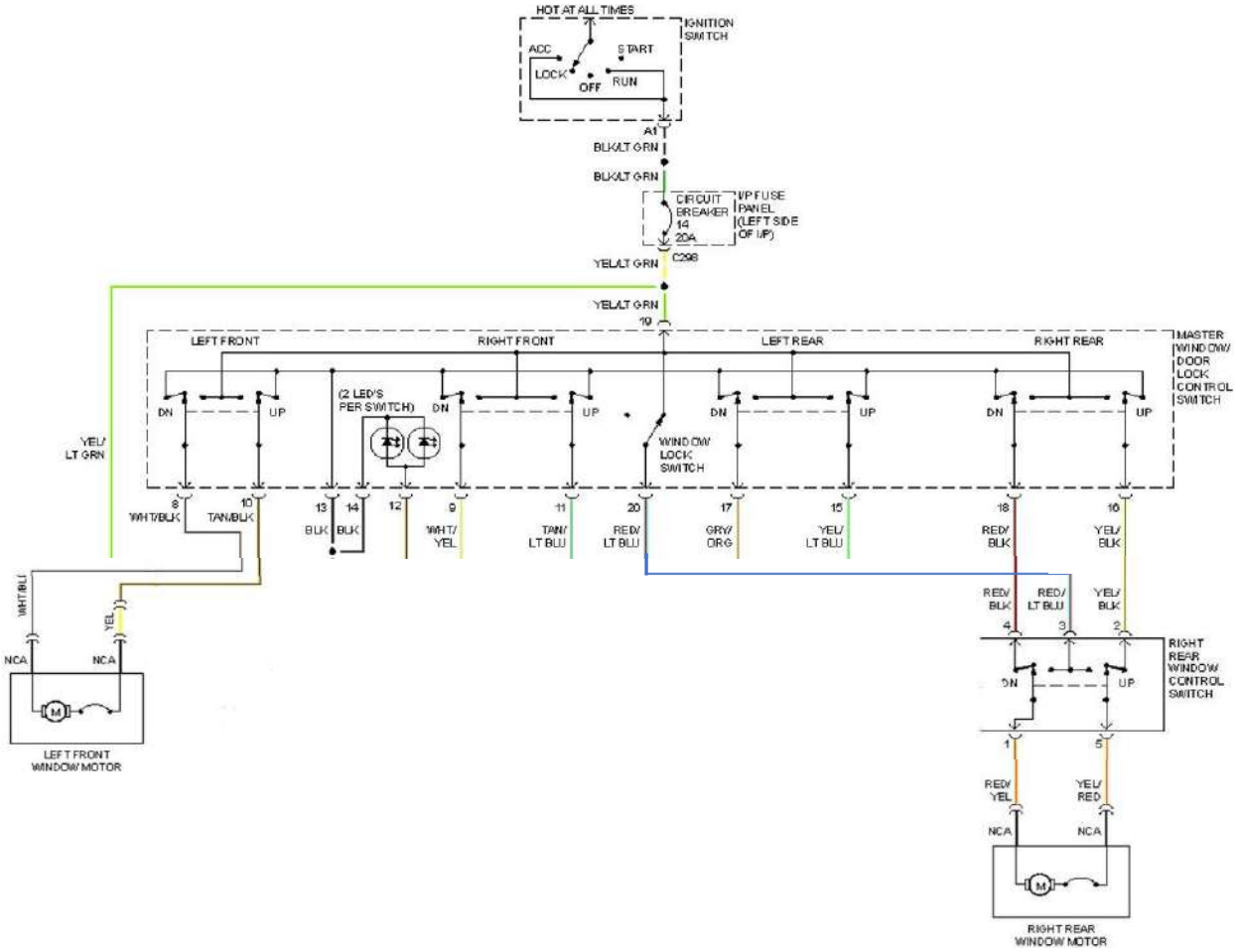


Figura 3.2 Fragmento de Diagrama Eléctrico de Ventanillas Mercury Gran Marquiz

En el Anexo 6 agregado en el presente documento se representa el diagrama eléctrico completo del sistema de ventanillas del Vehículo Mercury Gran Marquiz.

3.2.2 Bloque II: Sistema eléctrico de luces frontales y traseras

Este sistema surge a partir de una propuesta de diseño propia, dado que los componentes representativos utilizados como lo son: faros, mando de luces, interruptores, módulos leds, etc. Cumplen las funciones básicas requeridas para la circulación vial de cualquier vehículo.

Puebla (2018) establece en el Artículo 18 Fracción VI dentro de las Normas Generales de Circulación los siguiente:

- Faros principales: cumplen la función de emitir luz baja y alta en la parte frontal del vehículo.
- Luces de estacionamiento o posición: focos color ámbar en los extremos delanteros del vehículo y focos color ámbar o rojo en los extremos de la parte posterior.
- Luces posteriores:
    - Focos color rojo que representa la función de Stop (frenado del vehículo)
    - Luz blanca que indique la acción de movimiento en reversa.
    - Usar luces direccionales o intermitentes en parte trasera como frontal del vehículo.

Adicional a las funciones anteriormente mencionadas se anexa la función de neblineros para ocasiones dónde no haya suficiente visibilidad debido a las condiciones de camino, climatológicas o físicas.

3.3 Etapa 2 Diseño

Con los requisitos definidos se diseña el sistema estableciendo la arquitectura completa, no hay ninguna codificación durante esta fase, pero se establecen especificaciones tales como el lenguaje de programación y a grandes rasgos, se describen las partes (hardware) que deben formar el producto o servicio final.

Son tareas que se plasman en esta etapa de la metodología:

- Diseñar la organización de la estructura del circuito de control CAN BUS de los sistemas a abordar cumpliendo con los objetivos definidos en la etapa de análisis

de este proyecto.

- Establecer las funciones específicas de cada ECU que integran la estructura del circuito.
- Llevar a cabo la selección de los componentes para el funcionamiento de las ECU dentro del circuito de control.

3.3.1 Diseño organizacional de estructura de circuito de control CAN BUS

De acuerdo con lo indicado en la etapa anterior de Análisis se realiza el circuito de control integrando por varios nodos (ECU) los cuales se encontrarán interconectados en una red de comunicación bajo protocolo CAN. Concretamente se conectarán seis nodos encargados del control y manipulación lógica de los tres sistemas electrónicos a tratar. Cabe mencionar que el tiempo es algo muy crítico ya que una rápida valoración de los datos resulta en una mayor anticipación a los problemas y por tanto una mayor seguridad. Cada nodo que envía información se le asigna una ID única que lo distinguirá de los demás y que le otorgará una mayor o menor prioridad en el caso de que dos nodos manden mensaje a la red al mismo tiempo (ver Figura 3.3).

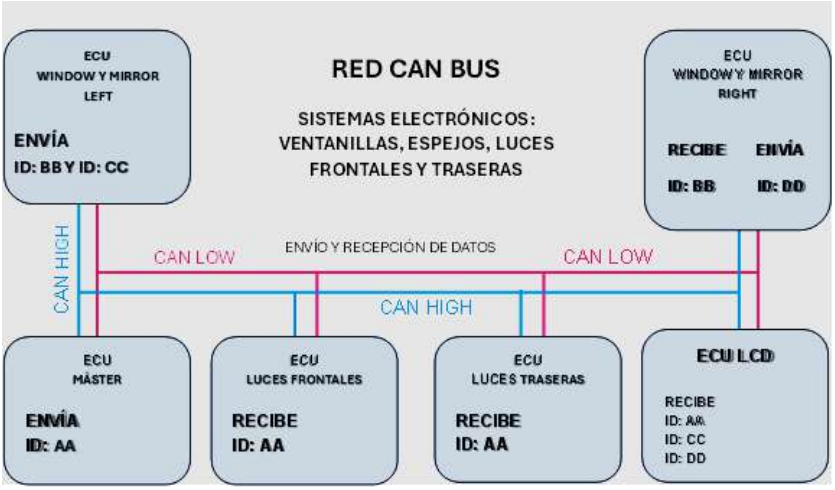


Figura 3.3 Estructura Organizacional de Circuito de Control CAN BUS

Como se observa en la Figura 3.3 durante la transmisión de datos la ECU Máster es la de mayor prioridad al contar con el ID más bajo en este caso el número hexadecimal “AA”, seguida de la ECU Window y Mirror Left que de manera específica

funciona con dos ID consecutivos; “BB y CC” y por último la ECU Window y Mirror Right con ID “DD”. Mientras que el resto de ECU´S solo reciben información con estos ID y ejecutan instrucciones cuando a estas se les asigne.

3.3.2 Codificación de Tramas CAN

De acuerdo con lo anterior las tramas CAN BUS son enlistadas conforme a su ID y el orden en la que este les permite enviar información a la red de manera ordenada por lo que en este subtema se describe el mensaje que transmite cada una. Cabe mencionar que para fines de este proyecto dentro de la comunicación CAN de las ECUS se emplearan solo tramas de datos del tipo estándar con un campo de datos de máximo ocho bits. (ver Tabla 3.1).

Trama	Campo de Datos	Descripción
Trama ID ”AA”	8 bits	Los 8 bits de datos es información de diferentes switches que manipulan el estado de encendido o apagado de luces frontales y traseras.
Trama ID “BB”	8 bits	Conlleva información de pulsadores que son manipulados por el piloto y pueden ordenar alguna acción de movimiento para el espejo y ventanilla right.
Trama ID “CC”	8 bits	Transmite información de algún accionamiento de espejo y ventanilla left; Accionamiento comandado por pulsadores manipulados por el conductor.
Trama ID “DD”	8 bits	Inspirada en la trama con ID BB pero a diferencia de esta, se modifica el movimiento generado para la ventanilla right debido a que esta ventanilla puede ser manipulada por el conductor y el copiloto.

Tabla 3.1 Tabla comparativa codificación de tramas CAN

3.3.3 Especificación de funciones de ECU´S en el Circuito

En el subtema 3.3.1 se representan las tramas CAN a emplearse, si bien se conoce que las tramas CAN contienen un ID identificador, este se puede complementar con delays (pausa en milisegundos) al final de la programación, esto para garantizar que todos los nodos encargados de transmitir información puedan hacerlo respetando su turno (ver Figura 3.4, 3.5 y 3.6).

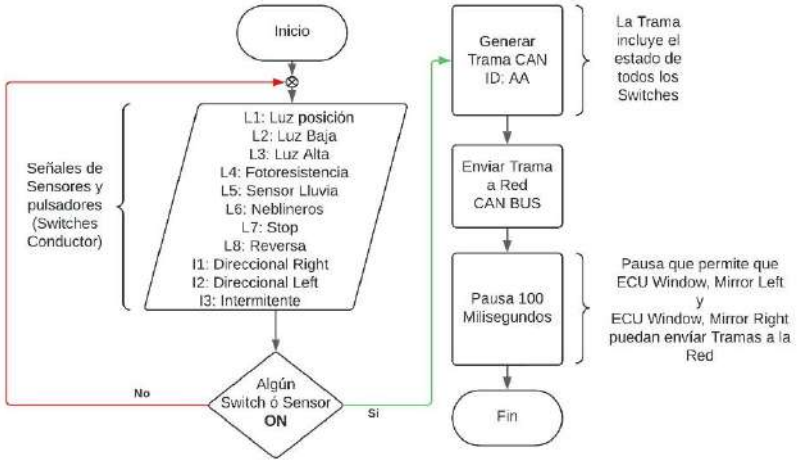


Figura 3.4 Secuencia lógica para ECU Máster

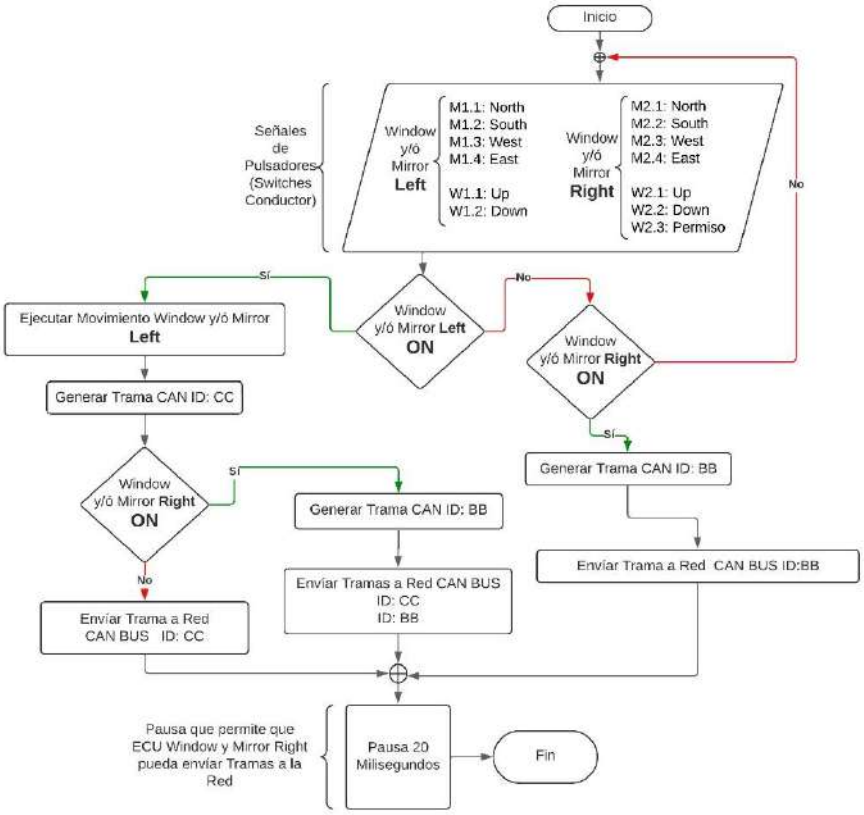


Figura 3.5 Secuencia lógica para ECU Window y Mirror Left

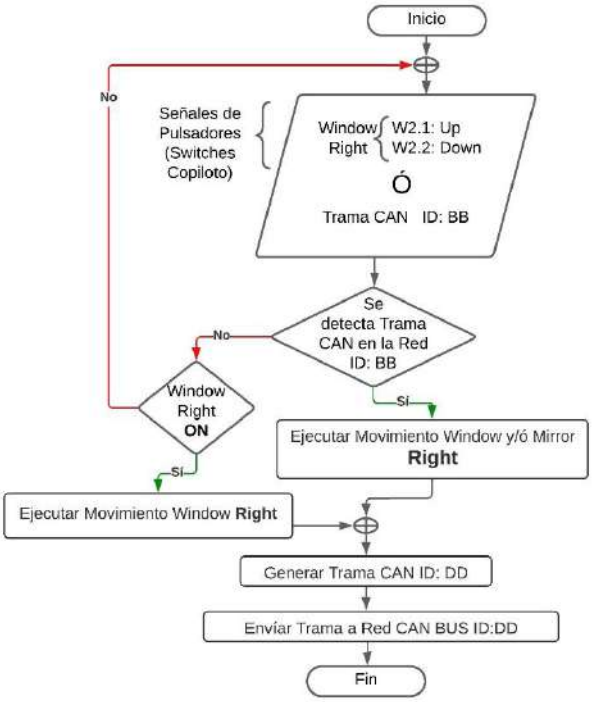


Figura 3.6 Secuencia lógica para ECU Window y Mirror Right

Las demás ECU´S complementarias (ECU luces frontales, ECU Luces traseras y ECU LCD) no es necesario integrarles algún delay debido a que estas ECU´S están destinadas a recibir información y ejecutar las instrucciones que se les ordene (ver Figuras 3.7, 3.8 y 3.9).

Figura 3.7 Secuencia lógica para ECU luces frontales

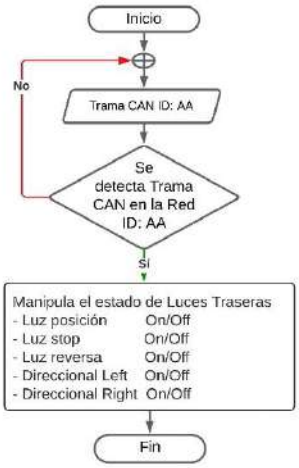


Figura 3.8 Secuencia lógica para ECU luces traseras

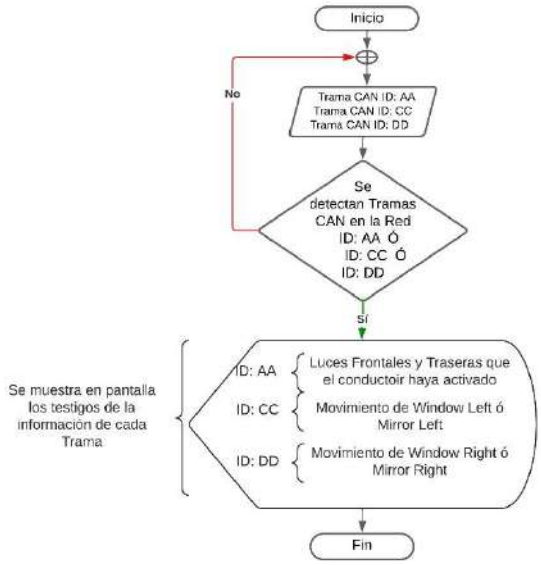


Figura 3.9 Secuencia lógica para ECU LCD

3.3.4 Selección de los Componentes

En la siguiente tabla comparativa se representan algunos componentes electrónicos empleados para el desarrollo del proyecto tomando en cuenta aspectos como: características, la facilidad de manejo y accesibilidad. (Ver Tabla 3.2)

Microcontrolador			
Pensado para las ECU	ATmega328P Características: • 32KB de memoria flash • Tamaño de datos RAM: 2KB • Interfaz: 2-wire, SPI, USART • Velocidad: 20MHz • Puertos de entrada/salidas programables: 23 • Temporizadores: 3 • Canales de ADC: 6 canales de 10 bits • Empaquetado: PDIP-28	PIC16F887A Características: • 14KB de memoria flash • RAM de 368KB • EEPROM de 256KB • Velocidad: 20MHz • 36 puertos de entrada/salida • 14 canales de ADC de 10 bits. • 3 Timers • Interfaz: SCI, UART/USART, I2C, SPI • Voltage de alimentación 5V • Encapsulado: DIP-40	Tarjeta Arduino Mega 2560 Características: • Microcontrolador: ATmeg2560 • Velocidad de reloj: 16 MHz • Voltaje de trabajo: 5V • Pinout: 54 pines digitales (15 PWM) y 16pines analógicos • 4 UARTs • Conexión USB • Conector ICSP
	✓	✗	✗
Controlador y Transceptor CAN BUS			
Existen módulos que Integran Estos componentes como los siguientes:	Módulo CAN MCP2515 Características: • Integra un Controlador CAN (DIP MCP2515) • Integra un Transceptor CAN (DIP MCP2551) • Comunicación SPI • Cristal 16 MHz • Terminal CAN • Velocidad Max. 10 Mbits/s (a 12 Mtrs.) • Longitud Max. de alcance de 1200 Mtrs. (a 100 Kbit/s) 	Tarjeta CAN BUS Shield V2.0 Características: • Integra un Controlador CAN (DIP MCP2515) • Integra un Transceptor CAN (DIP MCP2551) • Comunicación ISP • Cristal 16 MHz • Conector DB9 • Terminal CAN 
	✓	✗	

Tabla 3.2 Tabla Comparativa de Componentes Electrónicos

En la siguiente Tabla 3.3 se describen las características de los componentes adicionales a integrar en el área de electrónica dentro del desarrollo del circuito de control.

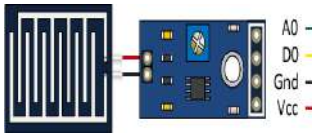
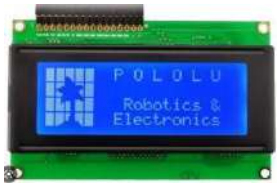
Componente	Características	Imagen
Sensor de lluvia	• Compatible con Arduino • Alimentación: 3.3 ó 5V • Salida digital (TTL): hasta 100 mA • La placa tiene un tratamiento de níquel Antioxidante	
Sensor foto-resistencia LDR	• Voltaje de alimentación: 3.3 – 5V • Salida: Digital y Analógica • Sensibilidad ajustable con potenciómetro • Número de pines: 3(VCC, GND y Señal)	
Pantalla LCD I2C	• Display LCD alfanumérico, monocromático • Resolución 2 líneas de 16 caracteres de 8X5 pixeles cada uno • Voltaje de alimentación 5V	
Módulo de Relevadores	• El relé es accionado por corriente con circuito de control por transistor • El voltaje de la bobina del relé puede ser de 5 ó 12V • Puede ser controlado directamente por circuitos lógicos	
Optoacoplador PC817	• Paquete: PDIP-4 • Tipo de salida: PhotoTriac • Número de canales: 1 • Corriente directa: 50 mA	
Resistencias	• Se miden en Ohmios • Es una medida de la oposición a la circulación de la corriente eléctrica • Existe de distintos valores que van desde los Ohmios a Kilo Ohmios	

Tabla 3.3 Componentes Electrónicos Adicionales para Circuito de Control

A continuación, en la Tabla 3.4 se enlistan los demás componentes eléctricos a integrar en el circuito de control.

Componente	Ilustración	Características
Actuador eléctrico lineal		• 12 V • 600 mA • 300 mm de desplazamiento
Fuente de CD		• Proporciona línea de voltaje de 5 V 25 A y 12 V 10 A
Espejos retrovisores eléctricos		• 12 V • 200 mA
Faros, Neblineros y Reversa		• 12 V • 80 mA
Luces de posición		• 12 V • 128 mA
		• 12 V • 70 mA
Mando central de pulsadores de espejos y elevalunas (Conductor)		• Control para ventanilla Left y Right • Control para espejo Left y Right
Mando de pulsadores elevalunas (Copiloto)		• Control para ventanilla Right
Palanca de activación de luces y direccionales		• Control para luces de posición, luz baja, luz alta y direccionales.
Switches & Botón de intermitentes		• Interruptores adicionales para activación de luces Stop, Reversa, Neblinero e Intermitentes.

Tabla 3.4 Componentes Eléctricos a emplear en Circuito de Control

3.4 Etapa 3 Implementación

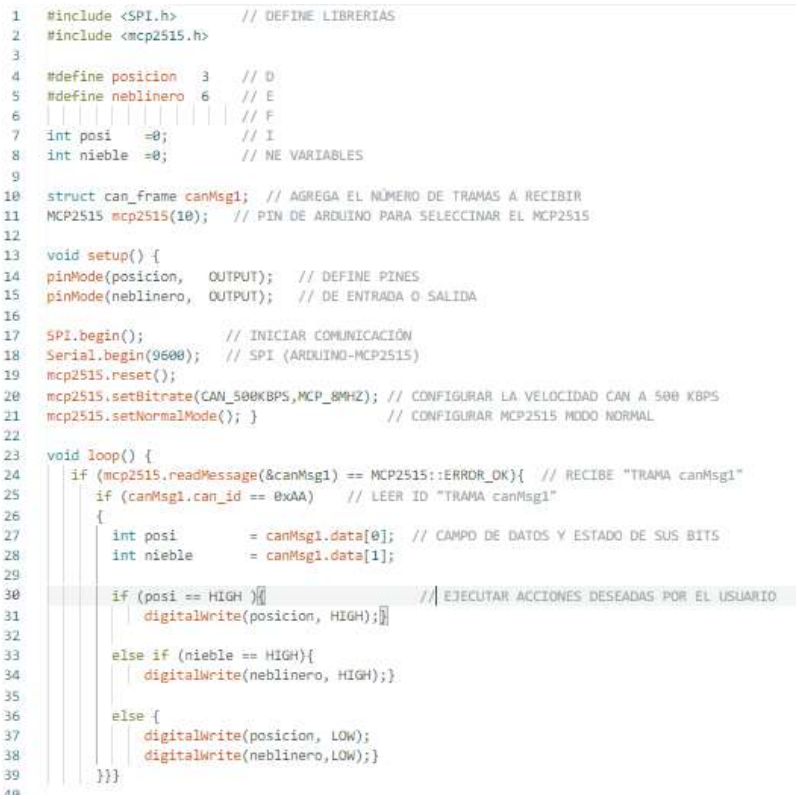
Para esta etapa de la metodología se desarrolló el código fuente de programación en Arduino IDE que integrará a las ECUS previamente descritas en la etapa anterior para el desarrollo del circuito de control. Se opto por emplear Arduino IDE para la programación debido a que hoy día ya se cuenta con librerías que fungen de apoyo para el control de módulos MCP 2515 dentro de una Red CAN BUS.

3.4.1 Estructura de librerías para envío y recepción de tramas

A continuación, se representan estructuras de las librerías “SPI.h” y “mcp2515.h” para programar un código fuente destinado al envío o recepción de tramas de datos CAN BUS (ver Figuras 3.10 y 3.11).

```
#include <SPI.h>            //  DEFINE LIBRERIAS
#include <mcp2515.h>

#define sensorlluvia    A0  // D
#define fotoresistencia A1  // E
                            // F
int senslluvia     =0;      // I
int fotoresis      =0;      // NE  VARIABLES

struct can_frame canMsg1;  // AGREGA EL NÚMERO DE
struct can_frame canMsg2;  // TRAMAS A ENVIAR
MCP2515 mcp2515(10);    // PIN DE ARDUINO PARA SELECCIONAR EL MCP2515

void setup() {
pinMode(sensorlluvia,    INPUT); //DEFINE PINES
pinMode(fotoresistencia, INPUT); //DE ENTRADA O SALIDA

SPI.begin();          // INICIAR COMUNICACIÓN
Serial.begin(9600);   // SPI (ARDUINO-MCP2515)
mcp2515.reset();
mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);  // CONFIGURAR LA VELOCIDAD CAN A 500 KBPS
mcp2515.setNormalMode();                   // CONFIGURAR MCP2515 MODO NORMAL

canMsg1.can_id  = 0xAA;  // ASIGNAR ID HEXADECIMAL
canMsg1.can_dlc = 8;  }  // LONGITUD DE DATOS EN LA TRAMA CAN BUS
canMsg2.can_id  = 0x8B;
canMsg2.can_dlc = 8;

void loop() {
    int senslluvia     = digitalRead(sensorlluvia);
    int fotoresis      = digitalRead(fotoresistencia);

      canMsg1.data[0]  = sensluvia;   // CAMPO DE DATOS Y ESTADO DE SUS BITS
      canMsg1.data[1]  = fotoresis;
      canMsg1.data[2]  = ...;
      canMsg1.data[3]  = ...;
      canMsg1.data[4]  = ...;
      canMsg1.data[5]  = ...;
      canMsg1.data[6]  = ...;
      canMsg1.data[7]  = ...;
      mcp2515.sendMessage(&canMsg1);  //ENVÍO DE "TRAMA canMsg1"
      delay(100); }
```

Figura 3.10 Ejemplo de código fuente para envío de tramas de datos CAN BUS



```
#include <SPI.h>        // DEFINE LIBRERIAS
#include <mcp2515.h>

#define posicion   3    // D
#define neblinero  6    // E
                        // F
int posi    =0;         // I
int nieble  =0;         // NE VARIABLES

struct can_frame canMsg1;  // AGREGA EL NÚMERO DE TRAMAS A RECIBIR
MCP2515 mcp2515(10);   // PIN DE ARDUINO PARA SELECCINAR EL MCP2515

void setup() {
pinMode(posicion,   OUTPUT);   // DEFINE PINES
pinMode(neblinero,  OUTPUT);   // DE ENTRADA O SALIDA

SPI.begin();          // INICIAR COMUNICACIÓN
Serial.begin(9600);   // SPI (ARDUINO-MCP2515)
mcp2515.reset();
mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ); // CONFIGURAR LA VELOCIDAD CAN A 500 KBPS
mcp2515.setNormalMode(); }               // CONFIGURAR MCP2515 MODO NORMAL

void loop() {
  if (mcp2515.readMessage(&canMsg1) == MCP2515::ERROR_OK){  // RECIBE "TRAMA canMsg1"
    if (canMsg1.can_id == 0xAA)    // LEER ID "TRAMA canMsg1"
    {
      int posi         = canMsg1.data[0];  // CAMPO DE DATOS Y ESTADO DE SUS BITS
      int nieble       = canMsg1.data[1];

      if (posi == HIGH ){                  // EJECUTAR ACCIONES DESEADAS POR EL USUARIO
        digitalWrite(posicion, HIGH);}

      else if (nieble == HIGH){
        digitalWrite(neblinero, HIGH);}

      else {
        digitalWrite(posicion, LOW);
        digitalWrite(neblinero,LOW);}
    }}}
```

Figura 3.11 Ejemplo de código fuente para recepción de tramas de datos CAN BUS

Como se observa en la Figura 3.10 se debe asignar el ID a la trama en número Hexadecimal, seguido de establecer el tamaño de la misma en el campo de datos; En el ejemplo se aprecia un campo de datos de 8 bits y por último se representa la instrucción “mcp2515.sendMessage(canMsg1)” la cual indica el envío de la trama CAN. Mientras que una trama destinada a la recepción de mensajes (Figura 3.11) se diferencia por emplear la instrucción “mcp2515.readMessage(canMsg1)” seguido de ello corroborar el ID identificador y ejecutar las instrucciones solicitadas.

La programación de cada una de las ECU´S destinadas para el circuito de control se representan en los Anexos del presente documento de la siguiente manera:

- Anexo 7: Programación ECU Master
- Anexo 8: Programación ECU Window y Mirror Left
- Anexo 9: Programación ECU Window y Mirror Right
- Anexo 10: Programación ECU Luces Frontales
- Anexo 11: Programación ECU Luces Traseras

- Anexo 12: Programación ECU LCD

3.4.2 Circuitos Eléctricos de ECU´S

Una vez realizada la descripción de la programación de las ECU se procede a establecer los circuitos eléctricos que integrarán a cada una contemplado la parte lógica compuesta por el microcontrolador Atmega 328P y el módulo MCP 2515 seguido de la parte de potencia que permite la recepción de señales de sensores, pulsadores y/o switches, así como el control de actuadores que se desea manipular.

3.4.2.1 Circuito Eléctrico de ECU Máster

En la siguiente figura se representa la propuesta de diseño del circuito eléctrico/electrónico que integra la ECU Máster. En el mercado existen variedad de software de simulación para circuitos electrónicos, para fines de este proyecto se optó por emplear uno de ellos. (ver Figura 3.12).

También en la Figura 3.12, se consideran las entradas para las conexiones provenientes de los sensores de lluvia, fotoresistencia, interruptores para la función de stop, marcha atrás y neblineros, aunado a ello, señales provenientes del mando de luces.

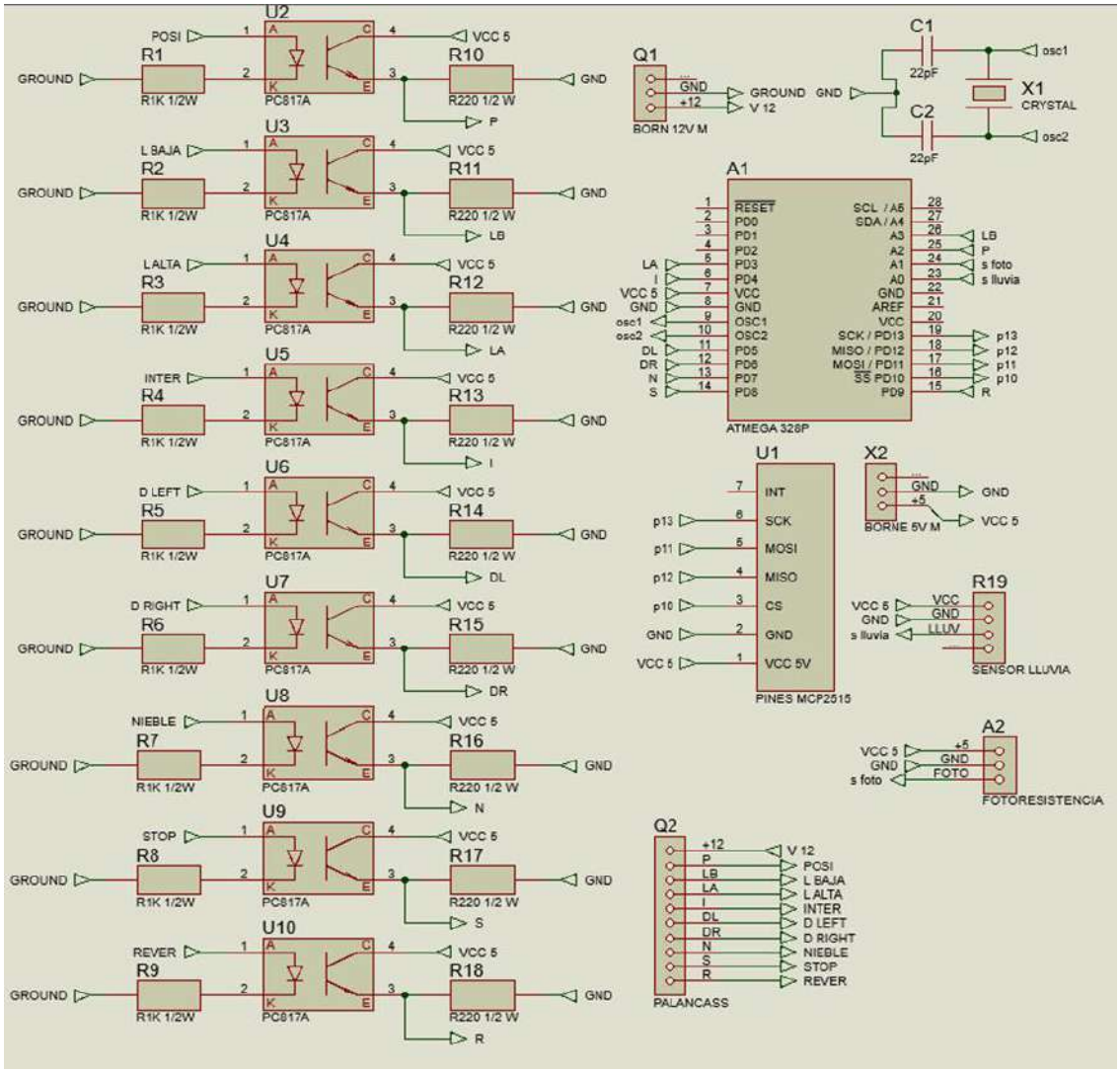


Figura 3.12 Circuito Eléctrico de ECU Máster

3.4.2.2 Circuito Eléctrico de ECU Window y Mirror Left

Ecu diseñada para la recepción de las señales provenientes del mando central de ventanillas y espejos retrovisores que es manipulado por el conductor, además de enviar información a la ECU Window y Mirror Right para activar los actuadores así como a la ECU LCD la cual representara las acciones que se ejecuten.

En la Figura 3.13 se representa el circuito eléctrico de dicha ECU; Cabe mencionar que dentro de la ECU se integran módulos L298N para el control de giro de motores de ventanillas y espejos.

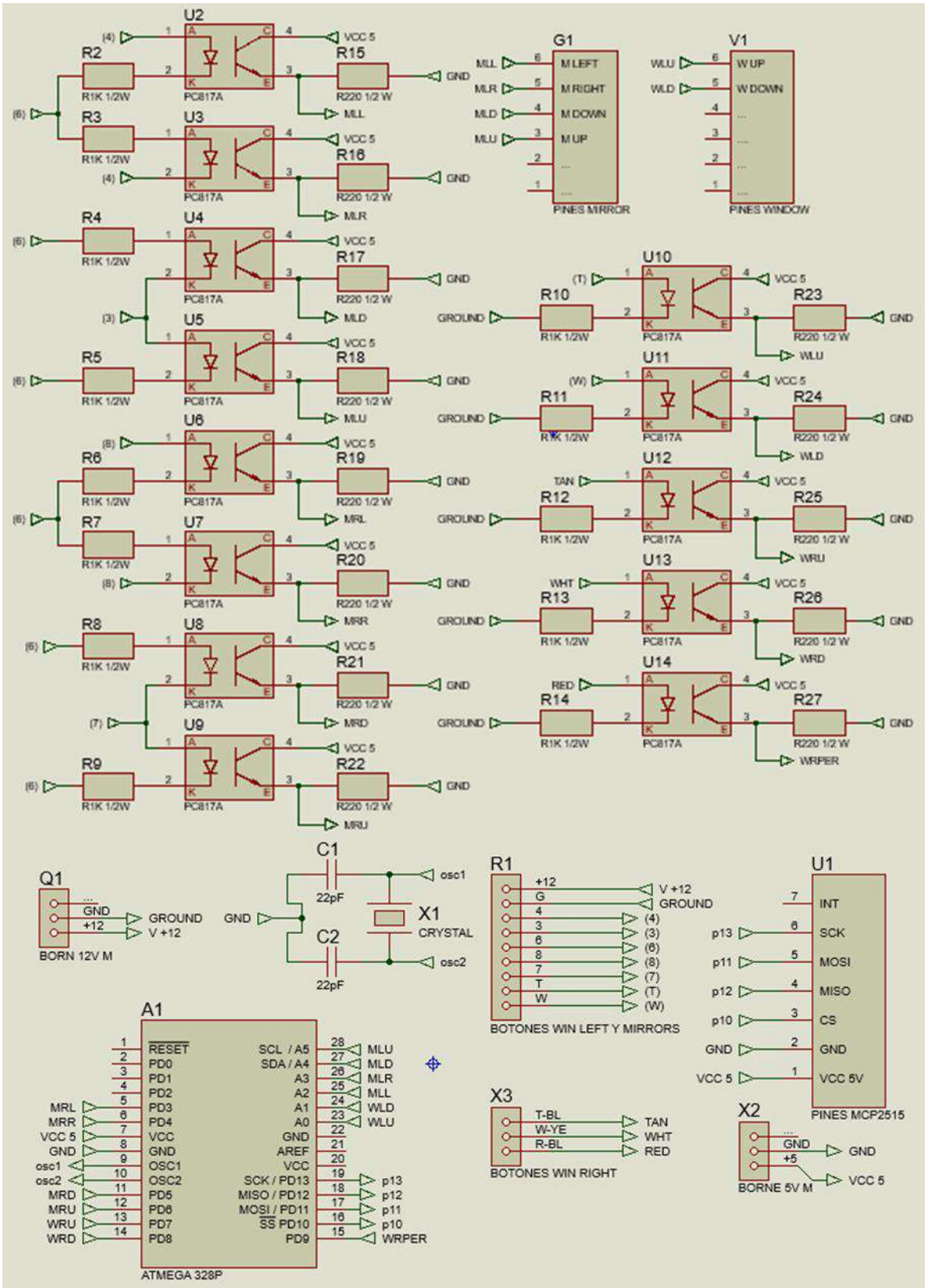


Figura 3.13 Circuito Eléctrico de ECU Window y Mirror Left

3.4.2.3 Circuito Eléctrico de ECU Window y Mirror Right

Esta ECU recibe información proveniente de la ECU Window y Mirror Left para la manipulación del espejo y ventanilla del copiloto, además, se encarga de verificar el estado del mando para la ventanilla del pasajero y permite que el mismo pueda realizar cambios en la posición. Adicionalmente, envía tramas CAN a la ECU LCD sobre las acciones que se lleven a cabo.

En la Figura 3.14 se ilustra la propuesta de diseño del circuito eléctrico de esta ECU.

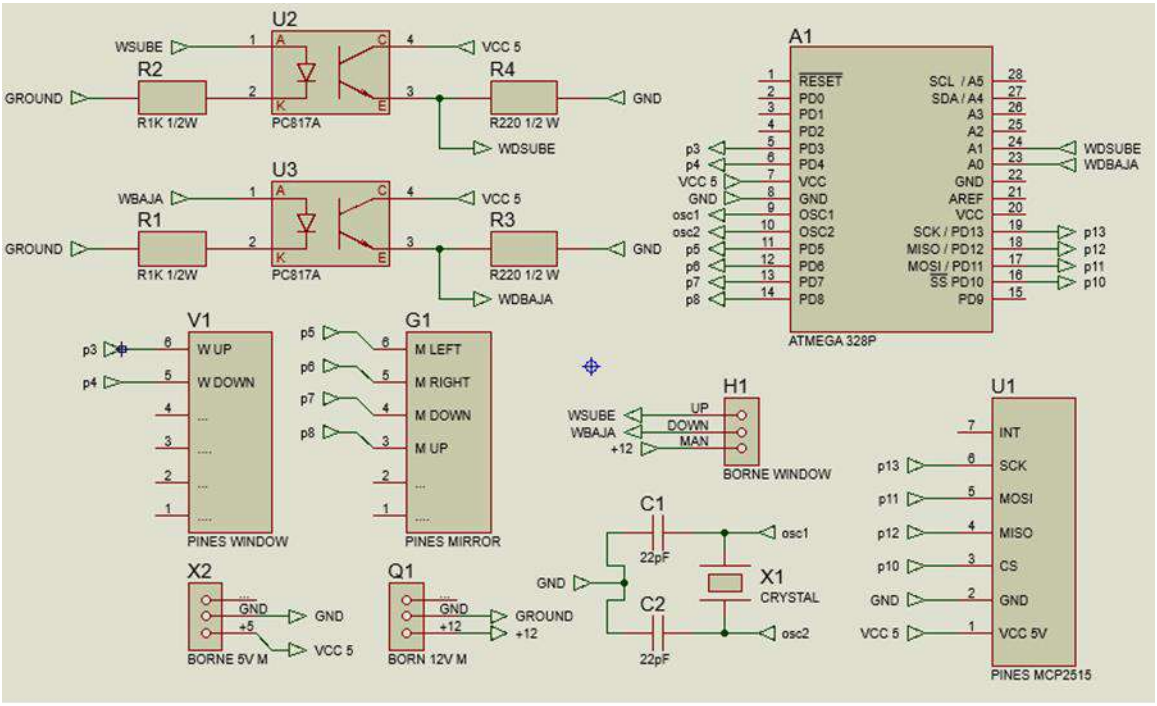


Figura 3.14 Circuito Eléctrico de ECU Window y Mirror Right

3.4.2.4 Circuito Eléctrico de ECU Luces Frontales

Está pensada para manipular el estado del sistema de luces frontales como lo son: posición, luz baja y/o alta, neblineros, direccionales e intermitentes; Se recibirán las tramas CAN BUS provenientes de la ECU Master y se llevarán a cabo las instrucciones para el encendido y/o apagado de las luces. Al igual que en la ECU Luces traseras se emplean módulos de relevadores compatibles para Arduino.

En la Figura 3.15 se representa la propuesta de diseño del circuito eléctrico de esta ECU de luces frontales.

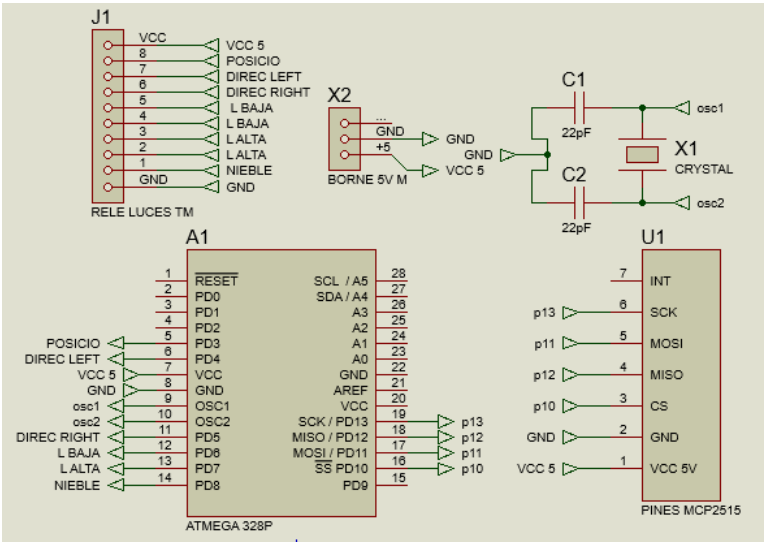


Figura 3.15 Circuito Eléctrico de ECU Luces Frontales

3.4.2.5 Circuito Eléctrico de ECU Luces Traseras

Al igual que el sistema de luces frontales, en esta ECU se cumplen las instrucciones proporcionadas por la ECU Master. Se encarga del control de las luces de posición, stop, marcha atrás, direccionales e intermitentes traseros respectivamente. En la Figura 3.16 se muestra el circuito eléctrico de esta ECU de luces traseras.

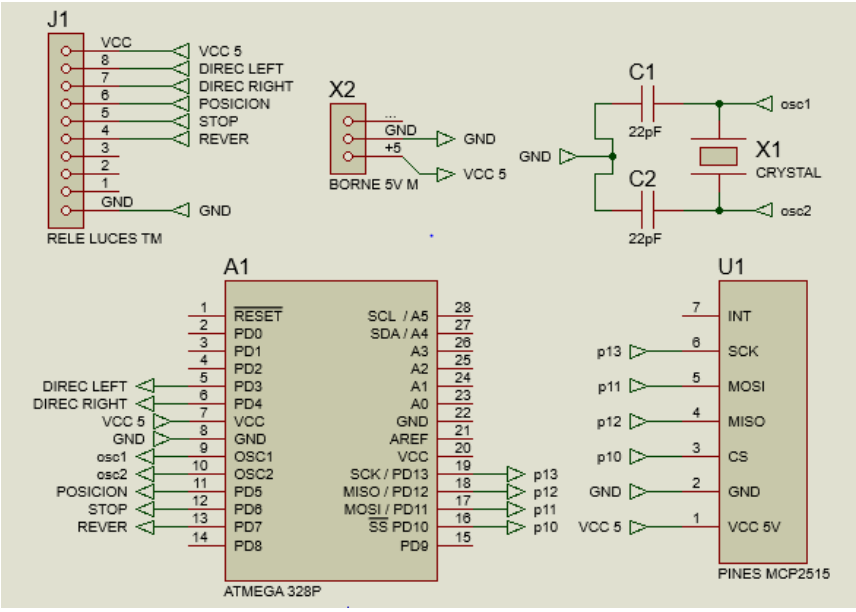


Figura 3.16 Circuito Eléctrico de ECU Luces Traseras

3.4.2.6 Circuito Eléctrico de ECU LCD

La ECU LCD funge como apoyo visual para el conductor o copiloto en la cual se representa por medio de simbología las acciones que esté llevando a cabo todo el circuito de control en tiempo real. En la Figura 3.17 se representa las conexiones del circuito eléctrico de la ECU LCD.

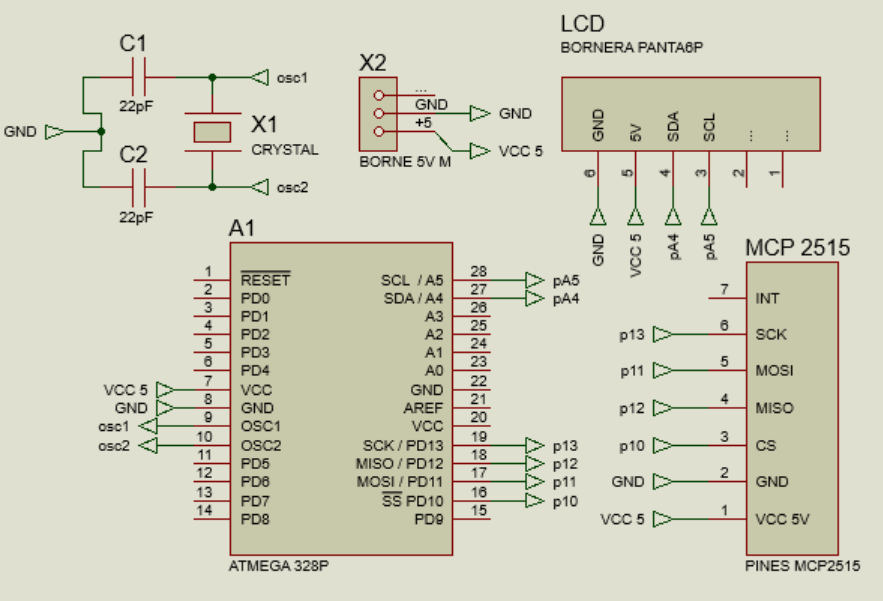


Figura 3.17 Circuito Eléctrico de ECU LCD

3.5 Etapa 4 Verificación

Cada una de las partes del software que conforman el producto final se integran y prueban como un sistema completo para asegurar que cumple con todos los requisitos deseados.

Para esta etapa se deben cubrir tres tareas importantes para corroborar que nuestra programación y las propuestas de los circuitos eléctricos de las ECU antes descritos en la etapa de implementación puedan trabajar conjuntamente sin ningún problema, asegurando que se cumplen los requisitos planteados en un principio. Las tareas por cubrir son las siguientes:

- **Verificación de los códigos fuente de programación**: Es lo que en lenguaje informático se conoce como “compilar”. Esta acción transforma el código fuente (el programa tal y como lo leemos al escribirlo) en código máquina (el programa tal y como lo ejecuta el microcontrolador ATmega 328P), durante esta acción también se verifican los posibles errores de redacción y/o ausencia de símbolos en el código fuente.
- **Cargar los códigos de programación en su respectivo microcontrolador ATmega328P para cada ECU:** Las tarjetas de Arduino permiten que por medio de su interfaz ISP se puedan grabar microcontroladores independientes como en este caso, los microcontroladores ATmega328P siguiendo los pasos que se describen a continuación.

  Paso 1: Conecte la placa Arduino UNO y el microcontrolador como se muestra en la Figura 3.18 integrando dos capacitores de 22 nF y un cristal de 16MHz. (ver Figura 3.18).

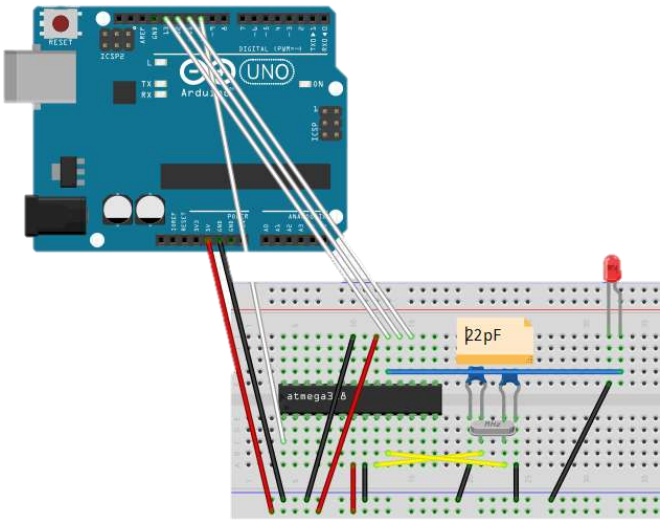


Figura 3.18 Conexión Arduino Uno y ATmega 328P mediante interfaz ISP

Paso 2: Conectar la placa Arduino y seleccionar el puerto en Arduino IDE seguido de Abrir el Sketch ArduinoISP ubicado en la siguiente dirección: Archivo\Ejemplos\ArduinoISP.

Paso 3: Una vez dentro del Sketch, en la barra de Herramientas seleccionar Grabar Bootloader y posterior seleccionar (Programador: “Arduino as ISP”).

Para finalmente subir el Sketch de ArduinoISP a tu Arduino.

Paso 4: Finalmente abrir el Sketch deseado a subir en el Atmega 328P, dirigirse a la barra “Sketch” y seleccionar “Cargar usando el programador”.

- **Realizar pruebas y corroborar su comportamiento de las ECU**

  Una vez verificados y grabados los códigos de programación a los microcontroladores ATmega328P se procede a la integración de los mismos en las ECU´S correspondientes para realizar las conexiones requeridas de cada circuito y corroborar su correcto funcionamiento y a partir de ello poder integrarlas al circuito de control de los sistemas electrónicos de ventanillas, espejos, luces frontales y traseras.

3.6 Etapa 5 Mantenimiento

A partir de este punto, el desarrollo se centra en la corrección de errores no descubiertos en etapas anteriores, corroborar que las conexiones eléctricas mantengan un buen contacto físico entre si dentro del tablero didáctico con la finalidad de evitar posibles fallas y/o cortos circuitos. Asegurarse que los sensores de lluvia y luz se encuentren descubiertos y sin obstrucciones externas que puedan afectar el correcto funcionamiento del circuito de control.

CAPÍTULO IV

ANÁLISIS Y DISCUSIÓN DE RESULTADOS

4.1 Elaboración de tarjetas PCB (ECUS)

A continuación, se presentan los resultados obtenidos del diseño estructural, de la Figura 4.1, 4.2, 4.3, 4.4, 4.5 y Figura 4.6 ilustran las ECU´S desarrolladas en placa PCB integrando todos los componentes electrónicos necesarios como se mencionó en la etapa de implementación.

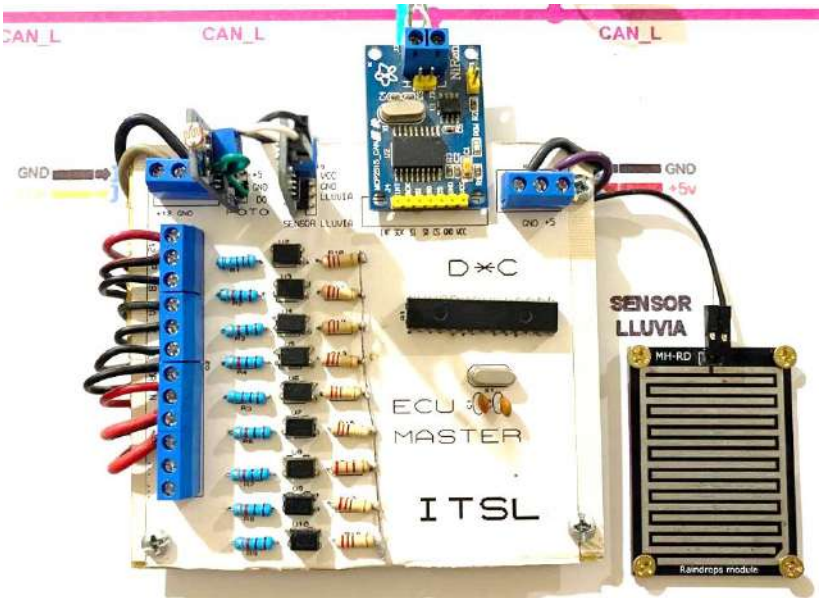


Figura 4.1 Tarjeta PCB de ECU Master

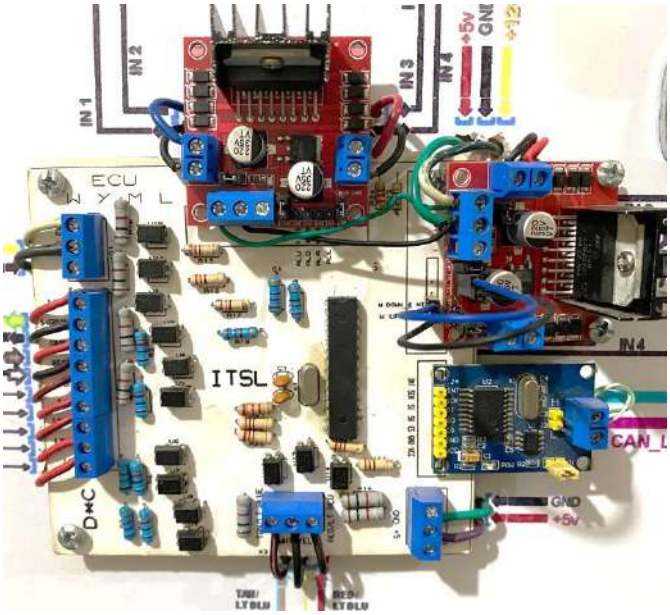


Figura 4.2 Tarjeta PCB de ECU Window y Mirror Left

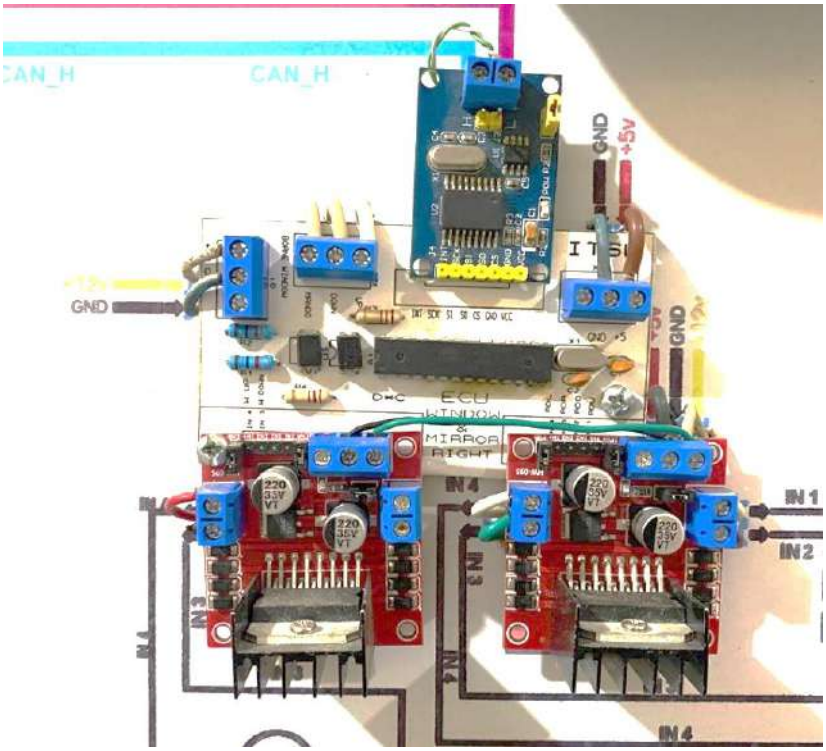


Figura 4.3 Tarjeta PCB de ECU Window y Mirror Right

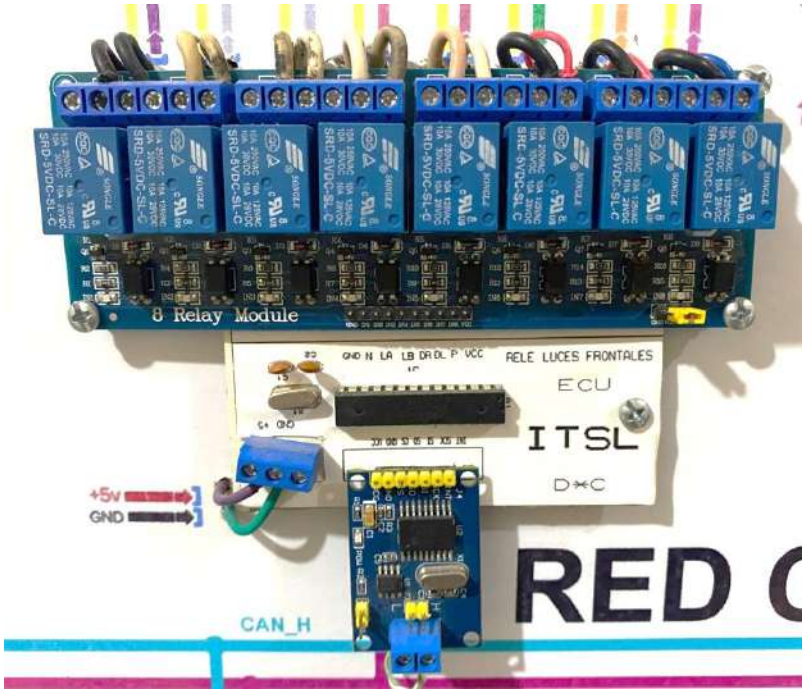


Figura 4.4 Tarjeta PCB de ECU Luces Frontales

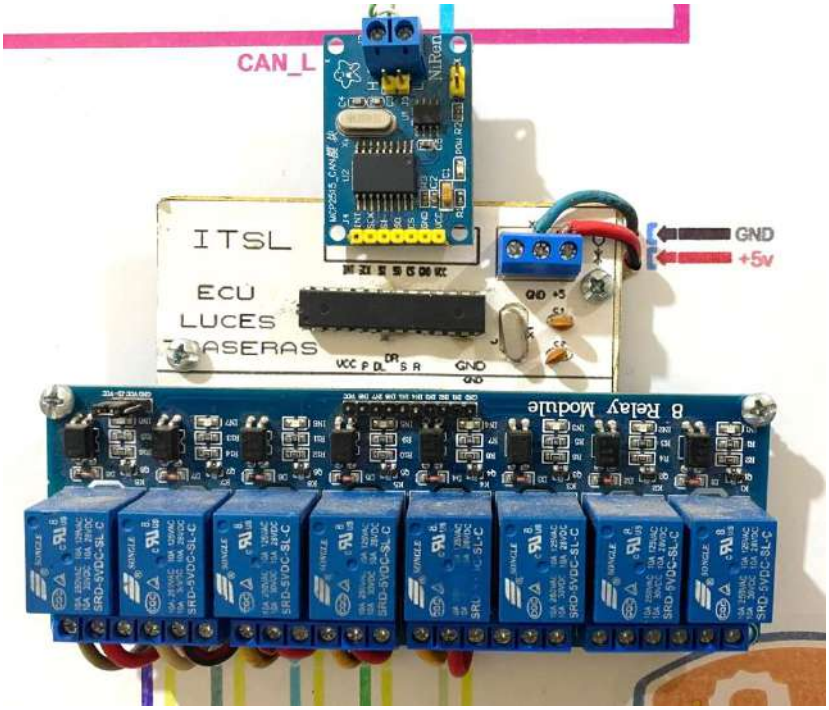


Figura 4.5 Tarjeta PCB de ECU Luces Traseras

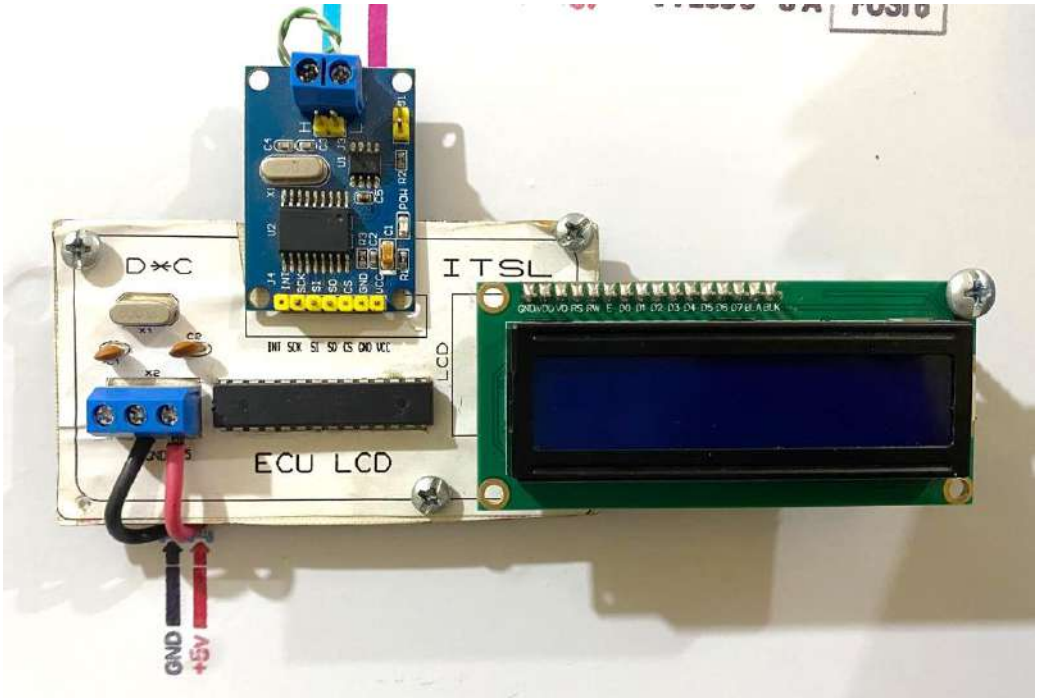


Figura 4.6 Tarjeta PCB de ECU LCD

4.2 Integración de ECU´S al circuito de control electrónico

En la Figura 4.7 y 4.8 se representan la integración de las ECU´S anteriormente ilustradas aunado a demás componentes eléctricos propios del vehículo dando como resultado la creación del circuito de control de los sistemas electrónicos de ventanillas, espejos, luces frontales y traseras. Este circuito manipula la activación o desactivación de los sistemas anteriormente mencionados en función de las necesidades del usuario, es decir, una vez encendido el circuito, este se mantendrá siempre pendiente de las instrucciones que el usuario pueda generar por medio de la interacción con los interruptores o sensores (sensor de lluvia y sensor de luz) que integra el tablero didáctico y así poder ejecutar algún cambio en los actuadores como ventanillas, espejos, luces frontales y traseras.

El circuito de control maneja un sistema de comunicación CAN BUS para la transferencia de información entre las distintas ECUS que se encuentran interconectadas dentro del tablero, estas con la capacidad de tomar lecturas en tiempo real de las instrucciones que él o los usuarios puedan generar.

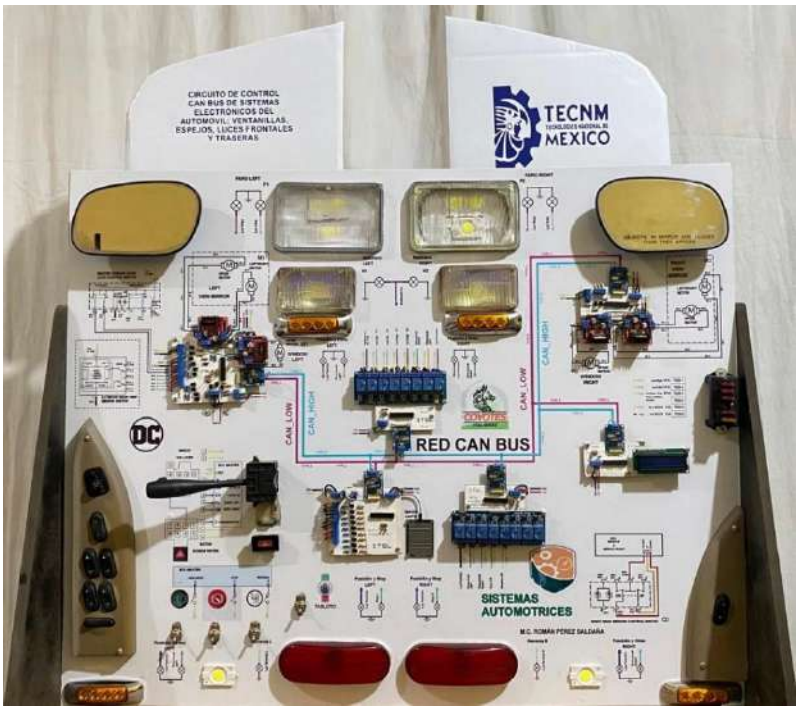


Figura 4.7 Tablero didáctico de circuito de control CAN BUS de sistemas electrónicos de ventanillas, espejos, luces frontales y traseras

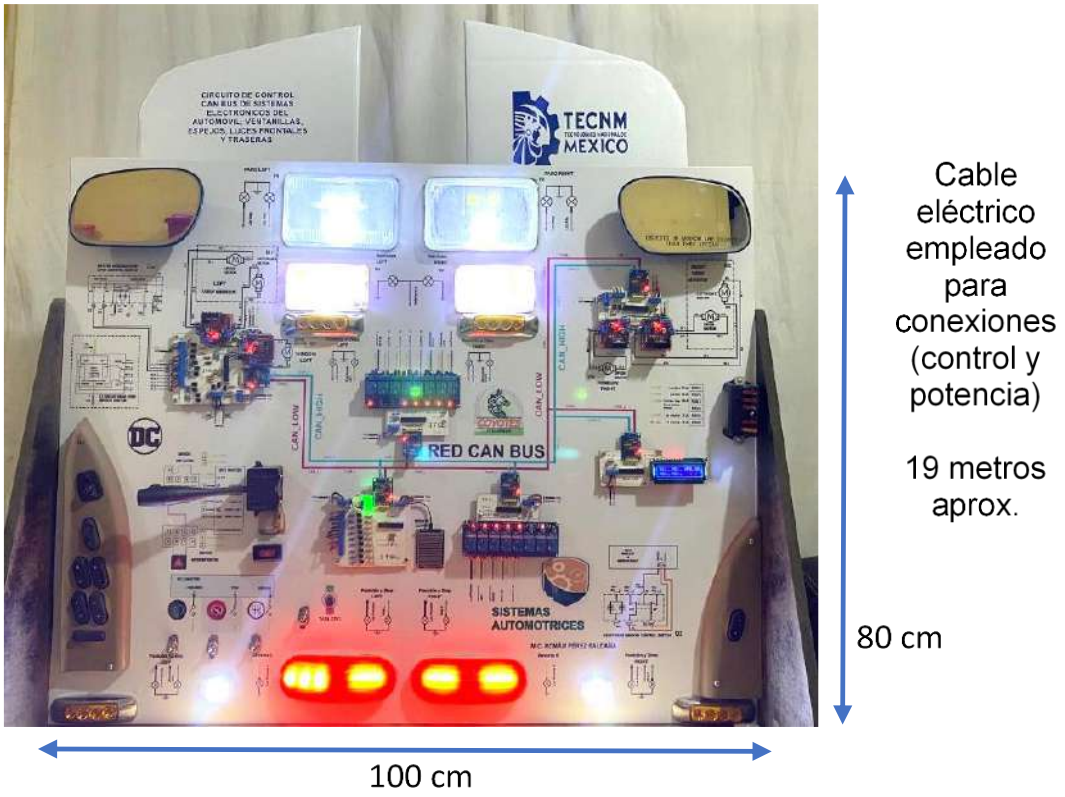


Figura 4.8 Encendido de Tablero Didáctico

4.3 El CAN BUS y la reducción de cableado

El CAN ofrece un método simplificado para el desarrollo de controles electrónicos sobre los sistemas eléctricos que se venían utilizado tradicionalmente en la industria automotriz. A continuación, en las Figuras 4.9 y 4.10 se representa un panorama general de los sistemas eléctricos de un automóvil con y sin el empleo del protocolo CAN respectivamente.

Como se observa, la implementación de un bus CAN permite la reducción significativa del cableado requerido para los sistemas del automóvil. Dicha reducción dependerá en gran medida de las características y equipamiento propio de cada vehículo, por ejemplo; dimensiones, numero de puertas, gama, etc.

En cuanto a la elaboración de este proyecto en el que se obtuvo el circuito de control de los sistemas electrónicos de ventanillas, espejos, luces frontales y traseras

utilizando el protocolo de comunicación CAN BUS plasmados en un tablero didáctico con medidas de 100 cm de base y 80 cm de altura (ver figura 4.8) también se refleja una reducción significativa del cableado eléctrico requerido de aproximadamente 70 % en comparación al empleo de un sistema de conexión tradicional sin CAN BUS (ver Figura 4.11)

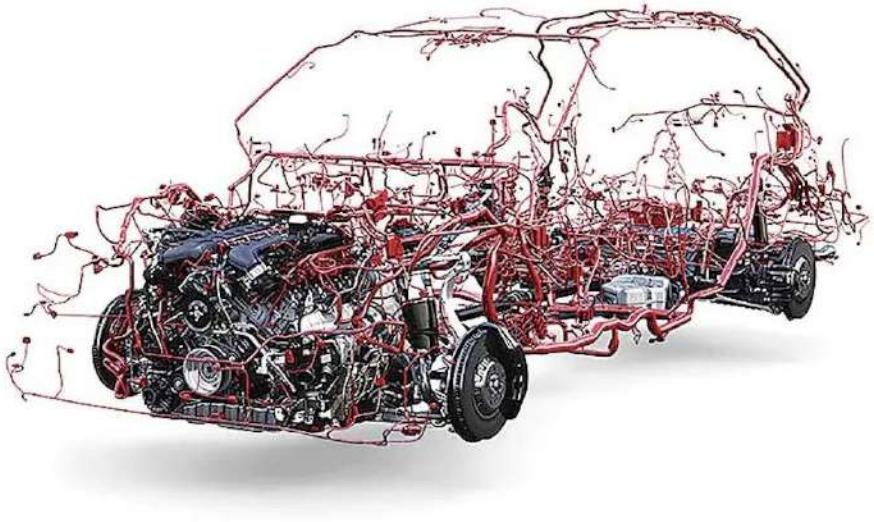


Figura 4.9 Cableado eléctrico típico en un automóvil de pasajeros

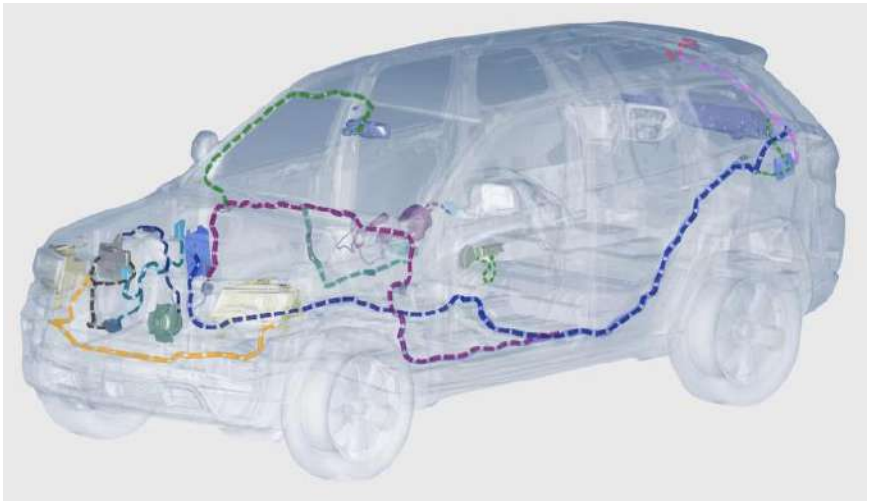


Figura 4.10 CAN BUS de 2 hilos en los automóviles y camiones de hoy día

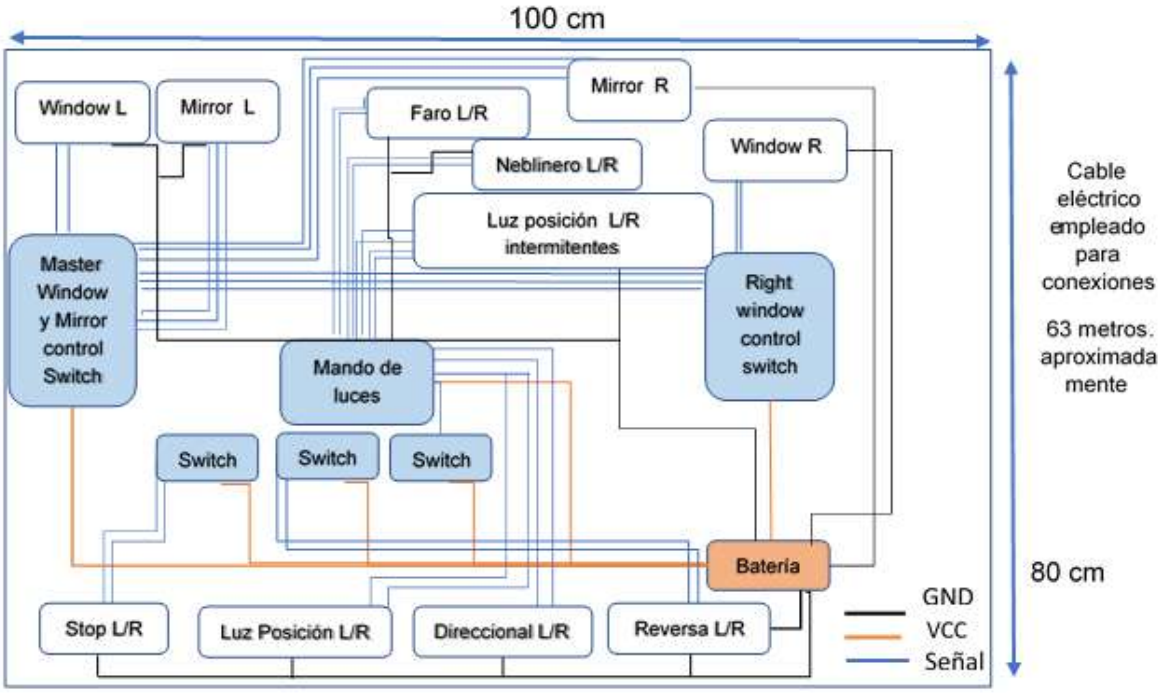


Figura 4.11 Propuesta de diseño propio (tablero didáctico) de diagramas del cableado para sistemas eléctricos de ventanillas, espejos, luces frontales y traseras sin CAN BUS

4.4 Pruebas técnicas del circuito de control

Una vez elaborada la conexión del circuito de control se procedió a la realización de pruebas técnicas con ayuda del osciloscopio para la verificación del correcto funcionamiento de las tramas CAN BUS encargadas de la intercomunicación de las ECU´S dentro de la red.

4.4.1 Verificación de funcionamiento de tramas CAN BUS

A continuación, en las Figuras 4.12 y 4.13 se muestran algunas de las tramas CAN BUS generadas a partir de la activación de ciertas funciones como lo son luces intermitentes y ventanilla de copiloto sucesivamente. Cabe mencionar que los rangos de voltaje de las líneas CAN HIGH y CAN LOW se representan en la parte inferior derecha de cada imagen. Teóricamente los valores de las líneas CAN HIGH Y CAN

LOW en promedio son valores que van de 2.5 a 3.75 V y 2.5 a 1.25 V por lo que al realizar una comparativa con los valores de las siguientes imágenes resultan ser semejantes.

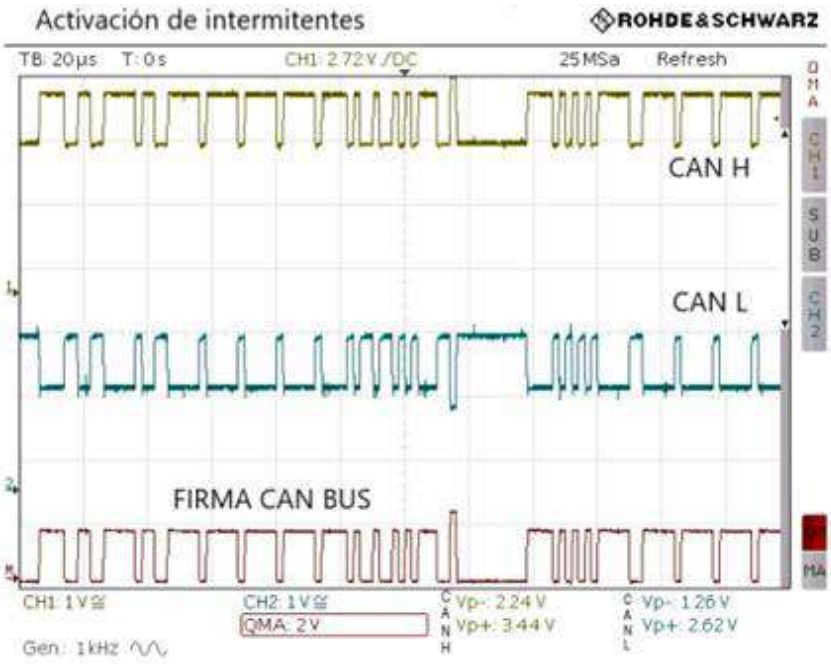


Figura 4.12 Tramas CAN BUS de activación de luces intermitentes

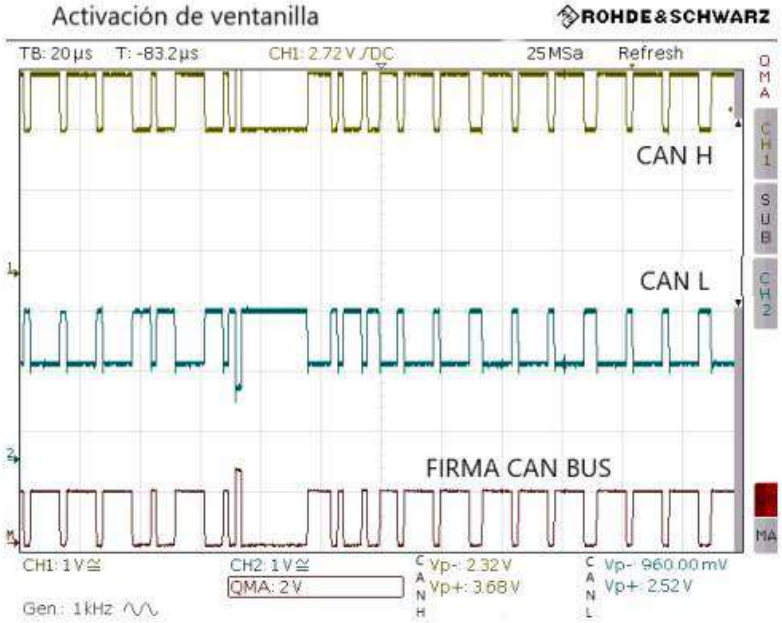


Figura 4.13 Tramas CAN BUS de activación de ventanilla copiloto

También, otro punto importante a destacar es la velocidad de transmisión de las tramas CAN dentro de la red como se muestra a continuación en la Figura 4.14, la cual representa la medición de velocidad de un bit correspondiente a una trama tipo estándar (compuesta por 55 bits) de luces de posición dando como resultado 490 kHz, o bien, 490 kbit/s, por lo que es posible el envío de 8909 mensajes por segundo. Aunado a ello, en el subtema 3.4.1 (Estructura de librerías para envío y recepción de tramas) se estableció dentro de la programación una velocidad para el CAN BUS de 500 kbit/s, lo cual corrobora que la medición realizada arroja un valor cercano al asignado confirmando así la hipótesis planteada.

El circuito de control ofrece una relevante velocidad de respuesta ante las demandas de los usuarios, además, este se encontrará funcionando en tiempo real mientras que el tablero se mantenga encendido.

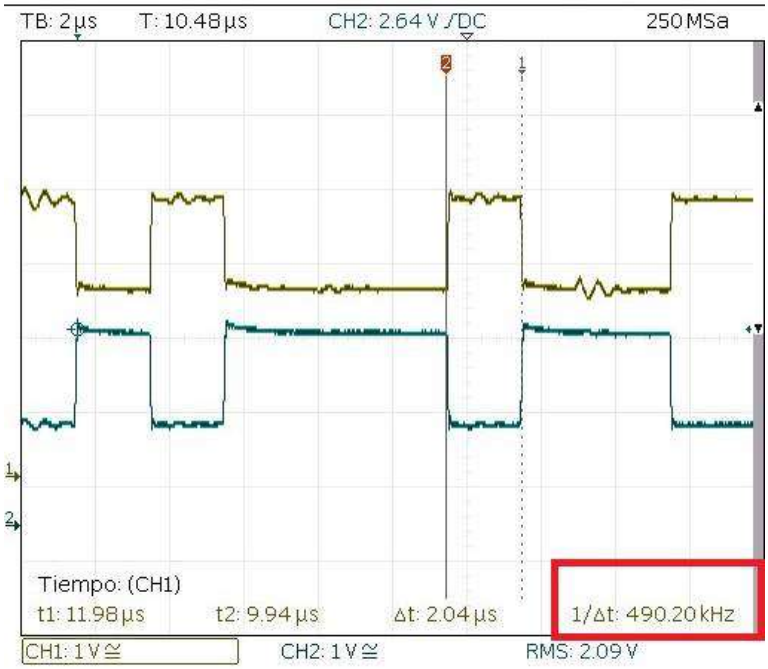


Figura 4.14 Medición de velocidad de transmisión de un bit en estado dominante de una trama de luces de posición

4.4.2 Activación de testigos en pantalla LCD

Durante la activación de cada uno de los sistemas se representa un testigo en la pantalla LCD como medio de comprobación visual de que el circuito se mantiene trabajando correctamente. En la Figura 4.15 se muestran los testigos activos en la pantalla LCD, así como en la Tabla 4.1 se describe detalladamente la simbología.

Figura 4.15 Testigos activos en pantalla LCD

Testigo	Descripción	
W1: U o W1: D	W1: “Up”	Ventanilla de conductor en movimiento (Subiendo o Bajando)
	W1: “Down”	
W2: U o W2: D	W2: “Up”	Ventanilla de Copiloto en movimiento (Subiendo o Bajando)
	W2: “Down”	
M1: N	M1: “North”	Espejo Retrovisor Left en movimiento hacia algún punto cardinal
M1: S	M1: “South”	
M1: E	M1: “East”	
M1: W	M1: “West”	
M2: N	M2: “North”	Espejo Retrovisor Right en movimiento hacia algún punto cardinal
M2: S	M2: “South”	
M2: E	M2: “East”	
M2: W	M2: “West”	
< >	Intermitentes o algún direccional activo	
N	Luz Neblineros	
P	Luz Posición	
B	Luz Baja	
A	Luz Alta	
S	Stop	
R	Reversa	

Tabla 4.1 Descripción de Simbología en pantalla LCD

4.4.3 Verificación de consumo de corriente continua del tablero didáctico

Si bien se tiene conocimiento del consumo eléctrico de cada uno de los componentes que integran al circuito de control, es conveniente corroborar mediante mediciones físicas el consumo de corriente continua que genera el tablero con los sistemas activos en cuanto a la parte de control (ECU´S) así como a la parte de potencia (actuadores de ventanillas, espejos y luces). En la Tabla 4.2 se muestran los valores de las mediciones realizadas con ayuda del multímetro.

<table>
<tr><td colspan="2">Parte de control</td><td colspan="3">Parte de potencia (Actuadores)</td></tr>
<tr><td>Línea</td><td>Consumo</td><td colspan="2">Línea</td><td>Consumo</td></tr>
<tr><td rowspan="4">• 5V ECU´S</td><td rowspan="4">• 1.07 A</td><td>• 12 V</td><td>• ECU´S para Sistema de ventanillas y/o espejos</td><td>• 1.48 A</td></tr>
<tr><td>• 12 V</td><td>• luces de posición, Stop, Reversa, Neblineros e intermitentes)</td><td>• 1.22 A</td></tr>
<tr><td>• 12 V</td><td>• Luz baja</td><td>• 280 mA</td></tr>
<tr><td>• 12 V</td><td>• Luz alta</td><td>• 560 mA</td></tr>
<tr><td>Total de Consumo (A)</td><td>= 1.07 A</td><td></td><td>Total, de Consumo (A)</td><td>= 3.54 A</td></tr>
</table>

Tabla 4.2 Consumo de Corriente Continua de Tablero Didáctico

Como se muestra en la Tabla 4.2, la parte de control del circuito tiene un consumo de 1.07 A, mientras que para la parte de potencia se tiene un consumo total de 3.54 A. Por lo tanto, al contar con una fuente de alimentación de 5V a 25A y 12V a 10A esta suministra la energía requerida para que el circuito pueda trabajar en óptimas condiciones.

CONCLUSIONES

Derivado de la metodología aplicada para la realización de este proyecto y el desarrollo e implementación de la programación planteada para la creación del circuito de control electrónico los resultados obtenidos fueron de carácter satisfactorio, comenzando con la obtención de las placas PCB catalogadas como unidades de control (ECU´S) para cada sistema del mismo. Posteriormente se procedió con la aplicación de pruebas técnicas verificando la funcionalidad del circuito de manera visual, mediante la acción de pulsar los botones de mando para cada sistema y corroborando la activación o desactivación de los componentes eléctricos que lo conforman como son: luces frontales, traseras, ventanillas y espejos piloto/copiloto, a su vez, de reflejarse los testigos en la pantalla LCD integrada en el circuito de la(s) tarea(s) que se está(n) realizando. También se realizaron mediciones bajo el osciloscopio mismas que demostraron señales de funcionamiento optimo representadas por la generación de tramas CAN BUS tipo estándar (55 bits) trabajando a una velocidad de 490 kbit/s; valor cercano al establecido inicialmente (500 kbit/s) dentro de la programación para la comunicación CAN logrando así el envío 8909 mensajes por segundo.

A su vez, las tramas CAN reflejan cambios en los nieles de voltaje para las líneas CAN HIGH y CAN LOW, entre los rangos de 2.36 a 3.72 V y de 2.6 a 1.16V respectivamente, mismos valores que se encuentran dentro de los parámetros teóricamente establecidos para las líneas de 2.5 a 3.75 V (CAN HIGH) y 2.5 a 1.25 V (CAN LOW). Tomando como referencia que el protocolo de comunicación CAN se basa en un voltaje diferencial entre estas dos líneas del BUS se dice que a partir de ello surge la encriptación de la información en código binario para la recepción y transmisión de datos dentro de la red de comunicación.

RECOMENDACIONES

- Para la obtención de resultados positivos en la generación de circuitos de control se recomienda el análisis previo de diagramas eléctricos de los sistemas a abordar, así como de los componentes a emplear tomando en cuenta características, funcionalidad y especificaciones.

- Si bien, a las ECUS destinadas al envío de información se les asigno un ID identificador propio, que les proporciono una mayor o menor prioridad al momento de transmitir, esto no logro garantizar que las ECUS pudieran hacerlo de manera consecutiva, ya que, por ejemplo, si la ECU de primer orden decide mandar mensajes todo el tiempo, las ECUS de segundo y tercer orden difícilmente podrán transmitir. Es por ello que se debe garantizar que las ECUS puedan enviar información respetando su turno. Una solución practica dentro de este proyecto fue el empleo de delays al final de la programación de cada ECU. Para futuros trabajos que demanden programaciones más complejas con mayor número de sistemas y ECUS se recomienda sustituir a los delays por el uso de interrupciones dentro de la programación debido a que estas ofrecen respuestas mucho más rápidas en la comunicación CAN BUS.

Con este proyecto se espera aportar información de relevancia que sirva como base a futuros proyectos referentes al uso del protocolo de comunicación CAN BUS por parte de los alumnos de la Ingeniería en Sistemas Automotrices del ITSLibres exhortándolos a aplicar dicho protocolo a demás sistemas electrónicos del automóvil debido que este proyecto solo va enfocado en los sistemas de ventanillas, espejos, luces frontales y traseras, de este modo ampliar las herramientas de apoyo para futuras generaciones.

REFERENCIAS

Alarms, A. S. (17 de Diciembre de 2020). *CAN BUS: componentes y características*. Obtenido de https://asiro.com/can-bus-componentes-y-caracteristicas/

Estudio, E. (17 de Abril de 2024). *¿Qué es un microcontrolador?* Obtenido de https://www.estudioelectronica.com/que-es-un-microcontrolador/

Ford. (2024). *Cosas que no sabías de los faros de tu vehículo*. Obtenido de https://www.ford.mx/blog/experto/curiosidades-faros-vehiculo-junio2023/#:~:text=Los%20faros%20de%20un%20veh%C3%ADculo,con%20poca%20o%20nula%20iluminaci%C3%B3n.

HELLA. (2024). *REVISIÓN DEL SENSOR DE LLUVIA - SENSOR DE LUZ*. Obtenido de https://www.hella.com/techworld/mx/Informacion-Tecnica/Electricidad-y-electronica-del-automovil/Revision-del-sensor-de-lluvia-Sensor-de-luz-42078/

Ltda, O. E. (14 de Junio de 2022). *Switches - Interruptores*. Obtenido de https://osakaelectronicsltda.com/switches-pulsadores/switches-interruptores/#:~:text=Los%20Switches%20Interruptores,el%20encendido%20y%20el%20apagado.

MotorOK. (28 de Julio de 2020). *¿Cómo funciona el protocolo de comunicación CAN bus en automoción?* Obtenido de https://motorok.com/noticias/protocolo-comunicacion-can-bus-automocion/

Puebla, G. d. (Julio de 2018). *Reglamento de la Ley de Vialidad para el Estado Libre y Soberano de Puebla .* Obtenido de https://www.tjaep.gob.mx/

Valle, J. S. (2024). *CLASIFICACIÓN DE MOTORES DE CORRIENTE CONTINUA SEGÚN SU FORMA DE EXCITACIÓN*. Obtenido de chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.citeenergia.com.pe/wp-content/uploads/2022/07/ARTICULO-WORD-1.pdf

AG, V. (14 de Septiembre de 2022). *MECANICO AUTOMOTRIZ* . Obtenido de file:///C:/Users/Dany7/Downloads/manual-can-bus-datos-bosch-red-area-controladordiseno-funcionamiento-transmision-sistema-confort-traccion.pdf

AUTOMOTRIZ, I. Y. (17 de MARZO de 2020). *INGENIERÍA Y MECANICA AUTOMOTRIZ* . Obtenido de https://www.ingenieriaymecanicaautomotriz.com/quees-el-can-bus-y-como-funciona/

Barrón Ruiz , M. (28 de Septiembre de 2022). USO DIDÁCTICO DEL SOFTWARE DE AYUDA AL DISEÑO ELECTRÓNICO "PROTEUS" . Leioa, Biscay, España .

Basco, B. (10 de Septiembre de 2019). *e-Automotive*. Obtenido de https://noticias-renting.aldautomotive.es/saber-fusiblescoche/#:~:text=Los%20fusibles%20son%20peque%C3%B1as%20piezas,sistemas% 20vitales%20puedan%20verse%20da%C3%B1ados.

Fernández, Y. (23 de Septiembre de 2022). *Xataka Basics*. Obtenido de https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-haceruno#:~:text=Arduino%20ofrece%20la%20plataforma%20Arduino,dar%20todo%20tip o%20de%20utilidades.

FiNAL TEST. (04 de Febrero de 2023). *FINAL TEST*. Obtenido de https://www.finaltest.com.mx/product-p/art-9.htm

García Osés , A. (24 de Junio de 2015). Diseño de una red CAN BUS con Arduino . Pamplona , España.

Iglesias Paz, R. (16 de Enero de 2019*). CAN BUS DE TRACCION J 587 UCE PALANCA ID J 104 UCE ABS ID 4A0 J 431 UCE ALCANCE DE FAROS ID G 85 TRANSMISOR ANGULO DE GIRO ID 0C2*. Obtenido de https://docplayer.es/95373978-Can-bus-de-traccion-j-587-uce-palanca-id-j-104-uceabs-id-4a0-j-431-uce-alcance-de-faros-id-g-85-transmisor-angulo-de-giro-id-0c2.html 56

Martínez Requena , A., & García Martín, J. (04 de Julio de 2017). Introducción a CAN bus: Descripción, ejemplos y aplicaciones de tiempo real. Madrid, España .

Sánchez Vela , L., Molano Clemente , M., Fabela Gallegos , M., Martínez Madrid, M., Hernández Jiménez, J., Vázquez Vega , D., & Flores Centeno, O. (2016). Revisión documental del protocolo CAN como herramienta de comunicación y aplicación en vehículos . Sanfandila , Queretaro , México .

SDI. (15 de Febrero de 2022). *SDI.* Obtenido de https://sdindustrial.com.mx/blog/relevadores/#:~:text=%C2%BFQu%C3%A9%20son %20los%20relevadores%3F,este%20mismo%20tipo%20de%20energ%C3%ADa.

Suárez Guedes , R. (16 de Febrero de 2023). *Blog de tecnologías* . Obtenido de https://www3.gobiernodecanarias.org/medusa/ecoblog/rsuagued/arduino/

GLOSARIO

ECU: Se hace referencia a una unidad de control electrónico que tiene como objetivo controlar y regular los sistemas electrónicos del automóvil. Es decir, la principal tarea de la Ecu es controlar cada uno de los procesos que mantienen al vehículo funcionando de manera correcta, y controlar toda la información que este pueda disponer.

Delay: Dentro de la programación el uso de “Dalay()” detiene la programación durante una cantidad específica de milisegundos que se hayan especificado.

CAN BUS: Significa Controller Area Network (Red de área de control) y BUS, en informática, se entiende como un elemento que permite transportar información.

Bit: En informática, bit es la unidad mínima de información. Según esta definición, un bit es un dígito del sistema de numeración binario, que se representa con dos valores, el 0 y el 1.

Byte: Un byte es una unida de datos en informática, que representa un grupo de los bits que se puede utilizar para codificar un solo carácter (letra, número, símbolo). El byte estándar consta de ocho bits.

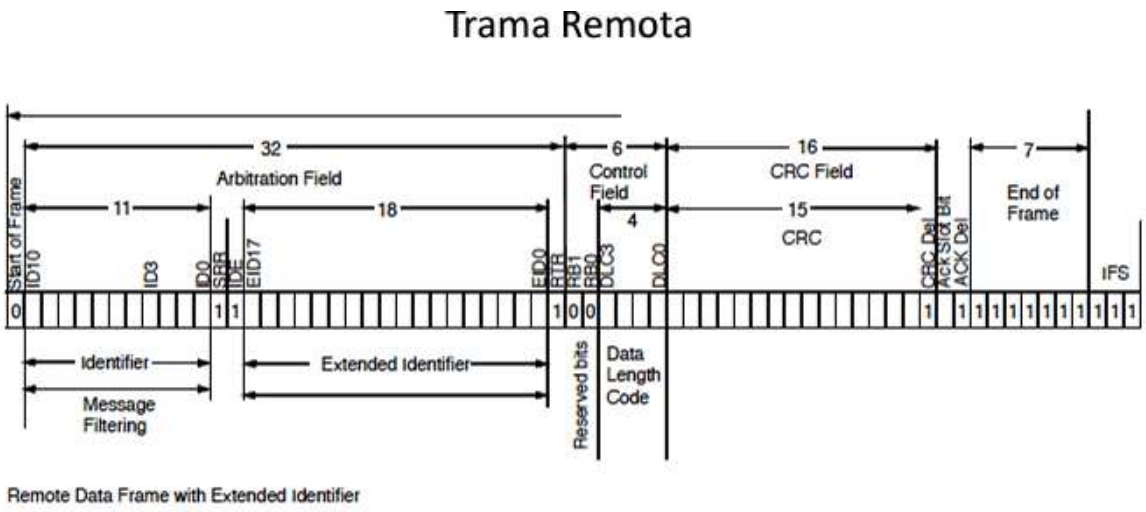
Trama CAN: Es una estructura definida de cierto número bits necesarios para la creación de mensajes CAN.

Modulo L298N: Es un controlador de motor de alto voltaje, alta corriente, doble puente completo, diseñado para aceptar estándares de niveles lógicos TTL y manejar cargas inductivas tales como relés, solenoides, motores DC y de paso.

Interrupciones: Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción.

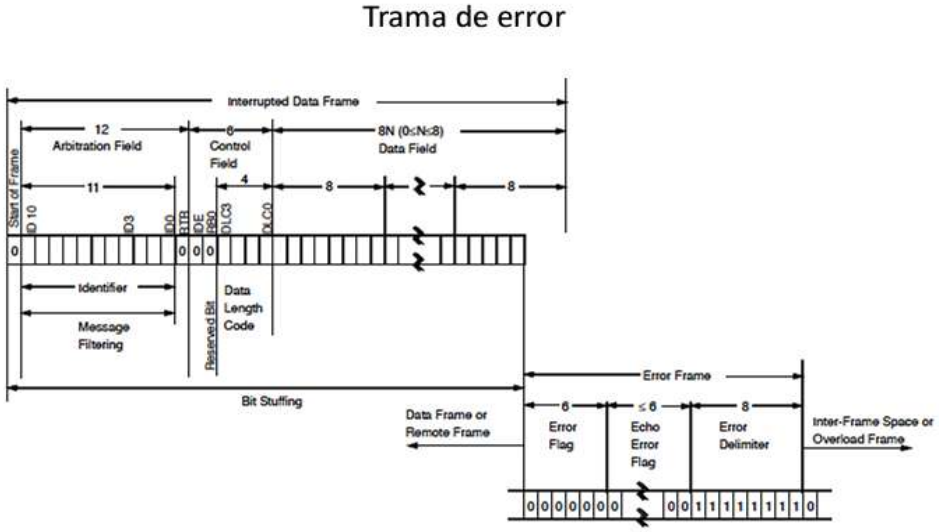
ANEXOS

Anexo 1. Formato de trama remota CAN BUS

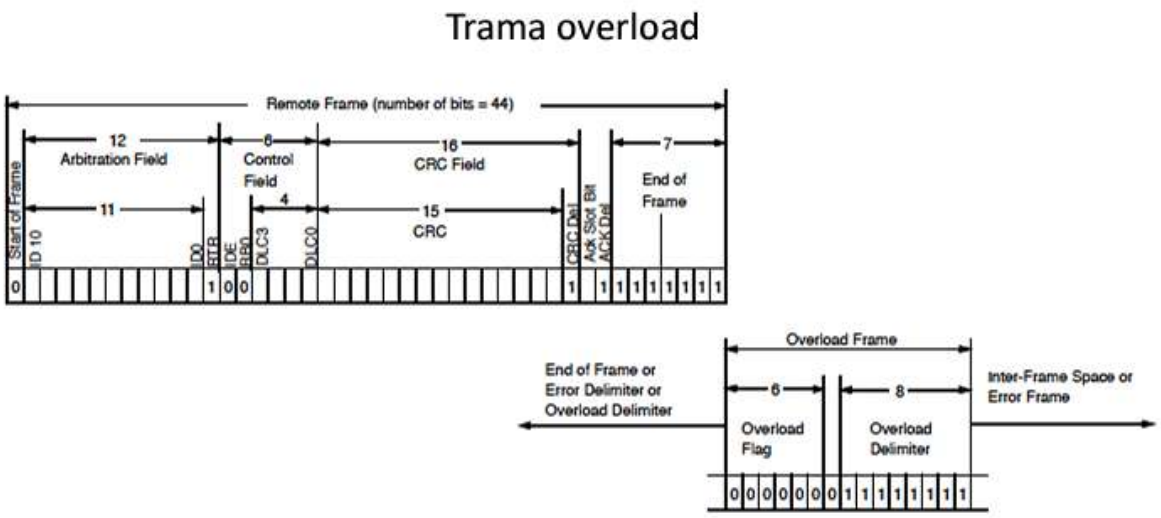


Remote Data Frame with Extended Identifier

Anexo 2. Formato de trama de error CAN BUS



Anexo 3. Formato de trama de saturación (Overload frame) CAN BUS



Anexo 4. Data sheet de Microcontrolador Atmega 328P

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

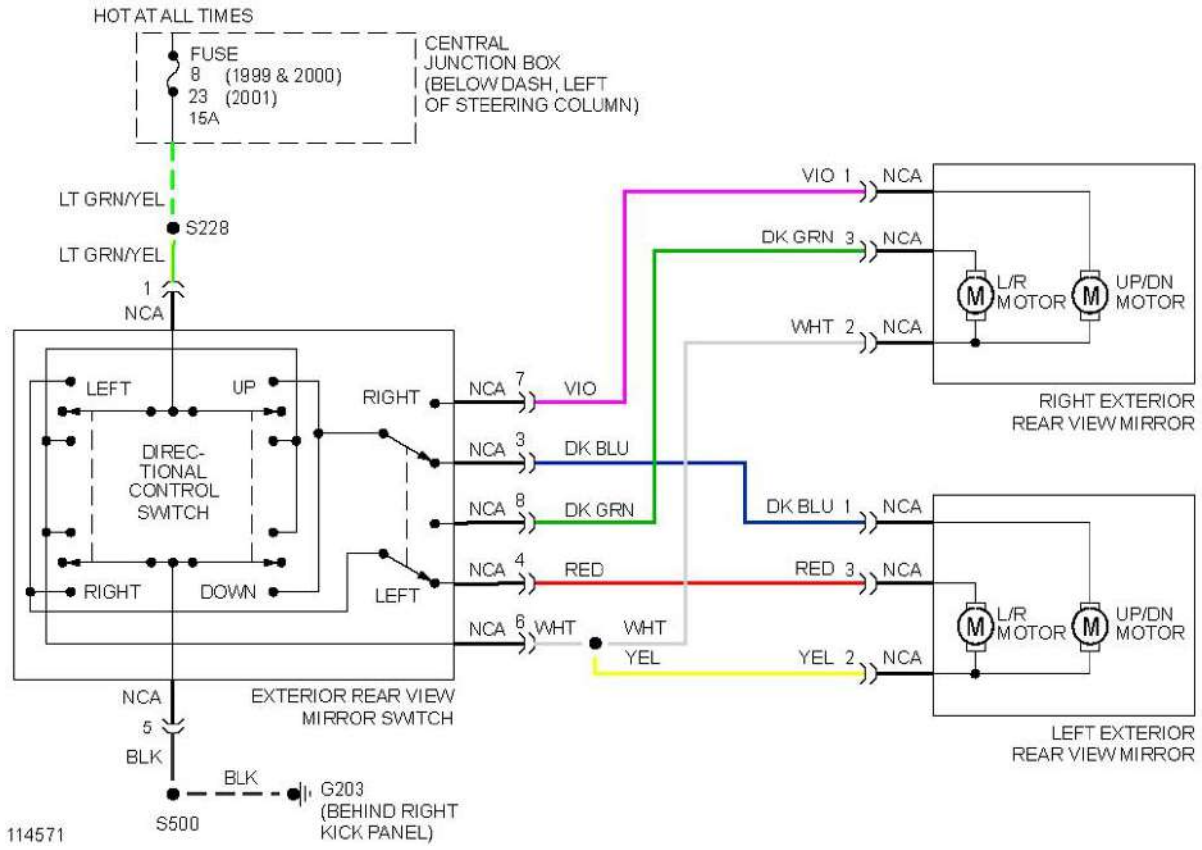
ATMEL

8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

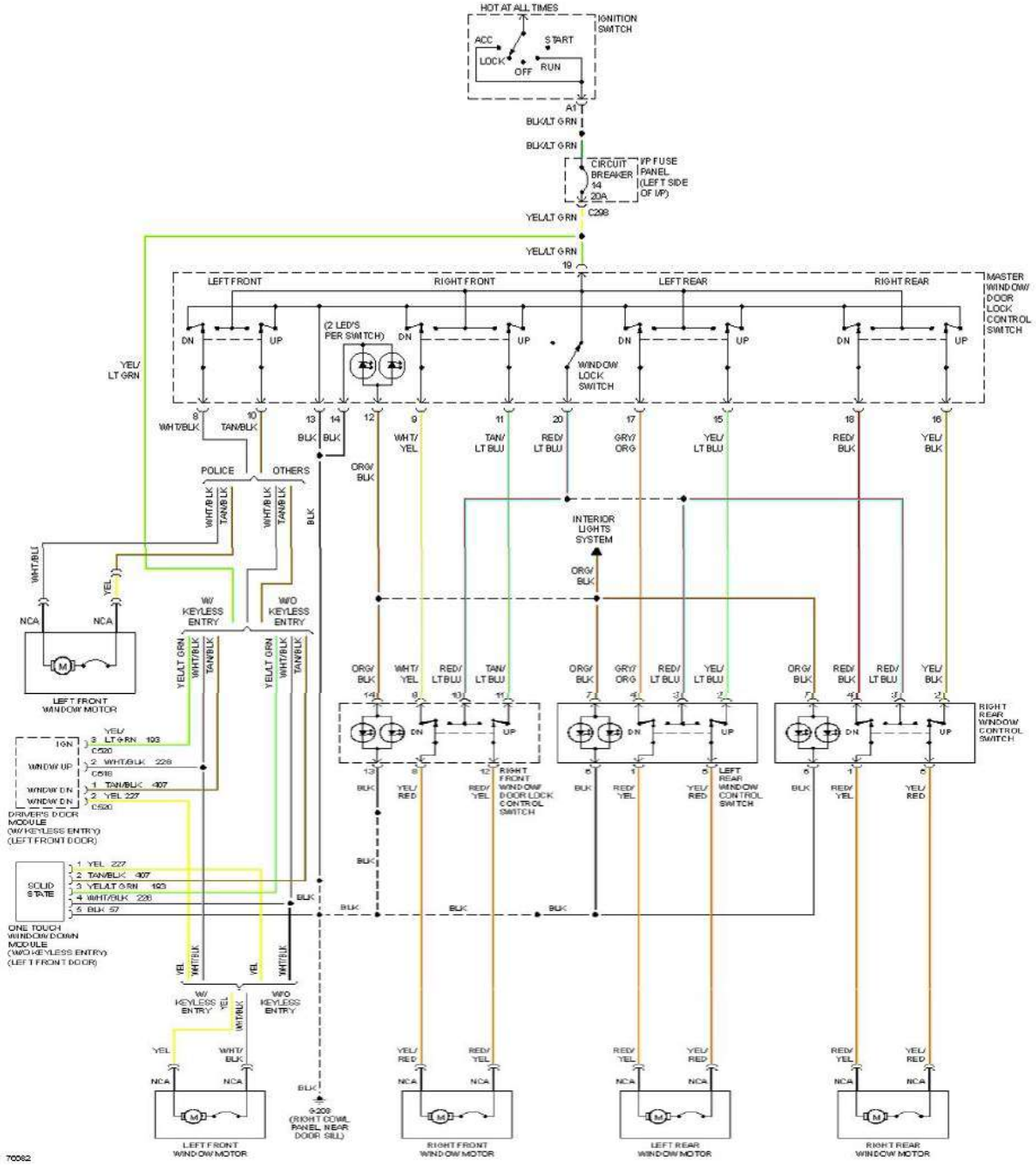
ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Summary

Anexo 5. Diagrama eléctrico de espejos de Mercury Gran Marquiz modelo 2000



Anexo 6. Diagrama de sistema eléctrico de ventanillas de Mercury Gran Marquiz modelo 2000



Anexo 7. Código de programación en Arduino IDE de ECU Máster

```
#include <SPI.h>              // Se define Librerias
#include <mcp2515.h>

#define sensorlluvia      A0
#define fotoresistencia   A1
#define posicion          A2
#define luzbaja           A3
#define luzalta            3
#define intermitente       4
#define direcizq           5
#define direcder           6
#define botonnieblero      7
#define botonstop          8
#define botonreversa       9

int senslluvia     =0;
int fotoresis      =0;
int botonnieble    =0;
int botonposi      =0;
int direccionalizq =0;
int direccionalder =0;
int inter          =0;
int posi           =0;    // variable de luz posicion
int nieble         =0;    // variable de luz nieblero
int dd             =0;    // variable direccional derecho
int di             =0;    // variable direccional izquierdo
int lbaja          =0;    // variable de luz baja
int lalta          =0;    // variable de luz alta
int stop           =0;    // variable de stop
int reversa        =0;    // variable de reversa

struct can_frame canMsg1;  // tramas de Sistena de Luces
MCP2515 mcp2515(10);

void setup() {


pinMode(sensorlluvia,       INPUT);
pinMode(fotoresistencia,    INPUT);
pinMode(posicion,           INPUT);
pinMode(luzbaja,            INPUT);
pinMode(luzalta,            INPUT);
pinMode(intermitente,       INPUT);
pinMode(direcizq,           INPUT);
pinMode(direcder,           INPUT);
pinMode(botonnieblero,      INPUT);
pinMode(botonstop,          INPUT);
pinMode(botonreversa,       INPUT);

```

```
50 SPI.begin();
51 Serial.begin(9600);
52 mcp2515.reset();
53 mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);  // declarar la velocidad
54 mcp2515.setNormalMode();                   // del Bus Can
55
56 canMsg1.can_id  = 0xAA;  // ID Trama de luces
57 canMsg1.can_dlc = 8;     // define el tamaño del campo de datos
58
59 }
60
61 void loop() {
62
63     int senslluvia     = digitalRead(sensorlluvia);
64     int fotoresis      = digitalRead(fotoresistencia);
65     int botonnieble    = digitalRead(botonnieblero);
66     int botonposi      = digitalRead(posicion);
67     int direccionalizq = digitalRead(direcizq);
68     int direccionalder = digitalRead(direcder);
69     int inter          = digitalRead(intermitente);
70     int lbaja          = digitalRead(luzbaja);
71     int lalta          = digitalRead(luzalta);
72     int stop           = digitalRead(botonstop);
73     int reversa        = digitalRead(botonreversa);
74
75     if (fotoresis == HIGH || botonposi == HIGH )
76     {
77          posi = HIGH ;
78     }
79     else
80     {
81         posi = LOW;
82     }
83     if (senslluvia == LOW || botonnieble == HIGH )
84     {
85          nieble = HIGH;
86     }
87     else
88     {
89          nieble = LOW;
90     }
91     if (inter == HIGH & direccionalizq == LOW & direccionalder == LOW)
92     {
93          di = HIGH;
94          dd = HIGH;
95     }
96     else if (inter == LOW & direccionalizq == HIGH)
97     {
98          di = HIGH;
99          dd = LOW;
100    }
101    else if (inter == LOW & direccionalder == HIGH)
```

```
102     {
103          di = LOW;
104          dd = HIGH;
105     }
106     else
107     {
108       di     = LOW;
109       dd     = LOW;
110     }
111
112        canMsg1.data[0]  = posi;  // Campo de datos y estado de sus Bits
113        canMsg1.data[1]  = nieble;
114        canMsg1.data[2]  = dd;
115        canMsg1.data[3]  = di;
116        canMsg1.data[4]  = lbaja;
117        canMsg1.data[5]  = lalta;
118        canMsg1.data[6]  = stop;
119        canMsg1.data[7]  = reversa;
120        mcp2515.sendMessage(&canMsg1);  //envio de mensaje
121        delay(100);
122 }
```

Anexo 8. Código de programación en Arduino IDE de ECU Window y Mirror Left

```
 1 #include <SPI.h>              // Se define Librerias
 2 #include <mcp2515.h>
 3
 4 #define RetroIzqleft       A2
 5 #define RetroIzqright      A3
 6 #define RetroIzqdown       A4
 7 #define RetroIzqup         A5
 8 #define RetroDerleft       3
 9 #define RetroDerright      4
10 #define RetroDerdown       5
11 #define RetroDerup         6
12 #define VentaIzqsube       A0
13 #define VentaIzqbaja       A1
14 #define VentaDersube       7
15 #define VentaDerbaja       8
16 #define VentaDerpermiso    9
17
18 int VDsube    =0;
19 int VDbaja    =0;
20 int VDpermiso =0;
21 int RDleft    =0;
22 int RDright   =0;
23 int RDdown    =0;
24 int RDup      =0;
```

```
25
26 struct can_frame canMsg2;  // tramas para ECU copiloto
27 struct can_frame canMsg3;  // tramas para ECU LCD
28 MCP2515 mcp2515(10);
29
30 void setup(){
31
32   pinMode(VentaIzqsube,           INPUT);
33   pinMode(VentaIzqbaja,           INPUT);
34   pinMode(VentaDersube,           INPUT);
35   pinMode(VentaDerbaja,           INPUT);
36   pinMode(VentaDerpermiso,        INPUT);
37   pinMode(RetroIzqleft,           INPUT);
38   pinMode(RetroIzqright,          INPUT);
39   pinMode(RetroIzqdown,           INPUT);
40   pinMode(RetroIzqup,             INPUT);
41   pinMode(RetroDerleft,           INPUT);
42   pinMode(RetroDerright,          INPUT);
43   pinMode(RetroDerdown,           INPUT);
44   pinMode(RetroDerup,             INPUT);
45
46   Serial.begin(9600);
47   SPI.begin();
48   mcp2515.reset();
49   mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);  // declarar la velocidad
50   mcp2515.setNormalMode();                   // del Bus Can
51   canMsg2.can_id  = 0xBB;  // identificador de la trama de copiloto
52   canMsg2.can_dlc = 7;     // tamaño de campo de datos
53   canMsg3.can_id  = 0xCC;  // identificador de la trama de copiloto
54   canMsg3.can_dlc = 6;     // tamaño de campo de datos
55 }
56
57 void loop() {
58
59   int VIsube    = digitalRead(VentaIzqsube);
60   int VIbaja    = digitalRead(VentaIzqbaja);
61   int VDsube    = digitalRead(VentaDersube);
62   int VDbaja    = digitalRead(VentaDerbaja);
63   int VDpermiso = digitalRead(VentaDerpermiso);
64   int RIleft    = digitalRead(RetroIzqleft);
65   int RIright   = digitalRead(RetroIzqright);
66   int RIdown    = digitalRead(RetroIzqdown);
67   int RIup      = digitalRead(RetroIzqup);
68   int RDleft    = digitalRead(RetroDerleft);
69   int RDright   = digitalRead(RetroDerright);
70   int RDdown    = digitalRead(RetroDerdown);
71   int RDup      = digitalRead(RetroDerup);
72
73   canMsg2.data[0] = RDleft;    // Tramas de
74   canMsg2.data[1] = RDright;  // actuadores
75   canMsg2.data[2] = RDdown;  // de puerta RIGHT
76   canMsg2.data[3] = RDup;
```

```
77   canMsg2.data[4] = VDsube;
78   canMsg2.data[5] = VDbaja;
79   canMsg2.data[6] = VDpermiso;
80   mcp2515.sendMessage(&canMsg2);  //envio de mensaje
81
82   canMsg3.data[0] = RIleft;    // Tramat de
83   canMsg3.data[1] = RIright;  // actuadores
84   canMsg3.data[2] = RIdown;  // de puerta LEFT
85   canMsg3.data[3] = RIup;
86   canMsg3.data[4] = VIsube;
87   canMsg3.data[5] = VIbaja;
88   mcp2515.sendMessage(&canMsg3);  //envio de mensaje
89   delay(10);
90 }
```

Anexo 9. Código de programación en Arduino IDE de ECU Window y Mirror Right

```
 1 #include <SPI.h>        // definir librerías
 2 #include <mcp2515.h>
 3
 4 #define VentaDersube       A1   // a los pines del Arduino
 5 #define VentaDerbaja       A0  // se le asigna valor
 6 #define subirventa         3
 7 #define bajarventa         4
 8 #define RDL                5
 9 #define RDR                6
10 #define RDD                7
11 #define RDU                8
12
13 int sventa      =0;
14 int bventa      =0;
15 int vventa      =0;   // variables de ventanilla derecha
16 int RDleft      =0;
17 int RDright     =0;
18 int RDdown      =0;
19 int RDup        =0;
20 int dsubirventa =0;
21 int dbajarventa =0;
22 int subirw      =0;
23 int bajarw      =0;
24
25 struct can_frame canMsg2;  // Tramas de espejo Retrovisor
26 struct can_frame canMsg4;
27 MCP2515 mcp2515(10);
28
29 void setup() {
30
```

```
  pinMode(VentaDersube,    INPUT);
  pinMode(VentaDerbaja,    INPUT);//Señales de Copiloto para ventanilla
  pinMode(subirventa,     OUTPUT);
  pinMode(bajarventa,     OUTPUT);
  pinMode(RDL,            OUTPUT);//Declarar pines de salida espejo R
  pinMode(RDR,            OUTPUT);
  pinMode(RDD,            OUTPUT);
  pinMode(RDU,            OUTPUT);

  Serial.begin(9600);
  SPI.begin();
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);  // velocidad del Bus CAN
  mcp2515.setNormalMode();
  canMsg4.can_id  = 0xDD;  // identificador de la trama de copiloto
  canMsg4.can_dlc = 6;     // tamaño de campo de datos
}

void loop() {

   int dbajarventa = digitalRead(VentaDerbaja);//Asignación de variable
   int dsubirventa = digitalRead(VentaDersube); // pines del Atmega328P
   if (mcp2515.readMessage(&canMsg2) == MCP2515::ERROR_OK)
   {    //CODIGO ESPEJO y VENTANILLA DERECHO
      if (canMsg2.can_id == 0xBB)
      {
        int RDleft       = canMsg2.data[0];  //estado de los bits
        int RDright      = canMsg2.data[1];  //en el campo de datos
        int RDdown       = canMsg2.data[2];
        int RDup         = canMsg2.data[3];
        int sventa       = canMsg2.data[4];//Permiso de condu a copilo
        int bventa       = canMsg2.data[5];//condu manipula venta copil
        int vventa       = canMsg2.data[6];//condu manipula venta copil

        if (RDup == HIGH)
        {
            digitalWrite(RDL,  LOW);  // Mover espejo al norte
            digitalWrite(RDR,  LOW);
            digitalWrite(RDD,  LOW);
            digitalWrite(RDU, HIGH);
        }
        else if (RDdown == HIGH)
        {
            digitalWrite(RDL,  LOW);
            digitalWrite(RDR,  LOW);  //Mover espejo al sur
            digitalWrite(RDD, HIGH);
            digitalWrite(RDU,  LOW);
        }
        else if (RDleft == HIGH)
        {
            digitalWrite(RDL, HIGH);
            digitalWrite(RDR,  LOW);
```

```
 83            digitalWrite(RDD,  LOW);  //Mover espejo al Oeste
 84            digitalWrite(RDU,  LOW);
 85        }
 86        else if (RDright == HIGH)
 87        {
 88            digitalWrite(RDL,  LOW);
 89            digitalWrite(RDR, HIGH);
 90            digitalWrite(RDD,  LOW);
 91            digitalWrite(RDU,  LOW);   //Mover espejo al Este
 92        }
 93        else
 94        {
 95            digitalWrite(RDL,  LOW);
 96            digitalWrite(RDR,  LOW);
 97            digitalWrite(RDD,  LOW);
 98            digitalWrite(RDU,  LOW);
 99        }
100
101        if ( bventa == HIGH & vventa == LOW)//VENTANILLA DERECHO
102        {
103            digitalWrite(bajarventa, HIGH);
104            digitalWrite(subirventa, LOW);  //bajar ventanilla
105            bajarw = HIGH;
106            subirw = LOW;
107        }
108        else if ( sventa == HIGH & vventa == LOW)
109        {
110            digitalWrite(bajarventa, LOW);  //Subir ventanilla
111            digitalWrite(subirventa, HIGH);
112            bajarw = LOW;
113            subirw = HIGH;
114        }
115        else if ( bventa == HIGH & vventa == HIGH & dsubirventa == LOW)
116        {
117            digitalWrite(bajarventa, HIGH);
118            digitalWrite(subirventa, LOW);  //bajar ventanilla
119            bajarw = HIGH;
120            subirw = LOW;
121        }
122        else if ( sventa == HIGH & vventa == HIGH & dbajarventa == LOW)
123        {
124            digitalWrite(bajarventa, LOW);  //Subir ventanilla
125            digitalWrite(subirventa, HIGH);
126            bajarw = LOW;
127            subirw = HIGH;
128        }
129        else if (vventa == HIGH & dbajarventa == HIGH )
130        {
131          if ( sventa == HIGH)
132          {
133              digitalWrite(bajarventa, LOW);  //Subir ventanilla
134              digitalWrite(subirventa, HIGH);
```

```
135             bajarw = LOW;
136             subirw = HIGH;
137           }
138           else
139           {
140             digitalWrite(bajarventa, HIGH);
141             digitalWrite(subirventa, LOW);  //bajar ventanilla
142             bajarw = HIGH;
143             subirw = LOW;
144           }
145         }
146         else if (vventa == HIGH & dsubirventa == HIGH )
147         {
148           if (bventa == HIGH)
149           {
150             digitalWrite(bajarventa, HIGH);
151             digitalWrite(subirventa, LOW);  //bajar ventanilla
152             bajarw = HIGH;
153             subirw = LOW;
154           }
155           else
156           {
157             digitalWrite(bajarventa, LOW);  //subir ventanilla
158             digitalWrite(subirventa, HIGH);
159             bajarw = LOW;
160             subirw = HIGH;
161           }
162         }
163         else
164         {
165             digitalWrite(bajarventa, LOW);
166             digitalWrite(subirventa, LOW);
167             bajarw = LOW;
168             subirw = LOW;
169         }
170         canMsg4.data[0] = RDleft;    // Tramas de
171         canMsg4.data[1] = RDright;  // actuadores
172         canMsg4.data[2] = RDdown;  // de puerta RIGHT
173         canMsg4.data[3] = RDup;
174         canMsg4.data[4] = subirw;
175         canMsg4.data[5] = bajarw;
176         mcp2515.sendMessage(&canMsg4);  //envio de mensaje
177       }
178     }
179 }
```

Anexo 10. Código de programación en Arduino IDE de ECU Luces Frontales

```
#include <SPI.h>     //Definir librerias
#include <mcp2515.h>

#define posicion   3
#define direcleft  4
#define direcright 5
#define luzbaja    6
#define luzalta    7
#define nieblero   8

int posi    = 0;
int nieble  = 0;
int dd      = 0;
int di      = 0;
int lbaja   = 0;
int lalta   = 0;
int stop    = 0;
int reversa = 0;

struct can_frame canMsg1;   //tramas sistema de luces
MCP2515 mcp2515(10);

void setup() {
//define los pines de salida del sistema de luces
pinMode(posicion,  OUTPUT);     //Luz Direccional Izq
pinMode(direcleft, OUTPUT);     //luz Direccional Der
pinMode(direcright,OUTPUT);     //luz Posicion
pinMode(luzbaja,   OUTPUT);     //luz Nieblero
pinMode(luzalta,   OUTPUT);     //Luz Baja
pinMode(nieblero,  OUTPUT);     //LUZ Alta

SPI.begin();
Serial.begin(9600);
mcp2515.reset();
mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ); //velocidad del BUS CAN
mcp2515.setNormalMode();
}

void loop()
{

   if (mcp2515.readMessage(&canMsg1) == MCP2515::ERROR_OK)
  {    //trama de sistema de luces
      if (canMsg1.can_id == 0xAA)     //identificador sistema de luces
      {
        int posi         = canMsg1.data[0];  //estado de los bits
        int nieble       = canMsg1.data[1];  //en el campo de datos
        int dd           = canMsg1.data[2];
        int di           = canMsg1.data[3];
```

```
50          int lbaja         = canMsg1.data[4];
51          int lalta         = canMsg1.data[5];
52          int stop          = canMsg1.data[6];
53          int reversa       = canMsg1.data[7];
54
55          if (di == HIGH & dd == LOW )  //Activar direccional izquierdo
56          {
57              digitalWrite(direcleft,   LOW);
58              digitalWrite(direcright, HIGH);
59          }
60          else if (dd == HIGH & di == LOW) //Activar direccional derecho
61          {
62              digitalWrite(direcleft, HIGH);
63              digitalWrite(direcright, LOW);
64          }
65          else if (dd == HIGH & di == HIGH ) //Activar intermitentes
66          {
67              digitalWrite(direcleft,  LOW);
68              digitalWrite(direcright, LOW);
69          }
70          else
71          {
72              digitalWrite(direcleft,  HIGH);
73              digitalWrite(direcright, HIGH);
74          }
75
76          if (posi == LOW & nieble == LOW & lbaja == LOW & lalta == HIGH)
77          {   //lalta ON
78              digitalWrite(posicion,   HIGH);
79              digitalWrite(nieblero,   HIGH);
80              digitalWrite(luzbaja,    HIGH);
81              digitalWrite(luzalta,     LOW);
82          }
83          else if (nieble == HIGH & posi == LOW  )
84          {    // nieblero, posición, lbaja y/o lalta ON
85              if (lalta == HIGH)
86              {
87                digitalWrite(posicion, LOW);
88                digitalWrite(nieblero, LOW);
89                digitalWrite(luzbaja, HIGH);
90                digitalWrite(luzalta,  LOW);
91              }
92              else
93              {
94                digitalWrite(posicion, LOW);
95                digitalWrite(nieblero, LOW);
96                digitalWrite(luzbaja, HIGH);
97                digitalWrite(luzalta, HIGH);
98              }
99          }
100         else if (nieble == HIGH & posi == HIGH & lbaja == LOW )
101         {    // nieblero, posición, lbaja y/o lalta ON
```

```
102             if (lalta == HIGH)
103             {
104               digitalWrite(posicion,  LOW);
105               digitalWrite(nieblero,  LOW);
106               digitalWrite(luzbaja,  HIGH);
107               digitalWrite(luzalta,   LOW);
108             }
109             else
110             {
111               digitalWrite(posicion,  LOW);
112               digitalWrite(nieblero,  LOW);
113               digitalWrite(luzbaja,  HIGH);
114               digitalWrite(luzalta,  HIGH);
115             }
116         }
117         else if (nieble == HIGH & posi == HIGH & lbaja == HIGH)
118         {    // nieblero, posición, lbaja y/o lalta ON
119             if (lalta == HIGH)
120             {
121               digitalWrite(posicion,  LOW);
122               digitalWrite(nieblero,  LOW);
123               digitalWrite(luzbaja,  HIGH);
124               digitalWrite(luzalta,   LOW);
125             }
126             else
127             {
128               digitalWrite(posicion,  LOW);
129               digitalWrite(nieblero,  LOW);
130               digitalWrite(luzbaja,   LOW);
131               digitalWrite(luzalta,  HIGH);
132             }
133         }
134         else if (nieble == LOW & posi == HIGH & lbaja == LOW)
135         {    //Posi Y/O lalta ON
136             if (lalta == HIGH)
137             {
138               digitalWrite(posicion,  LOW);
139               digitalWrite(nieblero, HIGH);
140               digitalWrite(luzbaja,  HIGH);
141               digitalWrite(luzalta,   LOW);
142             }
143             else
144             {
145               digitalWrite(posicion,   LOW);
146               digitalWrite(nieblero,  HIGH);
147               digitalWrite(luzbaja,   HIGH);
148               digitalWrite(luzalta,   HIGH);
149             }
150         }
151         else if (nieble == LOW & posi == HIGH & lbaja == HIGH)
152         {   //Posi, lbaja y/o lalta ON
153             if (lalta == HIGH)
```

```
154             {
155               digitalWrite(posicion,   LOW);
156               digitalWrite(nieblero,  HIGH);
157               digitalWrite(luzbaja,   HIGH);
158               digitalWrite(luzalta,    LOW);
159             }
160             else
161             {
162               digitalWrite(posicion,   LOW);
163               digitalWrite(nieblero,  HIGH);
164               digitalWrite(luzbaja,    LOW);
165               digitalWrite(luzalta,   HIGH);
166             }
167         }
168         else
169         {
170             digitalWrite(posicion,  HIGH);
171             digitalWrite(nieblero,  HIGH);
172             digitalWrite(luzbaja,   HIGH);
173             digitalWrite(luzalta,   HIGH);
174         }
175       }
176   }
177 }
```

Anexo 11. Código de programación en Arduino IDE de ECU Luces Traseras

```
 1 #include <SPI.h>        //definir librerías
 2 #include <mcp2515.h>
 3
 4 #define direcleft  4
 5 #define direcright 5
 6 #define posicion   3
 7 #define luzstop    6
 8 #define luzreversa 7
 9
10 int posi    =0;
11 int nieble  =0;
12 int dd      =0;
13 int di      =0;
14 int lbaja   =0;
15 int lalta   =0;
16 int stop    =0;
17 int reversa =0;
18
19 struct can_frame canMsg1;  //tramas de sistema de luces
20 MCP2515 mcp2515(10);
21
```

```
22 void setup() {
23      //define los pines de salida del sistema de luces
24 pinMode(direcleft,  OUTPUT);   //direccional izquierdo
25 pinMode(direcright, OUTPUT);   //direccional derecho
26 pinMode(posicion,   OUTPUT);   //luces posicion
27 pinMode(luzstop,    OUTPUT);   //luz stop
28 pinMode(luzreversa, OUTPUT);  // lUZ REVERSA
29
30 SPI.begin();
31 Serial.begin(9600);
32 mcp2515.reset();
33 mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ); //velocidad del CAN BUS
34 mcp2515.setNormalMode();
35 }
36 void loop()
37 {
38    if (mcp2515.readMessage(&canMsg1) == MCP2515::ERROR_OK)
39   {      //tramas del sistema de luces
40       if (canMsg1.can_id == 0xAA)     //ID del sistema de luces
41       {
42         int posi         = canMsg1.data[0];  //estado de los bits
43         int nieble       = canMsg1.data[1];  //en el campo de datos
44         int dd           = canMsg1.data[2];
45         int di           = canMsg1.data[3];
46         int lbaja        = canMsg1.data[4];
47         int lalta        = canMsg1.data[5];
48         int stop         = canMsg1.data[6];
49         int reversa      = canMsg1.data[7];
50
51         if (di == HIGH & dd == LOW )    //Activar direccional izquierdo
52         {
53             digitalWrite(direcleft,    LOW);
54             digitalWrite(direcright,  HIGH);
55         }
56         else if (dd == HIGH & di == LOW) //Activar direccional derecho
57         {
58             digitalWrite(direcleft,  HIGH);
59             digitalWrite(direcright,  LOW);
60         }
61         else if (dd == HIGH & di == HIGH ) //Activar intermitentes
62         {
63             digitalWrite(direcleft,   LOW);
64             digitalWrite(direcright,  LOW);
65         }
66         else
67         {
68             digitalWrite(direcleft,  HIGH);
69             digitalWrite(direcright, HIGH);
70         }
71         if (posi == HIGH || nieble == HIGH)
72         {
73           if (stop == LOW & reversa == HIGH)
```

```
74            {  //Posi y Reversa ON
75              digitalWrite(posicion,   LOW);
76              digitalWrite(luzstop,   HIGH);
77              digitalWrite(luzreversa, LOW);
78            }
79            else if (stop == HIGH & reversa == LOW)
80            {    //Posi & STOP ON
81              digitalWrite(posicion,    LOW);
82              digitalWrite(luzstop,     LOW);
83              digitalWrite(luzreversa, HIGH);
84            }
85            else if (stop == HIGH & reversa == HIGH)
86            {   //Posi, Reversa y Stop ON
87              digitalWrite(posicion,   LOW);
88              digitalWrite(luzstop,    LOW);
89              digitalWrite(luzreversa, LOW);
90            }
91            else                      //Posi ON
92            {
93              digitalWrite(posicion,    LOW);
94              digitalWrite(luzstop,    HIGH);
95              digitalWrite(luzreversa, HIGH);
96            }
97        }
98        else if (posi == LOW & nieble == LOW )
99        {
100           if (stop == LOW & reversa == HIGH)  //Reversa ON
101           {
102             digitalWrite(posicion,  HIGH);
103             digitalWrite(luzstop,   HIGH);
104             digitalWrite(luzreversa, LOW);
105           }
106           else if (stop == HIGH & reversa == LOW)  // STOP ON
107           {
108             digitalWrite(posicion,   HIGH);
109             digitalWrite(luzstop,     LOW);
110             digitalWrite(luzreversa, HIGH);
111           }
112           else if (stop == HIGH & reversa == HIGH)//Reversa y Stop ON
113           {
114             digitalWrite(posicion,  HIGH);
115             digitalWrite(luzstop,    LOW);
116             digitalWrite(luzreversa, LOW);
117           }
118           else
119           {
120             digitalWrite(posicion,  HIGH);
121             digitalWrite(luzstop,   HIGH);
122             digitalWrite(luzreversa,HIGH);
123           }
124       }
125       else
```

```
126         {
127                 digitalWrite(posicion,  HIGH);
128                 digitalWrite(luzstop,   HIGH);
129                 digitalWrite(luzreversa,HIGH);
130         }
131       }}}
```

Anexo 12. Código de programación en Arduino IDE de ECU LCD

```
 1 #include <SPI.h>        // definir librerías
 2 #include <mcp2515.h>
 3 #include <Wire.h>
 4 #include <LiquidCrystal_I2C.h>  //Librería para patalla LCD
 5 LiquidCrystal_I2C lcd(0x3f, 16, 2);  //define el tamaño de la LCD
 6
 7 int posi       = 0;  //estado de los bits
 8 int nieble     = 0;  //en el campo de datos
 9 int dd         = 0;
10 int di         = 0;
11 int lbaja      = 0;
12 int lalta      = 0;
13 int stop       = 0;
14 int reversa    = 0;
15
16 int RDleft     = 0;    // Tramas de
17 int RDright    = 0;  // actuadores
18 int RDdown     = 0;  // de puerta RIGHT
19 int RDup       = 0;
20 int subirventa = 0;
21 int bajarventa = 0;
22
23 int RIleft     = 0;    // Tramas de
24 int RIright    = 0;  // actuadores
25 int RIdown     = 0;  // de puerta RIGHT
26 int RIup       = 0;
27 int VIsube     = 0;
28 int VIbaja     = 0;
29
30 struct can_frame canMsg1;  // Tramas de espejo Retrovisor
31 struct can_frame canMsg3;
32 struct can_frame canMsg4;
33 MCP2515 mcp2515(10);
34
35 void setup() {
36
37   lcd.init();          //inicializar la pantalla lcd
38   lcd.backlight();
39   lcd.setCursor(0,0);
```

```
  lcd.print("W1:_ M1:_ ______");
  lcd.setCursor(0,1);
  lcd.print("W2:_ M2:_   _ _ ");
  Serial.begin(9600);
  SPI.begin();
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);  // velocidad del Bus CAN
  mcp2515.setNormalMode();
}

void loop() {

  if (mcp2515.readMessage(&canMsg1) == MCP2515::ERROR_OK)
  {   //trama de sistema de luces
      if (canMsg1.can_id == 0xAA)     //identificador sistema de luces
      {
        int posi         = canMsg1.data[0];  //estado de los bits
        int nieble       = canMsg1.data[1];  //en el campo de datos
        int dd           = canMsg1.data[2];
        int di           = canMsg1.data[3];
        int lbaja        = canMsg1.data[4];
        int lalta        = canMsg1.data[5];
        int stop         = canMsg1.data[6];
        int reversa      = canMsg1.data[7];

        if (di == HIGH & dd == LOW ) //Activar direccional izquierdo
        {
          lcd.setCursor(12,1);
          lcd.print("< _");   //testigo en pantalla

        }
        else if (dd == HIGH & di == LOW) //Activar direccional derecho
        {
          lcd.setCursor(12,1);
          lcd.print("_ >");   //testigo en pantalla

        }
        else if (dd == HIGH & di == HIGH )  //Activar intermitentes
        {
          lcd.setCursor(12,1);
          lcd.print("< >");   //testigo en pantalla
        }
        else
        {
          lcd.setCursor(12,1);
          lcd.print("_ _");   //testigo en pantalla
        }
        if (posi == LOW & nieble == LOW & lbaja == LOW & lalta == HIGH)
        {   //lalta ON
          lcd.setCursor(13,0);
          lcd.print("A");   //testigo en pantalla
        }
```

```
92          else if (nieble == HIGH & posi == LOW  )
93          {     // nieblero, posición, lbaja y/o lalta ON
94              if (lalta == HIGH)
95              {
96                lcd.setCursor(10,0);
97                lcd.print("NP_A");   //testigo en pantalla
98              }
99              else
100             {
101               lcd.setCursor(10,0);
102               lcd.print("NP__");   //testigo en pantalla
103             }
104         }
105         else if (nieble == HIGH & posi == HIGH & lbaja == LOW )
106         {     // nieblero, posición, lbaja y/o lalta ON
107             if (lalta == HIGH)
108             {
109               lcd.setCursor(10,0);
110               lcd.print("NP_A");   //testigo en pantalla
111             }
112             else
113             {
114               lcd.setCursor(10,0);
115               lcd.print("NP__");   //testigo en pantalla
116             }
117         }
118         else if (nieble == HIGH & posi == HIGH & lbaja == HIGH)
119         {     // nieblero, posición, lbaja y/o lalta ON
120             if (lalta == HIGH)
121             {
122               lcd.setCursor(10,0);
123               lcd.print("NP_A");   //testigo en pantalla
124             }
125             else
126             {
127               lcd.setCursor(10,0);
128               lcd.print("NPB_");   //testigo en pantalla
129             }
130         }
131         else if (nieble == LOW & posi == HIGH & lbaja == LOW)
132         {     //Posi Y/O lalta ON
133             if (lalta == HIGH)
134             {
135               lcd.setCursor(10,0);
136               lcd.print("_P_A");   //testigo en pantalla
137             }
138             else
139             {
140               lcd.setCursor(10,0);
141               lcd.print("_P__");   //testigo en pantalla
142             }
143         }
```

```
144         else if (nieble == LOW & posi == HIGH & lbaja == HIGH)
145         {     //Posi, lbaja y/o lalta ON
146             if (lalta == HIGH)
147             {
148               lcd.setCursor(10,0);
149               lcd.print("_P_A");   //testigo en pantalla
150             }
151             else
152             {
153               lcd.setCursor(10,0);
154               lcd.print("_PB_");   //testigo en pantalla
155             }
156         }
157         else
158         {
159             lcd.setCursor(10,0);
160             lcd.print("____");   //testigo en pantalla
161         }
162         if (stop == LOW & reversa == HIGH)   //Reversa ON
163         {
164           lcd.setCursor(14,0);
165           lcd.print("_R");   //testigo en pantalla
166         }
167         else if (stop == HIGH & reversa == LOW)   //STOP ON
168         {
169           lcd.setCursor(14,0);
170           lcd.print("S_");   //testigo en pantalla
171         }
172         else if (stop == HIGH & reversa == HIGH)//Reversa y Stop ON
173         {
174           lcd.setCursor(14,0);
175           lcd.print("SR");   //testigo en pantalla
176         }
177         else
178         {
179           lcd.setCursor(14,0);
180           lcd.print("__");   //testigo en pantalla
181         }
182     }
183   }
184
185   else if (mcp2515.readMessage(&canMsg4) == MCP2515::ERROR_OK)
186   {     //trama window y mirror Right
187       if (canMsg4.can_id == 0xDD)
188       {
189         RDleft     = canMsg4.data[0];    // Tramas de
190         RDright    = canMsg4.data[1];  // actuadores
191         RDdown     = canMsg4.data[2];  // de puerta RIGHT
192         RDup       = canMsg4.data[3];
193         subirventa = canMsg4.data[4];
194         bajarventa = canMsg4.data[5];
195
```

```
196         if (RDup == HIGH)
197         {
198           lcd.setCursor(8,1);
199           lcd.print("N");   //testigo en pantalla
200         }
201         else if (RDdown == HIGH)
202         {
203           lcd.setCursor(8,1);
204           lcd.print("S");   //testigo en pantalla
205         }
206         else if (RDleft == HIGH)
207         {
208           lcd.setCursor(8,1);
209           lcd.print("W");   //testigo en pantalla
210         }
211         else if (RDright == HIGH)
212         {
213           lcd.setCursor(8,1);
214           lcd.print("E");   //testigo en pantalla
215         }
216         else
217         {
218           lcd.setCursor(8,1);
219           lcd.print("_");   //testigo en pantalla
220         }
221         //delay(20) ;
222         if (subirventa == HIGH & bajarventa == LOW)
223         {
224           lcd.setCursor(3,1);
225           lcd.print("U");   //testigo en pantalla
226         }
227         else if (bajarventa == HIGH & subirventa == LOW)
228         {
229           lcd.setCursor(3,1);
230           lcd.print("D");   //testigo en pantalla
231         }
232         else
233         {
234           lcd.setCursor(3,1);
235           lcd.print("_");   //testigo en pantalla
236         }
237       }
238   }
239
240   else if (mcp2515.readMessage(&canMsg3) == MCP2515::ERROR_OK)
241   {       //trama window y mirror LEFT
242       if (canMsg3.can_id == 0xCC)
243       {
244         RIleft      = canMsg3.data[0];    // Trama de
245         RIright     = canMsg3.data[1];  // actuadores
246         RIdown      = canMsg3.data[2];  // de puerta LEFT
247         RIup        = canMsg3.data[3];
```

```
248         VIsube      = canMsg3.data[4];
249         VIbaja      = canMsg3.data[5];
250 
251         if (RIup == HIGH)
252         {
253           lcd.setCursor(8,0);
254           lcd.print("N");   //testigo en pantalla
255         }
256         else if (RIdown == HIGH)
257         {
258           lcd.setCursor(8,0);
259           lcd.print("S");   //testigo en pantalla
260         }
261         else if (RIleft == HIGH)
262         {
263           lcd.setCursor(8,0);
264           lcd.print("W");   //testigo en pantalla
265         }
266         else if (RIright == HIGH)
267         {
268           lcd.setCursor(8,0);
269           lcd.print("E");   //testigo en pantalla
270         }
271         else
272         {
273           lcd.setCursor(8,0);
274           lcd.print("_");   //testigo en pantalla
275         }
276         //delay(20) ;
277         if (VIsube == HIGH & VIbaja == LOW)
278         {
279           lcd.setCursor(3,0);
280           lcd.print("U");   //testigo en pantalla
281         }
282         else if (VIbaja == HIGH & VIsube == LOW)
283         {
284           lcd.setCursor(3,0);
285           lcd.print("D");   //testigo en pantalla
286         }
287         else
288         {
289           lcd.setCursor(3,0);
290           lcd.print("_");   //testigo en pantalla
291         }
292       }
293   }
294 
```