UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INTEGRACIÓN DE UN SISTEMA PARA LA OBTENCIÓN DE DATOS DE VEHÍCULOS AUTOMOTORES BASADO EN LOS PROTOCOLOS CAN BUS Y OBD-II

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO ELECTRÓNICO

(Á R E A : E L E C T R Ó N I C A)

P R E S E N T A :

MIGUEL IBAÑEZ GALINDO

DIRECTOR DE TESIS: M. I. LAURO SANTIAGO CRUZ

MÉXICO, D.F. 2015

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

CONTENIDO

Índice de Figuras

Capítulo 2

Capítulo 3

Capítulo 4

Índice de Tablas

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 1
Introducción

Debido a que un vehículo a motor de combustión interna evidentemente contamina, y es la causa actualmente de la quinta parte de la contaminación mundial, el gobierno federal de los Estados Unidos de Norteamérica comenzó a aprobar legislaciones que pretendían mejorar la calidad del aire. Como resultado de los esfuerzos para mejorar la calidad del aire, en 1970 se elaboró un documento para el decreto sobre Aire Limpio, con esto se formó la Agencia de protección Ambiental (EPA - Enviroment Protection Agency) adquiriendo ésta una amplia autoridad para regular la contaminación vehicular.

Después de que el congreso aprobó el Decreto sobre Aire limpio en 1970, el estado de California creo la Comisión de Recursos del Aire (California Air Resources Board – CARB). Ésta tenía como objetivo principal regular de manera más rigurosa los niveles de emisión de gases en los vehículos que se vendían en ese estado. Otros estados también tomaron las medidas dictadas por la CARB.

Durante los años 70 y principios de los 80 algunos fabricantes empezaron a usar componentes electrónicos de control y diagnóstico de errores en sus automóviles, así se originó el OBD (On Board Diagnostic) usado por primera vez en vehículos en 1988, y fue obligatorio su uso en Estados Unidos en 1991. Al principio fue sólo para conocer y controlar las emisiones del vehículo y adaptarlas a los estándares exigidos, pero con el paso del tiempo estos sistemas fueron volviéndose cada vez más sofisticados, hasta los años 90, que es cuando surgió el estándar OBD II (On Board Diagnostic segunda generación). Este estándar está integrado en todos los vehículos de pasajeros y los camiones de gasolina y combustibles alternativos, los vehículos cuyo modelo data de 1996 deben contar obligatoriamente con sistemas OBD II, al igual que todos los vehículos de pasajeros y camiones de diésel a partir de 1997. Además, un pequeño número de vehículos de gas fueron equipados con sistemas OBD II. Los vehículos pesados utilizan una norma diferente llamada HDOBD, aunque también usan el protocolo CAN para la comunicación entre sus dispositivos. Cabe comentar que la EPA define como vehículo pesado a aquellos que sobrepasan las 1400 libras de peso (aproximadamente 6 350 kg).

El OBD II es un sistema que permite diagnosticar los errores que se producen en el vehículo, sin necesidad de desmontar partes para descubrir la procedencia de dicho error. Este sistema de codificación única se encuentra actualmente implantado en todos los vehículos particulares y vehículos industriales ligeros, ésta última denominación se emplea corrientemente para indicar todos los vehículos automóviles destinados al transporte de mercancías por carretera o al transporte colectivo de personas, y cuyo peso está entre 2.5 y 3.2 toneladas; sin embargo,

dicho término no indica técnicamente ningún tipo determinado de vehículo. A diferencia de otros sistemas desarrollados antes de 1996, OBD-II se caracteriza por ser un sistema estandarizado, que permite, de manera fácil, ver que errores se han producido en un vehículo cualquiera, utilizando una única codificación así como un conector estandarizado.

La gran inconveniencia de la primera versión de OBD fue que cada fabricante cumplía con las especificaciones de la CARB de forma diferente. Cada fabricante equipó sus vehículos con OBD de forma muy variada, entonces también cada uno poseía sus propios códigos de errores y sus propias herramientas para interpretar esos códigos. Luego para modelos a partir de 1994, la CARB y EPA aumentaron los requerimientos del sistema OBD, creando una extensa lista de procedimientos y estándares, lo que se convirtió en la segunda generación de diagnósticos a bordo, lo que hoy es el OBD-II. Esto justifica su uso, porque el OBD-II está estandarizado, desde sus códigos, herramientas de lectura y conector.

Los vehículos al ir evolucionando a sistemas más complejos para adaptarse a los requerimientos de los usuarios, que cada vez exigen más calidad, seguridad y confort en sus vehículos, así como cumplir con las exigencias cada vez mayores para el control de emisiones, han aumentado el número de componentes y por lo tanto la longitud y cantidad del cableado. Para disminuir este problema manteniendo la cantidad de dichos componentes o incluso aumentarlo, se empezaron a introducir buses de comunicación. Uno de esos buses, el más difundido y usado, es el bus de comunicación CAN (Controller Area Network) desarrollado por Bosh.

La utilización de sistemas electrónicos en la gestión del motor, con sistemas como el OBD, aumentaron su vida útil y su eficiencia; sobre todo se mejoró en la disminución de emisiones contaminantes hacia la atmósfera.

Con la intensión de aprovechar la información que se puede obtener a través del sistema OBDII-CAN, el objetivo del presente trabajo de tesis es solicitar y obtener información relacionada con el control de emisiones de contaminantes de los vehículos, almacenarla y presentarla, de manera que sirva a los trabajos que se están desarrollando en el Laboratorio de Control de Emisiones (LCE) de la Facultad de Ingeniería de la UNAM.

Con base en lo expuesto anteriormente, en cuanto a la instrumentación electrónica del automóvil, a la estandarización del sistema OBD-II y de sus protocolos de comunicación, y a la obligatoriedad de su uso, se desarrollará un sistema para la obtención y monitoreo de datos de vehículos automotores basado en el bus CAN.

Cabe comentar que en el LCE se han desarrollado ciclos de manejo para vehículos y motocicletas, el laboratorio utiliza los ciclos de manejo como una herramienta útil para evaluar el consumo de combustible y las emisiones contaminantes del autotransporte, estos ciclos últimamente se han desarrollado para el laboratorio utilizando diferentes herramientas. El sistema que se integrará

podría servir de apoyo en el desarrollo de ciclos de manejo. Pero vayamos un poco más allá, el sistema a integrar monitorea en tiempo real datos de sensores y actuadores que intervienen en el control del motor y por ello intervienen en el control de emisiones, al evaluar estos datos se obtienen interpretaciones directas acerca de las emisiones contaminantes de los vehículos, pero esto es sólo es un ejemplo, porque con los datos monitoreados gente experta en el campo de la mecánica podría hacer interpretaciones de diversa índole.

La presente tesis está organizada en 5 capítulos, la bibliografía y los apéndices. El contenido general de cada uno de los apartados mencionados es:

1.- Introducción: En este capítulo se da la idea del motivo del trabajo y se presenta un panorama general de la organización del escrito.

2.- Generalidades: En este capítulo se describen los conceptos relevantes de cada concepto para ayudar al lector a entender el desarrollo del trabajo. Este capítulo proporciona información general del sistema OBD-II y del bus-CAN; incluye información acerca de alguno de los circuitos integrados utilizados responsables del funcionamiento del sistema y del monitoreo.

3.- Desarrollo: Aquí se explica desde qué dispositivos y tarjetas se utilizaron hasta la integración del sistema en una sola tarjeta, pasando por la modificación del software y en cierta medida del hardware.

4.- Pruebas al sistema: Las pruebas que se realizaron con el sistema, a cada uno de sus componentes por separado, así como al conjunto de tarjetas hasta la integración en una sola, se reportan en este capítulo. Se reportan qué fallas se presentaron en las pruebas en cada etapa del sistema, cuáles no se resolvieron y cuáles y como tuvieron solución.

5.- Resultados y conclusiones: En cuanto a resultados se mencionarán: de los obtenidos satisfactoriamente, si éstos eran los que se esperaban, si surgieron otros a lo largo del desarrollo; de los que no fueron satisfactorios, se mencionará la posible causa y se sugerirá a quien continúe con el trabajo que ponga atención a esas fallas.

Finalmente se presentará la bibliografía consultada y los apéndices generados para una mejor compresión del presente trabajo.

Capítulo 2
Generalidades

En este capítulo se aborda de manera general cada uno de los conceptos, sistemas, dispositivos y componentes relacionados con el proyecto, como son: protocolos, circuitos integrados, entre otros.

2.1. El sistema OBD-II

OBD (On Board Diagnostics) es un sistema de diagnóstico a bordo para automóviles y camiones; se emplean los estándares OBD-II (Estados Unidos), EOBD (Europa) y JOBD (Japón), los cuales permiten el monitoreo y control completo del motor y otros dispositivos del vehículo, detectan, almacenan, y evalúan fallos, guardan códigos de error (DTC - Diagnostic Trouble Code), por mencionar las más importantes. En principio las fallas y sensores que se monitorean están relacionados con las emisiones contaminantes. El uso más importante que se le da a los DTC es diagnosticar un problema, sin necesidad de desmontar partes del automóvil; proceso que resultaría laborioso en la actualidad por la gran cantidad de sistemas mecánicos y electrónicos incluidos en los automóviles. Estos códigos se obtienen desde el sistema a través del Conector de datos (DLC - Data Link Connector).

Al sistema OBD-II llegan las señales de todos los sensores y salen señales hacía los actuadores repartidos por todo el vehículo, todo esto puede ser controlado por 1 o varias ECU (Electrónic Control Unit) que se comunican entre sí y con los dispositivos electrónicos a través de cableado. Cuando ocurre una falla el sistema OBD-II debe informar al conductor por medio de un testigo, en este caso, la Lámpara de mal-funcionamiento (MIL - Malfunction Indicator Lamp).

OBD-II es una mejora del OBD, dicho sistema surgió por mandato del gobierno de California para controlar las emisiones contaminantes de los automóviles, sistema del que hacemos uso en el presente trabajo.

Después de creada la CARB (California Air Resources Board) se comenzó a legislar el OBD-I, sistema que se empezó a utilizar en automóviles vendidos en California en 1988. Esta primera versión del sistema monitorea: el sistema de medición del combustible, el sistema EGR (Exhaust Gas Recirculation) y otras mediciones relacionadas con componentes eléctricos.

Con las revisiones al decreto sobre aire limpio de 1990, la CARB desarrolló nuevas regulaciones para establecer la segunda generación de OBD (OBD-II). Entonces, a partir de 1996, todos los camiones, camionetas y automóviles vendidos en Estados Unidos debían tener implementados el sistema OBD-II. Esta

segunda generación de diagnóstico está diseñada para detectar fallas eléctricas, químicas y mecánicas, y se especializa en el monitoreo del motor, en el sistema ABS (Anti-lock Break Sistem), entre otros sistemas; en su funcionamiento general se dedica a vigilar continuamente los componentes que intervienen en las emisiones contaminantes. Cabe comentar que el funcionamiento de este sistema difiere en cierta medida en vehículos con motor a gasolina con respecto a los que usan Diesel.

Para saber si un vehículo determinado cuenta con el sistema OBD-II, podemos encontrar en el vehículo una etiqueta que lo indique, como la siguiente:

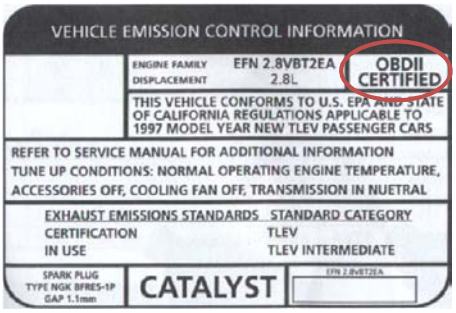


Figura 2.1. Certificado OBD-II.

Las funciones principales del sistema OBD-II en vehículos a gasolina o Diesel son:

En vehículos con motor a gasolina:

- Vigila el rendimiento del catalizador.
- Diagnóstico del envejecimiento de las sondas lambda (sensores de oxígeno).
- Prueba de voltaje en las sondas lambda.
- Sistema de aire secundario (si el vehículo lo incorpora).
- Sistema de recuperación de vapores de combustible del sistema de control de evaporaciones de gases (EVAP o cánister).
- Prueba de diagnóstico de fugas.
- Sistema de alimentación de combustible.
- Fallos de combustión.
- Funcionamiento del sistema de comunicación entre ECUs (Engine Control Unit) o PCM (Powertrain Control Module).
- Control del sistema de gestión electrónica.
- Sensores y actuadores del sistema electrónico que intervienen en la gestión del motor o que están relacionados con las emisiones contaminantes.

En los motores Diesel:

- Fallos de la combustión.
- Regulación del comienzo de la inyección.
- Regulación de la presión de sobrealimentación.
- Recirculación de gases de escape.

- Funcionamiento del sistema de comunicación entre ECUs.
- Control del sistema de gestión electrónica.

El sistema OBD-II, además de las mejoras añadidas, estandarizó los códigos de error para las diferentes funciones del vehículo y para todos los fabricantes; también permite acceder a esos códigos con equipos de diagnóstico universales, por lo tanto el conector de acceso al OBD-II está también normalizado.

El conector para el sistema OBD-II, figura 2.2 debe ser accesible y estar en la zona del conductor (incluye el lado del copiloto). Tiene asignado en sus terminales, de manera estandarizada, los protocolos por los cuales se accede al sistema OBD-II.

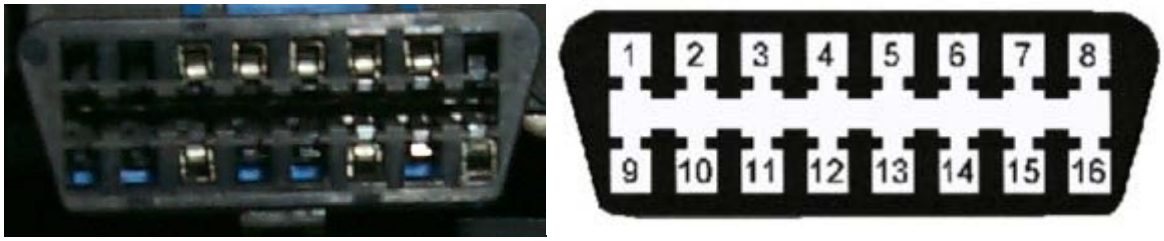


Figura 2.2. Conector OBD-II y sus terminales.

Es importante mencionar que antes del sistema OBD-II, cada fabricante establecía sus propios códigos, con el inconveniente de que se necesitaba un equipo para cada modelo de automóvil y un compendio diferente de códigos con su significado.

La tabla 2.1 enlista los protocolos presentes en cada terminal del conector.

Terminal	Uso	Nivel [V]	Terminal	Uso	Nivel [V]
1	Reservado	----	9	Reservado	----
2	J1850 PWM y VPW (+)	5 y 7	10	J1850 PWM (-)	0
3	Reservado	----	11	Reservado	----
4	Negativo de la batería	0	12	Reservado	----
5	Tierra de la señal	0	13	Reservado	----
6	CAN (high)	3.5	14	CAN (low)	1.5
7	ISO 9141-2 e ISO 14230-4 (Línea K)	12	15	ISO 9141-2 e ISO 14230-4 (Línea L)	0
8	Reservado	----	16	Positivo de la Batería	

Tabla 2.1. Terminales del conector OBD-II.

En el sistema OBD-II cuenta con cinco protocolos de comunicación, y un automóvil normalmente usa sólo uno de ellos, estos son: **J1850 PWM** (Pulse Width Modulation), usado por Ford y Mazda; **J1850 VPW** (Variable Width Modulation), usado por General Motors y camiones ligeros; **ISO9141-2,** protocolo más viejo en

vehículos Chrysler, europeos y asiáticos, entre 2000-2004; **ISO14230-4 KWP2000** (Keyword Protocol 2000), comúnmente usado en vehículos del 2003; **ISO15765-4 CAN-BUS,** primero introducido en 2004 y obligatorio en Estados Unidos en vehículos desde 2008.

2.1.1. Modos de funcionamiento del sistema OBD-II

El sistema OBD-II tiene varios modos de funcionamiento, también estandarizados, lo que significa que el modo con el que se acceda al sistema no dependerá del fabricante del vehículo. El total de modos de funcionamiento son nueve, aunque los fabricantes no están obligados a incluir todos, son:

Modo 1. Muestra datos en tiempo real.- Identificación de Parámetro (PID - Parameter Identification), es el acceso a datos en tiempo real de valores analógicos o digitales de salidas y entradas a la ECU. Este modo es también llamado flujo de datos. Aquí es posible ver, por ejemplo, la temperatura de motor o el voltaje generado por una sonda lambda. *El proyecto del que trata el presente trabajo se centra en este modo de funcionamiento.*

Modo 2. Muestra los datos guardados en memoria (Freeze Frame).- Los PID son los mismos que en el modo 1, con el mismo significado, pero el acceso es al cuadro de datos congelados, con los valores de los sensores al momento de ocurrir una falla. Esta es una función muy útil del OBD-II, porque al recuperar estos datos, se pueden conocer las condiciones exactas en las que ocurrió dicho fallo. El cuadro de datos corresponde al primer fallo detectado.

Modo 3. Petición de códigos de error o fallo.- Este modo permite extraer de la memoria de la ECU todos los códigos de fallo (DTC - Data Trouble Dode) almacenados. En talleres de mantenimiento y reparación de vehículos los escáneres trabajan principalmente con este modo, para detectar la posible zona de la falla.

Modo 4. Borrado de DTCs y valores almacenados.- Con este modo se pueden borrar todos los códigos almacenados en la PCM, incluyendo los DTCs y el cuadro de datos grabados.

Modo 5. Solicitud de los diagnósticos de los sensores de oxígeno.- El propósito de este modo es permitir el acceso a los resultados de las mediciones de voltaje hechas a los sensores de oxígeno del módulo de control del motor, y así determinar el funcionamiento de los mismos y la eficiencia del convertidor catalítico. La información de este modo no está disponible en vehículos que utilizan el sistema CAN. Para esos vehículos se requiere utilizar el modo 6.

Modo 6. Diagnóstico en el sistema OBD-II.- El propósito de este modo es permitir el acceso a los resultados de las pruebas hechas a componentes que no están monitoreados continuamente, pero sólo en periodos de tiempo y condiciones de operación determinadas, estos resultados reportan rangos de valores máximos y mínimos. Este modo también sirve para verificar que todos los sensores y otros componentes de control de emisiones contaminantes estén funcionando adecuadamente. También muestra las pruebas hechas a los sensores de oxígeno de la misma manera que en el modo 5 pero sólo para CAN.

Modo 7. Diagnóstico de DTCs pendientes.- Este modo permite leer de la memoria de la ECU todos los DTCs pendientes.

Modo 8. Prueba a bordo.- Este modo nos permite realizar un control de operación a bordo de componentes y/o sistemas del vehículo, así, este modo permite realizar la prueba de actuadores, con lo que el mecánico puede activar y desactivar actuadores como bombas de combustible, válvula de ralentí, etc.

Modo 9. Información del vehículo.- Solicita información del vehículo como el VIN (Vehicle Identification Number). Este número es una secuencia de dígitos que identifica los vehículos de motor de cualquier tipo, y los remolques a partir de un cierto peso, es un código específico y único para cada unidad fabricada.

Para comunicarnos con el sistema OB-II se debe utilizar alguno de sus nueve modos de funcionamiento. La información de petición de datos se envía dentro de una trama del protocolo que se use, esta trama incluye, el modo de funcionamiento con el cual se trabajará, así como cuál es la información que se requiere. Con la finalidad de especificar el dato que se pide se hace uso de los PID's, algún dispositivo del sistema OBD-II conectado en el bus reconoce ese PID como uno al que tiene que responder, y reporta el valor para ese PID por medio del mismo bus; donde el número de bytes enviados y leídos tanto como su codificación dependen del dato. La mayoría de los datos están codificados con una fórmula, otros están codificados de forma especial.

La comunicación con el sistema OBD-II se lleva a cabo a través de alguno de los protocolos, en este caso, nuestro sistema utiliza el protocolo CAN (Controller Area Network).

2.2. El protocolo CAN

Desde principios de los 40's, los fabricantes continuamente han mejorado la tecnología de sus automóviles, integrando e incrementando la cantidad de sistemas electrónicos en ellos. Dichos sistemas tienen que comunicarse entre sí y pueden ser módulos de control, sensores, actuadores, etc. En forma general llamaremos "nodo" a cualquier sistema conectado al bus CAN. La comunicación se establece de manera paralela o en serie; en un principio la comunicación se hacía punto a punto, lo que implicaba que al menos hubieran dos cables hacia cada lugar donde se requería comunicar. Conforme la tecnología ha ido progresando, los vehículos se han vuelto más complejos; con el reemplazo de sistemas mecánicos por componentes electrónicos y al proporcionarle comodidades adicionales al usuario, conveniencia y características de seguridad se ha ido incrementando el cableado.

Tener una cantidad muy grande de cableado tiene muchos problemas, como el incremento del peso del vehículo, menor maniobrabilidad, mayor interferencia, necesidad de más espacio, más propenso a averías, etc. En la actualidad las señales que se transmiten y procesan son digitales, entonces una solución es utilizar comunicación serial, con un único bus de comunicación, que sea resistente al ruido.

El bus CAN es un bus automotriz desarrollado en 1980 por Robert Bosch. Es un estándar que rápidamente ha ganado una amplia aceptación dentro de la industria automotriz y aeroespacial. CAN es un protocolo de bus serial para conectar sistemas y sensores individuales, como alternativa al cableado múltiple, esto permite a los componentes automotrices comunicarse sobre un bus de datos en red de uno o dos cables de hasta 1 Mbps, gracias a este bus la cantidad de cableado necesario para interconectar todos los sistemas se redujo enormemente. En la figura 2.3 vemos con bloques un ejemplo de conexión sin bus CAN.

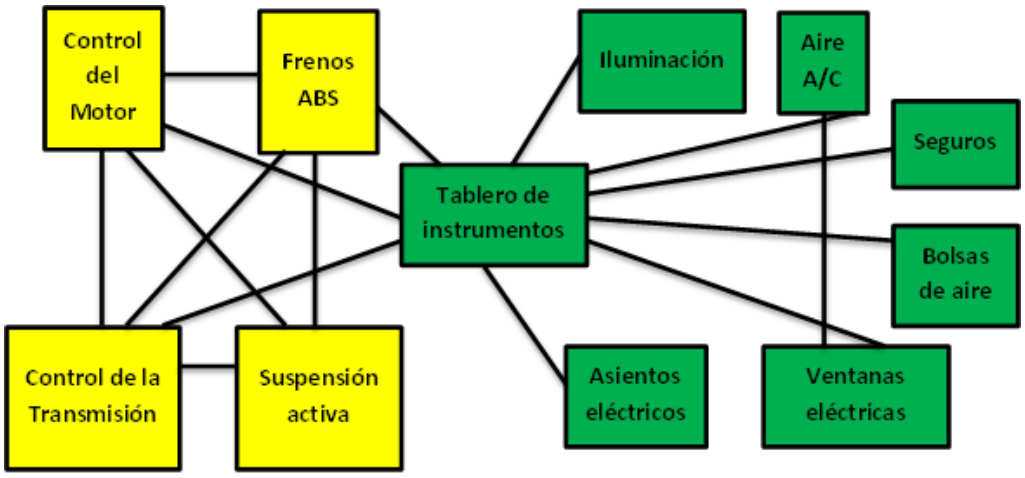


Figura 2.3. Conexión sin bus CAN.

En la figura 2.4 presentamos un ejemplo de conexión con CAN.

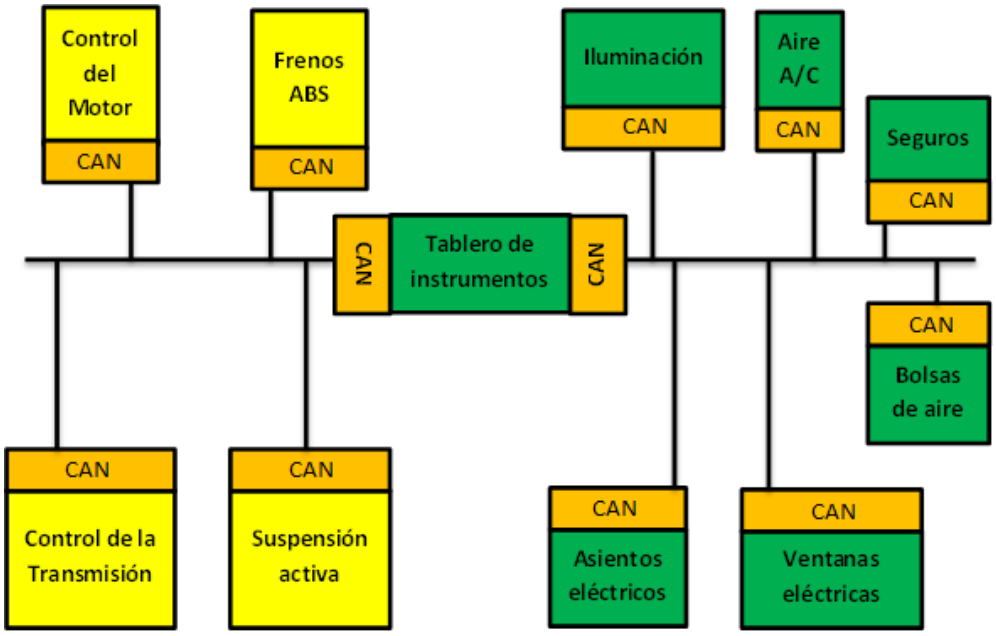


Figura 2.4. Conexión con bus CAN.

Al comparar estas dos imágenes nos damos idea de la gran importancia del uso del protocolo CAN, requerido de forma obligatoria en vehículos a partir del 2008 en Estados Unidos.

Las principales razones por las que se utiliza el protocolo CAN son:

- Es un estándar maduro
  - CAN es un protocolo usado por más de 20 años
  - Existen numerosos productos y herramientas CAN en el mercado
- Implementación de hardware del protocolo
  - Combinación del manejo de error y aislamiento de fallas con una alta velocidad de transmisión
- Medio de transmisión simple
  - El estándar es un cable de par trenzado, pero incluso sólo con un alambre puede funcionar
  - Puede trabajar con otros medios: enlaces ópticos o de radio
- Excelente manejo de error
  - Mecanismo de detección de errores con CRC
- Aislamiento de fallas
  - Característica incluida para prevenir que un nodo que falle bloquee al sistema
- Protocolo más usado en el mundo industrial y automotriz
- Buena relación precio-desempeño

Algunas de las desventajas que puede presentar el bus CAN son un mayor coste y la necesidad de conocimientos especializados a la hora del mantenimiento y reparación del vehículo.

Es importante señalar que el protocolo CAN Bus define su manera de trabajar sobre las primeras capas del modelo OSI (Open System Interconection); siendo éstas la capa de enlace de datos y parte de la capa física. Estas capas están estandarizadas conforme el estándar ISO-11898, para velocidades de hasta 1Mbps, denominado CAN de alta velocidad (*High-Speed CAN)* y el estándar ISO-11519, para velocidades de hasta 125 kbps, denominado CAN de baja velocidad (Low-Speed CAN) también llamado “tolerante a fallas” o (*Fault Tolerant Low-Speed CAN)*. En nuestro caso utilizamos CAN de alta velocidad, porque esta velocidad se usa en aplicaciones críticas, como en la gestión del motor, sistemas de seguridad y monitoreo.

Las características principales de la operación de CAN en las capas mencionadas se describen a continuación:

2.2.1. Capa de enlace

Controla el enlace lógico de los nodos del bus, esta capa se encarga de los filtros con los cuales los nodos deciden qué mensajes recibidos se aceptan. Proporciona

servicios durante la transferencia y petición de datos entre los nodos y proporciona medios para el restablecimiento y para notificar la sobrecarga del bus. Esta capa también controla el acceso al medio y representa el núcleo del protocolo CAN; presenta los mensajes recibidos al control de enlace lógico, también es responsable de la trama de mensajes, arbitraje, reconocimiento, señalización y detección de error. El control de acceso al medio decide si el bus está libre para comenzar una nueva transmisión o si la recepción acaba de comenzar.

2.2.2 Capa física

La capa física en CAN es responsable de la transferencia de bits entre los distintos nodos que componen la red. Define aspectos como niveles de señal, codificación, sincronización y tiempos en que los bits se transfieren al bus. En la especificación original de CAN la capa física no fue definida, permitiendo con ello diferentes opciones para la elección del medio y niveles eléctricos de transmisión. Las características de las señales eléctricas en el bus fueron establecidas más tarde por el estándar ISO 11898. La especificación CiA (CAN in AUTOMATION), complementó las definiciones respecto al medio físico y conectores. El resto de la capa física (y para las demás capas del modelo OSI) no están especificadas para el protocolo CAN y pueden ser definidas por el diseñador del sistema.

Hemos mencionado que CAN es un protocolo de comunicación serial, está basado en una topología de bus para la transmisión de mensajes en ambientes distribuidos, es para aplicaciones de control en tiempo real, con una velocidad de comunicación de hasta 1 Mbit por segundo, y tiene excelente capacidad de detección y aislamiento de errores; además, ofrece una solución a la gestión de la comunicación entre múltiples CPUs (unidades centrales de proceso) como las ECU y PCM.

El bus de comunicación de CAN consta de dos cables de par trenzado (aunque pueden ser otros medios), en los que viaja una señal diferencial, estas dos características hacen que la comunicación sea en gran medida inmune al ruido, tomando en cuenta que está sometido al ruido proveniente de la temperatura del motor, de la chispa de ignición y de todos los demás dispositivos con los que cuenta el automóvil. Todos los dispositivos que se comunican están conectados en estos dos cables, llamados respectivamente “CAN-High” (línea “H”) y “CAN-Low” (línea “L”), y todos los dispositivos conectados deben comunicarse a la misma velocidad. Estos cables o bus tiene una impedancia característica de 120 Ohms, por lo que debe tener una resistencia de terminación del mismo valor en cada extremo para evitar que se refleje la señal.

El protocolo CAN es multi-maestro, esto permite que todos los dispositivos conectados tengan la misma oportunidad de acceder al bus para enviar mensajes. La comunicación se lleva a cabo por un “mecanismo de difusión”, el cual se basa en un protocolo de transmisión orientado a mensajes, esto quiere decir que define el contenido del mensaje en lugar de definir al nodo o a la dirección del nodo. Este mecanismo de comunicación permite que el acceso al bus se realice por prioridad

del mensaje utilizando CSMA/CR (Carrier Sense Multiple Access/Collision Resolution). CSMA significa que los nodos tienen la misma oportunidad de enviar mensajes (MA) y también que todos están escuchando los mensajes del bus (CS). En un determinado momento más de un nodo tratará de acceder al bus, y los mensajes colisionan, esta colisión se soluciona por arbitraje basado en prioridades (CR). Este método se basa en una topología eléctrica que aplica una función lógica determinista a cada bit; la prioridad se le da a un nivel de bit de tipo dominante. Se define como bit *dominante* al nivel lógico '0', este bit fuerza su valor en el bus sin importar el valor de los demás bits; el valor lógico '1' es un bit *recesivo*, este valor permanece si todos los bits tienen este valor.

Para la solución a una colisión de mensajes se aplica una función AND a todos los bits transmitidos simultáneamente. Cada transmisor escucha continuamente el valor presente en el bus; el transmisor se retira cuando el valor en el bus es dominante y éste ha enviado un bit recesivo. Mientras el valor del bus y el enviado coincidan, el transmisor permanecerá; finalmente el mensaje con identificador de máxima prioridad sobrevive, y los demás nodos reintentarán la transmisión lo antes posible. En concreto, la resolución de colisión (CR) hace que la comunicación sea no-destructiva y se garantiza que el mensaje que tenga la mayor prioridad no se detenga y se transmite sin perder tiempo. La prioridad del mensaje se establece dentro de la misma trama que contiene al mensaje, en una sección dentro del campo "arbitraje" llamada identificador. Cada nodo, con base en el identificador, debe decidir si el mensaje debe atenderse o ignorarse.

2.2.3. Codificación

El protocolo CAN utiliza una codificación llamada NRZ (Non Return to Zero), en la cual cada bit está representado por un solo nivel, esto quiere decir que el nivel de la señal permanece constante a lo largo del tiempo que dura un bit; un nivel lógico '0' se representa con un nivel de voltaje bajo, y un nivel lógico '1' se representa con un nivel de voltaje alto. Usar la codificación NRZ asegura mensajes compactos con un número mínimo de transiciones y una resistencia alta a perturbaciones externas.[1] Un detalle de la codificación NRZ es que el nivel de la señal puede permanecer constante por un periodo de tiempo más largo; por lo tanto se deben tomar medidas para asegurar que no se exceda el intervalo máximo permisible entre dos flancos de la señal, esto es importante para propósitos de sincronización En la figura 2.5 se muestra un ejemplo de la codificación NRZ, donde se ve que el nivel permanece constante al enviarse varios bits seguidos del mismo valor.

[1] Di Natale, Marco. *Understanding and using the Controller Area Network.* P. 8. Recuperado 24 de Septiembre de 2014 de http://inst.cs.berkeley.edu/~ee249/fa08/Lectures/handout_canbus2.pdf

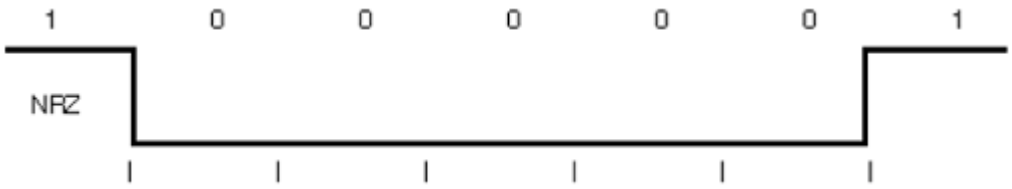


Figura 2.5. Ejemplo de codificación NRZ.

Dado que cada nodo conectado al bus tiene su propio oscilador pueden presentarse desfasamiento entre ellos. Todos los nodos se sincronizan cuando alguno comienza la transmisión y se re-sincronizan con cada cambio de nivel, pero, debido a la característica de la codificación NRZ, cuando el nivel permanece constante mucho tiempo se aplica el relleno de bits (Bit Stuffing), insertando bits complementarios después de cinco bits del mismo valor; el receptor tiene que quitar estos bits de relleno para poder procesar el mensaje original. En la figura 2.6 podemos ver el relleno con bits.

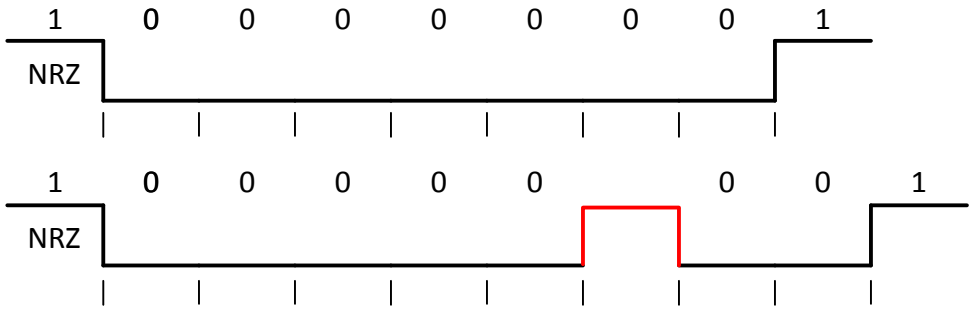


Figura 2.6. Datos sin relleno (arriba) y datos con relleno (abajo).

2.2.4. Niveles eléctricos del bus

El controlador CAN es el elemento encargado de la comunicación entre el microprocesador de la ECU y el transmisor-receptor CAN, este controlador trabaja acondicionando la información que entra y sale entre ambos componentes, y también es el que determina la velocidad de trasmisión de los mensajes; además interviene en la necesaria sincronización entre los diferentes nodos para la correcta emisión y recepción de los mensajes. El transceptor CAN es el elemento que tiene la misión de recibir y de trasmitir los datos, además de acondicionar y preparar la información para que pueda ser utilizada por los controladores; esta preparación consiste en situar los niveles de tensión de forma adecuada, amplificando la señal cuando la información se vuelca en la línea y reduciéndola cuando es recogida de la misma y suministrada al controlador; el transceptor en ningún caso interviene modificando el contenido del mensaje. Funcionalmente el transceptor está situado entre los cables que forman la línea “bus CAN” y el controlador.

Hemos definido al bit “dominante” (‘0’) y al bit “recesivo” (‘1’), estos bits tienen niveles TTL (5 V para alto ó ‘1’ y 0 V para bajo ó ‘0’) y están presentes a la salida

del controlador CAN hacia el transceptor CAN, en las líneas de Transmisión (Tx) y Recepción (Rx) correspondientes. Cuando se presenta un estado recesivo el voltaje nominal en las líneas “H” y “L” es de 2.5 V para cada una de ellas, lo que arroja un voltaje diferencial nominal de 0 V; para un estado dominante el voltaje nominal en “H” es de 3.5 V y para “L” es de 1.5 V, esto arroja un voltaje diferencial nominal de 2 V.

Se ha de tomar en cuenta que la especificación CAN de Bosh no establece cómo se han de traducir estos niveles a variable física, (entendiéndose con esto si el bus es óptico, eléctrico, electromagnético, etc.). Cuando se utiliza el par trenzado, según ISO 11898, el nivel dominante es un voltaje diferencial positivo en el bus; el nivel recesivo es ausencia de voltaje, o cierto valor negativo. En términos simples, los valores dominante y recesivo, descritos anteriormente, están definidos a nivel de protocolo y se presentan eléctricamente, generalmente con niveles TTL, entre el controlador y transceptor CAN; entonces, el valor dominante ‘0’ provoca un voltaje diferencial positivo en el bus a través del transceptor (así como voltaje diferencial positivo provoca un valor dominante ‘0’ a través del transceptor), y un valor recesivo ‘1’ provoca un voltaje diferencial cero o negativo en el bus a través del transceptor (así como un voltaje diferencial cero o negativo provoca un valor recesivo a través del transceptor). La figura 2.7 muestra los niveles en el bus antes descritos.

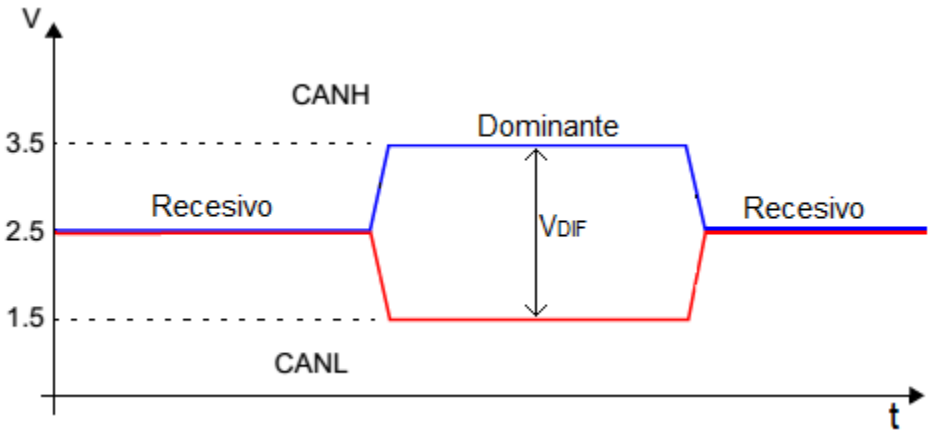


Figura 2.7. Niveles del bus.

2.2.5. Formato de tramas

CAN utiliza mensajes de estructura predefinida, tramas, para la gestión de la comunicación. Se distinguen entre dos variantes de CAN, el definido en CAN 2.A o "CAN Estándar" y el definido en CAN 2.B o "CAN Extendido". En estas variantes de CAN los formatos de trama son análogos, diferenciándose básicamente en el número de bits que se utilizan para el identificador de mensaje: de 11 bits, dando 2048 identificadores diferentes; sin embargo la especificación dice que los 7 bits más significativos del identificador no pueden ser simultáneamente recesivos, por lo tanto debido a los 4 bits restantes hay 16 combinaciones en los que estos siete bits son todos recesivos y el número de identificadores se reduce a 2032 ($2048-2^4$)

en CAN estándar; y de 29 bits que dan 536.870.912 identificadores para CAN Extendido. Las tramas CAN son de longitud reducida, la trama más larga es de 130 bits en CAN Estándar y 154 bits en CAN Extendido.

Los cuatro tipos de trama utilizados en las dos variantes son: **trama de datos (Data Frame),** lleva datos de un transmisor a los receptores, puede incluir entre 0 y 8 bytes de información útil; **trama remota (Remote Frame),** es transmitida por algún nodo del bus que requiere la transmisión de una *trama de datos* con el mismo *identificador*; **trama de error (Error Frame),** es transmitida por alguna unidad que detecte algún error; **trama de sobrecarga (Overload Frame),** permite que un nodo fuerce a los demás a alargar el tiempo entre transmisión de tramas sucesivas.

Trama de datos

Una trama de datos, figura 2.8, se compone de siete diferentes campos de bits: Inicio de trama (SOF – Start Of Frame), arbitraje, control, datos, CRC (Cyclic Redundancy Check), reconocimiento, fin de trama.

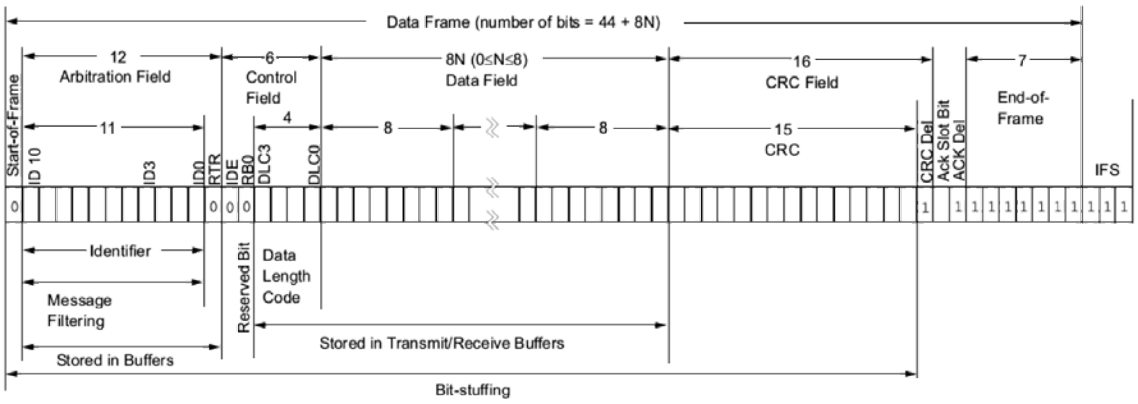


Figura 2.8. Trama de datos estándar.

Inicio de trama (SOF): El inicio de trama es un campo de un solo bit siempre dominante, que indica el inicio de la transmisión. Los nodos receptores se sincronizan con el flanco de bajada de este bit.

Arbitraje: El campo de identificación está formado por el identificador de mensaje (11 bits) más el bit RTR (Remote Transmission Request). En una trama de datos el bit RTR es dominante, en una trama remota es recesivo. Los bits de identificador se transmiten en orden de más significativo a menos significativo.

Control: El campo de control, mostrado en la figura 2.9, está formado por dos bits reservados para uso futuro y cuatro bits adicionales que indican el número de bytes de datos. En realidad el primero de estos bits, IDE (Identifier Extension), se utiliza para indicar si la trama es de CAN Estándar (IDE dominante) o Extendido (IDE recesivo). El segundo bit, RB0 (Reserved Bit Zero), siempre es recesivo. Los

cuatro bits de código de longitud (DLC- Data Lenght Code) indican en binario el número de bytes de datos en el mensaje (0 a 8, otros valores no pueden ser usados).

La tabla 2.2 muestra el número de bytes de datos que se pueden obtener, cantidad codificada por los bits de código de longitud de datos, donde "D" es para 'dominante' y "R" para 'recesivo'.

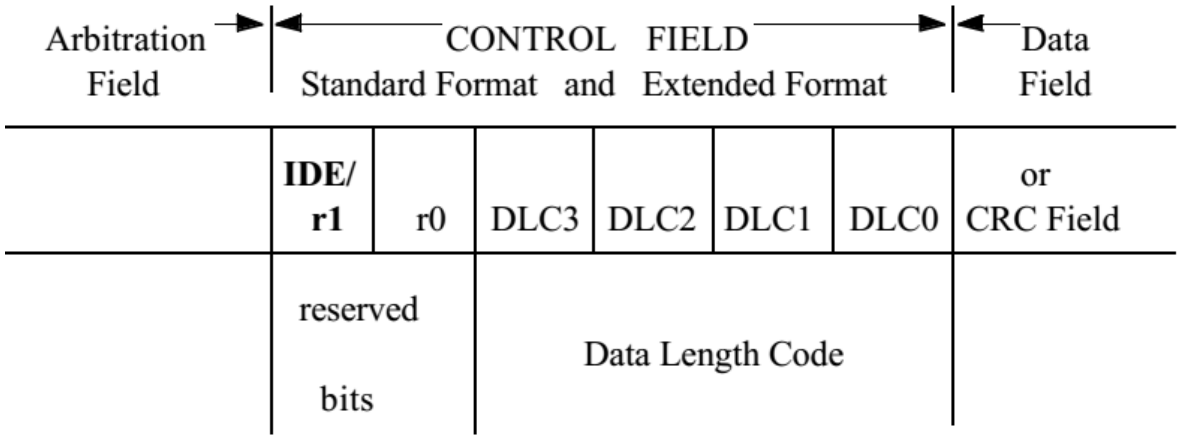


Figura 2.9. Campo de control.

Número de bytes de Datos	Código de longitud de datos			
	DLC3	*DCL2*	*DLC1*	*DLC0*
0	D	D	D	D
1	D	D	D	R
2	D	D	R	D
3	D	D	R	R
4	D	R	D	D
5	D	R	D	R
6	D	R	R	D
7	D	R	R	R
8	R	D	D	D

Tabla 2.2. Número de bytes de datos en el DLC. Dominante "D", Recesivo "R".

Datos: Es un campo formado por 0 a 8 bytes de datos, es decir 0 a 64 bits. Cada byte se transmite con el bit más significativo primero.

CRC: Código de redundancia cíclica que genera el transmisor por la división módulo 2 de todos los bits precedentes del mensaje, incluyendo los de relleno si existen, por el polinomio generador: $X^{15}+X^{14}+X^{8}+X^{7}+X^{4}+X^{3}+X^{1}+1$, el residuo de esta división es el código CRC transmitido. Los receptores comprueban este código. Tras el código CRC se incluye un bit recesivo (delimitador de CRC).

Campo de reconocimiento (ACK): es un campo de dos bits que el transmisor envía como recesivos. El primero de estos bits es cambiado a un bit dominante

por los nodos que han recibido el mensaje correctamente. El bit de ACK queda así insertado entre dos bits dominantes de delimitación.
Fin de trama (EOF): Cierra la trama, consiste en 7 bits recesivos sucesivos.

Espaciado entre tramas (IFS). Consta de un mínimo de 3 bits recesivos.

Trama de datos extendida

En la trama de datos extendida en CAN, mostrada en la figura 2.10, al bit SOF le sigue el campo de arbitraje, el cual consta de 32 bits. Los primeros 11 bits son los bits más significativos (MSB, Most Significant Bits) (identificador base) de los 29 bits del identificador. Esos 11 bits son seguidos por el bit sustituto de petición remota (SRR – Substitue Remote Request), el cual es recesivo y aparece en la posición del bit RTR de la trama estándar. Después del bit SRR está el bit IDE, éste es recesivo para indicar que se trata de una trama extendida.

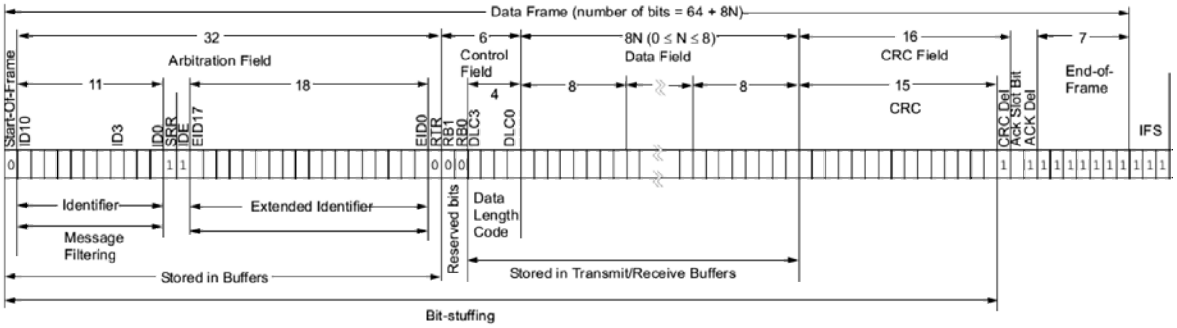


Figura 2.10. Trama de datos extendida.

Es importante hacer notar que si el arbitraje permanece sin resolverse después de la transmisión de los primeros 11 bits del identificador, y uno de los nodos involucrados en el arbitraje está enviando una trama CAN estándar (11 bits de identificación), la trama CAN estándar ganará el arbitraje debido a la aserción del bit dominante IDE. También, el bit SRR en una trama CAN extendida debe ser recesivo para permitir la aserción del bit dominante RTR por el nodo que está enviando una trama remota estándar.

A los bits SRR e IDE les siguen los restantes 18 bits del identificador (identificador extendido) y el bit de petición de transmisión remota.

Para habilitar el envío de tramas estándar y extendidas a través de una red compartida, los 29 bits de identificador de mensaje extendido se dividen en secciones de 11 bits (Más Significativos) y 18 bits (Menos Significativos). Esta división asegura que el bit IDE pueda permanecer en la misma posición de bit tanto en la trama estándar como en la trama extendida. Seguido del campo de arbitraje está el campo de control de seis bits. Los primeros dos bits de este campo están reservados y deben ser dominantes. Los cuatro bits restantes del

campo de control son el DLC, el cual especifica el número de bytes de datos contenidos en el mensaje.

La porción restante de la trama (campo de datos, campo del CRC, campo de reconocimiento, fin de trama y el espacio entre tramas) está construida de la misma manera que una trama de datos estándar.

Trama Remota

El formato de la trama remota es análogo a la trama de datos (estándar o extendida) pero con el bit RTR recesivo. Por otra parte una trama remota, mostrada en la figura 2.11, no incluye nunca datos. El identificador es el del mensaje que se solicita, el campo longitud corresponde a la longitud de ese mensaje. Por lo tanto una trama remota se usa para pedir la transmisión de un mensaje con un identificador dado, esto desde un nodo remoto. Las características de la trama remota en resumen son:

- El campo de *identificación,* se usa para indicar el identificador del mensaje pedido.
- El campo de datos siempre está vacío (0 bytes).
- El campo DLC indica la longitud del dato del mensaje pedido (no del mensaje transmitido).
- El bit RTR en el campo de arbitraje siempre se establece como recesivo.

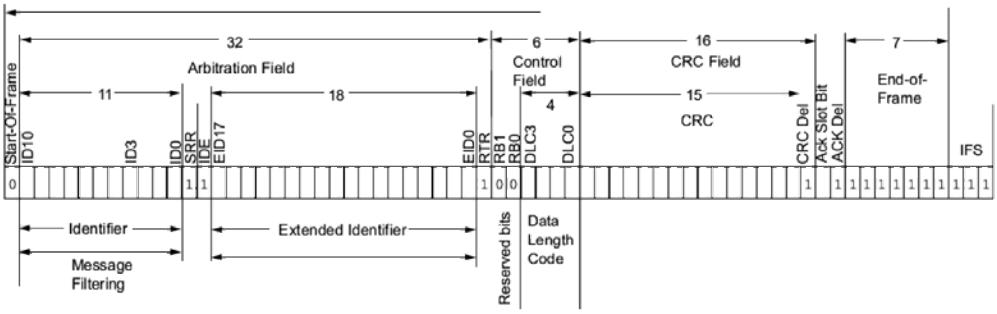


Figura 2.11. Trama Remota con identificador extendido. Sin campo de datos.

Trama de Error

La trama de error no es una trama como las anteriores. En la figura 2.12 se puede observar que está conformada por dos tramas; la primera es una trama de datos interrumpida por la trama de error mostrada en la parte inferior de la figura. Las tramas de error son generadas por cualquier nodo que detecta un error. Consiste de dos campos: Bandera de error (Error Flag) y Delimitador de error. La bandera de error es distinta según el estado de error del nodo que detecta el error (los estados de error de nodo, “activo” y “pasivo”, se describirán más adelante). El delimitador de error consta de 8 bits recesivos consecutivos y permite a los nodos reiniciar la comunicación limpiamente tras el error.

Si un nodo en estado de error "Activo" detecta un error en el bus, interrumpe la comunicación del mensaje en proceso generando una "Bandera de error activo", que consiste en una secuencia de 6 bits dominantes sucesivos. Esta secuencia rompe la regla de relleno de bits y provocará la generación de tramas de error en otros nodos, por lo que la bandera de error puede extenderse entre 6 y 12 bits dominantes sucesivos. Finalmente se espera el campo de delimitación de error formado por los 8 bits recesivos. Entonces la comunicación se reinicia y el nodo que había sido interrumpido reintenta la transmisión del mensaje.

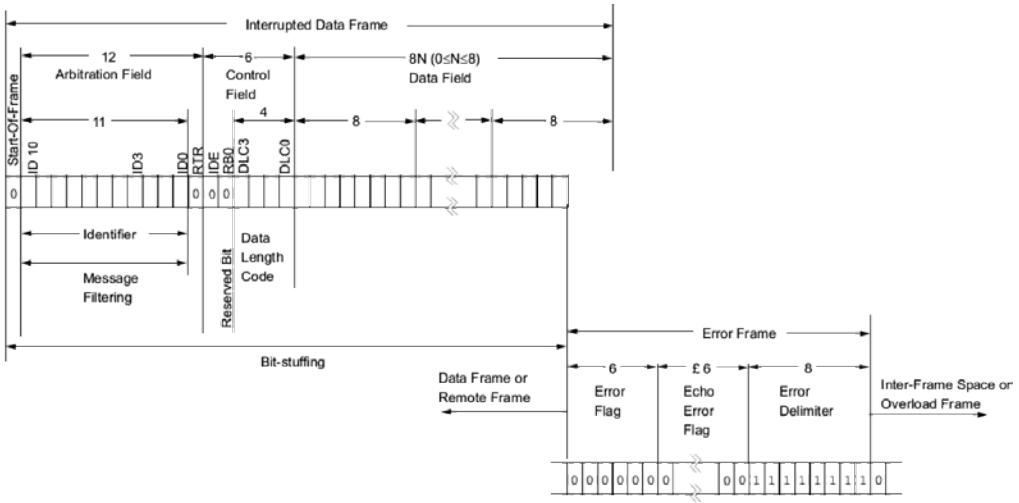


Figura 2.12. Trama de error.

Si un nodo en estado de error "Pasivo" detecta un error, el nodo transmite una "Bandera de error pasivo", seguido, de nuevo, por el campo delimitador de error. La Bandera de error de tipo pasivo consiste en 6 bits recesivos consecutivos y el delimitador de tipo pasivo es de 8 bit, por tanto, la trama de error para un nodo pasivo es una secuencia de 14 bits recesivos. De aquí se deduce que la transmisión de una trama de error de tipo pasivo no afectará a ningún nodo en la red, excepto cuando el error es detectado por el propio nodo que está transmitiendo. En ese caso los demás nodos detectarán una violación de las reglas de bits de relleno y transmitirán a su vez tramas de error.

Tras señalar un error por medio de la trama de error apropiada, cada nodo transmite bits recesivos hasta que recibe un bit también recesivo, luego transmite 7 bits recesivos consecutivos antes de finalizar el tratamiento de error.

Espacio entre tramas

El espacio entre tramas separa una trama (de cualquier tipo) de la siguiente trama de datos o interrogación remota. El espacio entre tramas ha de constar de, al menos, 3 bits recesivos. Esta secuencia de bits se denomina "intermisión". Una vez transcurrida esta secuencia, un nodo en estado de error activo puede iniciar una nueva transmisión o el bus permanecerá en reposo. Para un nodo en estado de error pasivo la situación es diferente, éste deberá esperar una secuencia

adicional de 8 bits recesivos antes de poder iniciar una transmisión. De esta forma se asegura una ventaja en el inicio de transmisión de información a los nodos en estado activo frente a los nodos en estado pasivo.

Bus inactivo: este estado del bus puede ser de una longitud arbitraria. Los nodos reconocen al bus como libre y cualquiera de ellos que tenga algo qué transmitir puede acceder al bus. El mensaje que esté pendiente de transmitir durante la transmisión de otro mensaje, comienza en el primer bit al terminar la intermisión.

Trama de Sobrecarga

La trama de sobrecarga se nuestra en la parte inferior derecha de la figura 2.13, después del final de una trama de datos. Una trama de sobrecarga tiene el mismo formato que una trama de error activo. Sin embargo, la trama de sobrecarga sólo puede generarse durante el espacio entre tramas. De esta forma se diferencia de una trama de error, que sólo puede ser transmitida durante la transmisión de un mensaje. La trama de sobrecarga consta de dos campos, la bandera de Sobrecarga, y el delimitador. La bandera de sobrecarga consta de 6 bits dominantes que pueden ser seguidos por los generados por otros nodos, dando lugar a un máximo de 12 bits dominantes. El delimitador es de 8 bits recesivos.

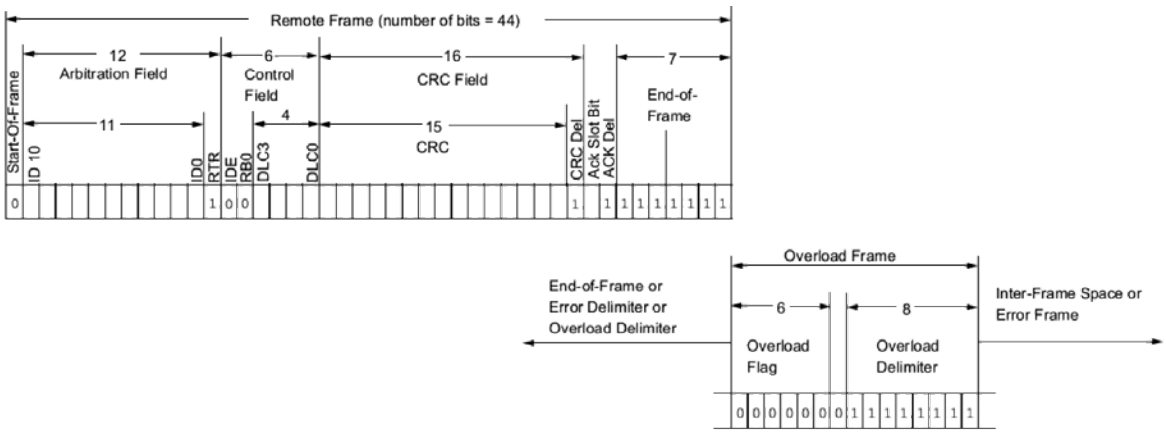


Figura 2.13. Trama de sobrecarga.

Una trama de sobrecarga puede ser generada por cualquier nodo como resultado de dos condiciones:

1. Debido a sus condiciones internas el nodo no puede iniciar la recepción de un nuevo mensaje. De esta forma retrasa el inicio de transmisión de un nuevo mensaje. Un nodo puede generar como máximo 2 tramas de sobrecarga consecutivas para retrasar un mensaje.
2. Por la detección por cualquier nodo de un bit dominante en los 3 bits de "intermisión".

Por todo ello una trama de sobrecarga generada por un nodo dará normalmente lugar a la generación de tramas de sobrecarga por los demás nodos, dando lugar,

como se ha indicado, a un máximo de 12 bits dominantes para la bandera de sobrecarga.

2.2.6. Manejo de error

El protocolo CAN está diseñado para la transferencia segura de mensajes y para proveer mecanismos para la detección de errores, señalización y autodiagnóstico, incluyendo medidas para el aislamiento de fallos, el cual previene que nodos defectuosos afecten el estado de la red.

Las medidas generales para la detección de errores se basan en la capacidad de cada nodo de controlar la difusión de las transmisiones sobre el bus, si se trata de un transmisor o de un receptor, y para señalar las condiciones de error resultantes de varias fuentes. Los mensajes dañados se marcan por cualquier nodo que detecta un error, tales mensajes se abortan y serán retransmitidos automáticamente.

Pasos en el tratamiento de errores en CAN

1. Se detecta un error global o local
2. Será transmitida una “Bandera de Error” (globalización del error)
3. En el caso de un error local la “Bandera de Error” procederá a sobreponerse a la “Bandera de Error” anterior seguido por el “Delimitador de Error”
4. El mensaje será descartado por cada nodo
5. Los contadores de error de cada nodo del bus son incrementados
6. La transmisión del mensaje interrumpido será repetida automáticamente

Detección de error

Hay 5 tipos diferentes de error (los cuales no son mutuamente excluyentes):

- **ERROR DE BIT.** El nodo que esté enviando un bit sobre el bus también lo monitorea. Un ERROR DE BIT debe ser detectado en el tiempo que dura ese bit, cuando el valor del bit que se monitorea es diferente al valor del bit enviado. Como excepciones están los bits recesivos enviados como parte del proceso de arbitraje o el espacio de ACK.
- **ERROR DE RELLENO.-** Un error de relleno se detecta en el sexto bit consecutivo del mismo valor en el campo de mensaje, el cual debe estar codificado por el método de relleno (bit stuffing).
- **ERROR DE CRC.-** La secuencia del CRC consiste del resultado del cálculo del CRC por el transmisor. El receptor calcula el CRC de la misma forma que el transmisor. Un ERROR DE CRC se detecta si el resultado del cálculo no es el mismo que el recibido en la secuencia CRC.
- **ERROR DE FORMA.-** Este error de detecta cuando la forma fija de un campo de bits contiene uno o más bits ilegales.

- **ERROR DE RECONOCIMIENTO.-** Un ERROR DE RECONOCIMIENTO se detecta por el transmisor cuando no detecta un bit 'dominante' durante el espacio ACK.

Señalización de error

Un nodo que detecta una condición de error lo señala transmitiendo una bandera de error. La trama de error no es una trama verdadera, en realidad es el resultado de una secuencia de señalización de error que consiste en la superposición de banderas de error, transmitido desde diferentes nodos, posiblemente en diferentes momentos, seguido por un campo delimitador de error. Siempre que un nodo detecta un ERROR DE BIT, un ERROR DE RELLENO, un ERROR DE FORMA o un ERROR DE RECONOCIEMIENTO, se empieza la transmisión de una bandera de error en el respectivo nodo en el siguiente bit. Siempre que se detecta un ERROR DE CRC, la transmisión de una bandera de error empieza en el siguiente bit al delimitador ACK, al menos que la bandera de error ya haya comenzado debido a otra condición. Para un nodo en error activo es una bandera de ERROR ACTIVO, el cual consta de 6 bits 'dominantes', para un nodo en error pasivo es una bandera de ERROR PASIVO, que consiste de 6 bits recesivos consecutivos.

Las banderas de ERROR violan la regla de relleno de bits o destruye la forma fija del campo de reconocimiento o el de fin de trama. Como consecuencia, todos los demás nodos detectan una condición de error y comienzan la transmisión de una bandera de ERROR por su parte. Así que, la secuencia de bits dominantes que pueden ser monitoreados por los niveles del bus, es una superposición de diferentes banderas de error transmitidos por los nodos individuales. La longitud total de esta secuencia varía entre un mínimo de seis y un máximo de doce bits. La bandera de error pasivo enviado por un nodo en error pasivo no tiene efecto sobre el bus. Sin embargo, el nodo de señalización todavía tendrá que esperar seis bits consecutivos de igual nivel, que comienza en el principio de la bandera de error pasivo antes de continuar.

El tiempo de recuperación desde la detección de un error hasta el comienzo del siguiente mensaje, la mayoría de las veces, es de 31 bits.

2.2.7. Aislamiento de fallos

El protocolo CAN tiene un protocolo de aislamiento de fallos que detecta unidades defectuosas y las pone en estados pasivos o desactivadas, de manera que no puedan afectar el estado del bus con sus salidas. El protocolo asigna a cada nodo uno de tres estados:

- Error activo
- Error pasivo
- Fuera de bus

Las unidades cambian sus estados de acuerdo al valor de dos contadores enteros:

Una unidad en ‘error activo’ normalmente puede tomar parte de las comunicaciones del bus y envía una bandera de ERROR ACTIVO cuando se ha detectado un error.

Una unidad en ‘error pasivo’ no debe enviar una bandera de ERROR ACTIVO. Éste toma parte de las comunicaciones en el bus, pero cuando un error ha sido detectado, sólo se envía una bandera de ERROR PASIVO. También después de una transmisión, una unidad en ‘error pasivo’ esperará antes de iniciar una nueva transmisión

A una unida en ‘fuera de bus’ no se le permite tener alguna influencia sobre el bus. Para el aislamiento de fallos están implementados dos contadores en cada unidad en el bus:

1) Contador de error de transmisión
2) Contador de error de recepción

Esos contadores se modifican de acuerdo a las siguientes reglas:

1. Cuando un RECEPTOR detecta un error, el CONTADOR DE ERROR DE RECEPCIÓN se incrementará en 1, excepto cuando el error detectado fue un ERROR DE BIT, durante el envío de una BANDERA DE ERROR ACTIVO o de una BANDERA DE SOBRECARGA.
2. Cuando un RECEPTOR detecta un bit ‘dominante’, como el primer bit después de enviar una BANDERA DE ERROR, el CONTADOR DE ERROR DE RECEPCIÓN se incrementará en 8.
3. Cuando un TRANSMISOR envía una BANDERA DE ERROR, el CONTADOR DE ERROR DE TRANSMISIÓN se incrementa en 8.

Excepción 1: Si el transmisor está en ‘error pasivo’ y detecta un ERROR DE RECONOCIMIENTO, porque no detecta un ACK ‘dominante’ y no detecta un bit ‘dominante’ al enviar su BANDERA DE ERROR PASIVO.

Excepción 2: Si el TRANSMISOR envía una BANDERA DE ERROR porque ocurrió un ERROR DE RELLENO durante el ARBITRAJE, y debe haber sido ‘recesivo’, y ha sido enviado como ‘recesivo’ pero monitoreado como ‘dominante’.

En las excepciones 1 y 2 el CONTADOR DE TRANSMISIÓN no cambia.

4. Si un TRANSMISOR detecta un ERROR DE BIT mientras envía una BANDERA DE ERROR ACTIVO o una BANDERA DE SOBRECARGA, el CONTADOR DE ERROR DE TRANSMISIÓN se incrementa en 8.

5. Si un RECEPTOR detecta un ERROR DE BIT mientras envía una BANDERA DE ERROR ACTIVO o una BANDERA DE SOBRECARGA, el CONTADOR DE ERROR DE RECEPCIÓN se incrementa en 8.
6. Cualquier nodo tolera hasta 7 bits 'dominantes' consecutivos después de enviar una BANDERA DE ERROR ACTIVO, BANDERA DE ERROR PASIVO o BANDERA DE SOBRACARGA. Después de detectar el décimo cuarto bit 'dominante' consecutivo (en el caso de una BANDERA DE ERROR ACTIVO o de una BANDERA DE SOBRECARGA) o después de detectar el octavo bit 'dominante' consecutivo seguido de una BANDERA DE ERROR PASIVO, y después de cada secuencia de ocho bits 'dominantes' consecutivos adicionales cada TRANSMISOR incrementa su CONTADOR DE ERROR DE TRANSMISIÓN en 8 y cada RECEPTOR incrementa su CONTADOR DE ERROR DE RECEPCIÓN en 8.
7. Después de una transmisión exitosa de un mensaje (obtener un ACK y sin error hasta finalizar FIN DE TRAMA) El CONTADOR DE ERROR DE TRANSMISIÓN se disminuye en 1, al menos que ya esté en 0.
8. Después de una recepción exitosa de un mensaje (recepción sin error hasta el ESPACIO ACK y el envío exitoso del bit ACK), el CONTADOR DE ERROR DE RECEPCIÓN se disminuye en 1, si estaba entre 1 y 127. Si el CONTADOR DE ERROR DE RECEPCIÓN estaba en 0, se queda en 0, y si era mayor a 127, entonces se establecerá en un valor de entre 119 y 127.
9. Un nodo está en 'error pasivo' cuando el CONTADOR DE ERROR DE TRANSMISIÓN es igual o mayor a 128, o cuando el CONTADOR DE ERROR DE RECEPCIÓN es igual o mayor a 128. Una condición de error que deja que un nodo se convierta a 'error pasivo' provoca que el nodo envíe una BANDERA DE ERROR ACTIVO.
10. Un nodo esta en 'fuera de bus' cuando el CONTADOR DE ERROR DE TRANSMISIÓN es mayor o igual a 256.
11. Un nodo en 'error pasivo' se vuelve a 'error activo' otra vez cuando, tanto el CONTADOR DE ERROR DE TRANSMISIÓN como el CONTADOR DE ERROR DE RECEPCIÓN son menores o igual a 127.
12. A un nodo que esté 'fuera de bus' se le permite convertirse a 'error activo' (ya no más 'fuera de bus') con sus dos contadores de error puestos en 0, y después de 128 apariciones de 11 bits 'recesivos' consecutivos que han sido monitoreados en el bus.

Nota:
Una cuenta de error mayor de alrededor a 96 indica una fuerte perturbación en el bus. Puede tener ventaja proveer medios para poner a prueba esta condición.

Nota:
Inicio: Si durante la puesta en marcha del sistema sólo está en línea un nodo, y éste nodo transmite algún mensaje, no recibirá ningún reconocimiento, detecta un error y repite el mensaje. Este nodo se pone en 'error pasivo' pero no en 'fuera de bus' debido a la razón mencionada.

2.2.8. Tolerancia del oscilador en el bus CAN

La tolerancia máxima permitida del oscilador en el bus CAN es de 1.58%. La oscilación está dada por un resonador cerámico, por lo tanto se requiere de su uso en un bus con una velocidad de hasta 125 kbits/s como regla de oro.[2]

Para una mayor velocidad y mejor precisión del bus para el protocolo CAN se necesita de un oscilador de cuarzo.

2.2.9. Requerimientos para el tiempo de bit

Todos los dispositivos en el bus CAN deben tener la misma velocidad de bits. Sin embargo, no se requiere que todos los nodos tengan la misma frecuencia del oscilador del reloj maestro. Por las diferencias en las frecuencias de los relojes de cada nodo, la velocidad de bits tiene que ser ajustada estableciendo apropiadamente el pre-escalador del 'Baud Rate' y el número de cuantos de tiempo en cada segmento.

El tiempo de los bits en CAN está compuesto por segmentos no superpuestos. Cada uno de esos segmentos está compuesto por unidades llamados "Cuantos de Tiempo" (TQ – Time Quanta). La velocidad nominal de bits (NBR – Nominal Bit Rate) está definida en CAN como el número de bits transmitidos por segundo por un transmisor ideal sin re-sincronización. Se puede describir con la ecuación 2.1:

$$NBR = f_{bit} = \frac{1}{t_{bit}} \tag{2.1}$$

Tiempo nominal de bit.- El tiempo nominal de bit (NBT) (t_{bit}) está compuesto por segmentos no superpuestos, mostrados en la figura 2.14.

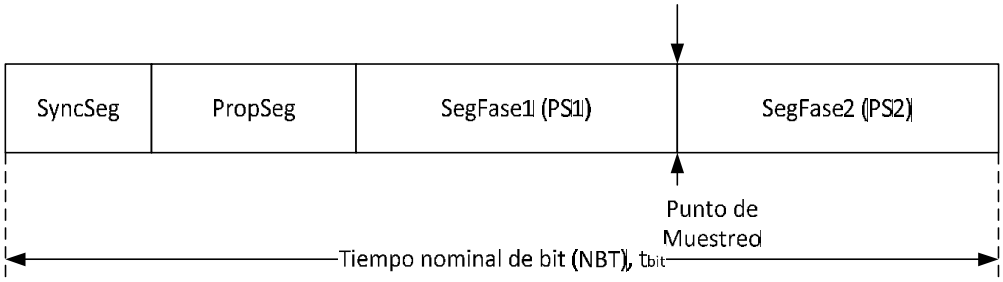


Figura 2.14. Segmentos del tiempo de bit en CAN.

El NBT es la suma de los siguientes segmentos:

$$t_{bit} = t_{SegSinc} + t_{SegProp} + t_{PS1} + t_{PS2} \tag{2.2}$$

[2] Dais, S; Chapman, M. *Impact of Bit Representation on Transport Capacity and Clock Accuracy in Serial Data Streams,* SAE Technical Paper Series 890532, Multiplexing in Automobil SP-773, March 1989.

Asociado con el NBT está el punto de muestreo, el Ancho de Salto de sincronización (SJW – Synchronization Jump Width) y el Tiempo de Procesamiento de la Información (IPT – Information Processing Time).

Se requiere que los nodos se sincronicen en los flancos de los bits, así que cada nodo concuerda con el valor del bit transmitido actualmente en el bus. Para sincronizarse, cada nodo implementa un protocolo de sincronización que mantiene la velocidad de bits del receptor alineado con la velocidad actual de los bits transmitidos. El protocolo de sincronización usa los flancos de transición para re-sincronizar a los nodos. Por lo tanto, se deben evitar las largas secuencias sin transición de bits para evitar desfasamientos entre los bits de los nodos. Esta es la razón por la que el protocolo emplea la técnica llamada “bit de relleno”, lo que fuerza a complementar el flujo después de transmitir 5 bits del mismo tipo. Los bits de relleno son insertados automáticamente por el nodo transmisor y removidos en el lado receptor antes del procesamiento del contenido de la trama.

La transmisión síncrona de los bits permite el protocolo de arbitraje en CAN y simplifica el manejo del flujo de bits, pero también requiere un sofisticado protocolo de sincronización. La sincronización de los bits se realiza primero al recibir el bit de inicio disponible con cada transmisión asíncrona. Después, para permitir que los receptores lean correctamente el contenido del mensaje se requiere de una continua re-sincronización. Otras características del protocolo influyen en la definición de la temporización del bit. Para propósito de arbitraje del bus, reconocimiento del mensaje y señalización de error, el protocolo requiere que los nodos puedan cambiar el estado de un bit transmitido de recesivo a dominante, informando del cambio de estado del bit a todos los demás nodos de la red antes de finalizar la transmisión. Esto significa que el tiempo del bit debe ser al menos lo suficiente grande para acomodar la propagación de la señal desde cualquier transmisor a cualquier receptor y de regreso al transmisor.

El tiempo del bit incluye un segmento de retardo de la propagación que toma en cuenta la propagación de la señal en el bus, así como los retardos causados por la transmisión y recepción en los nodos. En la práctica esto significa que la señal de propagación se determina por los dos nodos dentro del sistema que están más apartados entre sí.

El flanco del bit de un nodo transmisor alcanza a un nodo receptor, después la señal se propaga por todo el camino desde los dos nodos. En este punto el receptor puede cambiar su valor de recesivo a dominante, pero el nuevo valor no alcanzará el transmisor hasta que la señal se propague por todo el camino de regreso; sólo entonces el primer nodo puede decidir si el nivel de su propia señal (en este caso recesivo) es el valor actual en el bus o si éste ha sido remplazado por un nivel dominante por otro nodo.

Considerando el protocolo de sincronización y a la necesidad de que todos los nodos estén de acuerdo con el valor del bit, el tiempo nominal del bit puede definirse como la composición de cuatro segmentos. Estos segmentos son:

-Segmento de sincronización (SyncSeg)
-Segmento de propagación (PropSeg)
-Segmento de fase 1 (PS1)
-Segmento de fase 2 (PS2)

Segmento de sincronización (SyncSeg): Es el primer segmento en el NBT y se usa para sincronizar a los nodos en el bus. Se espera que los flancos de los bits ocurran dentro de este segmento.

Segmento de propagación (PropSeg): Este segmento sirve para compensar los retardos físicos dentro de la red. El retardo de propagación se define como dos veces la suma del tiempo de propagación de la señal en el cable del bus, el retardo del comparador de entrada y el retardo del controlador de salida.

Segmento de fase 1 y 2 (PS1 y PS2 respectivamente): Los dos segmentos de fase se usan para compensar el error de fase de los bordes de los bits dentro del bus. PS1 se puede alargar o PS2 acortar en la re-sincronización.

Punto de muestreo (SP): El punto de muestreo es el punto en el cual se lee e interpreta el nivel del bus como el valor respectivo del bit. El punto de muestro se localiza al final de PS1.

Tiempo de Cuanta (TQ): El tiempo de cuanta es una unidad de tiempo fija derivada del periodo del oscilador. Hay un pre-escalador programable, con valores enteros, que van desde 1 a 32. Empezando con el TIEMPO MINIMO DE CUANTA. El TIEMPO DE CUANTA puede tener una longitud de:

$$TIEMPO\ DE\ CUANTA = m * TIEMPO\ MÍNIMO\ DE\ CUANTA \qquad (2.3)$$

En la que m es el valor del pre-escalador. El TIEMPO DE CUANTA es la mínima resolución en la definición del tiempo del bit y el error máximo asumido por el protocolo de sincronización orientado a bits. La longitud de cada segmento queda:

- SyncSeg es de 1 TQ de longitud.
- PropSeg se programa para ser de longitud 1,2,…,8 TQ.
- PS1 se programa para ser de longitud 1,2,…,8 TQ.
- PS2 con un máximo de entre PS1 y del TIEMPO DE PROCESAMIENTO DE LA INFORMACIÓN, éste último debe ser menor o igual a 2 TQ.

El número total de TQ en el tiempo de un bit puede programarse desde al menos 8 hasta 25.

Sincronización

Para compensar los desplazamientos de fase entra las frecuencias de lo osciladores de cada nodo del bus, cada controlador CAN debe ser capaz de

sincronizarse con el flanco relevante de la señal entrante. Cuando se detecta un flanco en el dato transmitido, la lógica comparará la ubicación del flanco en el momento previsto (SyncSeg). Entonces el circuito ajustará los valores de PS1 y PS2 como sea necesario. Hay dos mecanismos de sincronización:

1. Hard synchronization
2. Re-sincronización

Hard Synchronización: Toma lugar al comienzo de la trama, cuando el bit de inicio de trama SOF cambia el estado del bus de recesivo a dominante. Después de la detección del flanco correspondiente, el tiempo de bit se reinicia al final del segmento de sincronización. Por lo tanto "hard synchronization" fuerza que el flanco del bit de inicio se encuentre dentro del segmento de sincronización del tiempo de bit reiniciado.

Re-sincronización: Como resultado de la re-sincronización el PS1 puede alargarse o el PS2 acortarse, ver figura 2.15.

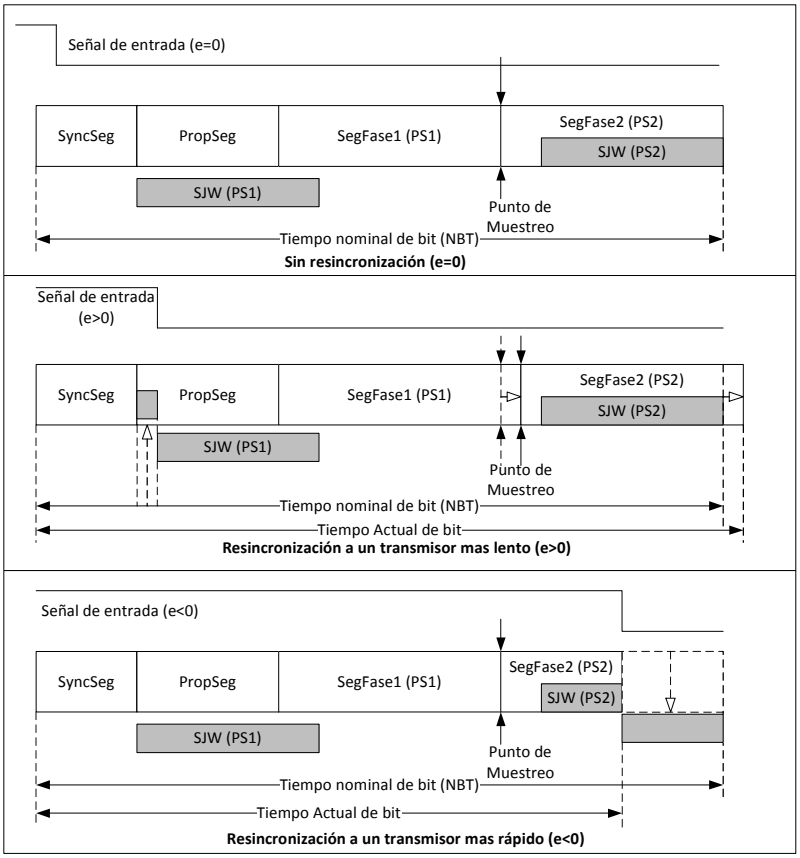


Figura 2.15. Re-sincronización el tiempo de bit.

La cantidad de alargamiento o acortamiento de PS1 y PS2 tiene un límite superior dado por SJW, el cual debe ser programado entre 1 TQ y 4TQ.

La información de sincronización sólo se puede derivar de la transición de un valor de bit a otro. Por lo tanto, la posibilidad de volver a sincronizar un nodo del bus con el flujo de bits durante una trama depende de la propiedad de que exista un intervalo máximo de tiempo entre dos transiciones de bit (aquí la importancia del protocolo de bits de relleno).

El diseñador del dispositivo puede programar los parámetros del tiempo de bit en el controlador CAN por medio de registros apropiados. Tomar en cuenta que, dependiendo del tamaño del segmento de propagación, se puede determinar la longitud máxima posible del bus a una velocidad de datos específica (o la velocidad máxima de datos con una longitud de bus específica).

Errores de fase

El ERROR DE FASE de un flanco está dado por la posición del flanco con relación al SyncSeg, se mide en TQ. El signo del ERROR DE FASE se define como sigue:

- e=0 si el flanco se encuentra dentro del SyncSeg.
- e>0 si el flanco se encuentra antes del punto de muestreo (se añaden TQ al PS1).
- e<0 si el flanco se encuentra después del punto de muestreo (se sustraen TQ del PS2).

Sin error de fase e=0: Si la magnitud del error de fase es menor o igual que el valor programado del SJW, el efecto de la re-sincronización es el mismo que en “hard synchronization”.

Error de fase positivo (e>0): Si la magnitud del error de fase es mayor que el SJW y, si el error de fase es positivo, PS1 es alargado una cantidad igual al SJW.

Error de fase negativo (e<0): Si la magnitud del error de fase es mayor que el SJW, y el error de fase es negativo, PS2 es acortado por una cantidad igual al SJW.

Reglas de sincronización

1. Sólo se usarán los flancos de recesivo a dominante para sincronización.
2. Se permite sólo una sincronización dentro de un tiempo de bit.
3. Un flanco podrá ser usado para sincronización sólo si el valor detectado en el punto de muestreo previo difiere del valor del bus inmediatamente después del flanco.
4. Un nodo transmisor no se sincronizará en un error de fase positivo (e>0).
5. Si la magnitud absoluta del error de fase es mayor que el SJW, el segmento de fase apropiado se ajustará por una cantidad igual al SJW.

2.3. Protocolo SPI

El protocolo SPI (Serial Peripheral Interface) es un protocolo de datos en serie síncrono, usado en microcontroladores para comunicarse rápidamente con uno o más dispositivos periféricos a distancias cortas. También se puede usar para comunicación entre microcontroladores.

SPI es un bus sobre el cual se transmiten paquetes de información de 8 bits. Cada una de estas líneas porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es *full dúplex*. Dos de estas líneas trasfieren los datos (una en cada dirección) y la tercer línea es la del reloj. Algunos dispositivos sólo pueden ser transmisores y otros sólo receptores, generalmente un dispositivo que tramite datos también puede recibir.

Con una conexión SPI siempre hay un dispositivo maestro (usualmente un microcontrolador) que controla a los dispositivos periféricos, inicia la transferencia de información sobre el bus y genera las señales de reloj. Un esclavo es un dispositivo controlado por el maestro. En la figura 2.16 se muestra una conexión SPI con un maestro y tres esclavos.

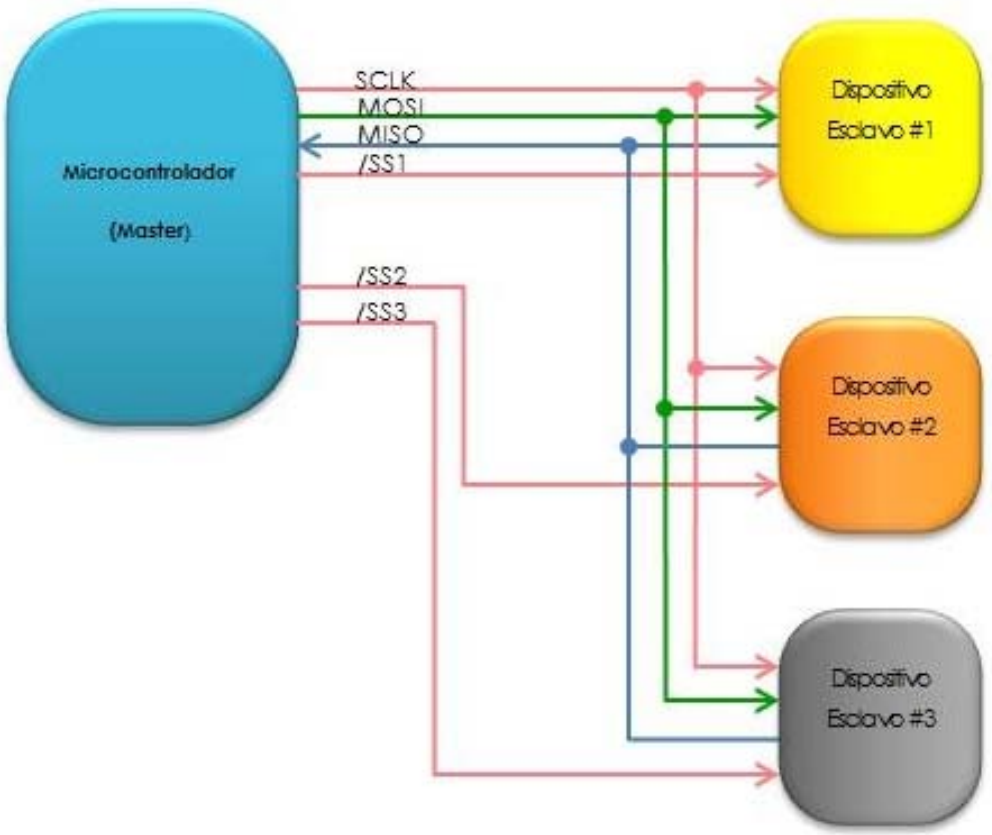


Figura 2.16. Conexión con SPI.

Típicamente las tres líneas en común para todos los dispositivos son:

- **MISO** (Master In Slave Out) – Línea del esclavo para enviar datos al maestro.
- **MOSI** (Master Out Slave In) – Línea del maestro para enviar datos a los periféricos (esclavos).
- **SCK** (Serial Clock) – Pulsos de reloj generados por el maestro que sincronizan la transmisión de datos.

Y una línea específica para cada dispositivo:

- **SS** (Slave Select) – es la terminal en cada dispositivo que el maestro puede usar para habilitar y deshabilitar dispositivos específicos.

Cuando la terminal SS de un dispositivo se selecciona mediante un estado “bajo”, éste se comunica con el maestro. Cuando está en estado “alto" el dispositivo ignora al maestro. Esto permite tener múltiples dispositivos SPI compartiendo las mismas líneas, MOSI, MISO y CLK.

Modos del Reloj en SPI

Existen cuatro modos de reloj definidos por el protocolo SPI, estos modos son:

- Modo A
- Modo B
- Modo C
- Modo D

Cada modo determina el valor de la polaridad del reloj con el bit CPOL (Clock Polarity) y de su fase con el bit CPHA (Clock Phase). La mayoría de los dispositivos SPI pueden soportar al menos 2 modos de los 4 antes mencionados. La polaridad del reloj determina el nivel del estado inactivo del reloj y la fase del reloj determina en qué flanco se lee el dato del bus.

La tabla 2.3 muestra la combinación de la polaridad y la fase del reloj para los cuatro modos y el flanco en el que se establecen los bits en el bus y en el que se muestrean:

Modo	Condición	Primer flanco	Siguiente flanco
Modo A	CPOL=0, CPHA=0	Muestreo (Subida)	Establece (Bajada)
Modo B	CPOL=0, CPHA=1	Establece (Subida)	Muestreo (Bajada)
Modo C	CPOL=1, CPHA=0	Muestreo (Bajada)	Establece (Subida)
Modo D	CPOL=1, CPHA=1	Establece (Bajada)	Muestreo (Subida)

Tabla 2.3. Combinaciones para los cuatro modos SPI.

2.4. Memorias de estado sólido

Una memoria de estado sólido es un dispositivo electrónico de almacenamiento de datos, pensada para usarse en equipos informáticos como apoyo o sustitución del disco duro convencional, el cual tiene partes móviles. Las memorias de estado sólido pueden estar constituidas por unidades volátiles, como la memoria SDRAM (Synchronous Dynamic Random Access Memory), o por unidades no volátiles, como la memoria flash o ROM (Read Only Memory).

Las memorias CMOS (complementary Metal-Oxide Semiconductor) pueden dividirse en dos categorías principales:

- Memoria de acceso aleatorio (RAM: Random Access Memory) es del tipo volátil, es decir, pierde la información cuando su fuente de alimentación es apagada.
- Memorias de sólo lectura (ROM), son del tipo no volátil, conserva la información aun cuando no están conectadas a la fuente de alimentación. Las memorias EEPROM (Electrically-Erasable Programmable Read-Only Memory) son un tipo de memoria ROM, que puede ser programada, borrada y reprogramada eléctricamente de 100,000 a 1,000,000 de veces.

La memoria Flash es un tipo de memoria basada en celdas microscópicas de semiconductores del tipo no volátil que al igual que las memorias EEPROM se programan, borran y reprograman eléctricamente, por esta razón en sus comienzos se le llamaba flash EEPROM, pero para evitar confusiones se le llama memoria flash. Sus datos no se eliminan al quitar la fuente de energía eléctrica. Permiten la lectura y escritura de múltiples posiciones de memoria en la misma operación, ofreciendo la posibilidad de trabajar con sectores, en vez de un solo bit (como es el caso de las memorias EEPROM), lo cual permite alcanzar velocidades de funcionamiento muy superiores frente a la tecnología EEPROM. Debido a su alta velocidad, durabilidad y bajo consumo de energía, la memoria flash resulta ideal para aplicaciones donde se requiere almacenamiento de grandes cantidades de información, a un costo razonable y donde el tamaño es un factor crítico, como es el caso de aplicaciones portátiles. Además, este tipo de memorias no tiene partes móviles, lo que la hace más resistente a posibles golpes, evitando la pérdida de información a causa de ellos. Entre las principales ventajas de las memorias flash se encuentran las siguientes:

- Gran resistencia a golpes
- Muy silenciosas
- Bajo consumo de energía
- Tamaño pequeño
- Elevada resistencia térmica

Hay dos tipos de clasificación de las memorias flash: según su formato físico y utilización, tarjetas de memoria y pendrives; y según su funcionamiento interno, NAND y NOR.

Según el formato físico

Dentro de la clasificación de las memorias flash según su formato físico están las *tarjetas de memoria*, las cuales son dispositivos pensados para estar dentro de otro dispositivo electrónico (cámaras, pdas, celular, etc.). Algunos de los tipos de memoria flash que podemos encontrar son, Compact flash, Multimedia Card o MMC, tarjeta SD, XD y Memory stick.

También con respecto al formato físico encontramos a los *"Pen drive",* éstos son dispositivos pensados para ser usados como discos duros portátiles. Llevan una memoria flash en su interior y un conector usb que nos permite conectarlos directamente a una computadora. Son de tamaño reducido lo que permite transportarlos sin problemas, siendo también bastante resistentes a caídas y golpes. La ventaja de estos dispositivos es que se conectan directamente a una computadora y no requieren instalación de software adicional.

Una de las formas más comunes en la que podemos encontrar memorias flash, de acuerdo a la clasificación del formato físico, es en las tarjetas de memoria. En la tabla 2.4 se muestran los formatos más comunes de tarjetas de memoria flash que existen, con sus capacidades máximas de almacenamiento.

Formato	Sigla	Capacidad
PC Card	PCMCIA	8 GB
Compact Flash I	CF-I	12 GB
Compact Flash II	CF-II	48 GB
Smart Media	SM/SMC	128 MB
Memory Stick	MS	128 MB
Memory Stick Duo	MSD	256 MB
Memory Stick PRO Duo	MSPD	32 GB
Memory Stick PRO-HG Duo	MSPDX	32 GB
Memory Stick Micro M2	M2	32 GB
MultiMedia Card	MMC	4 GB
Reduce Size Multimedia Card	RE-MMC	2 GB
MMC micro card	MMC micro	2 GB
Secure Digital Card	SD	Hasta 2 GB
miniSD card	miniSD	Hasta 2 GB
microSD card	microSD	Hasta 2 GB
SD card High Capacity (mini, micro)	SDHC	4GB-32 GB
SD card Extended Capacity (mini, micro)	SDXC	32GB-2TB

Tabla 2.4. Tarjetas y capacidades de memoria flash.

Según su funcionamiento interno

Según su funcionamiento interno, las memorias flash están fabricadas con compuertas lógicas NOR y NAND para almacenar los 0's ó 1's correspondientes.

El tipo NOR permite una lectura y escritura más lenta que NAND, pero archiva muy rápido las rutas de acceso aleatorias. Esto hace que NOR sea más adecuado para la ejecución y almacenamiento de comandos, mientras que NAND es más indicado para el almacenamiento masivo de datos.

En cuanto a la arquitectura, NAND puede almacenar más datos en un espacio de silicio más pequeño, lo que ahorra el coste por bit. En el pasado, cuando el almacenamiento de datos era más bajo, NOR tuvo mayor influencia en el mercado. Hoy, con el gran incremento de la necesidad de guardar más datos, el consumo de la electrónica y el negocio de los dispositivos, NAND ha superado por mucho a NOR.

2.4.1. La tarjeta SD

La tarjeta SD (Secure Digital card) es uno de los formatos de tarjeta de memoria flash más populares y ampliamente usado. El formato SD fue creado en colaboración por Panasonic Corporation, SanDisk Corporation y Toshiba Corporation. Estas empresas conforman actualmente la asociación SD (SD Association).

Las especificaciones técnicas describen la interfaz física y el protocolo de comandos usados por las tarjetas de memoria SD *Card*[3]. Su propósito es definir a las tarjetas de memoria SD en cuanto a su estructura y manipulación.

La tarjeta SD tiene diferentes tamaños, cada uno de los tamaños incluye a todas las capacidades de almacenamiento de la SD, estos tamaños son: Tarjeta de memoria SD, tarjeta de memoria miniSD y tarjeta de memoria microSD. Las capacidades de almacenamiento de las tarjetas de memoria SD son:

- **Capacidad de almacenamiento**
  - SDSC (SD Standar Capacity), hasta 2 GB
  - SDHC (SD High Capacity), de 4 GB a 32 GB
  - SDXC (SD Extended Capacity), arriba de 32 GB hasta 2 TB

- **Velocidad**

Están definidas cinco clases de velocidad e indican el desempeño mínimo de velocidad de las tarjetas

  - Clase 0 – Esta clase de tarjetas no especifica la velocidad. Incluye a todo el legado de tarjetas anterior a esta clasificación independientemente de su velocidad.
  - Clase 2, tiene velocidad mayor o igual a 2 MB/s.
  - Clase 4, tiene velocidad mayor o igual a 4 MB/s.
  - Clase 6, tiene velocidad mayor o igual a 6 MB/s.
  - Clase 10, tiene velocidad mayor o igual a 10 MB/s.

[3] SD Specifications part 1, *Physical Layer Simplified Specification*. Version 4.10, January 22, 2013. Recuperado Agosto de 2014 de https://www.sdcard.org/downloads/pls/simplified_specs/archive

Protocolos de comunicación con la tarjeta SD

Las tarjetas de memoria SD actuales soportan dos protocolos de comunicación: SD y SPI, la elección del protocolo depende de la aplicación y del dispositivo. El protocolo es seleccionado por el dispositivo cuando éste inicia la comunicación al enviar un comando de *reset*, a partir de ese momento la tarjeta trabajará con tal protocolo. El uso del protocolo SD requiere de una licencia, lo que implica un costo, por otra parte, el protocolo SPI puede ser implementado, sin costo, con la mayoría de microcontroladores. La tabla 2.5 muestra las terminales de la tarjeta utilizadas con cada protocolo.

Terminal	SD	SPI
1	DAT2	N/C
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	N/C

Tabla 2.5. Terminales de la tarjeta micro SD.

Protocolo SD

El protocolo SD es nativo, esto significa que fue desarrollado para este tipo de tarjetas y por default trabaja con este modo. Este protocolo se basa en la transmisión de bits de comandos y bits de datos, inicia la comunicación con un bit de inicio y la termina con un bit de paro; se diseñó para trabajar a una frecuencia máxima de 50 MHz a través de 8 terminales. La figura 2.17 muestra una conexión de una tarjeta con un dispositivo en modo SD.

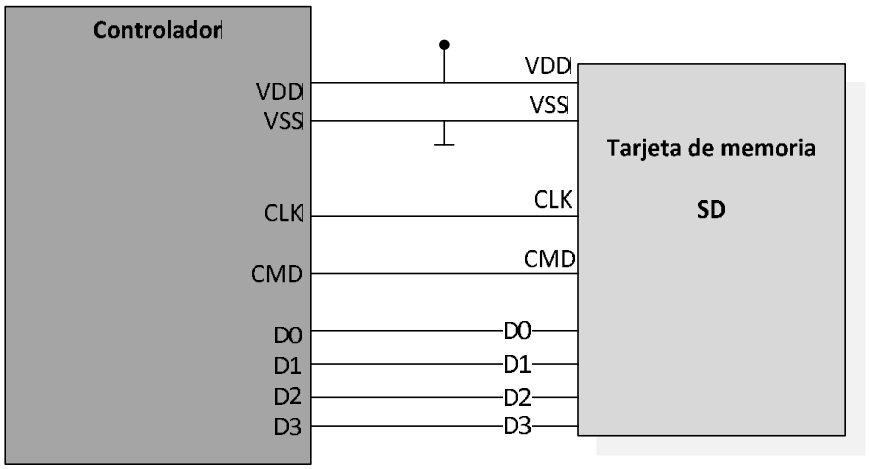


Figura 2.17. Modo SD.

Las terminales que se usan en este protocolo se describen a continuación:

- **CMD** es la línea de comandos. Un comando indica que va a iniciar una operación, éste se envía dese *host* a una sola tarjeta (por direccionamiento) o a todas las tarjetas conectadas (por difusión). Por la línea CMD también se envía la respuesta de la tarjeta o todas las tarjetas hacia el *host* como resultado de un comando previamente enviado.
- **DAT0…3** son las cuatro líneas de datos. Los datos pueden transmitirse desde la tarjeta al *host* y viceversa.
- **CLK** es la línea de la señal de reloj.
- **VSS, VDD** son las líneas de alimentación de la tarjeta.

Protocolo SPI

El modo SPI es el protocolo de comunicación secundario soportado por las memorias SD. Este modo es una división del protocolo de las tarjetas de memoria SD, diseñado para comunicarse a través del canal SPI, encontrado en la mayoría de los microcontroladores del mercado.

La interfaz o modo es seleccionado durante el primer comando de reset (CMD0), después de ser energizada la tarjeta. En el estándar SPI se define, únicamente, la conexión física; la implementación del bus SPI en las tarjetas de memoria SD utiliza sólo una parte del protocolo y de la serie de comandos; cabe comentar que se utilizan solamente 7 de las 9 señales del bus SD. En la figura 2.18 se muestra dicha conexión.

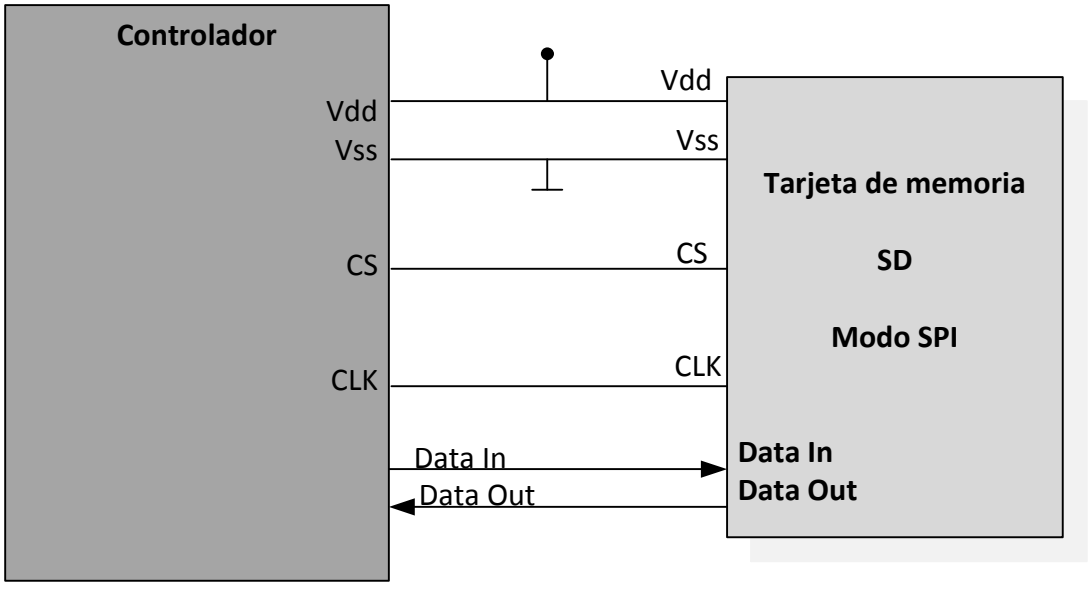


Figura 2.18. Comunicación con SPI.

Con el protocolo SPI se usan dos terminales menos que con el protocolo SD, las terminales usadas son:

- CS (Chip Select): Línea de selección de dispositivo.
- CLK: Señal de reloj.
- Data In: Datos hacia la tarjeta de memoria SD.
- Data Out: Datos desde la tarjeta de memoria SD.

La ventaja de usar este protocolo con la tarjeta SD es que casi cualquier microcontrolador de propósito general cuenta con SPI, y que no necesita de licencia para su uso, como es el caso del protocolo SD. La desventaja, como puede deducirse al observar la figura, es que tiene una menor velocidad de comunicación debido a que usa una línea.

2.4.2. El protocolo SPI para la tarjeta SD

A diferencia del modo SD, el cual se basa en comandos y flujo de bits de datos que comienzan con un bit de inicio y terminan con un bit de paro, el modo SPI está orientado a bytes. Cada comando o bloque de datos está compuesto de bytes, los bytes están alineados con la señal CS (la longitud es múltiplo de ocho ciclos de reloj). La tarjeta empieza a contar los ciclos de reloj del bus SPI al activarse la señal de CS. Algunas de las características de SPI con SD son:

- Los mensajes consisten de comandos, respuestas y transferencia de datos.
- Toda comunicación entre el host y la tarjeta está controlado por el host (maestro); el maestro inicia cada actividad en el bus con la señal de CS.
- La tarjeta seleccionada siempre responde a los comandos.
- Cuando la tarjeta encuentra problemas en recuperar los datos en la operación de lectura, enviará una respuesta de error.
- En el caso de las tarjetas de capacidad estándar (SDSC), un bloque de datos debe ser tan grande como un bloque de escritura (512 bytes) de la tarjeta y tan pequeño como un solo byte.
- Los comandos de protección contra escritura no son soportados.
- El código de detección de errores CRC (*Cyclic Redundancy Code*) está deshabilitado y es ignorado por el *host.*
- El *host* debe tratar a todas las tarjetas como si fueran de clase 0, en cuanto a velocidad.

Selección de modo e inicialización

Cada vez que la tarjeta SD se energiza entra en modo SD, para que entre en modo SPI la señal de CS debe estar activa durante la recepción del comando de *reset* (CMD0), si la tarjeta se puso en modo SPI envía como respuesta R1. En la figura 2.19 se observa el diagrama de flujo de la secuencia de inicio en modo SPI.

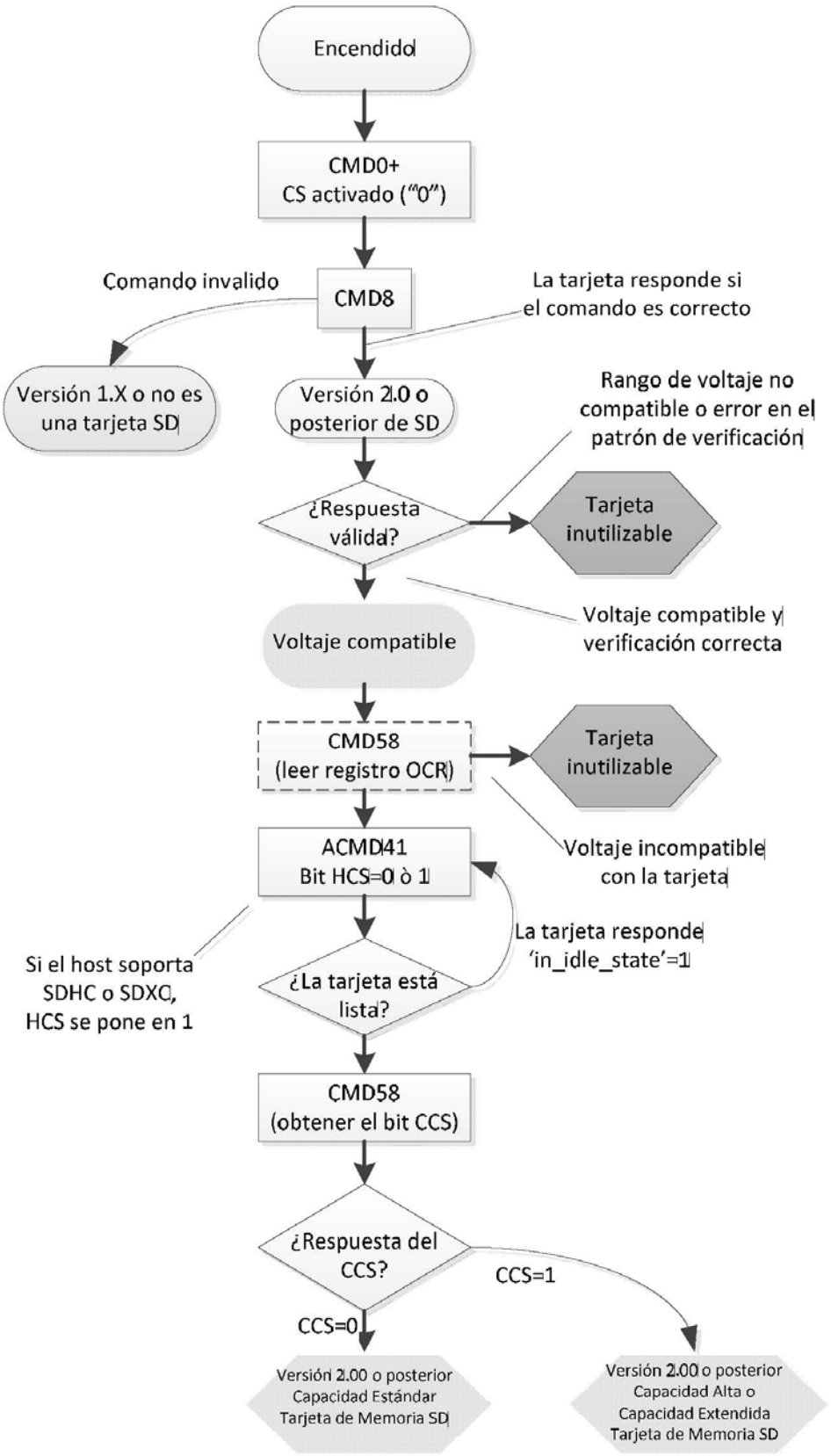


Figura 2.19. Flujo de la secuencia de inicio en modo SPI.

Según el diagrama de flujo proporcionado por las especificaciones reducidas, la inicialización de la tarjeta se lleva a cabo enviado la secuencia de comandos indicada en el diagrama de flujo anterior como “CMD” y el número representa el número del comando. La tarjeta analizará el argumento de los comandos y responderá según éstos. Al finalizar el proceso, el *host* tendrá información suficiente para saber la versión de la tarjeta y si puede o no trabajar con ella. A continuación se explican las funciones realizadas por los comandos:

1. El comando CMD8 (*Envía las condiciones de operación de la interfaz*), es utilizado para verificar las condiciones de operación de la conexión establecida entre la tarjeta de memoria SD y el *host*. La tarjeta comprueba la validez de estas condiciones analizando el argumento del CMD8, a su vez, el *host* lo hace analizando la respuesta R7, en la que se envía el voltaje aceptado por la tarjeta.

2. El comando CMD58 (*Lee OCR:Operation Condition Register*), es utilizado por el *host* como un mecanismo para identificar a las tarjetas que no pueden trabajar con el margen de voltaje VDD suministrado. Si existe incompatibilidad de voltajes no se continuará con el proceso de inicialización. La respuesta asociada a este comando es R3.

3. El comando ACMD41 (*Envío de condiciones de operación*), es utilizado para comenzar la secuencia de inicialización y para verificar si la tarjeta ha completado ésta. Es obligatorio que el *host* envíe el CMD8 antes de enviar por primera vez el ACMD41, de esta manera se expanden las funciones de los comandos CMD58 y ACMD41; se podrá utilizar el campo HCS (*High Capacity Suport*) en el argumento del ACMD41 y el CCS (*Card Capacity Status*) en la respuesta al CMD58.

4. El bit “*in_idle_state*” en la respuesta R1 al ACMD41 es utilizado por la tarjeta para informar al *host* si el procedimiento de inicialización ha terminado; el tener dicho bit un valor de “1” indica que la tarjeta SD aún se encuentra en el ciclo de inicialización, y entregando un valor de “0” significará que se ha completado la inicialización.

Si se empleó la secuencia de inicialización correctamente, el *host* deberá obtener la información del campo CCS, en la que; **CCS = 1** significa que la tarjeta recién inicializada es de Alta Capacidad (*High Capacity SD*) y **CCS = 0** significa que se trata de una tarjeta de Capacidad Estándar (*Standard Capacity SD*).

En la tabla 2.6 se despliega el significado de los campos del registro de condición de operación OCR de la tarjeta de memoria SD, el cual es enviado como respuesta al comando CMD58. Las condiciones que nos permite saber este registro son: el rango de voltaje soportado por la tarjeta, si la tarjeta es o no de alta capacidad y, con el bit *Card Power up Status* se puede saber si la tarjeta a finalizado el ciclo de encendido.

Posición de Bits	Definición del campo OCR
0 – 6	Reservado
7	Reservado para bajo voltaje
8 – 14	Reservado
15	2.7 – 2.8
16	2.8 – 2.9
17	2.9 – 3.0
18	3.0 – 3.1
19	3.1 – 3.2
20	3.2 – 3.3
21	3.3 – 3.4
22	3.4 – 3.5
23	3.5 – 3.6
24 – 29	Reservado
30	CCS (*Card Capacity Status*)
31	Card power up status bit (busy)

Tabla 2.6. Campos del registro OCR.

Operaciones de lectura

Con el modo SPI se pueden realizar operaciones de lectura; tanto de uno o de múltiples bloques, con los comandos CMD17 (lectura de un bloque) y con el CMD18 (lectura de bloques múltiples). Después de recibir el comando correctamente, la tarjeta SD manda una respuesta seguida por un bloque de datos con una longitud de 512 bytes. En la figura 2.20 se muestra la operación de lectura por medio de un diagrama.

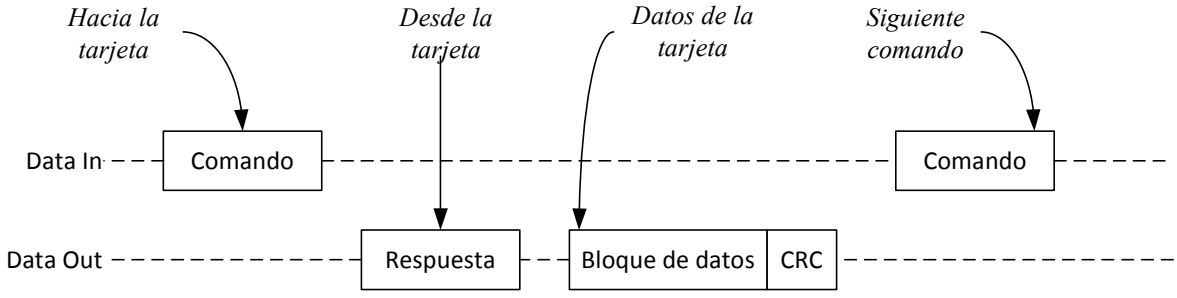


Figura 2.20. Diagrama del proceso de lectura de datos.

Después de los bloques de datos que vienen de la tarjeta sigue un bloque para detección de errores de tipo CRC de 16 bits. Generalmente con el modo SPI se ignoran los bits del CRC, pero con el primer comando que se envía se debe de respetar ese CRC, es así porque la tarjeta siempre inicia en modo SD y este modo no debe ignorar el CRC.

Ahora, si la lectura es de múltiples bloques, cada uno será enviado por la tarjeta SD con su bloque CRC correspondiente. Si ocurre un error de lectura, la tarjeta enviará al *host* la palabra de error en lugar de datos y esperará el envío de un nuevo comando.

La operación de lectura de múltiples bloques de datos, que se muestra en la figura 2.21, se realiza igual que la de lectura de un solo bloque, pero la tarjeta responde al comando enviando continuamente los bloques, hasta que reciba el comando de *fin de transmisión* (CMD12) para detener el envío de bloques y enviar la respuesta correspondiente.

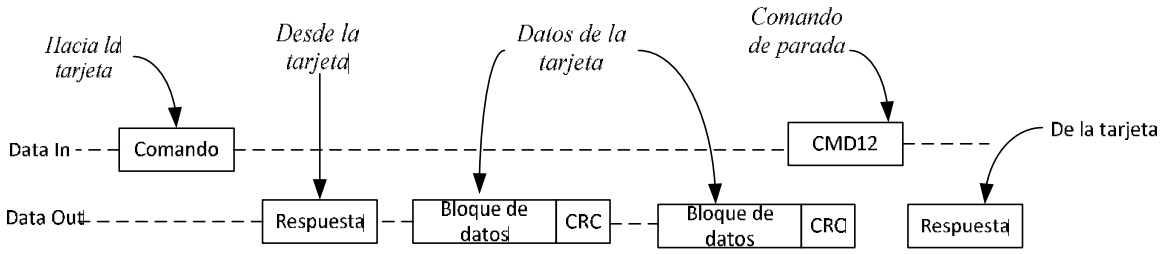


Figura 2.21. Diagrama del proceso de lectura de múltiples datos.

Operaciones de Escritura

Con el modo SPI también se pueden realizar operaciones de escritura de uno y múltiples bloques. El comando CMD24 sirve para escribir un solo bloque y el comando CMD25 para escribir múltiples bloques. El procedimiento para la escritura de un bloque es el siguiente: después de recibir un comando de escritura, la tarjeta envía la respuesta correspondiente y espera el bloque de datos que enviará el *host*. Cada bloque de datos que manda el *host* lleva un byte de inicio de bloque (Start Block Token). Cuando la tarjeta ha recibido el bloque de datos envía la respuesta de datos (Data_Response). Mientras el bloque de datos se está guardando en la tarjeta SD ésta enviará la señal de ocupado (busy). La figura 2.22 muestra la operación de escritura definida para un bloque de datos.

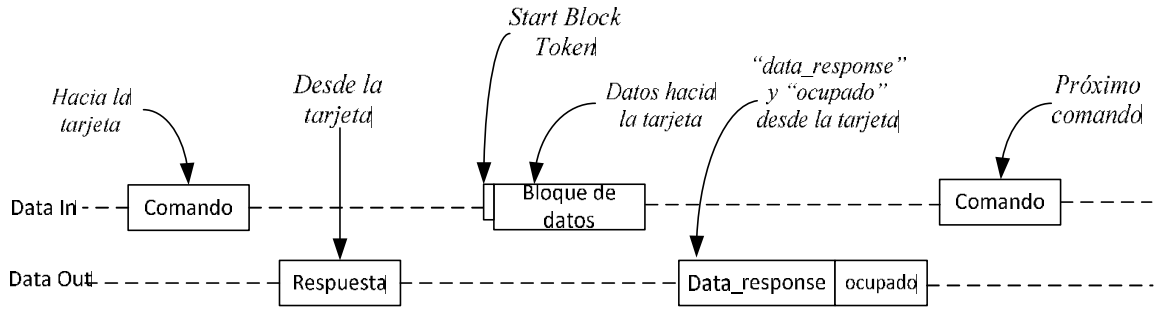


Figura 2.22. Diagrama del proceso de escritura en la SD.

Similar a la lectura de varios bloques, se debe indicar el fin de envío de bloques de datos que se guardaran en la tarjeta SD, en este caso el fin se señala enviando un byte de fin de transmisión, llamado *"stop tran"*, después del último bloque transmitido. La figura 2.23 muestra el proceso de escritura de múltiples bloques.

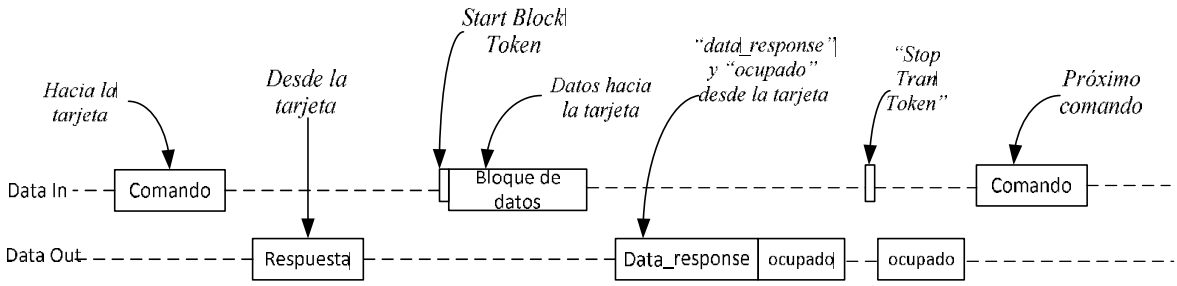


Figura 2.23. Diagrama del proceso de escritura de múltiples datos.

La tarjeta SD siempre debe responder a cada comando que reciba con el protocolo SPI, en esa respuesta indicará si se aceptó o rechazó el comando, donde las causas de rechazo son:

- Si se envía el comando durante la operación de lectura
- Si la tarjeta se encuentra en estado ocupado o *"busy"*
- Si la tarjeta no es SD
- Si hay un error de CRC
- Si el argumento del comando es ilegal

En caso de que ocurra alguna de las causas mencionadas podría haber pérdida de información.

Formato de los comandos

En el modo SPI, todos los comandos de la tarjeta de memoria SD son de 6 bytes de longitud. Los comandos se transmiten con su bit más significativo primero (MSB: Most Significant Bit). En la tabla 2.7 se enlistan los comandos y sus argumentos.

Posición del bit	47	46	[45:40]	[39:8]	[7:1]	0
Ancho (bits)	1	1	6	32	7	1
Valor	'0'	'1'	X	X	x	'1'
Descripción	Bit de inicio	Bit de transmisión	Índice de comando	argumento	CRC7	Bit de fin

Tabla 2.7. Formato de los comandos en SPI.

Formato de las respuestas

También las respuestas de la tarjeta SD a los comandos tienen un formato definido, son varios tipos de respuesta, éstas también se envían con su MSB primero. A continuación se exponen las características principales y el formato de las respuestas:

Respuesta R1.- Esta respuesta es enviada por la tarjeta después de recibir cada comando, con excepción al comando CMD13 (send_status). Tiene una longitud de un byte y su MSB siempre vale cero, los demás bits indican condiciones de error dependiendo en qué bit se presente un '1'. Su formato se muestra en la figura 2.24.

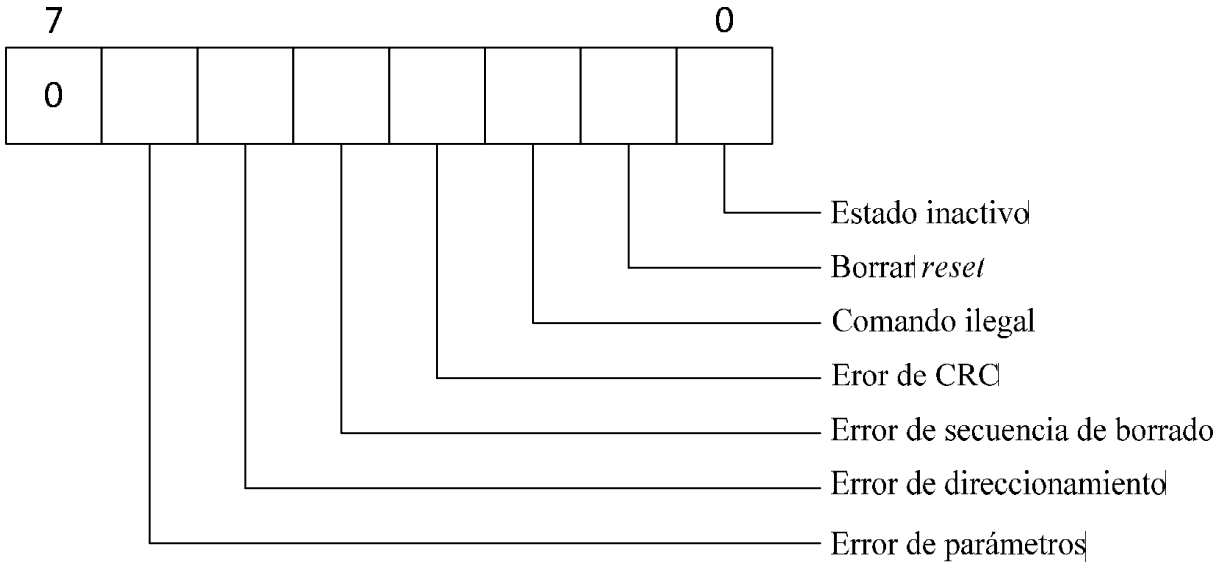


Figura 2.24. Formato de la respuesta R1.

El significado de cada uno de los bits que forman a la respuesta **R1** es:

- **Estado inactivo:** indica que la tarjeta está inactiva o está ejecutando la secuencia de inicio.
- **Borrar *reset:*** la secuencia de borrado fue cancelada por recibir un comando de *reset*.
- **Comando ilegal:** se detectó un comando no válido.
- **Error de CRC:** falló la verificación de los bits de CRC del último comando.
- **Error de secuencia de borrado:** no se ha completado la secuencia de borrado.
- **Error de direccionamiento:** el tamaño de la dirección especificada no coincide con el tamaño de los bloques.
- **Error de parámetro:** El argumento del comando (por ejemplo, dirección, longitud del bloque) está fuera de rango permitido para esta tarjeta.

Respuesta R2

Esta respuesta es de 2 bytes de longitud, se envía como respuesta al comando *SEND_STATUS* (CMD13), su formato se ve en la figura 2.25.

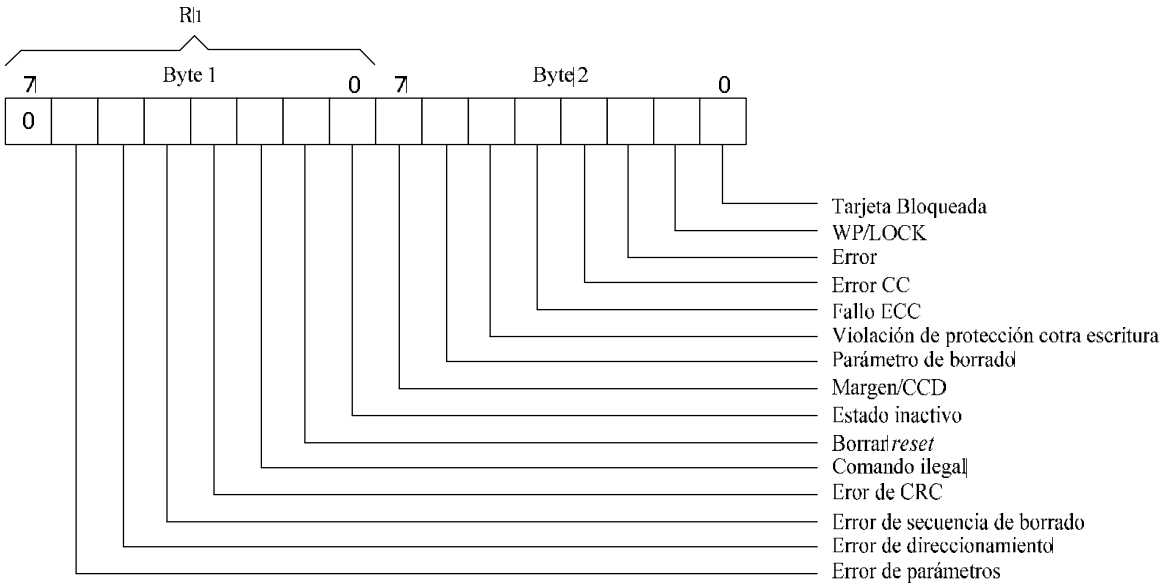


Figura 2.25. Formato de la respuesta R2.

Como se indica en la figura, el primer byte corresponde a la respuesta R1. A continuación se muestra el significado de los bits del byte 2:

- **Parámetro de borrado:** Selección inválida de sectores o grupos para borrar.
- **Violación de protección de lectura:** El comando trató de escribir en un bloque protegido contra escritura.
- **Falló ECC (Error Correction Code):** Se aplicó ECC pero falló la corrección de los errores de los datos.
- **Error CC (Card Controller):** Error en el controlador interno de la tarjeta.
- **Error:** Ocurrió un error general o desconocido durante la operación.
- **WP/Lock:** Este bit de estado tiene dos funciones: está activo cuando el *host* intenta borrar un sector protegido contra escritura o ejecuta una secuencia u ocurren errores en la contraseña durante la operación bloqueo/desbloqueo de la tarjeta.
- **Tarjeta bloqueada:** Este bit está activo cuando la tarjeta ha sido bloqueada por el usuario, se desactiva cuando se desbloquea (protección contra escritura).

Respuesta R3

Es la respuesta al comando de lectura del OCR (CMD58), tiene una longitud de 5 bytes. La estructura del primer byte (MSB) es idéntico a la respuesta R1, los 4 bytes restantes contienen al registro OCR, ver figura 2.26.

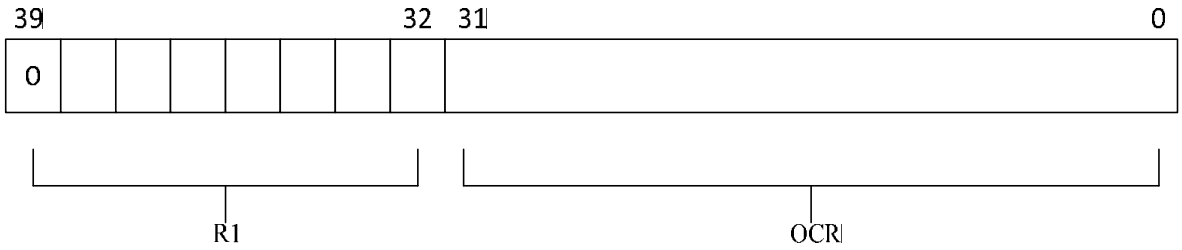


Figura 2.26. Formato de la respuesta R3.

Respuesta R7

La tarjeta envía la respuesta R7 cuando recibe correctamente el comando CMD8 *SEND_IF_CONDITION* (Enviar Condiciones de Interfaz). Esta respuesta consta de 5 bytes, en la que la estructura del primero es idéntica a la respuesta R1. Los otros cuatro bytes contienen información del voltaje de operación de la tarjeta y regresa el patrón de verificación que le envío el *host* en el argumento del comando mencionado. La figura 2.27 muestra el formato de esta respuesta.

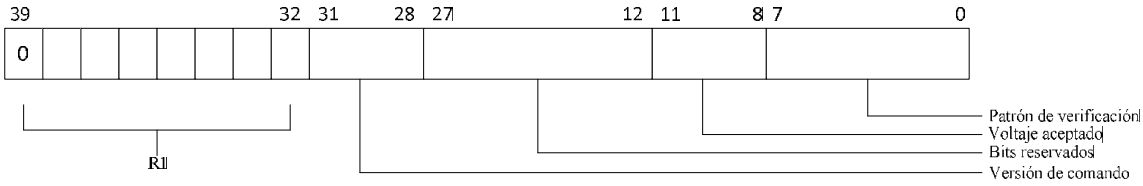


Figura 2.27. Formato de la respuesta R7.

Además de las respuestas mencionadas anteriormente, están definidas señales adicionales para cada intervención entre la tarjeta y el *host*, descritas a continuación.

Respuesta *'Respuesta_a_datos'*

Por cada bloque escrito, la tarjeta enviará a manera de acuse de recibido, la señal *Respuesta_a_datos,* que consta de 1 byte. La estructura del byte se muestra en la figura 2.28.

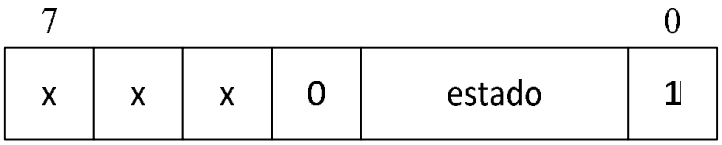


Figura 2.28. Formato de la *Respuesta_a_datos.*

Los valores que puede tomar el campo "estado" son:

'010' – los datos fueron aceptados.
'101' – los datos fueron rechazados debido a un error de CRC.
'110' – los datos fueron rechazados debido a un error de escritura.

Esta señal de respuesta le sirve al *host* para determinar la posible causa del error cuando éste ocurre.

Señales *Inicio de Bloque* y *Finalizar Transmisión*

En las operaciones de lectura y de escritura existen comandos que están asociados a ciertas señales; los datos son transmitidos o recibidos por la tarjeta a través de las llamadas "*señales de datos"*. Estas señales tienen una longitud que va desde los 4 hasta los 515 bytes, el formato general se muestra en la figura 2.29.

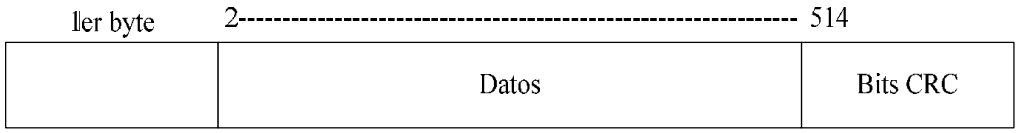


Figura 2.29. Formato de los bloques de datos.

Para operaciones de lectura y escritura de un solo bloque y lectura de varios bloques se tiene que:

- El primer byte es el bloque de inicio (start_block token) y tiene el formato mostrado en la figura 2.30.

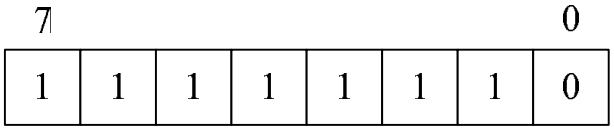


Figura 2.30. Bloque de inicio, lectura y escritura individual y lectura múltiple.

- Los bytes 2 a 513 corresponden a los datos que se leen o escriben.

- Los dos últimos bytes son los bits del CRC, que como se mencionó anteriormente se ignoran en el modo SPI.

Para operaciones de escritura de múltiples bloques:

- El primer byte de cada bloque:
  - Si los datos se van a transmitir, se envía la señal de inicio de bloque (Start_block token), mostrada en la figura 2.31.

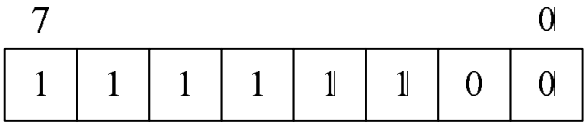


Figura 2.31. Bloque de inicio para escritura múltiple.

- Si se quiere terminar la transmisión de los bloques, se envía la señal *stop_tran_token,* que se observa en la figura 2.32.

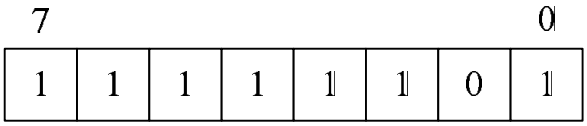


Figura 2.32. Bloque de paro de escritura múltiple (Stop_tran_token).

Este formato se usa únicamente para detener la escritura de múltiples bloques. En el caso de lectura de múltiples bloques la transmisión de paro se lleva a cabo usando el comando *STOP_TRAN* (CMD12).

Señal de error de datos

Cuando la operación de lectura falla y la tarjeta no puede transmitir la información requerida, en lugar de datos se debe enviar una señal de error de datos. Esta señal de error es de un byte de longitud, en los que los últimos 4 bits (LSB) son los bits de error de la respuesta R2, los cuales se ven en la figura 2.33.

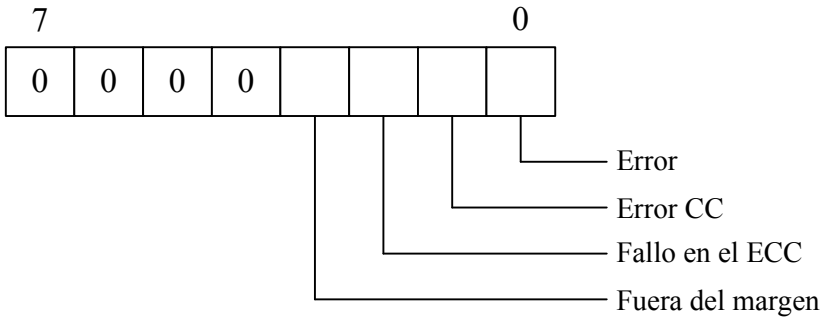


Figura 2.33. Bloque de la señal de "error de datos".

2.5. Sistema de archivos

Los dispositivos de almacenamiento masivo, incluyendo a las tarjetas de memoria flash, requieren que los datos guardados en ellos estén organizados para poder encontrarlos y acceder a ellos de manera eficiente. En una computadora personal el Sistema Operativo (SO) tiene como una de sus funciones principales controlar el acceso a los archivos. Para esto, debe conocer la naturaleza del dispositivo que alberga la información (*hardware* de almacenamiento) y el sistema de archivos con el que cumplen los datos registrados. El SO sirve como interfaz entre las aplicaciones utilizadas por el usuario y el *hardware* del equipo de cómputo.

Alguna definición de sistema de archivos podría ser que: "un *sistema de archivos* son los métodos y estructuras de datos que un sistema operativo utiliza para seguir la pista de los archivos de un disco o partición; es decir, es la manera en la que se organizan los archivos en el disco", por lo que se puede pensar en un sistema de archivos como una base de datos de propósito específico.

Se han desarrollado muchos sistemas de archivos, la mayoría de los sistemas operativos utiliza su propio sistema de archivos y permiten compatibilidad con otros. Algunos ejemplos de sistemas operativos y los sistemas de archivos soportados se presentan en la tabla 2.8. De los sistemas operativos presentados en dicha tabla, y que podemos encontrar en la gran mayoría de las computadoras personales actuales, *Windows* de *Microsoft* es de los más utilizados a nivel comercial. Este SO utiliza el sistema de archivos FAT (**File Allocation Table**) para el manejo de la información almacenada o por almacenar.

Sistema operativo	Sistema de archivos
Linux	Ext, Ext2, Ext3, Ext4: Extended file system
Linux	XFS: X file system
Mac OS	HFS, HFS+: Hierarquical File System
OS/2	HPFS: High Performance File System
Windows	FAT12, FAT16, FAT32: File Allocation Table
Windows	NTFS: New Technology File System

Tabla 2.8. Sistemas operativos y sus sistemas de archivos.

2.5.1. Sistema de archivos FAT

El sistema de archivos FAT tiene sus orígenes entre finales de 1970 y principios de 1980, y fue el sistema de archivos soportado por el sistema operativo *Microsoft MS-DOS*. Originalmente fue desarrollado como un sistema de archivos simple, adecuado para las unidades de disco flexible con capacidad menor a 500 kB. FAT es un sistema de archivos relativamente simple y poco sofisticado que es entendido por casi todos los sistemas operativos, incluyendo Linux y MacOS, así que por lo general es una opción común para proyectos basados en *firmware* que necesitan acceder a discos duros.

FAT con el tiempo fue mejorado para soportar volúmenes de mayor capacidad. Existen tres tipos de sistemas de archivos FAT: FAT12, FAT16 y FAT32. La diferencia básica en estos subtipos de FAT, y el motivo de los nombres, es el tamaño en bits de las entradas en la región FAT en el disco; hay 12 bits en una entrada en FAT12, 16 bits en una entrada en FAT16 y 32 bits en una entrada en FAT32. Los sistemas de archivos FAT de mayor uso en la actualidad son FAT16 y FAT32; en el presente trabajo únicamente se hablará de los sistemas FAT16 y FAT32.

Un aspecto importante que se debe considerar al implementar este sistema de archivos es que, el espacio para datos de una unidad de almacenamiento se divide en pequeñas unidades de 512 bytes (tamaño mínimo), llamadas sectores, esto es a nivel físico. Sin embargo, a nivel lógico el sistema de archivos divide el medio de almacenamiento en unidades llamadas *clúster*; un *clúster* agrupa una cierta cantidad de sectores, determinada de forma automática por el SO. El SO toma en cuenta la capacidad de almacenamiento del dispositivo en cuestión, por lo que, cuanto más grande sea la unidad de almacenamiento mayor será el tamaño del clúster. La tabla 2.9 muestra el tamaño del clúster dependiendo de la capacidad de almacenamiento.

Capacidad de almacenamiento	Tamaño del clúster
34 MB, 64 MB	512 bytes
64 MB, 128 MB	1 kB
128 MB – 256 MB	2 kB
256 MB – 8 GB	4 kB
8 GB, 16 GB	8 kB
16 GB, 32 GB	16 kB
32 GB hasta 2 TB	No se admite
> 2 TB	No se admite

Tabla 2.9. Tamaños predeterminados de clúster para FAT32.

Para el sistema de archivos el clúster es la unidad mínima de almacenamiento que se puede asignar a un archivo y direccionar en la tabla FAT. El número de sectores por clúster es potencia de 2 (1, 2, 4, 8, 16, 32 en FAT32).

El sistema de archivos FAT32 está compuesto por tres regiones, que se encuentran en el siguiente orden dentro del medio de almacenamiento:

0. Región reservada
1. Región FAT
2. Región de datos: directorios y archivos

El primer sector físico de la unidad de almacenamiento es el llamado Sector Maestro de Arranque o MBR (*Master Boot Record*). En el caso de una PC los primeros 446 bytes del MBR contienen código que sirve para *arrancarla*. Después

se encuentran 64 bytes de una tabla de particiones, que es una serie de 4 registros de 16 bytes cada uno, conteniendo información sobre la división lógica de la memoria, es decir, información de las particiones, y los últimos 2 bytes siempre son 0x55 y 0xAA, ver figura 2.34. Después del MBR se encuentra la primera partición, que contendrá la estructura del sistema de archivos.

Una partición es una división del disco o memoria, no es una división física sino "imaginaria", o mejor dicho una división lógica, la cual tiene un sistema de archivos propio.

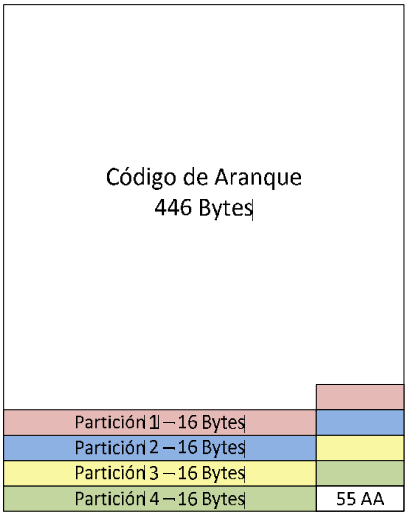


Figura 2.34. Disposición del sector MBR.

El MBR sólo puede representar cuatro particiones. Se utiliza una técnica llamada partición "extendida" para permitir más de cuatro particiones.

Cada descripción de partición es de 16 bytes, figura 2.35, pero generalmente se ignoran la mayoría de ellos, por lo que sólo mencionaré los que se toman en cuenta siempre. El quinto byte (azul) es un "código de tipo" que dice cuál es el sistema de archivos que se supone debería tener la partición (tabla 2.8), y del noveno hasta el decimosegundo byte (verde) indican al "*LBA begin*" (Logical Block Addresing), que es la dirección donde esa partición comienza en el disco.

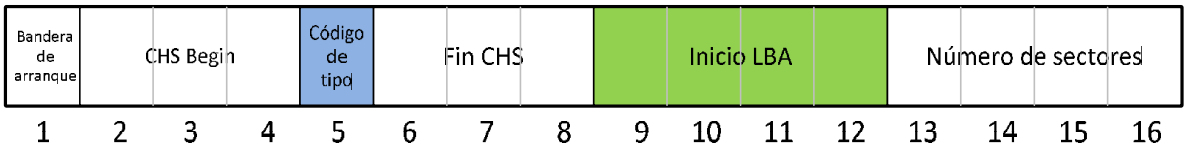


Figura 2.35. Entrada de particiones de 16 bytes.

Normalmente lo único que se necesita hacer para identificar y posteriormente usar el sistema de archivos es verificar “código de tipo” de cada entrada, observando, ya sea el valor 0x0b o 0x0c (los dos son usados por FAT32), y entonces se lee “inicio LBA” para saber dónde se localiza el sistema de archivos FAT32 en el disco.

El campo “Número de sectores” se puede verificar para asegurarse de que no se acceda (particularmente al escribir) más allá del final del espacio que es asignado para la partición. Sin embargo, el sistema de archivos FAT32 en sí mismo contiene información acerca de su tamaño, por lo que este campo “Número de sectores” es redundante. Varios de los sistemas operativos de Microsoft lo ignoran y en su lugar se basan en la información de tamaño incluido dentro del primer sector del sistema de archivos. Linux verifica el campo “Número de sectores” y previene adecuadamente el acceso más allá del espacio asignado. La mayoría del *firmware* lo ignorará, así como los demás campos deben ser ignorados.

0. Región reservada

El primer sector en la región reservada (***R0***) es el llamado *Boot Sector* o sector de arranque, en éste se encuentra una estructura llamada Bloque de Parámetros del BIOS (*Basic Input/Output System*) o BPB (*BIOS Parameter Block*). El BPB contiene la información sobre el sistema de archivos, que nos servirá para tener acceso al medio de almacenamiento o volumen, es decir, para poder leer y escribir en él.

En la tabla 2.10 se muestran los campos más importantes del *Boot Sector*, se señala un *offset* o desplazamiento que permite localizar al campo en cuestión. Dicho desplazamiento está contado a partir del cero lógico, que es donde empieza la partición.

Nombre	Offset (byte)	Tamaño (bytes)	Descripción
BS_jmpBoot	0	3	Instrucción de salto al código de arranque
BS_OEMName	3	8	Es una cadena de caracteres
BPB_BytsPerSec	11	2	Bytes por sector
BPB_SecPerClus	13	1	Sectores por cluster
BPB_RsvdSecCnt	14	2	Número de sectores reservados
BPB_NumFATs	16	1	Número de tablas FAT
BPB_RootEntCnt	17	2	Entradas en el directorio raíz en FAT12 y FAT16
BPB_TotSec16	19	2	Total de sectores en el volumen en FAT16
BPB_Media	21	1	Medios extraibles o no removibles
BPB_FATSz16	22	2	Sectores por FAT en FAT16
BPB_SecPerTrk	24	2	Sectores por pista. Válido sólo para dispositivos muy específicos como discos flexibles, cintas, discos duros, etc.
BPB_NumHeads	26	2	Número de cabezas. Válido sólo para dispositivos muy específicos como discos flexibles, cintas, discos duros, etc.
BPB_HiddSec	28	4	Número de sectores escondidos.
BPB_TotSec32	32	4	Total de sectores en el volume FAT

Tabla 2.10. Sector de arranque y estructura BPB.

1. Región FAT

La región FAT (***R1***) almacena la llamada Tabla de Asignación de Archivos, que está definida como una lista ligada de los clúster que componen a los distintos archivos y sirve como un mapa de la región de datos de la partición de acuerdo al número de clúster. Cada clúster es identificado por un número con el que se crea una entrada en la región FAT. Las entradas en la región FAT se crean en una posición correspondiente al número de clúster; el contenido de estas entradas indican el número del siguiente clúster de un archivo dado y con una marca especial el fin de dicho archivo. Por lo general, en la región FAT existen dos tablas FAT, en la que la segunda tabla es un duplicado exacto de la primera, lo que sirve como copia de seguridad.

La FAT es una tabla formada por elementos que se corresponden con cada uno de los clúster de la unidad, es decir, el elemento situado en la posición 40 de la FAT controla el *clúster* 40 del disco (que a su vez corresponderá a unos determinados sectores del disco).

En la figura 2.36 se muestra un ejemplo del seguimiento de archivos en una tabla *FAT*. En dicha figura, se presenta el aspecto que tendría la región FAT de un volumen que contiene a tres archivos y al directorio raíz. Los clúster que componen a cada archivo se en listan en los recuadros de la derecha. El Directorio Raíz ocupa los clúster 0x0002, 0x0009, 0x000A y 0x0011, por lo que las entradas correspondientes se encuentran en las posiciones 0x0002, 0x0009, 0x000A y 0x0011 respectivamente. Puede notarse que el contenido de la entrada localizada en la posición 0x0002 es 0x0009, es decir que indica el número del siguiente clúster. Esto aplica para el resto de las entradas, excepto en la última, en la que se indica con el valor 0xFFFFFFFF el fin del archivo. Esta es la lógica que se sigue para localizar a cualquier archivo.

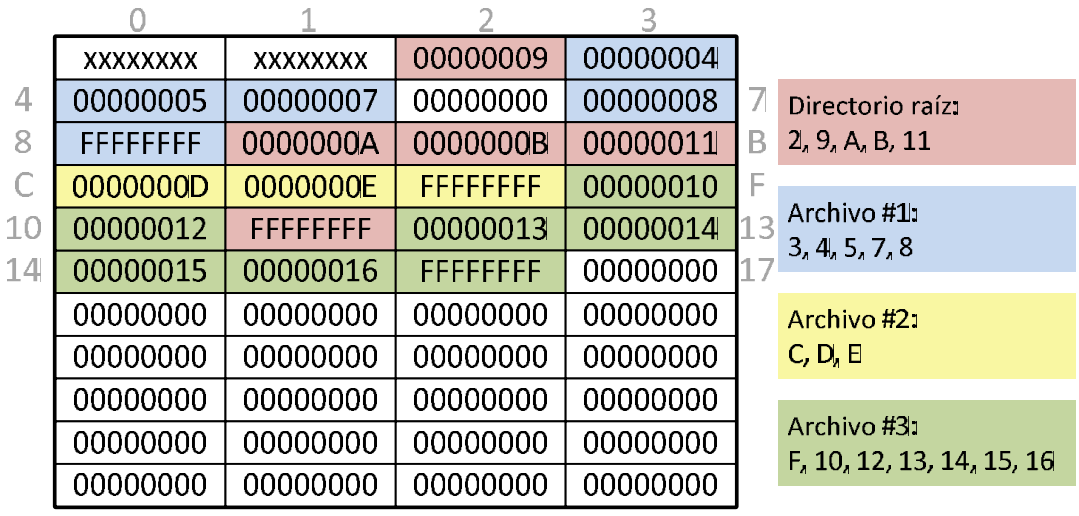


Figura 2.36. Sector FAT32, seguimiento del directorio raíz y tres archivos.

Para sistemas con FAT32 el Directorio Raíz es de tamaño variable, dependiendo de la cantidad de archivos y directorios que se vayan creando. Por lo general se encuentra en el primer clúster del área de datos.

El Directorio Raíz guarda la información relacionada a la disposición de los datos contenidos en el medio de almacenamiento; los datos son almacenados en una estructura de archivos y carpetas, en donde un directorio es un archivo con el atributo de directorio y su contenido serán las entradas de los archivos que lo conforman. El único directorio especial y que siempre debe estar presente es el directorio raíz.

Cada archivo o directorio (carpeta en Windows) creado tiene una entrada de 32 bytes en el Directorio Raíz, como lo muestra la figura 2.37; se utilizan ocho bytes para el nombre del archivo; tres para la extensión; un byte de atributos; ocho bytes reservados; dos bytes para la parte alta del número del primer clúster, cuatro bytes reservados, dos bytes para la parte baja del número del primer clúster y cuatro bytes para indicar el tamaño del archivo (en bytes). Esta información es utilizada por el SO para localizar al archivo deseado; obteniendo el número del primer clúster de un archivo encontrará el resto, entrando a la tabla FAT. Las entradas son iguales para los distintos tipos de FAT.

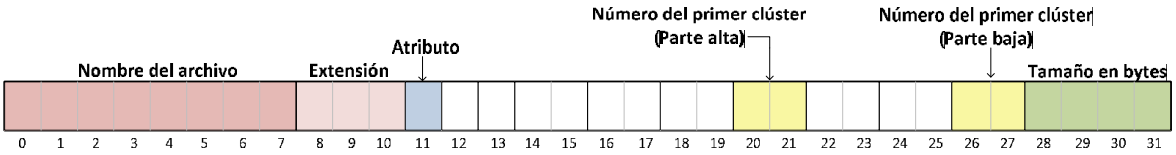


Figura 2.37. Estructura de entrada en el directorio raíz.

La última región corresponde al área de datos (***R2***), en la que son registrados los archivos y carpetas del usuario. Por lo general, la región de datos se encuentra a partir del clúster 2.

Determinación del tipo de FAT

El dispositivo que intente leer un medio de almacenamiento y localizar los archivos y carpetas guardados, tiene que conocer el tipo de sistema de archivos que tiene dicho medio para realizar las operaciones de lectura y escritura de manera correcta.

En el caso de los sistemas de archivos FAT, la única manera con la que se puede conocer el tipo de FAT que tiene el dispositivo de almacenamiento, es determinando el parámetro *Count_Of_Clusters*, que se refiere a la cantidad de clústers que tiene la unidad de almacenamiento en la región de datos. El procedimiento es el siguiente:

Primero, se determina la cantidad de sectores ocupados por el directorio raíz, realizando la siguiente operación, con algunos de los campos del BPB mostrados en la tabla 2.10:

$$RootDirSectors = \big((RootEntCnt * 32) + (BytesPerSec - 1)\big)/BytesPerSec \quad (2.4)$$

En un dispositivo con el sistema de archivos FAT32 el parámetro *RottDirSectors* siempre es 0, porque se ha definido al campo *RootEntCnt como cero.*

El siguiente paso consiste en determinar la cantidad de sectores en la región de datos. Teniendo en cuenta las siguientes consideraciones:

Si el sistema cuenta con FAT16: $FATsz = FATSz16$
$TotSec = TotSec16$

Si el sistema cuenta con FAT32: $FATsz = FATSz32$
$TotSec = TotSec32$

A partir de lo anterior se realiza la siguiente operación:

$$DataSec = TotSec - (ResvdSecCnt + (NumFATs * FATSz) + RootDirSectors) \quad (2.5)$$

Por último se obtiene el valor de *CountOfClusters,* de la siguiente manera:

$$CountOfClusters = DataSec / SecPerClus \quad (2.6)$$

Una vez determinado este valor se puede conocer qué tipo de FAT tiene el sistema, tomando en cuenta que siempre se cumple lo siguiente:

- En FAT12 $\boldsymbol{CountOfClusters} < 4084$
- En FAT16 $\boldsymbol{CountOfClusters} \leq 65524$
- En FAT32 $\boldsymbol{CountOfClusters} > 65524$

2.6. Microcontrolador

Un microprocesador puede entenderse como una Unidad Central de Procesamiento (*CPU: Central Processing Unit*), formada por una Unidad Aritmética-Lógica (*ALU: Arithmetic Logic Unit*), una unidad de control y algunos registros de transferencia, todo encapsulado en un solo circuito integrado.

Un microcontrolador es un sistema que incluye en un solo circuito integrado a un microprocesador y a un conjunto de subsistemas como memoria, puertos de entrada y salida, comunicación serie, convertidor analógico–digital, unidad de tiempo, oscilador, temporizador, entre otros.

Con lo anterior podemos decir que la más importante diferencia entre un microcontrolador y un microprocesador es su funcionalidad. Para darle un uso al microprocesador en una aplicación real, se debe de conectar con componentes tales como memoria o componentes buses de transmisión de datos. Aunque el microprocesador es considerado una máquina de computación poderosa, no está preparado para la comunicación con los dispositivos periféricos que se le conectan. Para que el microprocesador se comunique con algún periférico, se deben utilizar circuitos especiales. En cambio, al microcontrolador se le diseña de tal manera que tenga todas las componentes especiales integradas en el mismo circuito integrado. No necesita de otros componentes especializados para su aplicación, porque todos los circuitos necesarios, como los periféricos, ya se encuentran incorporados. Esto ahorra tiempo y espacio necesario para construir un dispositivo.

Algunos ejemplos del uso de un microcontrolador para el manejo de uno o varios procesos son: realizar tareas de procesamiento de datos provenientes de convertidores analógico digitales, realizar rutinas para almacenamiento de datos, adquirir los datos provenientes de un receptor GPS y realizar la interacción humano–máquina con ayuda de un *joystick* y mostrar resultados en una pantalla de cristal líquido.

Los sistemas internos más importantes con los que cuenta un microcontrolador se mencionan a continuación:

***-Reloj del sistema*:** El microcontrolador ejecuta las instrucciones del programa que se le cargó a cierta velocidad, la cual está determinada por la frecuencia del oscilador, llamada a veces “frecuencia del reloj” o “reloj del sistema”. Dicha frecuencia puede ser generada por un circuito interno de tipo RC o por circuitos externos del tipo RC, LC o a través de osciladores basados en cristal de cuarzo.

***-Memoria de programa*:** La memoria de programa es el espacio designado para almacenar las instrucciones que constituyen al código del programa. Es de tipo no-volátil, por lo general de tipo *EEPROM* o *FLASH*.

***-Memoria de lectura y escritura*:** Es la memoria utilizada por el microcontrolador para el almacenamiento de datos y variables. Por lo general es memoria tipo RAM.

***-Módulo MSSP* (*Master Synchronous Serial Port*):** Este módulo permite la comunicación a través del bus SPI y también por I2C (Inter-Integrated Circuit).

***-Módulo USART* (*Universal Synchronous Asynchronous Receiver Transmitter)*:** El módulo *USART* proporciona al microcontrolador la capacidad para comunicarse a través de las interfaces seriales RS–485 y RS–232.

***-Puerto E/S digital*:** Los puertos de entrada y salida digital proveen terminales que permiten controlar y recibir información de dispositivos periféricos, como teclados, *display's*, relevadores, motores y algunos otros.

***-Puerto de conversión analógico–digital*:** Este puerto es el que se encarga de realizar la conversión de señales analógicas a digitales; con ello se obtiene un número binario proporcional al voltaje de entrada detectado. Es utilizado para manipular la señal proveniente de algunos sensores.

***-Temporizador* (*Timmer*):** El temporizador es implementado como un contador que establece el tiempo preciso para la realización de alguna acción ante evento o rutina. También es útil como contador de eventos.

2.6.1. Arquitecturas de los microcontroladores

Los microcontroladores pueden ser clasificados de acuerdo a las características del conjunto de instrucciones que soportan:

- CISC (Complex Instruction Set Computer): Computadora con un conjunto complejo de instrucciones.

- RISC (Reduced Instruction Set Computer): Computadora con un conjunto reducido de instrucciones.

- MISC (Minimal Instruction Set Computer): Computadora con un conjunto mínimo de instrucciones.

La arquitectura interna de los microcontroladores está basada en la conexión de la CPU con sus periféricos, esta puede ser de dos tipos: **Von Neumann** o **Harvard.**

Arquitectura Von Neumann

En la arquitectura Von Neumann existe un bus por el cual viajan datos e instrucciones y comunica a la CPU con una memoria, la cual contiene a la vez el programa y los datos del sistema. Cuando la cantidad de datos por procesar es muy grande, la velocidad de procesamiento se reduce, y la arquitectura se satura, ya que el bus se utiliza para los datos y para las instrucciones, esto genera un cuello de botella; además puede ocurrir un traslape de zonas.

Algunas de las ventajas que ofrece la arquitectura Von Neumann son: el uso eficiente de la memoria, utilizando una jerarquía de memoria así que no se necesita que esté dividida en dos; es arquitectónicamente más simple, por lo que hay sólo un tipo de instrucciones para acceder a la memoria y una misma forma de acceso a datos y direcciones; tiene una mayor flexibilidad, esto lo hace útil para sistemas operativos y se minimiza la fragmentación de la momoria. En la figura 2.38 se muestra la arquitectura Von Neumann.

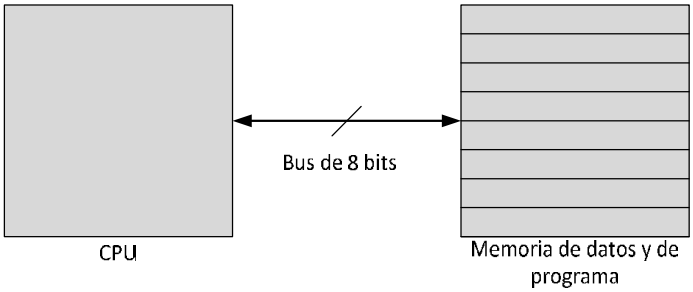


Figura 2.38. Arquitectura Von Neumann.

Arquitectura Harvard

La arquitectura Harvard fue desarrollada en 1970 para solucionar los problemas de velocidad de procesamiento que presentaba la arquitectura Von Neumann. En la arquitectura Harvard la CPU utiliza dos memorias, una de datos y otra de instrucciones, mediante dos buses distintos. Estos buses son independientes entre sí, por lo que pueden ser de distinto ancho, y el procesador puede recibir instrucciones por caminos diferentes, aprovechando el tiempo del ciclo de máquina y obteniendo a su vez un mayor desempeño en la ejecución de instrucciones. El concepto y nombre de la arquitectura, figura 2.39, proviene de la computadora MARK1, construida en la Universidad de Harvard en 1944.

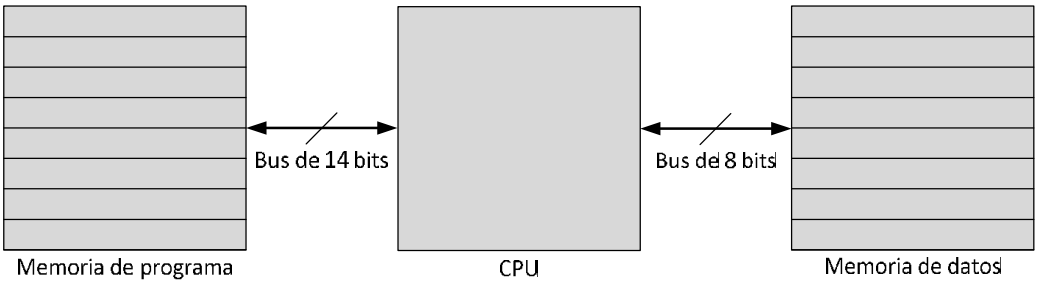


Figura 2.39. Arquitectura Harvard.

2.6.2. Familias de microcontroladores

Hay gran variedad de microcontroladores en el mercado, existen muchas marcas y modelos. Cada marca de microcontroladores cuenta con dispositivos tanto de propósito general como para uso específico y con características muy diversas. En las siguientes líneas se comentan de forma general los principales fabricantes de microcontroladores, sus modelos y algunas de las familias que producen.

ARM (*Advanced RISC Machine)*: Los microcontroladores ARM **son de una arquitectura de 32 bits, desarrollada en 1983 por la empresa Acorn Computers Ltd** para usarse en computadoras personales. Este microcontrolador maneja un sistema de instrucciones simple, esta característica le **permite ejecutar tareas con un mínimo consumo de energía**. En general, los atributos clave del

núcleo AMR son: el tamaño de la aplicación, desempeño y bajo consumo de energía. El núcleo de los microcontroladores ARM es utilizado por distintos fabricantes como base para sus propios diseños de microcontroladores. Dependiendo de la familia a la que pertenezca el microcontrolador ARM puede manejar la arquitectura Harvard o la arquitectura Von Neumann.

De los periféricos incorporados en microcontroladores ARM se encuentran: Convertidores Analógico – Digital (ADC: Analog to Digital Converter) y Digital – Analógico (DAC: Digital – Analog Converter), comunicación USB (Universal Serial Bus), comunicación serie SPI e I2C, módulo UART (Universal Asynchronous Receiver-Transmitter), módulo CAN, PWM (Pulse Wide Modulation) y Ethernet. Además, cuentan con memoria flash y SRAM (Static Random Access Memory).

Las principales familias de microcontroladores ARM son:

- Cortex – A. Está conformado por una serie de procesadores de aplicaciones que proporcionan una gama de soluciones a dispositivos que realizan tareas de cómputo complejas, tales como: contener una plataforma rica para un sistema operativo, ejecución de interfaces de usuario, y soporte para aplicaciones de software.
- Cortex – R. Son procesadores en tiempo real que ofrecen un alto desempeño en soluciones de cómputo para sistemas embebidos, en donde se quiere confiabilidad, alta disponibilidad, tolerancia a fallas, mantenibilidad y respuestas en tiempo real.
- Cortex – M. Es una familia optimizada en costo y en consumo de energía y son de señal mixta, se usan en aplicaciones como: internet de las cosas (IoT por sus siglas en inglés), conectividad, medición inteligente, dispositivos de interfaz humana, automotriz y sistemas de control industrial, electrodomésticos, productos de consumo e instrumentación médica.
- SecureCore. Es una familia de procesadores que ofrece poderosas soluciones de seguridad de 32 bits, diseñada principalmente para tarjetas inteligentes de uso rudo, incorpora varias características de seguridad lo que la hace una opción ideal para tales aplicaciones.

INTEL: Dentro de los microcontroladores de esta compañía se encuentra familia denominada 8051 (MCS 51, MCS 52). Estos dispositivos, como en el caso de los ARM, han sido utilizados como base para el desarrollo de otros microcontroladores, por parte de otros fabricantes. Estos últimos añaden algunas mejoras a sus características e incorporan otras características.

Los microcontroladores 8051 son de arquitectura Harvard, cuentan con convertidores ADC y DAC, comunicación USB, I2C y SPI, UART, temporizadores y contadores, módulo PWM, módulo CAN, y reloj de tiempo real (RTC: Real Time Clock), entre algunas otras funcionalidades.

ANALOG DEVICES: Esta empresa basa el diseño de sus microcontroladores en los núcleos ARM7 (de arquitectura Von Neumann) y 8052 (de arquitectura Harvard), ambos con un conjunto de instrucciones RISC. Estos dispositivos cuentan con amplias prestaciones analógicas, algunas como: ADC de 16 bits, DAC de 12 bits, referencia de voltaje, módulo PWM, sensor de temperatura y memoria flash.

Las principales familias de estos dispositivos son:

- ADuC7xx (Nucleo ARM7)
- ADuC8xx (Nucleo 8052)

Las aplicaciones principales de estos microcontroladores se encuentran en el área médica, de instrumentación y de las comunicaciones.

FREESCALE: Este fabricante produce microcontroladores de 8, 16 y 32 bits, de arquitectura Von Neumann y conjunto de instrucciones CISC. Estos microcontroladores cuentan con módulo SPI, I2C, PWM, temporizadores, ADC, entre otros. Algunos de estos dispositivos más específicos cuentan, además, con interfaz USB 2.0 y memoria flash, con la capacidad de ejecutar hasta 60 MIPS (Millones de Instrucciones por Segundo).

Dentro de las principales familias de microcontroladores FREESCALE hay:

- Microcontroladores de 8 bits:
- Microcontroladores de 16 bits:
- Microcontroladores de 32 bits:

Las aplicaciones más comunes de estos dispositivos son el control de displays, control de motores, comunicaciones, entre otras.

MICROCHIP: Los microcontroladores de Microchip PIC (Peripheral Interface Controller) son de arquitectura Harvard y contienen un conjunto de instrucciones RISC. Cuentan con varios modelos que incorporan periféricos básicos, como son: los módulos de comunicación serie MSSP, módulo USART y PWM, ADC, temporizadores, etcétera. En dispositivos de última generación se cuenta con interfaces USB, CAN, Ethernet y memoria FLASH.

Los PIC se clasifican en tres familias o gamas de acuerdo a las características incorporadas:

- Microcontroladores base o gama Baja

Los microcontroladores de gama baja tienen memoria de programa que puede contener 512, 1 k y 2 k palabras de 12 bits, y pueden ser de tipo ROM, EPROM y también hay modelos con memoria OTP (One Time Programmable). La memoria de datos puede tener una capacidad comprendida entre 25 y 73 bytes.

Los microcontroladores de gama baja sólo disponen de un temporizador (TMR0), un conjunto de 33 instrucciones y un número de terminales de E/S comprendido entre 12 y 20. El rango del voltaje de alimentación es muy flexible comprendido entre 2 y 6.25 V, lo que permite ser alimentado con baterías, teniendo en cuenta su bajo consumo (menos de 2 mA a 5 V y 4 MHz).

- Microcontroladores de gama Media

Los microcontroladores de gama media tienen prestaciones adicionales a las de los de la gama baja, haciéndolos más adecuados en las aplicaciones complejas. Admiten interrupciones, poseen comparadores analógicos, convertidores A/D, puertos serie y diversos temporizadores.

- Microcontroladores de gama Alta

Estos microcontroladores poseen un conjunto de 70 instrucciones, éstas son de 16 bits con bus de datos de 8 bits, pueden trabajar a 10 MIPs (Millones de Instrucciones por Segundo) con un cristal de 40 MHz, tienen prioridad de interrupciones, un multiplicador en hardware de 8 x 8, que funciona en un solo ciclo de máquina, tres terminales para manejo de interrupciones externas, maneja niveles de corriente de 25 mA; cuentan con un temporizador de 16 bits, con un segundo temporizador de 8 bits y con un tercer temporizador de 16 bits; tienen dos módulos de captura/comparación/PWM, y para comunicarse tienen un módulo de comunicación serial que soporta las interfaces RS-485 y RS-232.

ATMEL: Los microcontroladores AVR de ATMEL son dispositivos de arquitectura Harvard y conjunto de instrucciones RISC. Dentro de las características más comunes de estos microcontroladores se encuentran los ADC, comunicación SPI, módulo UART, módulo PWM, módulo de captura y comparación y temporizadores, además que utilizan memoria flash (hasta 128 kbytes). En general, los microcontroladores de ATMEL ejecutan hasta 16 MIPS. El AVR fue diseñado en un principio para la ejecución eficiente en código C.

Las principales familias de microcontroladores AVR son:

- **Tiny AVR.** Esta familia está orientada a aplicaciones que requieren encapsulados de microcontrolador pequeños, con un número reducido de terminales, parten de 6 terminales, con memoria flash de hasta 8 kB.

- **Mega AVR.** Esta familia cuenta con dispositivos con una cantidad mayor de terminales con respecto a tinyAVR, y una memoria flash de máximo 256 kB.

- **Xmega AVR.** Estos microcontroladores están orientados a aplicaciones en tiempo real, tienen de 44 a 100 terminales, contienen periféricos como ADCs y DACs de 12 bits, módulos para encriptado de datos y controladores DMA (Direct Memory Access), cuentan con mayor número de módulos UART, PWM, I2C, SPI.

- **UC3 AVR.** Estos microcontroladores son de arquitectura de 32 bits, son capaces de realizar de forma nativa operaciones de procesamiento digital, tienen controladores DMA; cuentan con conectividad CAN, LIN (Local Interconnect Network), Ethernet, salidas PWM en todas sus terminales de E/S, encapsulados de 48 a 144 terminales; contienen memoria flash de 512 kB.

Los microcontroladores AVR se encuentran en aplicaciones como son la automotriz, el control de displays, el control de motores e iluminación, también en aplicaciones de propósito general.

2.6.3. Los microcontroladores AVR

La arquitectura básica del microcontrolador AVR fue diseñada por dos estudiantes de NTH (Norwegian Institute of Technology), Alf-Egil Bogen y Vegard Wollan, después fue comprada y desarrollada por Atmel en 1996.

Existen varias versiones del significado de AVR; Attmel dice que no es más que el nombre de un producto, pero podría significar “Advanced Virtual RISC”, o “Alf and Vegard RISC (los nombres de los diseñadores del AVR).

Hay varios tipos de microcontroladores AVR con diferentes propiedades. A excepción del AVR32, que es un microcontrolador de 32 bits, todos los AVRs tienen microprocesadores de 8 bits, lo que significa que su CPU puede trabajar sólo con datos de 8 bits a la vez. Los datos mayores a 8 bits tienen que ser divididos en piezas de 8 bits para ser procesados por la CPU. Uno de los problemas con los microcontroladores AVR es que no todos son 100% compatibles en términos de software cuando se migra de una familia a otra. Por ejemplo, para ejecutar programas escritos para ATtiny25 en un ATmega64, se debe recompilar el programa y probablemente cambiar la ubicación de algunos registros antes de cargarlo al ATmega64.

Los microcontroladores AVR generalmente se clasifican en cuatro grandes grupos: Mega, Tiny, propósito especial y Clásico. La familia Mega son los microcontroladores mayormente usados.

Características de los AVR

Los AVR son microcontroladores RISC de 8 bits en sólo un circuito integrado con arquitectura Harvard, el cual tiene ciertas características estándar, como son: ROM para el programa, RAM de datos, EEPROM de datos, temporizadores y puertos de entrada/salida. La figura 2.40 muestra de manera simple a un AVR con las características mencionadas.

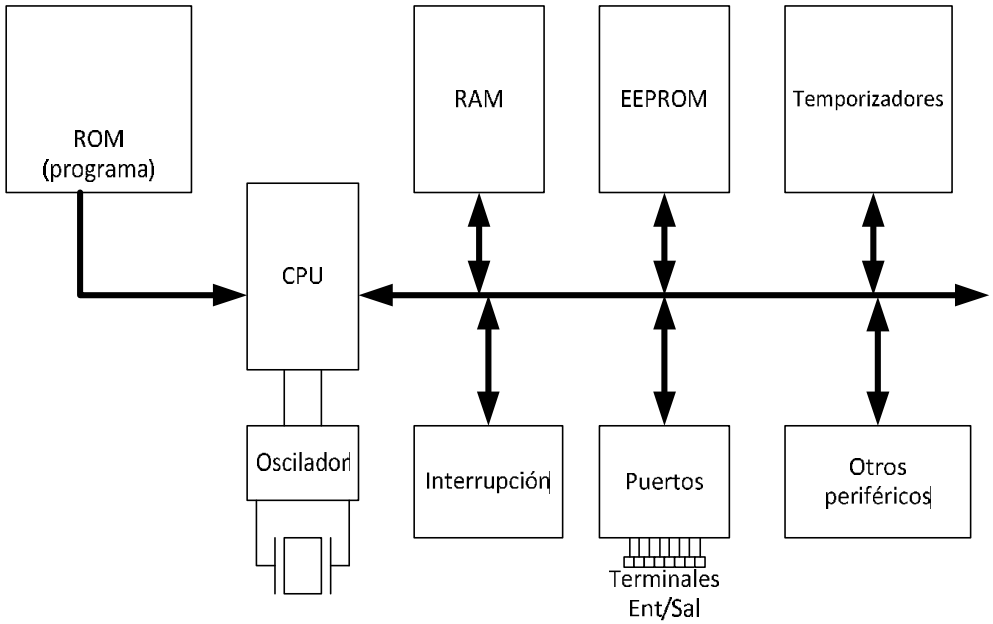


Figura 2.40. Vista simplificada de un microcontrolador AVR.

La mayoría de los AVR tienen algunas características adicionales como: ADC, PWM, y diferentes clases de interfaces seriales como USART, SPI, I2C (TWI: Two-Wire Interface), CAN, USB, etc.

ROM de programa de los microcontroladores AVR

En los microcontroladores la memoria ROM es usada para guardar programas, por esta razón se le conoce como ROM de programa o memoria de programa. Aunque el AVR tenga la capacidad de contener una ROM de 8MB, a los miembros de las familias no se les incluye esa cantidad de ROM. El tamaño de la ROM de programa puede variar desde 1K hasta 256K, dependiendo del miembro de la familia. El AVR fue uno de los primeros controladores en utilizar memoria ROM de tipo flash para almacenar el programa.

Memorias RAM y EEPROM

Mientras que la memoria ROM se usa para almacenar al programa, la memoria RAM es para el almacenamiento de datos. En general, el AVR tiene un máximo de 64kB de espacio en RAM, aunque no todos los miembros de la familia tienen esa cantidad de RAM. Esta RAM tiene tres componentes: registros de propósito general, memoria de entrada/salida, y SRAM interna. Hay 32 registros de propósito general en todos los AVR, pero el tamaño de la SRAM y el de la memoria de entrada/salida varía de integrado a integrado. En la hoja de datos de los AVR, siempre que se menciona el tamaño de la RAM se refiere al tamaño de la SRAM interna. En AVR hay también una pequeña cantidad de EEPROM para almacenar datos críticos que no se requieren cambiar muy a menudo, con

capacidad desde 512 Bytes como en el caso del ATmega8, hasta 4096 Bytes como en el caso del ATmega2560.

Terminales de entrada/salida en los microcontroladores AVR

Los microcontroladores AVR pueden tener de 3 a 86 terminales de entrada/salida. El número de terminales depende del y va desde 8 hasta 100. En el caso de un microcontrolador de 8 terminales, hay 3 terminales de entrada/salida, mientras en el caso de uno de 100 terminales se pueden usar hasta 86 terminales de entrada/salida.

Periféricos de los microcontroladores AVR

La mayoría de los AVR cuentan con ADC, temporizadores, y USART, como periféricos estándar. El ADC es de 10 bits y el número de canales ADC en los AVR varía y puede haber hasta 16, dependiendo del número de terminales del empaquetado. Los AVR pueden tener hasta 6 temporizadores, además de los temporizadores "watchdog". El periférico USART nos permite conectar sistemas basados en AVR a los puertos seriales, como los COM de las computadoras. La mayoría de los miembros de la familia AVR tienen buses I^2C y SPI y algunos de ellos tienen también buses USB o CAN.

Esquema del número de producto en los AVR

Todos los números de producto de AVR empiezan con AT, que significa Atmel. Las letras que siguen identifican el tipo de AVR del que se trata. Al final encontramos un número, donde el número más grande que es potencia de 2 muestra la cantidad de memoria de programa del microcontrolador. Por ejemplo, en ATmega**128**0 la mayor potencia de 2 que se encuentra es 128, así que tiene 128 kB de memoria de programa; en ATtiny**4**4, la cantidad de memoria de programa es 4 kB; etc. Aunque hay algunas excepciones como AT90PWM216, el cual tiene 16 kB de memoria programa en lugar de 2 kB.

2.6.4. Programación de los microcontroladores

La programación de microcontroladores puede realizarse entre tres tipos de niveles básicos de lenguajes: en código máquina, ensamblador y de alto nivel.

Lenguaje en código máquina

Es el lenguaje elemental del microprocesador, pero el más complicado de utilizar. Cada instrucción posee códigos hexadecimales que son específicos de ese procesador. Esto hace que la programación de las distintas familias de microprocesadores sea incompatible. Sólo se trabaja en código máquina con algunos periféricos que disponen de un repertorio determinado de comandos.

Los lenguajes diferentes al de máquina al final serán transformados a éste para ser introducidos en la memoria, ya que es el único lenguaje que entienden los microprocesadores. Pero esta conversión no la realiza el programador, sino que existe un software específico para este fin. Si se desea programar en código máquina, hay que entender previamente a fondo el microprocesador que se va a utilizar, ya que cada bit de cada instrucción tiene un significado concreto y es muy fácil equivocarse.

El lenguaje ensamblador

El lenguaje ensamblador es un tipo de lenguaje intermedio entre los de alto nivel y el lenguaje máquina. Cada microprocesador tiene su propio lenguaje ensamblador, que está en relación directa con su estructura. Este lenguaje usa las mismas instrucciones que posee el microprocesador, solo que el programador no emplea su correspondencia en hexadecimal, como en el lenguaje máquina, sino que utiliza los nemotécnicos de dichas instrucciones.

Lenguajes de alto nivel

Se llaman de alto nivel porque su sistema de programación está a la altura misma del lenguaje conceptual, matemático y de organización del propio hombre. El desarrollo de los lenguajes de alto nivel fue necesario como consecuencia de la adaptación de la máquina al hombre. Esto trajo muchas ventajas que hicieron que este tipo de lenguaje de programación se impusiera rápidamente. Por un lado, al ser un lenguaje próximo al del hombre –que, en definitiva, es quien tiene que programarlos–, permite la reducción de los costos de software, así como también del tiempo de desarrollo. Otras ventajas son su facilidad de aprendizaje, la posibilidad de realizar programación estructurada y el hecho de que para usarlo no es imprescindible tener conocimiento del hardware correspondiente.

Los lenguajes de alto nivel también fueron pensados para eliminar la incompatibilidad entre los de bajo nivel y los distintos sistemas de procesadores. Sin embargo, esto no es del todo cierto, ya que existen algunas diferencias dentro de un mismo lenguaje de alto nivel con los distintos sistemas que no proporcionan total compatibilidad. Obligatoriamente, un programa en lenguaje de alto nivel debe ser traducido a código máquina, para lo cual se utilizan programas intérpretes o compiladores.

2.7. Sistema de posicionamiento global

El sistema de posicionamiento global, (GPS: Global Positioning System), es un sistema compuesto por una red de satélites, radio bases terrestres y receptores GPS que permiten casi todas las posibilidades de navegación y posicionamiento en cualquier parte del mundo. Este sistema está compuesto por tres subsistemas principales, éstos son:

- Sistema satelital
- Sistema de control terrestre
- Sistema de receptores

Los sistemas de posicionamiento global de otros países ya están en órbita o en proyecto, pero al referirse al GPS generalmente las personas se refieren al sistema de posicionamiento operado por el gobierno de los Estados Unidos.

El **sistema satelital** está formado por una constelación de 24 satélites, denominada NAVSTAR (Navigation System Using Timing and Ranging), que giran alrededor de la Tierra en seis planos orbitales a unos 60° entre sí, con cuatro satélites en cada plano. Hay 21 satélites activos y 3 de reserva. En caso de que falle un satélite, uno de los de reserva puede ocupar su lugar. En la figura 2.41 se muestran las órbitas de los 21 satélites funcionales de la red NAVSTAR.

Figura 2.41. Constelación de satélites NAVSTAR y sus órbitas.

Cada satélite tiene tres piezas de hardware importante:

- ✓ **Computadora:** Mediante esta computadora a bordo se controla su vuelo y otras funciones.
- ✓ **Reloj atómico:** Éste mantiene la precisión del tiempo dentro de tres nanosegundos.
- ✓ **Radio transmisor:** Éste envía señales a la Tierra.

Los satélites NAVSTAR no son geosincrónicos, o geoestacionarios, debido a que giran en torno a la Tierra en órbitas circulares inclinadas. El ángulo de elevación en el modo ascendente es 55° con respecto al plano ecuatorial y la elevación promedio de un satélite NAVSTAR es de unos 20,200 km sobre la Tierra. Esta red de satélites fue desarrollada por el Departamento de Defensa de los Estados Unidos de América y declarada totalmente operativa desde el 27 de abril de 1995.

Al principio fue creada exclusivamente para fines militares y, posteriormente, dadas las aplicaciones tecnológicas, se puso al servicio de la comunidad civil.

El **sistema de control terrestre** también denominado "sistema de control de operación", incluye todas las estaciones monitoras terrestres fijas que se encuentran ubicadas en todo el mundo. Las estaciones monitoras no son más que receptores GPS que rastrean los satélites cuando pasan sobre ellas y acumulan datos de telemetría y efemérides de los mismos. Esta información se transmite a una estación de control maestro donde se procesa y determina si la posición real del satélite es igual a la calculada por el GPS. La estación de control – maestro recibe datos de las estaciones monitoras en tiempo real, con esa información se determina si los satélites sufren cambios de reloj o de información y detecta el mal funcionamiento de los equipos, por lo que se calcula nuevamente la información de navegación a partir de las señales monitoreadas y se envía a los satélites, junto con órdenes de mantenimiento rutinario.

Una efeméride es una lista de posiciones pronosticadas de cuerpos astronómicos como planetas o de la Luna. Las efemérides han existido desde hace miles de años debido a su importancia en la navegación astronómica. Las efemérides se compilan para rastrear las posiciones de los numerosos satélites que orbitan a la Tierra.

El **sistema de receptores** consiste en todos los receptores GPS y la comunidad de usuarios. Quien sea que tenga un receptor GPS puede recibir la señal de los satélites para determinar donde se localiza.

Funcionamiento del GPS

La base del GPS es la trilateración del receptor GPS con respecto a los satélites, al igual que la triangulación se usa para obtener posiciones relativas de objetos. Para realizar la trilateración, el receptor GPS mide la distancia hacia los satélites usando el tiempo de recorrido de la señal. Para medir el tiempo de recorrido el GPS necesita de una temporización muy precisa, y junto con la distancia se requiere saber dónde están los satélites en el espacio. Finalmente se deben corregir los retrasos que experimenta la señal en su viaje a través de la atmósfera.

Para determinar la posición de un punto en el espacio, es suficiente conocer las distancias a tres puntos de coordenadas conocidas. Se trata de una intersección espacial inversa. Es un problema geométrico relativamente simple porque se trata de una pirámide de base triangular. Desde el punto de vista geométrico el problema tiene dos soluciones, pero es fácil elegir la correcta, puesto que la otra está ubicada a unos 40.000 km. de la superficie terrestre.[4]

El problema a resolver es medir las distancias entre satélites y receptor. Para ello se utiliza el llamado código C/A (Código de Adquisición común) porque en GPS la

[4] Huerta, E., Mangiaterra, A., & Noguera, G. *GPS, Posicionamiento Global* P.8. Editorial UNR. 2005

medición es de vía única, es decir, no regresa la señal, por lo tanto no hay manera de que el receptor "sepa" en qué momento se envió y así calcular el tiempo que le tomo recorrer la distancia satélite-receptor. Puesto que se trata de medir tiempos es necesario contar con "relojes" adecuados tanto en los satélites como en el receptor. Los relojes constan de unos osciladores de frecuencias muy estables capaces de señalar medidas de tiempo en el orden de 10^{-13} segundos (o 10^{-14}) en los satélites y 10^{-8} segundos en los receptores.

Desde los satélites emiten dos ondas portadoras, llamadas L1 y L2; sobre L1 se monta la modulación correspondiente al código C/A. En la figura 2.42 se muestra al pseudo-ruido y a las señales que lo componen.

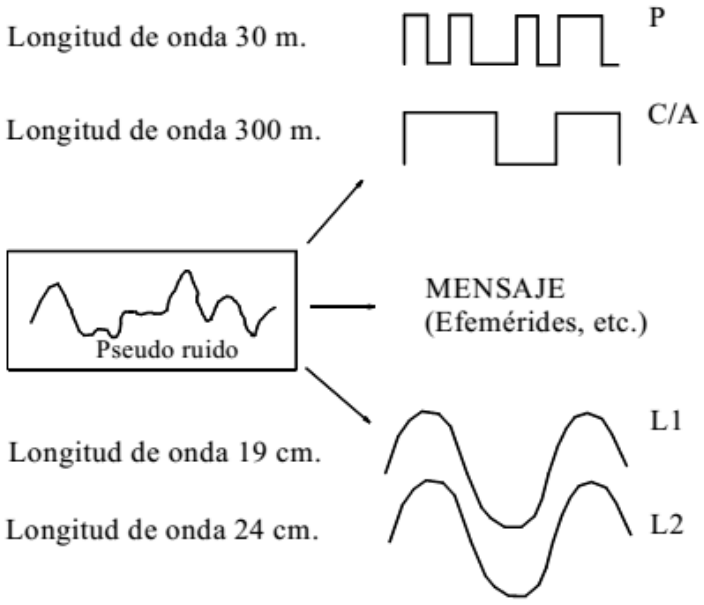


Figura 2.42. Señales: códigos (P y C/A), Portadoras (L1 y L2).

Suponiendo un satélite en particular, al código lo podemos imaginar como una serie de ceros y unos, o bien de "+1" y "-1", en un cierto orden. Al multiplicar la onda portadora por el código, aquella no se altera cuando se encuentra con los "+1", pero se invierte donde aparecen los "-1". Todo ello da como resultado una onda deformada, un seudo ruido aparentemente aleatorio, que es el mensaje que llega al receptor. La señal emitida es el resultado de multiplicar la onda portadora por el código. Es importante mencionar que cada satélite cuenta con un código C/A diferente, lo que genera una modulación específica de la señal, propia y exclusiva de ese satélite, de tal modo se obtiene un PRN (ruido seudo aleatorio) distintivo de ese satélite, haciendo analogía gráfica se diría que es un "dibujo" característico de ese satélite. Ese "dibujo", va asociado al tiempo; se repite cada milisegundo y le corresponde un instante determinado para comenzar cada repetición; ese instante no puede ser cualquiera, debe ser común a todo el sistema. Cada receptor tiene almacenadas en su memoria las réplicas de todos los PRN. Así, cuando recibe la emisión satelital, puede efectuar el reconocimiento

del satélite correspondiente. A continuación, procesando la señal recupera el código con el que fue modulada y, a la vez, genera interiormente una réplica del código recibido, éste está desfasado porque el recibido tuvo que viajar por el espacio.

La siguiente operación consiste en correlacionar el código recibido con el "local", lo que permite medir el tiempo y por lo tanto la distancia (considerando conocida la velocidad de la luz en el espacio). Pero lo que se miden son seudodistancias porque no puede haber una sincronización perfecta entre el reloj del satélite y el reloj del receptor. La consecuencia es que la distancia observada no es la real, sino un valor próximo que difiere en una longitud:

$$\delta d = c(-\delta R) \tag{2.7}$$

Donde c =velocidad de la luz. Surge así una incógnita imprevista: δR es una incógnita que representa el error del reloj del receptor respecto al sistema de tiempo GPS. Hay 4 incógnitas: 3 de posición $(x_p y_p z_p)$, 1 de reloj (δ_R). Esto se resuelve observando 4 satélites en vez de 3 y resolviendo un sistema de 4 ecuaciones con 4 incógnitas. Eso simplifica enormemente las cosas porque permite utilizar en los receptores osciladores menos precisos y con menor costo que los relojes atómicos de los satélites.

Intercambio de información

Para enviar la información del receptor GPS hacia el exterior para intercambio o para procesamiento se usan ciertos protocolos de comunicación estándares como: RINEX, NGS-SP3, RTCM SC-104 y NMEA 0183. El protocolo utilizado en este trabajo es el NMEA 0183.

NMEA es la abreviatura de *National Marine Electronics Association*. Es una asociación fundada en el año de 1957 por un grupo de fabricantes de electrónica, con la finalidad de obtener un sistema de comunicación común entre las diferentes marcas de electrónica naval. Poco a poco se fueron sumando todos los fabricantes a este estándar, además de organizaciones oficiales y gubernamentales. NMEA fue creado para el intercambio de información digital entre productos electrónicos marinos. El primer protocolo estándar se llamó NMEA 0183, y es el que todavía se utiliza y acepta en la mayoría de equipos electrónicos destinados para la navegación, como es el caso de las ecosondas, los sonares, los anemómetros, los girocompases y los receptores GPS.

El estándar NMEA 0183 utiliza un protocolo de comunicaciones serie ASCII, el cual define como se transmiten los datos en una cadena de un "locutor" a varios "oyentes" a la vez, los cuales son transmitidos mediante una comunicación serial. Para obtener los datos del receptor GPS es necesario conectar el receptor GPS con un equipo que procese la información de dicho dispositivo, por lo tanto se

debe configurar el puerto serial correctamente para enviar los datos del receptor GPS al equipo, dicha configuración se muestra en la tabla 2.11.

Baud Rate	4800
Bits de datos	8
Paridad	Ninguno
Bits de parada	1

Tabla 2.11. Valores del puerto serial para recibir señales del receptor GPS.

La cadena de datos del NMEA 0183 puede incluir información sobre la posición, fecha, altitud y otras variables. Las características globales a considerar son las siguientes:

- Los datos son enviados en forma de cadenas.
- Cada cadena comienza con el signo "$" y termina con los caracteres de retorno de carro y siguiente línea <CR><LF>.
- El signo "$" está seguido por un campo de 5 caracteres, generalmente en mayúsculas, que identifica al hablante (los primeros dos caracteres), al tipo de dato y el formato de los campos sucesivos (los últimos tres caracteres).
- El último campo de cualquier cadena es una suma de control, precedido por un carácter delimitador "*".

La siguiente es una cadena típica del estándar NMEA:

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Donde:

RMC→ Recommended Minimum Specific Data.
123519→ Hora, 12:35:19 UTC.
A→ Estado: A significa que el dato de posición se fijó y es correcto.
4807.038, N→ Latitud 48, 07.038 grados Norte.
01131.000, E→ Longitud 11, 31.000 grados Este.
022.4→ Velocidad en nudos.
084.4→ Orientación en grados del receptor GPS.
230394→ Fecha – 23 de Marzo de 1994.
003.1, W→ Variación magnética.
*64→ Suma de verificación de datos, siempre empieza con *

2.8. Dispositivos periféricos

Los dispositivos periféricos son aquellos que Pfaffenberger (1990), antropólogo social de tecnología, ofrece una definición precisa al señalar que los periféricos son aquellos dispositivos controlados por la computadora, pero externos a la unidad central de procesamiento (CPU). Por su parte, Larry Long (1999) asegura

que un periférico es un dispositivo de *hardware* que no sea el procesador central. Los dispositivos periféricos o simplemente periféricos son herramientas de *hardware* que sirven como interface entre el usuario y la computadora, y tienen la finalidad básica de satisfacer algún requerimiento, ya sea introducir, obtener o almacenar información. En la actualidad los dispositivos periféricos son muy variados, tan simples como un par de audífonos o tan complejos como una pantalla LED (Light-Emitting Diode) 3D.

Los periféricos se dividen fundamentalmente en los siguientes tres grandes grupos:

- Dispositivos de entrada
- Dispositivos de salida
- Dispositivos de entrada/salida

El microcontrolador cuenta con terminales designadas para la entrada y salida de datos (Puertos de E/S). Dichas terminales son de tipo digital, es decir, detectan o entregan los niveles lógicos ‘1’ y ‘0’.

2.8.1. Joystick

El joystick está conformado por un conjunto de interruptores. Mediante estos interruptores y un arreglo electrónico se podrán generar estados lógicos (‘1’ y ‘0’), cerrando o abriendo un circuito eléctrico, dependiendo de la posición en la que se encuentre.

De manera ideal, los arreglos entregarán una forma de onda como la mostrada en la figura 2.43, esto cuando se realizan activaciones y desactivaciones sucesivas.

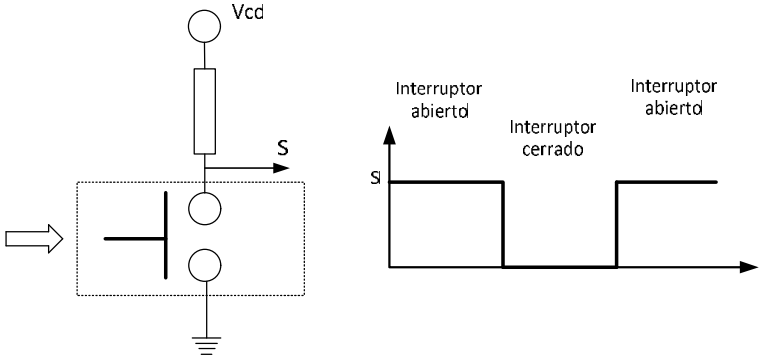


Figura 2.43. Señal ideal de cada posición de joystick.

Debido a las características mecánicas de los elementos que constituyen a los interruptores, al ser accionados éstos se generan vibraciones que alteran la forma de onda entregada, lo que es conocido como efecto de rebote. El efecto de rebote mecánico debe ser evitado, debido a que el microcontrolador puede tomarlo como

si se pulsara varias veces la tecla, generando errores en las rutinas programadas. Existen varias maneras de eliminar el efecto de rebote, algunas formas incluyen *hardware*. Una solución por medio de *software* es agregar retardos en la rutina que lee el estado del botón, con el fin de esperar a que se estabilice la señal proporcionada por el interruptor. Con una rutina de retardo, el programa descarta el ruido producido al accionar alguna tecla y sólo leerá el valor lógico cuando ya se encuentra estable.

2.8.2. Pantalla de cristal líquido serial

Dentro de los dispositivos de salida que permiten la visualización de datos podemos encontrar a la llamada pantalla de cristal líquido (LCD: *Liquid Crystal Display*). Una pantalla LCD se compone de dos placas de vidrio paralelas separadas por una capa de cristal líquido. Cuenta, además, con electrodos y filtros de polarización y una capa reflejante.

Los visualizadores LCD no emiten luz para mostrar el mensaje deseado, sino que trabajan con la luz que incide sobre ellos. Para ello aprovechan las propiedades de algunos materiales conocidos como *cristales líquidos*, que absorben o reflejan la luz dependiendo de la alineación que presentan sus moléculas. La luz suele venir incorporada en la parte posterior de la pantalla.

El cristal líquido es transparente en estado natural, en el que sus moléculas se encuentran alineadas de forma simétrica. Al aplicar un campo eléctrico a estos materiales se provocará un cambio en la alineación de su estructura, provocando que se vuelvan opacas a la luz, lo que crea un efecto de contraste. En un LCD se aplica el campo eléctrico a través de electrodos que tienen la forma de caracteres o de una matriz de pixeles en el caso del display gráfico (*GLCD Graphic Liquid Crystal Display*). En la figura 2.44 se muestra un LCD de 20x4 caracteres.

Figura 2.44. LCD de 20x4 caracteres.

Este tipo de pantallas utiliza hasta 8 líneas para recibir datos y 3 líneas de control, esto ocupa 11 terminales del microcontrolador, también tiene líneas para su alimentación y control de contraste.

Un LCD serial es básicamente el mismo LCD pero sólo usa tres cables, dos son de alimentación y uno es de datos, este utiliza entonces una terminal del microcontrolador y recibe la información de manera serial. La información serial la recibe un microcontrolador añadido y programado para tal efecto, éste transforma la información serial a la información paralela de 8 terminales o bits que requiere el LCD. El programa de este microcontrolador permite un número especial de comandos, así que se puede borrar la pantalla, ajustar el brillo de fondo, prender o apagar al LCD, entre otras cosas. En la figura 2.45 se muestra este tipo de LCD, donde se puede ver que es el mismo LCD básico, con las terminales necesarias para interactuar con él, en la parte superior derecha se pueden ver tres terminales juntas para utilizarlo como serie.

Figura 2.45. LCD serial.

La comunicación con este LCD requiere de una señal serie TTL con una velocidad de 9600bps. Se puede ajustar la velocidad de recepción para cualquier velocidad estándar entre 2400 y 38400bps.

Este capítulo trató de manera general cada uno de los conceptos, que servirán como antecedente en el desarrollo del proyecto de la presente tesis y ayudará en la comprensión del contenido de la misma. El siguiente capítulo trata lo relacionado al desarrollo del sistema adquisidor de datos automotrices.

Capítulo 3
Desarrollo del sistema

En este capítulo se abordará el diseño del sistema, considerando para ello los requerimientos y objetivos que se quieren alcanzar. Posteriormente se describen el hardware y software desarrollados.

3.1. Requerimientos

La UNAM se cuenta entre las pocas instituciones en el país que realizan la prueba de emisiones contaminantes a los vehículos de próxima introducción en el mercado nacional, muchos de los que se sumarán a los más de seis millones de automotores que circulan en la Ciudad de México, estas pruebas se realizan principalmente en el Laboratorio de Control de Emisiones (LCE) de la Facultad de Ingeniería (FI).

De acuerdo con el reglamento de las autoridades ecológicas señaladas en la Norma Oficial Mexicana 041, para acreditar la venta de automotores, las pruebas en el LCE tardaron un año, con muestreos por todo el Valle, por Naucalpan, Periférico, Aragón, Circunvalación, Eje Central, Iztapalapa y Ciudad Universitaria, para ver qué tanto influían los cambios climáticos en las emisiones y en el consumo de combustible.

Aunque hoy en día los catalizadores han mejorado considerablemente la calidad de las emisiones, según cálculos del LCE alrededor del 70 por ciento de quienes requieren el análisis de sus emisiones (un 90 por ciento empresas y el resto particulares) no aprueban el diagnóstico.

El grupo de trabajo del LCE ha desarrollado equipos como el dinamómetro de chasis, con un analizador de gases que puede simular las condiciones de operación de un vehículo, de acuerdo a cinco ciclos de manejo en distintas zonas del Valle de México, dos en el sur, dos en el norte y una en el centro. Dentro de los proyectos que se han desarrollado en el LCE se tienen:

- Instrumentación y control de un banco de ensayos para motores de combustión interna.
- Integración de un analizador de 5 gases para la medición de emisión de contaminantes.
- Instrumentación y control de un dinamómetro de chasis.

- Desarrollo de un sistema para el monitoreo de parámetros vehiculares en el desarrollo de ciclos de manejo, para autos y motocicletas.

Con la intención de actualizar dichos desarrollos; y elaborar nuevos equipos, se requiere seguir implementando y estableciendo pruebas así como sistemas para la medición de emisiones. Con base en lo anterior y aprovechando que los vehículos actuales cuentan con sistema de monitoreo a bordo, se planteó implementar un sistema para obtener datos de vehículos automotores con el sistema a bordo OBD-II, con la capacidad de comunicarse utilizando el protocolo CAN.

El sistema planteado que obtendrá datos de los vehículos debe tener la posibilidad de interactuar con un usuario, que es quien operará al sistema desarrollado; mostrar resultados y ser capaz de guardar los datos obtenidos en una memoria para su posterior análisis. Los datos registrados en la memoria deben contener fecha y hora junto con la ubicación del vehículo en prueba.

En las secciones siguientes se describen las piezas de hardware crítico para la operación del sistema, su integración y la interacción entre sí.

3.2. Hardware

El sistema para leer información de los automóviles está conformado por el hardware básico necesario para realizar su función, estos componentes son: el microcontrolador, es la parte central y gobierna al sistema, es en el que está basado el mismo, realiza el control y el procesamiento de la información de los demás componentes; un LCD serial, con el que podemos visualizar un menú de funciones, datos provenientes del automóvil, datos del GPS y estado del sistema; un receptor GPS, para obtener la información de la posición del automóvil bajo prueba en la Tierra y datos de la fecha y la hora; una tarjeta de memoria micro SD, donde se registran los datos obtenidos para su posterior análisis; un circuito integrado cambiador de nivel para adecuar los niveles de las señales de 5V, del microcontrolador, a 3.3V de la tarjeta de memoria; un Joystick, con el que podemos navegar por el menú de funciones y seleccionar alguna; un controlador CAN, éste genera y maneja las señales del protocolo CAN; un transceptor CAN que sirve de interfaz entre el controlador CAN y el bus físico acondicionando los niveles eléctricos de la señales hacia y desde el bus; un conector USB para programar al microcontrolador del sistema, para tal efecto hay un dispositivo FTDI (Future Technology Devices International) que transforma las señales del USB a señales entendibles por el microcontrolador y viceversa; un conector DB9, donde se encuentran las terminales para el bus CAN y de alimentación, en este conector se conecta un cable hacia el conector DLC del automóvil. La figura 3.1 muestra el sistema a desarrollar a nivel de diagrama de bloques.

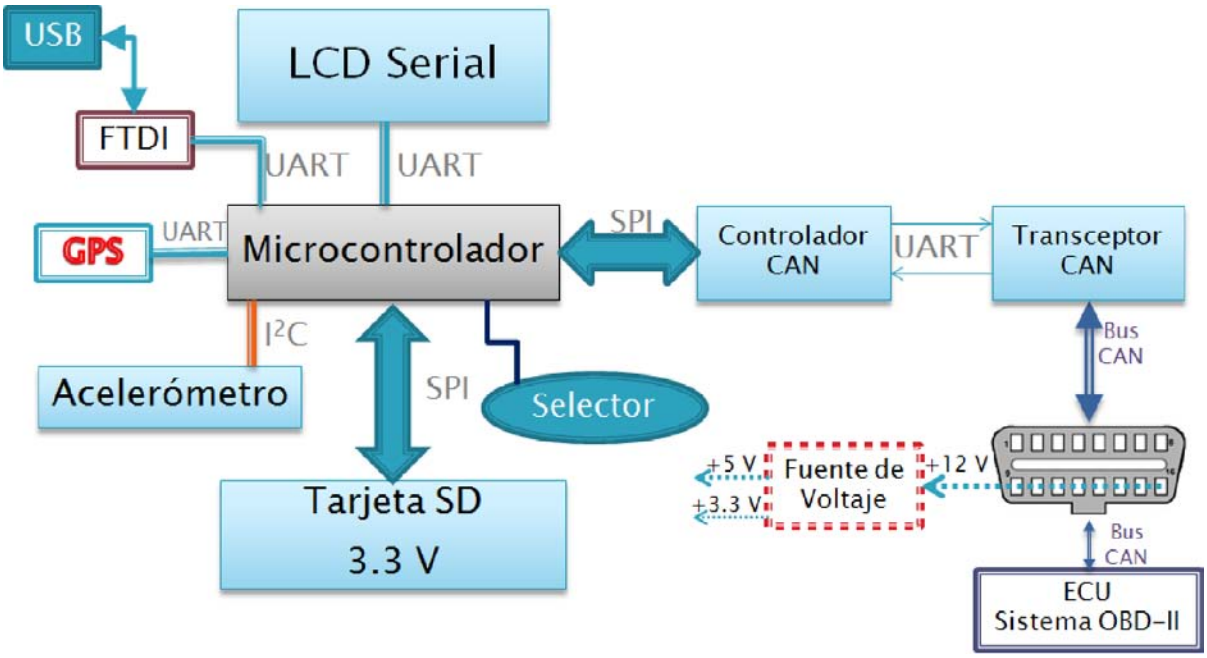


Figura 3.1. Diagrama de bloques del sistema.

3.2.1. Microcontrolador

El sistema tiene como base un microcontrolador AVR de Atmel. La decisión de utilizar este microcontrolador fue con base en los requerimientos para la primera versión del sistema, y por las siguientes razones:

Lo primero a considerar es la infraestructura con que cuenta el Laboratorio de Instrumentación del Instituto de Ingeniería, que es donde principalmente se desarrolló este trabajo. Dicho laboratorio cuenta con equipo para programar microcontroladores de las familias AVR de ATMEL y PIC de Microchip, así como microcontroladores ATmega328 y ATmega2560. Estos dos microcontroladores de Atmel también tienen la ventaja de tener una amplia disponibilidad en el mercado. El otro aspecto que se consideró fue que en el Instituto de Ingeniería ya se tiene experiencia trabajando con los microcontroladores PIC, se han desarrollado trabajos recientemente con los microcontroladores AVR, por lo que se requiere seguir generando experiencia con éstos últimos.

Tomando en cuenta las consideraciones anteriores y a las necesidades del proyecto, se seleccionó un microcontrolador AVR para trabajar con él. En la figura 3.2 se muestran dos empaquetados del un AVR, el tipo DIP (Dual In-line Package) es con el que se trabajó y el tipo TQFP (Thin Quad Flat Package) es el que se montó en el sistema.

Figura 3.2. AVR, empaquetado DIP (izquierda) y TQFP (derecha).

Las características principales de este microcontrolador están resumidas en la tabla 3.1:

Características del microcontrolador AVR seleccionado	
Frecuencia del reloj	Hasta 20 MHz
Memoria de programa	32 kBytes
Memoria RAM	2 kBytes
Memoria EEPROM	1 kByte
CPU	8 bits
Puertos de entrada/salida programables	23
Módulo *Master Synchronous Serial Port*	I^2C y SPI
Comunicación serie o módulo USART	1
Set de instrucciones	131 instrucciones RISC
Voltaje de alimentación	1.8 - 5.5 V
Número de terminales	28 en DIP y 32 en TQFP
Consumo de energía @ 1MHz, 1.8V, 25ºC	0.2 mA en modo activo
Rango de temperatura	-40 ºC a 85 ºC

Tabla 3.1. Características del microcontrolador seleccionado.

En la figura 3.3 se muestra la distribución de las terminales del microcontrolador AVR para el empaquetado DIP y TQFP.

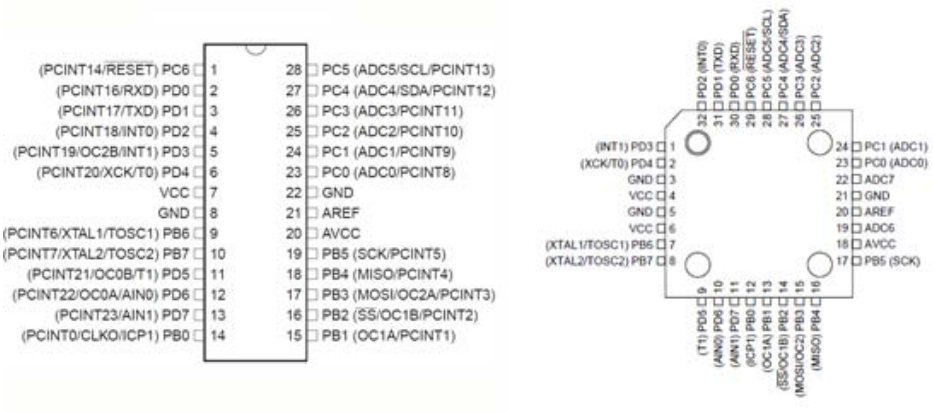


Figura 3.3. Distribución de terminales del ATmega DIP y TQFP.

3.2.2. LCD Serial

Se requiere que el sistema muestre resultados de las lecturas de los parámetros del vehículo en prueba, además el sistema también debe contar con un menú de funciones, por lo que tiene un LCD para dichas visualizaciones. Se eligió un LCD serial, de color negro sobre verde (caracteres negros, fondo verde) y puede mostrar un arreglo de 20X4 caracteres, su alimentación es de 5V y cuenta con una luz posterior producida por un LED.

En un mismo módulo está integrado el LCD 20x4 y el PIC 16F88. El PIC integrado recibe la señal serial TTL (Transistor-Transistor Logic) e imprime los caracteres que recibe en la pantalla. El firmware cargado también permite un número de comandos especiales, como por ejemplo para borrar la pantalla, ajustar el brillo de la luz, prender y apagar la pantalla, entre otras funciones.

La comunicación con el LCD serial requiere de una señal TTL con una velocidad de comunicación predeterminada de 9,600bps. Se puede ajustar la velocidad de entre 2,400 a 38,400bps. Este LCD tiene tres terminales, la alimentación (VDD), tierra (GND) y la recepción (RX).

Como se mencionó anteriormente, el LCD serial tiene un LED para iluminarlo desde atrás; existe un comando para atenuar la luz de este LED para conservar la energía cuando se requiera, también hay un potenciómetro en la parte posterior de la pantalla para ajustar el contraste.

El uso de este LCD fue de gran utilidad en esta primera fase del sistema porque sólo requiere de una terminal del microcontrolador, en lugar de las 11 que requieren otros para recibir los mensajes que se van a desplegar; además de que se ahorran instrucciones de control del LCD en el microcontrolador principal, por quedar a cargo del PIC integrado.

La alimentación del LCD deber sólo de 5V de corriente directa, con un voltaje mayor a 5.5V puede dañar al PIC, al LCD o al LED. La corriente consumida de este LCD es de 3mA, cuando la luz está apagada, y de aproximadamente 60mA con la luz activada totalmente. En la figura 3.4 se ve un ejemplo de mensaje en el LCD.

Figura 3.4. LCD con un mensaje.

3.2.3. Receptor GPS

Además de la información del vehículo en prueba, debe haber datos de la posición, fecha y hora en formato UTC de la prueba. Es por ello que el sistema debe ser capaz de obtener esos datos sin ser más complejo, por requerir otro tipo de conexiones, razón por la cual se optó por un receptor GPS. El receptor tiene un tamaño reducido, bajo costo y para su conexión al microcontrolador requiere de tres terminales. En la tabla 3.2 se resumen las características del receptor GPS utilizado para el sistema.

Características del receptor GPS	
Exactitud	**Posición:** 10 metros. **Velocidad:** 0.1 m/s. **Tiempo:** 1us (sincronizado con el reloj del GPS).
Tiempo de adquisición	**Readquisición:** 0.1 s en promedio **Arranque en frio:** 42 s en promedio **Arranque en tibio:** 38 s en promedio **Arranque en caliente:** 1 s en promedio
Condiciones dinámicas	**Altitud:** 18,000 metros. **Velocidad:** 515 m/s máximo **Aceleración:** Menos de 4g
Alimentación	**Voltaje de alimentación:** 4.5-6.5V **Corriente:** 44mA
Protocolo	**Niveles eléctricos:** Nivel TTL **Taza de comunicación:** 4,800bps **Mensajes de salida:** NMEA 0183 GGA, GSA, GSV, RMC, VTG, GLL

Tabla 3.2. Características del receptor GPS.

Este tipo de receptor GPS trabaja las cadenas de datos bajo el protocolo NMEA 0183; para procesar esta información es necesario conectar un microprocesador con el receptor GPS, por medio de la UART, con una velocidad de 4,800bps, como lo dicen las especificaciones. La figura 3.5 ilustra al conector del receptor GPS y sus respectivos usos.

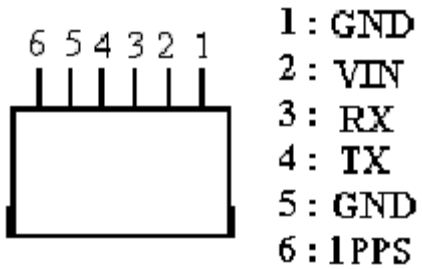


Figura 3.5. Conector del receptor GPS y sus terminales.

VIN: Es la entrada principal de la alimentación de 4.5-5.6V.
TX: Es el canal de salida principal de transmisión para los datos de posición y velocidad.
RX: Es el canal principal de recepción de comandos.
GND: Provee de conexión a tierra.
1PPS: Esta terminal provee un pulso por segundo

El receptor tiene incorporado un LED que, cuando prende, indica que hay voltaje de alimentación, después, cuando su luz es intermitente indica que recibe información de por lo menos tres satélites, entonces podemos obtener la información de la posición. La figura 3.6 muestra al receptor GPS utilizado en el sistema.

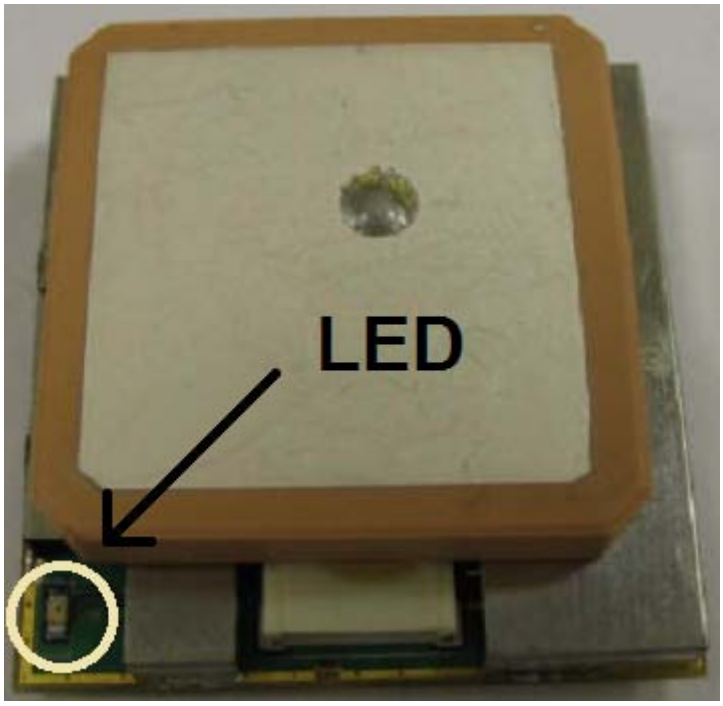


Figura 3.6. Receptor GPS, se muestra el LED integrado.

3.2.4. Tarjeta micro SD

Para leer y escribir datos en las tarjetas de memoria SD se requiere conocer sus características principales, que incluyen: la disposición de terminales, voltaje de alimentación, protocolos y topologías de comunicación.

En el subcapítulo 2.4.1 se explican las características importantes de la tarjeta de memoria SD, algunas de esas características se usaron en el desarrollo del sistema, la más destacada es la comunicación por SPI.

Alimentación y habilitación de la tarjeta SD

Las tarjetas de memoria SD trabajan con un voltaje de 3.3V para su alimentación y para la comunicación con el microcontrolador. El voltaje de alimentación se obtiene de un regulador de 3.3V integrado en la tarjeta del sistema; para la comunicación con el microcontrolador, debido a que éste provee señales de 5V en sus terminales digitales, se utilizó el cambiador de nivel a 3.3V. La figura 3.7 muestra la conexión descrita.

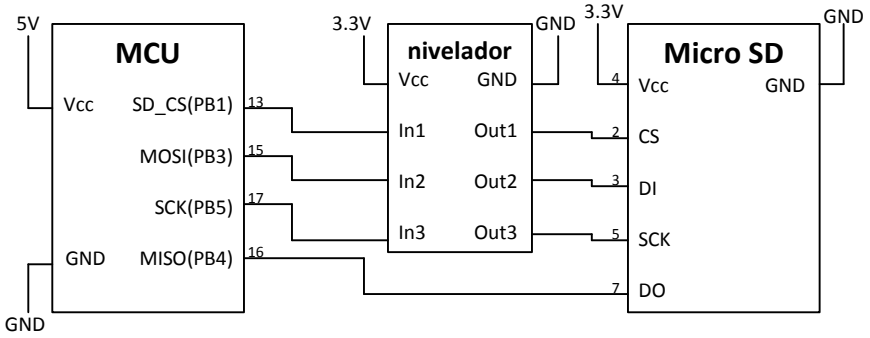


Figura 3.7. Diagrama de conexión de la memoria SD con el microcontrolador.

Para interactuar con la tarjeta de memoria SD se usó el protocolo SPI y las terminales del microcontrolador designadas para ello, y la habilitación o selección de la tarjeta SD se lleva a cabo por una terminal digital nombrada SD_CS (SD Chip Select) del microcontrolador hacia la terminal SC (Chip Select) de la tarjeta SD.

3.2.5. Joystick

El *joystick* del sistema es un interruptor de cinco vías, que se encuentra dentro de un encapsulado pequeño. Este dispositivo nos permite desplazarnos a través de los diferentes menús que se han programado en el microcontrolador y que son mostrados a través del *display*, con la finalidad de seleccionar alguna función, entre las cuales se encuentra la función de adquisición de datos de los vehículos. En la figura 3.8 se muestran las terminales del microcontrolador que están conectadas a las terminales del *joystick*, con resistencias de *pull-up,* por lo que al oprimir alguna de las cinco posiciones del joystick el microcontrolador detectará un estado lógico bajo y efectuará la operación programada.

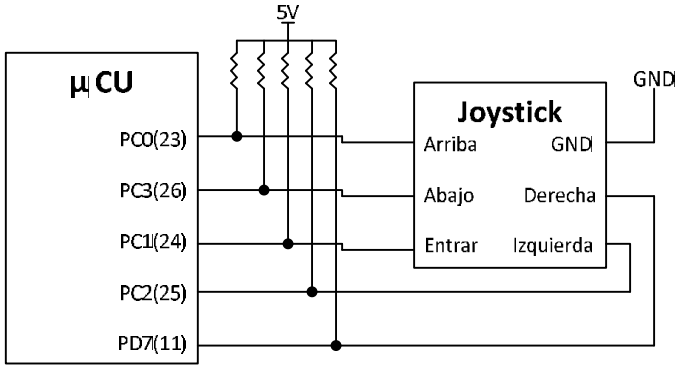


Figura 3.8. Diagrama de conexión del Joystick con el microcontrolador.

3.2.6. Controlador CAN

Con el fin de implementar el protocolo CAN, se utilizó un controlador CAN de Microchip. Este controlador consiste en un circuito integrado de 18 terminales con empaquetado SOIC (Small Outline Integrated Circuit) que es de montaje superficial.

El circuito integrado es un controlador CAN independiente o autónomo que implementa la versión 2.0B de CAN (29 bits en el campo de arbitraje). Es capaz de transmitir y recibir tramas de datos estándar y extendidas, tiene seis filtros de aceptación que se usan para filtrar mensajes no deseados y de esta manera se reduce la sobrecarga del microcontrolador principal. El circuito integrado se comunica con el microcontrolador por medio del protocolo SPI.

Las principales características del controlador CAN son:

- Implementa CAN versión 2.0B a 1 Mb/s máximo
  - Longitud del campo de datos de 0 – 8 bytes.
  - Tramas estándar y extendidas.
- Buffer de recepción, máscaras y filtros
  - Dos buffers de recepción con almacenamiento de mensaje de priorización.
  - Seis filtro de 29 bits.
  - Dos máscaras de 29 bits.
- Interfaz de alta velocidad con SPI (10MHz)
  - Modos SPI 0,0 y 1,1.
- Bajo consumo
  - Opera de 2.7V – 5.5V.
  - 5 mA en modo activo (típico).
  - 1 μA en espera (típico).

La figura 3.9 muestra la distribución de las terminales del circuito integrado.

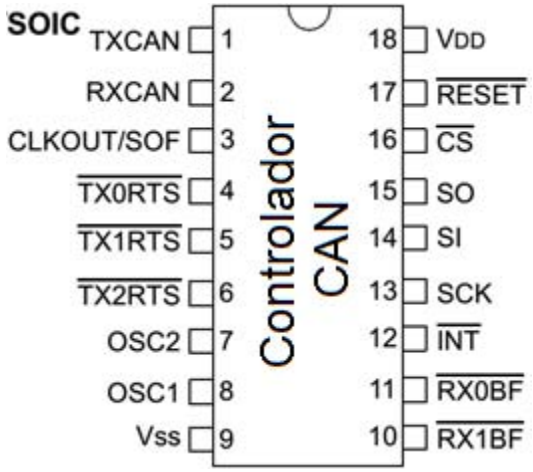


Figura 3.9. Configuración de terminales del integrado con encapsulado SOIC.

El circuito CAN fue desarrollado para simplificar las aplicaciones que requieren conexión con un bus CAN, la figura 3.10 muestra su diagrama de bloques. El diagrama podemos observar que el dispositivo consiste de tres bloques principales:

1. El módulo CAN, el cual incluye al motor del protocolo CAN, máscaras, filtros y buffers de transmisión y recepción.
2. El control lógico y registros que se usan para configuración del dispositivo y su operación.
3. El bloque del protocolo SPI.

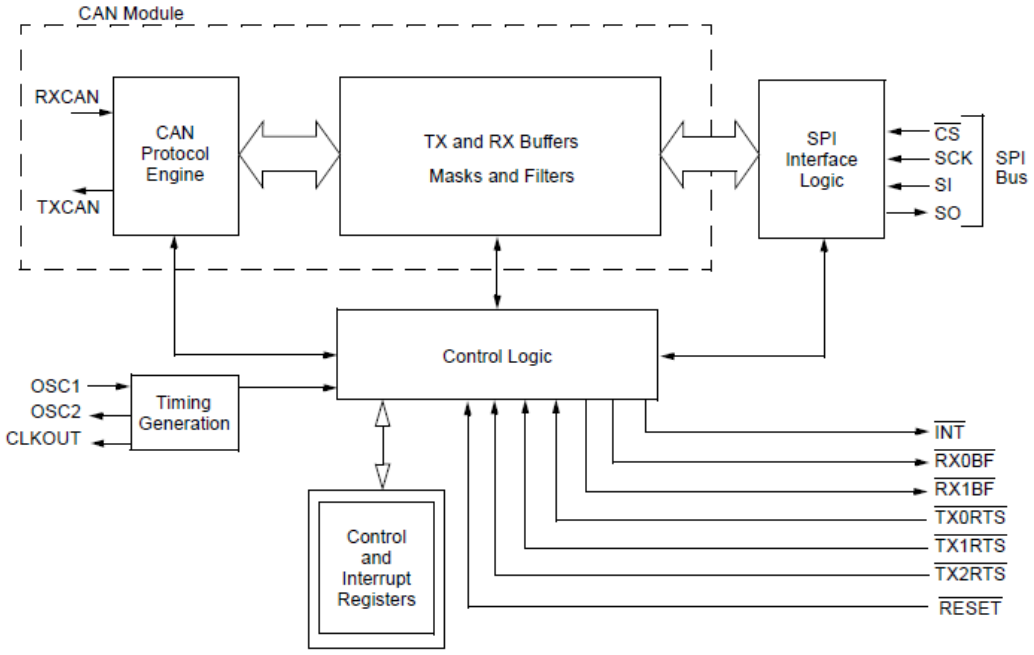


Figura 3.10. Diagrama de bloques del circuito integrado CAN.

3.2.7. Transceptor CAN

El transceptor CAN utilizado en el desarrollo del sistema es un circuito integrado. El circuito integrado es tolerante a fallos (hasta 125 kbit/s) y CAN de alta velocidad (hasta 1 Mbit/s), éste sirve de interface entre un controlador de protocolo CAN y el bus físico. Dicho circuito provee capacidad de transmisión y recepción diferencial para el controlador de protocolo CAN y es completamente compatible con el estándar ISO-11898 y opera a velocidades de hasta 1 Mb/s.

Típicamente, cada nodo en un sistema con comunicación CAN debe tener un dispositivo para convertir las señales digitales generadas por el controlador CAN a señales adecuadas para la transmisión sobre los alambres del bus (salida diferencial). También proporciona amortiguación entre el controlador CAN y los picos de alto voltaje que se pueden generar en el bus CAN debidos a fuentes externas (EMI – Electromagnetic Interference, ESD – Electrostatic Discharge, transitorios eléctricos, etc.).

Función del Transmisor

Como hemos visto, el bus CAN tiene dos estados: Dominante y Recesivo. Un estado dominante ocurre cuando el voltaje diferencial entre CANH y CANL es más grande que el voltaje umbral (> 1.2 V). Un estado recesivo ocurre cuando el voltaje diferencial es menor que el voltaje umbral (0 V). Los estados dominante y recesivo corresponden al estado bajo y alto respectivamente de la terminal de entrada TXD.

El circuito integrado puede manejar una carga mínima de 45Ω, lo que permite conectar un máximo de 112 nodos (dada una mínima resistencia diferencial de entrada de 20kΩ y un valor nominal de resistor de terminación de 120Ω).

Función del Receptor

La terminal de salida RXD refleja el voltaje diferencial del bus entre CANH y CANL. Los estados bajo y alto de la terminal RXD corresponden a los estados dominante y recesivo respectivamente del bus CAN.

La figura 3.11 muestra los niveles nominales de las líneas del bus y su diferencia, así como su tolerancia de salida y entrada de los dispositivos.

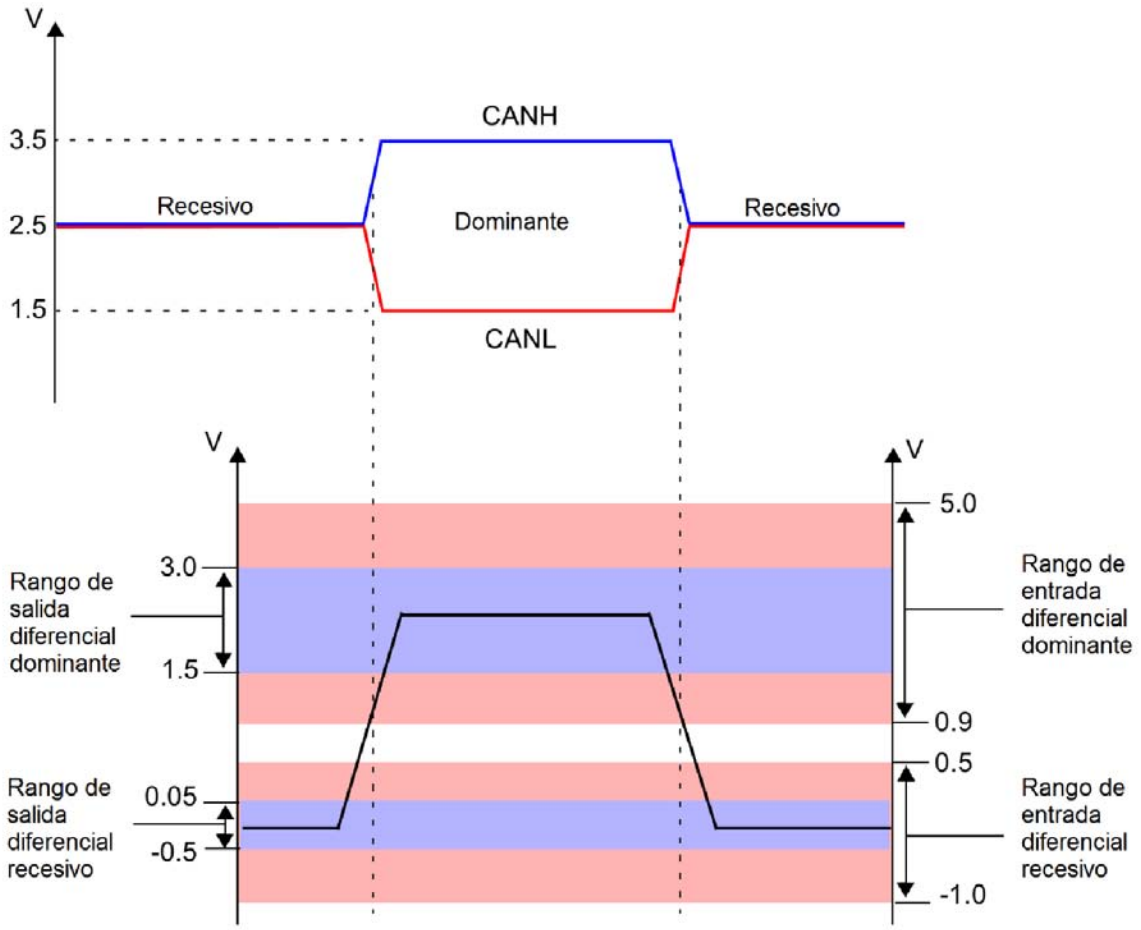


Figura 3.11. Valores nominales del bus CAN.

La norma ISO-11898-2 no especifica las características mecánicas de los cables ni de los conectores. Sin embargo, la especificación requiere que los cables y conectores cumplan con la especificación eléctrica. La especificación también requiere un resistor terminal de 120Ω (nominal) en cada extremo del bus.

Este mismo estándar especifica que el transceptor debe ser capaz de manejar un bus de 40m a 1 Mb/s. Una longitud mayor puede lograrse reduciendo la velocidad de transmisión. La limitación más grande para la longitud del bus es el retraso de propagación del transceptor.

Protección interna

CANH y CANL están protegidos contra cortocircuitos de hasta ±40 V y de voltajes transitorios de hasta ±250 V que pueden ocurrir en el bus CAN. Esta característica previene la destrucción de la etapa de salida del transmisor durante tal condición de fallo.

El dispositivo también está protegido de la corriente excesiva de carga, por un circuito de apagado térmico que deshabilita los controladores de salida cuando la temperatura de unión excede la temperatura límite nominal de 165ºC. Todas las partes restantes del dispositivo siguen operando y la temperatura del circuito integrado baja debido a que decrece la disipación de potencia en la salida del transmisor. Esta protección es esencial para proteger contra daños por cortocircuitos inducidos en la línea del bus. En el diagrama de bloques de la figura 3.12 se visualizan cada uno de los módulos del circuito, destacando entre ellos los de protección.

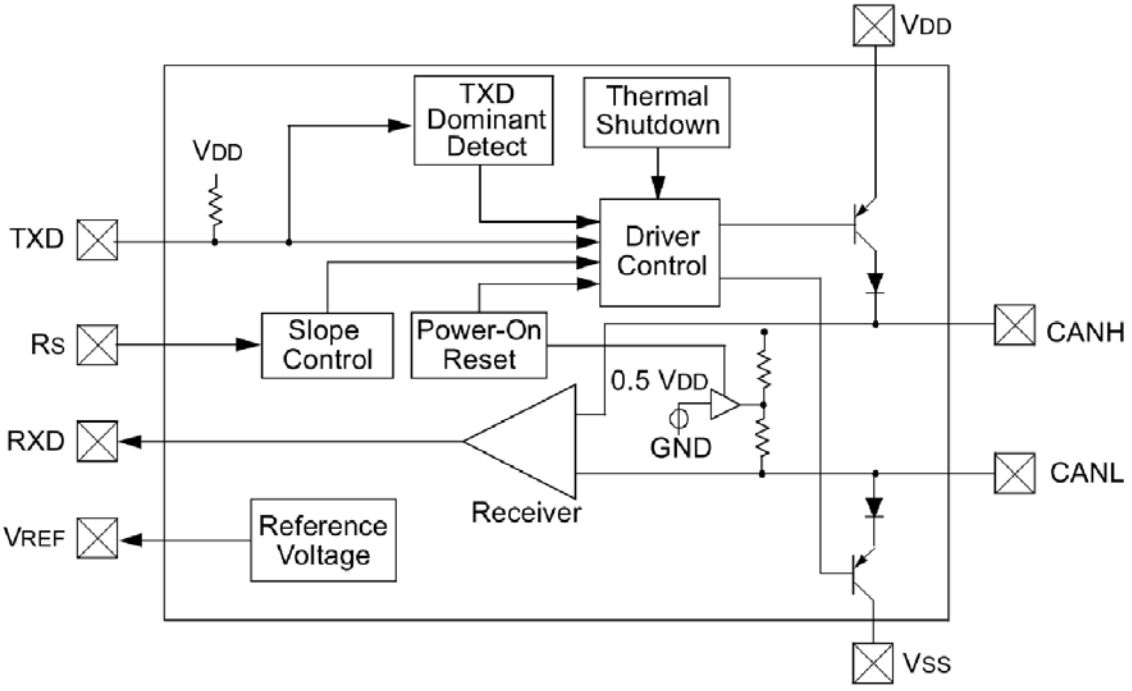


Figura 3.12. Diagrama de bloques del transceptor CAN.

Entrada de datos del transmisor (TXD)

TXD es una terminal de entrada compatible con TTL. Los datos colocados en esta terminal son enviados por las terminales de salida diferencial CANH y CANL. La terminal TXD se conecta con la salida de datos del transmisor del dispositivo controlador CAN. Cuando TXD está en bajo, CANH y CANL están en estado dominante. Cuando TXD está en alto, CANH y CANL están en estado recesivo, con la condición de que otro nodo no esté ocupando al bus con un estado dominante. TXD tiene un resistor de pull-up interno (nominal de 25kΩ a V_{DD}).

Salida de datos del receptor (RXD)

RXD es una salida compatible con CMOS, que se pone en alto o bajo dependiendo de las señales diferenciales en las terminales CANH y CANL. Esta terminal se conecta a la entrada de datos del receptor del dispositivo controlador CAN. RXD está en alto cuando el bus CAN está en recesivo y en bajo en el estado dominante.

CAN LOW (CANL)

La salida CANL controla el lado bajo del bus CAN. Esta terminal también está vinculada internamente con la entrada del comparador de recepción.

CAN HIGH (CANH)

La salida CANH controla el lado alto del bus CAN. Esta terminal también está vinculada internamente con la entrada del comparador de recepción.

Detección de TXD Dominante Permanente

La detección de la terminal TXD en estado dominante permanente sirve para proteger las señales del bus. Si el transceptor detecta un estado bajo extendido en la entrada TXD, deshabilitará a los controladores de las salidas CANH y CANL para prevenir la corrupción de los datos en el bus CAN. Los controladores están deshabilitados si TXD está en bajo por más de 1.25μs (mínimo). Esto implica un máximo tiempo de bit de 62.5μs (tasa de bus de 16kb/s), permitiendo hasta 20 bits consecutivos transmitidos como dominantes durante múltiples escenarios de error de bit y error de trama. Los controladores permanecen deshabilitados mientras que TXD permanezca en bajo. Un flanco de subida en TXD restablecerá la lógica del temporizador y habilita a los controladores de salida de CANH y CANL.

Cuando el dispositivo se prende, CANH y CANL permanece en estado de alta impedancia hasta que V_{DD} alcanza el nivel de voltaje V_{PORH} (Voltaje alto del reinicio de encendido). Además, CANH y CANL permanecerán en estado de alta impedancia si TXD está en bajo cuando V_{DD} alcance V_{PORH}. CANH y CANL se activarán sólo después de que en TXD se asegure un estado alto. Una vez prendido, CANH y CANL entrarán en estado de alta impedancia, esto sucede el nivel del voltaje en V_{DD} cae por debajo de V_{PORL} (Voltaje bajo por reiniciar el encendido), proporcionando detección de apagado durante la operación normal. La operación de detección de la terminal TXD se muestra gráficamente en la figura 3.13.

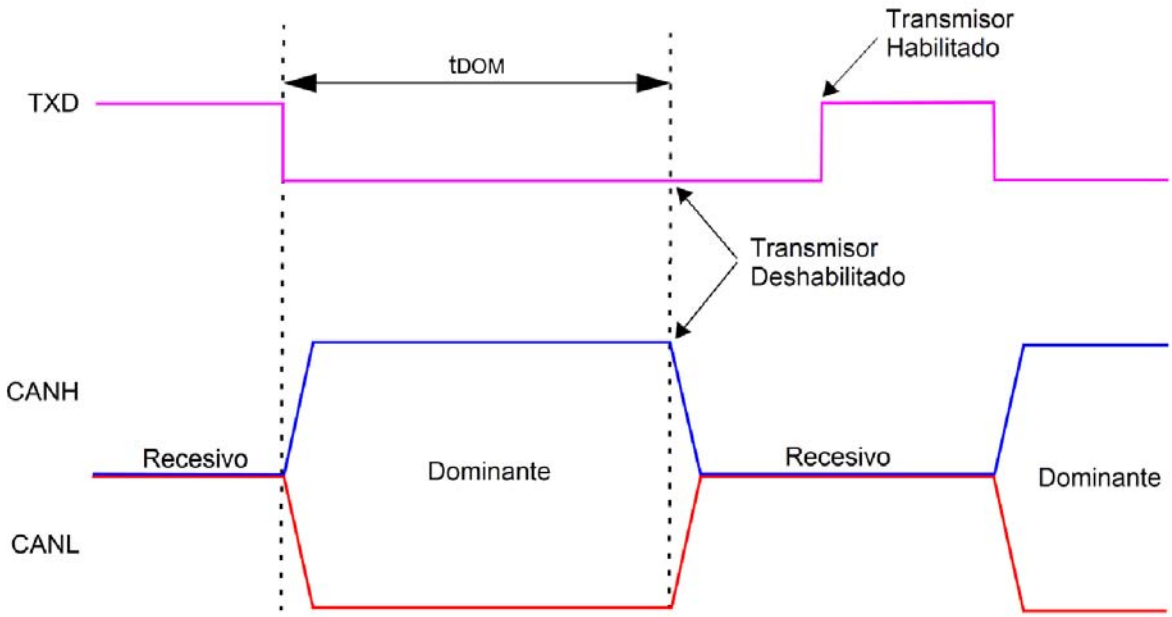


Figura 3.13. Detección de TXD en estado dominante permanente.

3.2.8. Conector USB

El sistema utilizará un conector USB tipo B hembra, como el que se muestra en la figura 3.14.

Figura 3.14. Cable y conector USB tipo B.

El conector USB con el que contará el sistema sirve para programar al microcontrolador principal, utilizando para esto un FTDI como interfaz con el UART del microcontrolador. El sistema obtiene también por este conector su alimentación, con esta característica además de programar al sistema se podrá utilizar para realizarle pruebas.

3.2.9. FTDI

Future Technology Devices International, comúnmente conocida por su abreviatura **FTDI**, es una empresa privada escocesa de dispositivos semiconductores, especializada en tecnología Universal Serial Bus (USB). Desarrolla, fabrica, y da apoyo a dispositivos y sus correspondientes controladores de software (drivers) para la conversión de transmisiones serie RS-232 o TTL a señales USB.

Se usará un dispositivo FTDI (figura 3.15) como interfaz USB a UART, dicho dispositivo maneja todo el protocolo USB sin necesidad de programarle un *firmware*, no requiere un cristal externo ya que tiene un reloj integrado.

Figura 3.15. Dispositivo FTDI.

3.2.10. Conector DB9

El conector DB9, mostrado en la figura 3.16, es un conector de 9 terminales de la familia de conectores D-Subminiature (D-Sub o Sub-D). El conector DB9 se utiliza principalmente para conexiones en serie, ya que permite una comunicación de datos asíncrona. Este conector proveerá una conexión del sistema con el automóvil por medio del cable DB9-OBDII como el de la figura 3.17. A través de las terminales de estos conectores se obtiene la alimentación para la tarjeta.

Figura 3.16. Conector DB9 macho.

Figura 3.17. Cable DB9 a OBD-II.

En la tabla 3.3 se enlistan las terminales del conector y cable que son utilizados, en la cual las "negritas" indican las terminales que se usarán en la tarjeta del sistema a desarrollar.

Uso de la terminal	OBDII	DB9
Bus J1850 +	2	7
Tierra del chasis	**4**	**2**
Negativo de la batería	**5**	**1**
CAN alto	**6**	**3**
Línea K del ISO 9141	7	4
Bus J1850 -	10	6
CAN bajo	**14**	**5**
Línea L del ISO 9141	15	8
Positivo de la batería	**16**	**9**

Tabla 3.3. Descripción de las terminales del cable DB9 a OBD-II.

3.2.11. Integración del sistema en una sola tarjeta

Para integrar todos los componentes del sistema se diseñará una PCB (Printed Circuit Board), pero antes se realizará una lista de los componentes que la conformarán, en la que se agrupan los componentes de acuerdo a “zonas”, para tener una organización al realizar la placa y a la hora de soldar los componentes. Como ejemplo de lo anterior, se necesitan: *headers* dobles y un conector USB tipo B hembra, para la programación del sistema; regulador de 5 y 3V, 2 capacitores de 47 uF para la fuente de alimentación; LEDs con sus resistencias como indicadores varios; conectores y terminales que sirven de entrada y salida de los datos de y hacia los periféricos como el GPS, LCD, Joystic, memoria micro SD, y Bus CAN.

Después de haber analizado y comprobado el funcionamiento básico de cada uno de los bloques y elementos que conforman al sistema, se procedió a montarlas en tarjetas, una de desarrollo donde se encuentra el microcontrolador y una de apoyo (shield), con la finalidad de corroborar el funcionamiento de todos los bloques en conjunto, además de ello, esto nos permitió realizar pruebas adicionales conforme se avanzaba en el desarrollo del prototipo. Las tarjetas mencionadas se muestran en la figura 3.18, montadas en una base de PCV espumado para manipularlas de forma fácil y segura.

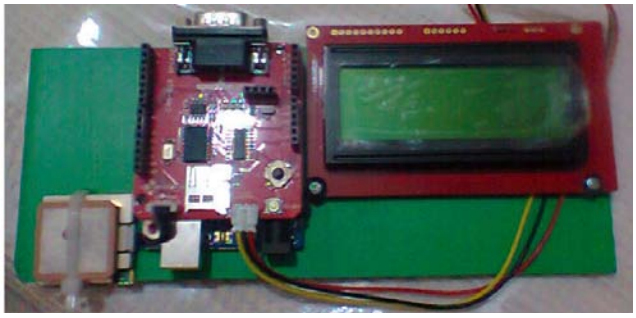


Figura 3.18. Tarjetas y componentes montados para realizar pruebas.

A manera de ejemplo se presentarán algunos de los diseños desarrollados: En la figura 3.19 se muestra el circuito de entrada del sistema por USB con el FTDI cuyas líneas TX y RX se conectan al microcontrolador.

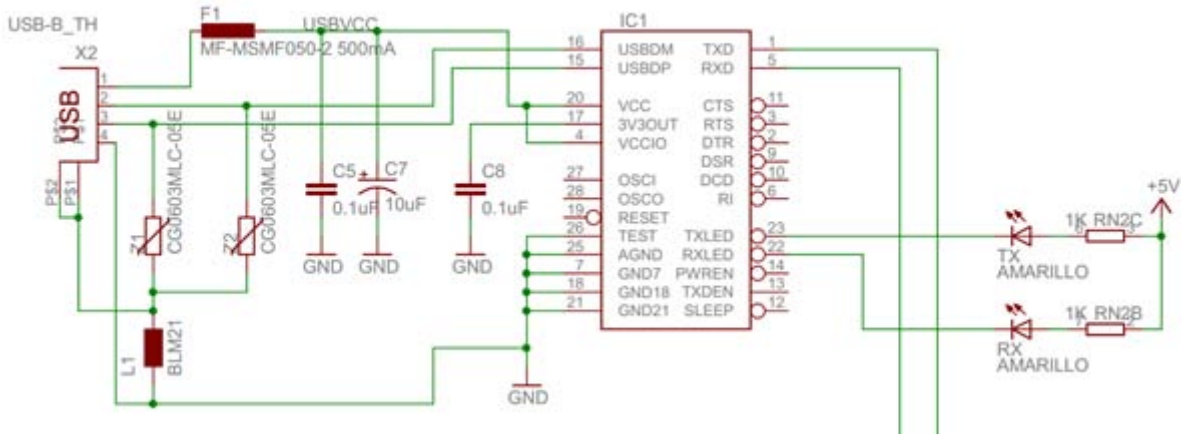


Figura 3.19. Diagrama eléctrico del USB y FTDI.

En la figura 3.20 se muestra el diagrama eléctrico alrededor del microcontrolador, en la que se observa el conector (ICSP – In-Circuit Serial Programming) para un programador externo.

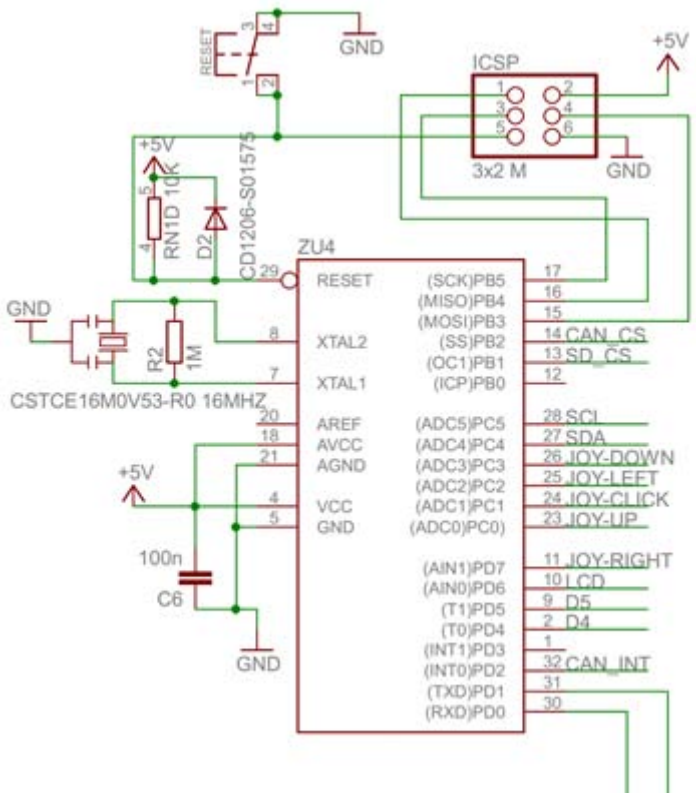


Figura 3.20. Conexión eléctrica con del microcontrolador.

PCB del sistema

Al llegar a la etapa final del desarrollo del sistema, después de comprobar que su funcionamiento es adecuado, y que no se requiere de ningún otro componente de hardware, se procedió al diseño e integración del sistema en una tarjeta de circuito impreso o PCB, un diseño como primer prototipo funcional.

Algunas de las consideraciones que se tomaron para el diseño del circuito impreso son:

- La ubicación de los componentes se realizó con la intención de hacer lo más compacto posible el prototipo.
- Debido a que se realizará un circuito impreso de doble cara, se optó por colocar en la capa superior los componentes y dispositivos con los que el usuario interactúa con el sistema, como por ejemplo: el *display*, el *joystick*, el conector de la memoria micro SD, componentes para la alimentación, así como los circuitos para la comunicación con bus CAN.
- Se colocaron indicadores (*LEDs*) para dar a conocer al usuario el estado del sistema como: encendido, comunicación serial y guardado de datos en la memoria micro SD.

La figura 3.21 se muestra la vista superior del circuito impreso y en la figura 3.22 la vista inferior.

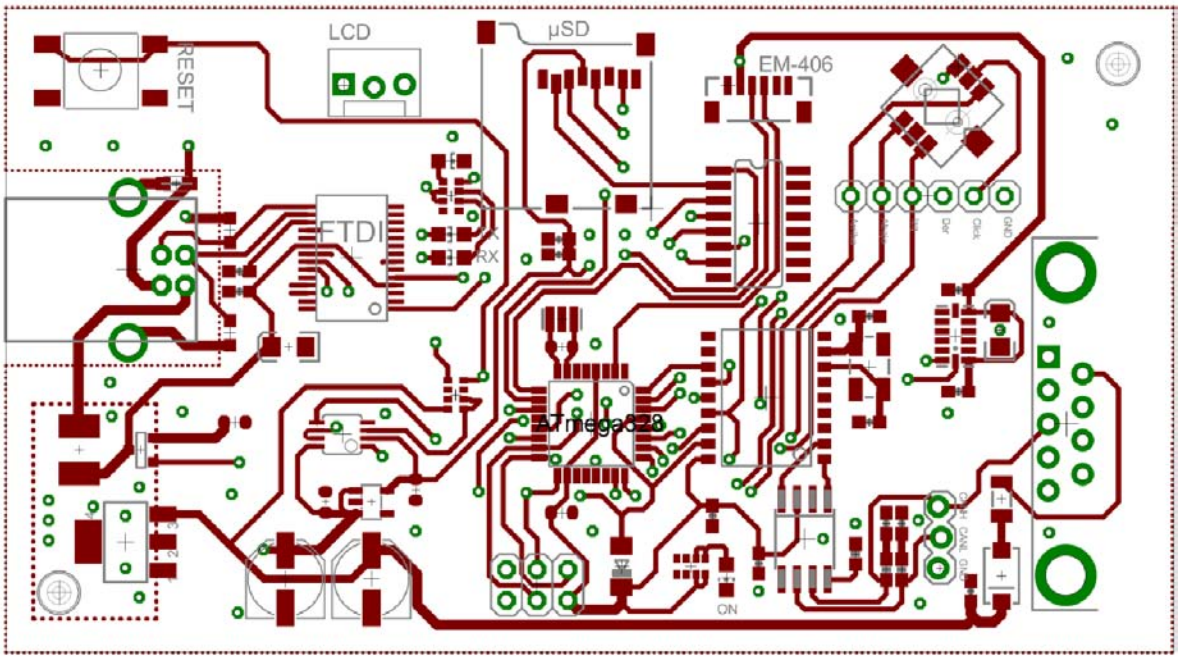


Figura 3.21. Vista superior del circuito impreso del sistema.

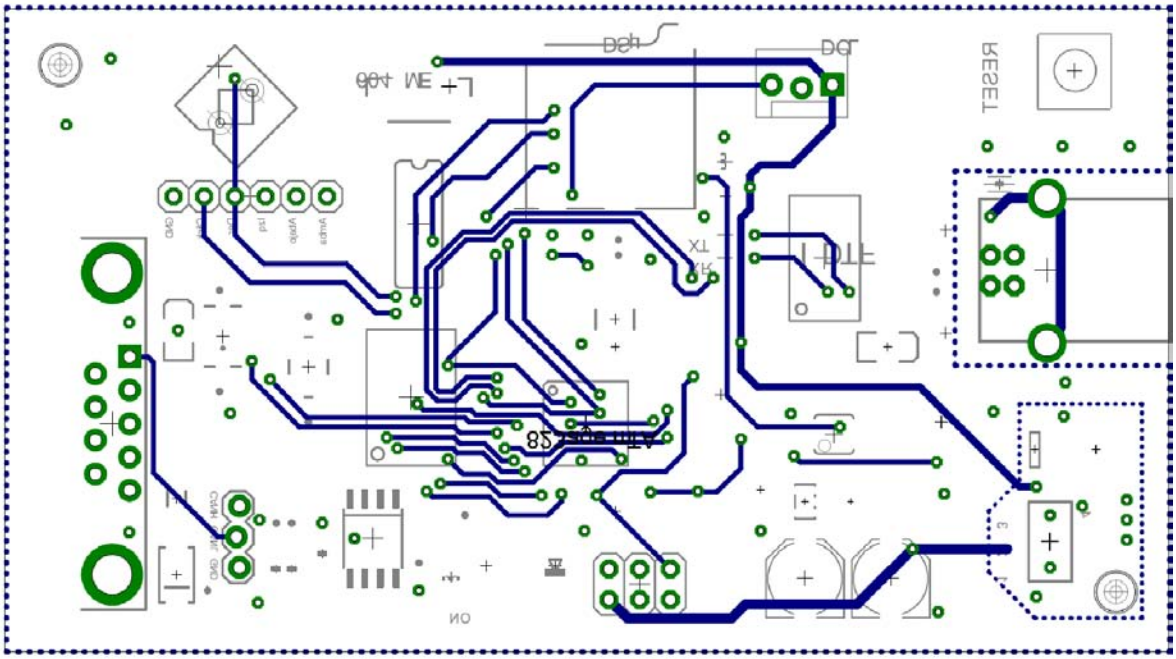


Figura 3.22. Vista inferior del circuito impreso del sistema.

3.3. Software

Para la elaboración del programa del microcontrolador, la compañía Atmel pone a disposición de los usuarios una plataforma de desarrollo para sus dispositivos; esta herramienta es ATmel Studio, la cual permite programar, depurar, simular y grabar el código al microcontrolador. La herramienta permite el uso de lenguajes de programación tales como: ensamblador, C y C++. Sin embargo, para el desarrollo del programa del sistema en cuestión, se optó por la plataforma de desarrollo de Arduino (tarjeta y ambiente de desarrollo), debido a que tiene implementada una librería que permite el almacenamiento de datos en las memorias SD estándar y micro SD. Las funciones implementadas en esta librería facilitan las tareas necesarias para la creación de archivos y directorios, así como el almacenamiento de información, y cumple con las especificaciones de los sistemas de archivos FAT16 y FAT32.

3.3.1. Estructura del programa del microcontrolador

La estructura del programa que se desarrolló está de acuerdo a las necesidades del usuario y con base en el *hardware* propuesto. A continuación, de manera general, se plantean las tareas que deberán programarse en el microcontrolador a las que llamaremos programa o *firmware*:

- Configuración del controlador CAN:
  - Establecer la comunicación por SPI.
  - Establecer la velocidad de transmisión de datos.

  - Verificar que la configuración sea correcta.

- Almacenamiento de datos en la tarjeta de memoria SD:
  - Creación de archivos.
  - Estructura del encabezado de los archivos.
  - Procesamiento y formato de datos para ser guardados.

- Comunicación por bus CAN del automóvil:
  - Comunicación por SPI con el controlador CAN.
  - Configuración e inicialización del controlador CAN.
  - Enviar y recibir datos del bus CAN.

- Manejo del teclado:
  - Detectar la tecla pulsada.
  - Manejar la correspondencia entre tecla pulsada y mensaje en el display.
  - Asignar una tarea de acuerdo a la tecla pulsada.

- Manejo del display LCD:
  - Mostrar menú de opciones.
  - Indicar la selección de una opción.
  - Indicar a través de mensajes el estado de algún proceso en curso.
  - Mostrar resultados de los distintos procesos.

3.3.2. Programa principal

El programa del microcontrolador está desarrollado de manera que cada una de las tareas o proceso mencionados pueden ser llamados como una subrutina. Al usuario se le presenta un menú de opciones en el que se puede desplazar y seleccionar una función con ayuda del joystick.

El programa general, del cual se muestra el diagrama de flujo en la figura 3.23, realiza la configuración de los puertos del microcontrolador, la inicialización del controlador CAN y la de la comunicación serial con los dispositivos externos. El programa general incluye la realización y muestra del menú y el uso del Joystick.

El programa principal debe ejecutarse al energizar al microcontrolador, después debe ejecutarse la parte del menú para seleccionar una subrutina. Al finalizar cada subrutina el programa debe regresar a ejecutar el programa principal. Es importante mencionar que en caso de oprimir el *reset* del sistema, el programa principal se ejecutará desde el principio, también lo hará al salir de la subrutina de guardar.

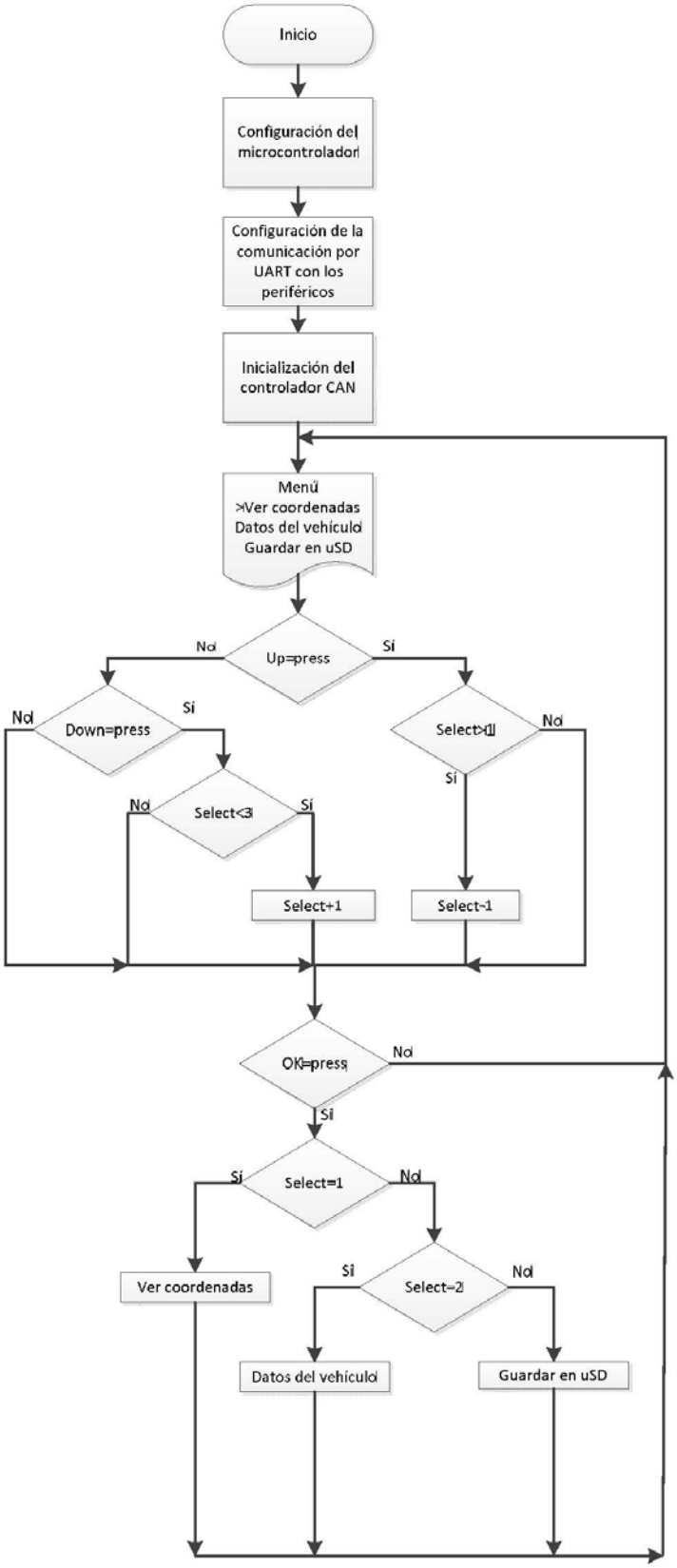


Figura 3.23. Diagrama de flujo del programa principal.

3.3.3. Leer y mostrar datos del GPS

En el caso de la subrutina de leer y mostrar datos del GPS, lo primero que debe hacer ésta, para leer los datos del GPS, es esperar la sincronización del receptor GPS con los satélites, mientras espera mostrará un mensaje en el LCD que indique que está en espera. Cuando ocurre la sincronización, el programa lee los datos disponibles en el módulo UART y verifica que tengan el comando $GPRMC, dado esto, entonces podrá obtener los datos de la cadena correspondientes a las coordenadas, la fecha y la hora y las despliega en el LCD. Este programa se ejecutará cíclicamente hasta que el usuario lo detenga oprimiendo el centro del Joystick, en ese momento se ejecutará el programa principal nuevamente. La figura 3.24 es el diagrama de flujo de la subrutina para leer y mostrar los datos del GPS.

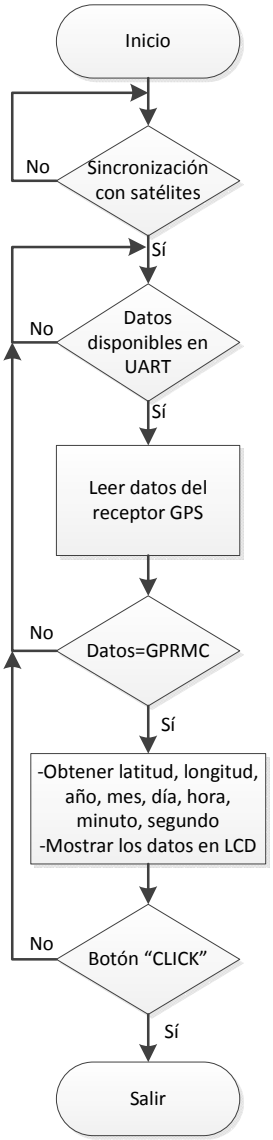


Figura 3.24. Diagrama de flujo para obtener y mostrar datos del GPS.

La velocidad de transmisión de datos de la UART para comunicarse con el dispositivo receptor GPS debe ser igual a 4,800 bauds, este valor es el que, según la hoja de especificaciones del receptor, está por defecto. Teniendo hecha esta configuración, el programa tendrá que leer de forma continua los datos de dicho puerto, sin embargo, para almacenar el mensaje RMC (Recommended Minimum Specific Data) se hace un recorrido de la cadena almacenada, buscando que el identificador de la cadena (los primeros seis caracteres comenzando con el signo $) tenga el comando *"$GPRMC"*, con esta acción se garantizará que se ha almacenado el mensaje RMC.

3.3.4. Leer y mostrar datos del vehículo

La figura 3.25 corresponde al diagrama de flujo para la lectura de datos del vehículo.

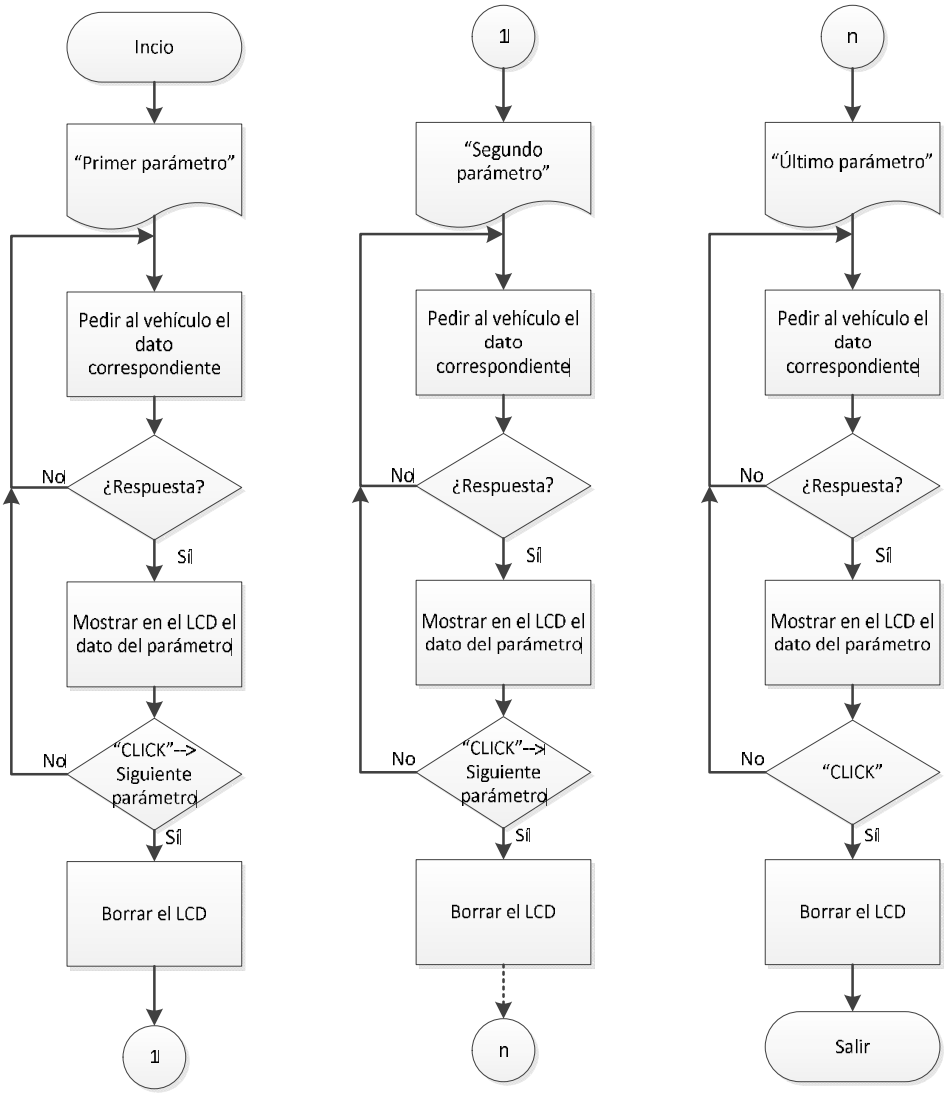


Figura 3.25. Diagrama de flujo para leer y mostrar datos del vehículo.

La subrutina para leer y mostrar datos de los vehículos utilizará la comunicación del bus CAN. La subrutina comenzará mostrando en el LCD el nombre del primer parámetro que se va a leer, después el programa debe pedir al vehículo la información de dicho parámetro, enviando con protocolo CAN el PID correspondiente, si no obtiene respuesta enviará nuevamente el mismo PID y así debe continuar hasta que obtenga una respuesta. Con la respuesta requerida se obtiene el valor del parámetro, éste dato se debe procesar para ser decodificado y después se adecuará para mostrarlo en el LCD. Con el fin de actualizar el valor del parámetro en cuestión se repite el proceso para pedirle la información al vehículo, continúa de esta manera hasta que el usuario por medio del botón central del Joystick indique que se leerá el siguiente parámetro.

El proceso de lectura de los parámetros siguientes es idéntico al del primer parámetro; al llegar al último dato, se pulsa del botón central del Joystick, con lo que se finaliza la subrutina y regresar al programa principal.

3.3.5. Guardar los datos del vehículo en la memoria

La subrutina que guarda los datos del vehículo en la memoria micro SD, lo hace en un archivo de texto, con los datos del vehículo en prueba y datos del GPS. En este proceso también se les da un formato a la distribución de los datos dentro del archivo.

Al iniciar la subrutina encargada de almacenar los datos en la memoria externa micro SD, se debe verificar que se encuentre insertada dicha memoria, en caso de no encontrarse ésta, se indica en el LCD; finalmente el programa sale de la subrutina y regresa al menú principal. Si existe una memoria, el siguiente paso sería verificar si el sistema de archivos es FAT32, si la tarjeta no tiene el sistema de archivos adecuado, termina el proceso y regresa al menú.

Después de hacer las verificaciones iniciales, la subrutina deberá esperar a que el receptor GPS se sincronice con los satélites y mostrará un mensaje en el LCD que lo indique, esto antes de iniciar el registro en la memoria, de lo contrario no se guardarían los datos del GPS; sin embargo, existe la opción de no esperar la sincronización con los satélites e iniciará el registro de datos. Cuando ocurra la sincronización satelital el sistema comenzará a registrar los datos en la memoria automáticamente, el nombre del archivo será la fecha proporcionada por el GPS. En caso de no esperar a la sincronización satelital, esto es, oprimiendo el botón central del joystick, comenzará el registro de datos con un nombre de archivo del tipo DATOSXX, donde “XX” indica el número del archivo comenzando por “00”.

El registro de datos durante una prueba, termina cuando el usuario oprime el botón central del joystick y se regresa al menú principal.

La figura 3.26 muestra el diagrama de flujo del proceso de guardado en la memoria.

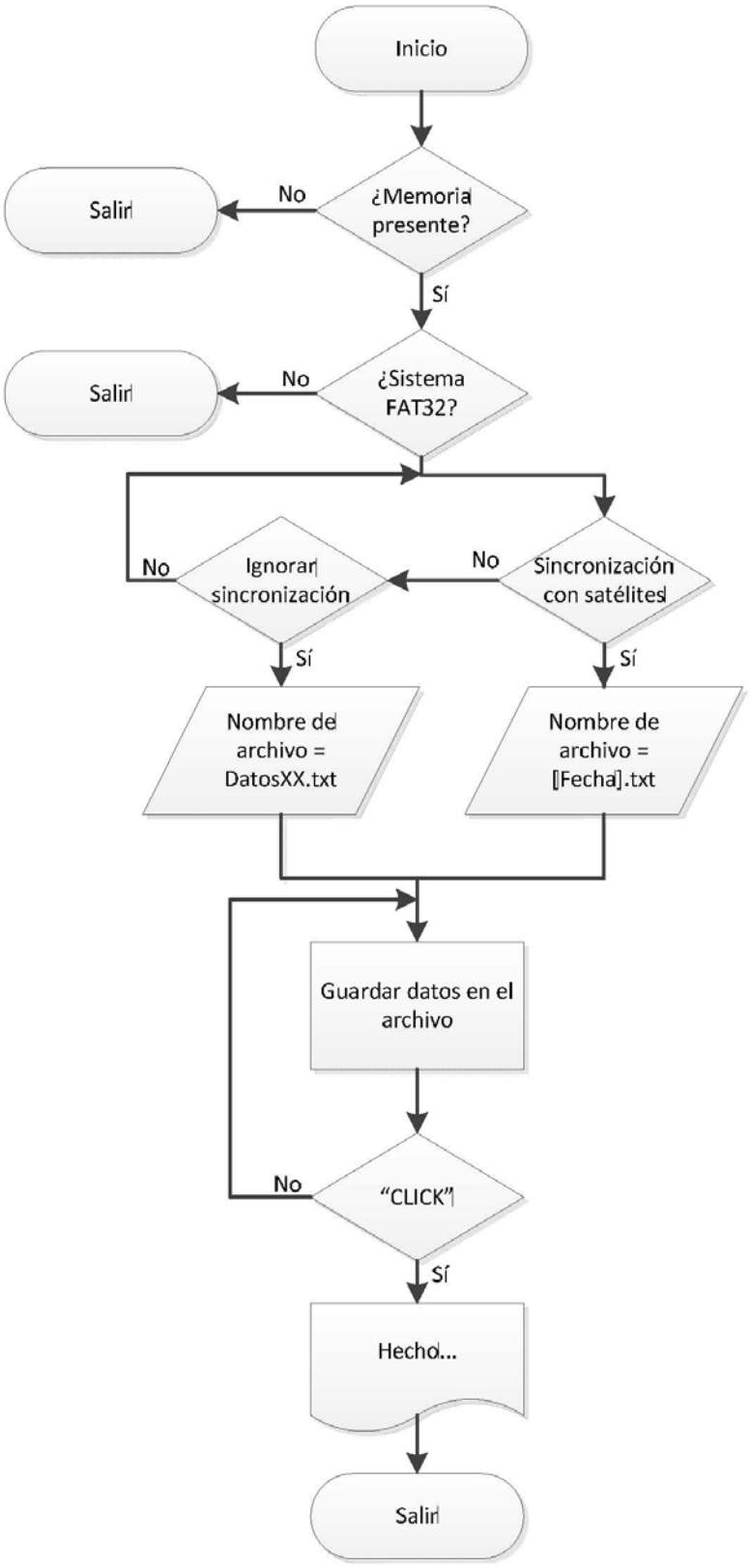


Figura 3.26. Diagrama de flujo del proceso de guardado de datos.

3.3.6. Manejo del Joystick

El *joystick* nos permitirá desplazarnos a través del menú mostrado en el *display* LCD. El *joystick* consta de 5 interruptores o "teclas", que han sido conectadas al microcontrolador para detectar la función que se les ha asignado, las cuales son:

- Terminal PC0 (23): Tecla arriba.
- Terminal PC3 (26): Tecla abajo.
- Terminal PC4 (27): Tecla centro ("CLICK").
- Terminal PC2 (25): Tecla izquierda (reservada para futuras mejoras).
- Terminal PD7 (11): Derecha (reservada para futuras mejoras).

La detección de algún evento en las teclas se realiza a través de una función que monitorea un cambio de estado lógico en algunas de las terminales descritas anteriormente, y en las cuales están conectadas los interruptores del *joystick*. Este programa puede considerarse parte del programa principal, por lo que puede observarse en el diagrama de flujo del mismo, figura 3.27.

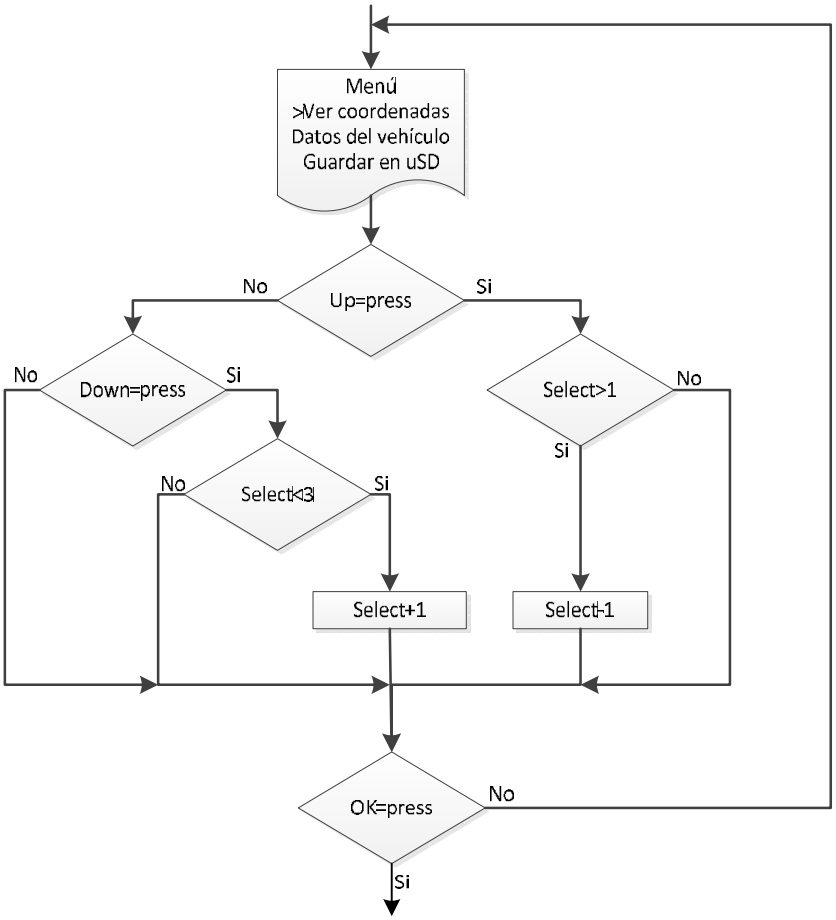


Figura 3.27. Manejo del Joystick.

Las distintas opciones de selección se presentarán en el *display*, las cuales serán elegidas a través del *joystick*. Esta selección se realizará desplazando el carácter ">" que se posicionará en la opción seleccionada. Cada opción tendrá un valor único que será asignada a la variable "pos", que servirá como señalador para desplegar el menú seleccionado, porque además de contener el número de función seleccionada indica la posición de ">" en el *display,* el valor de "opc" disminuye cuando se oprime el *joystick* hacia arriba y aumenta cuando se oprime hacia abajo.

3.3.7. Manejo del LCD

Para el manejo del *display* se creó una librería basada en otra que controla al LCD serial. La librería está pensada para controlar un LCD serial de 20x4, 20 columnas por 4 líneas, en la que la terminal de control del LCD está conectada a cualquier terminal del microcontrolador. Para inicializar la comunicación del *display* se deberá crear un objeto global de la clase LCDSerial con las terminales Rx y Tx que uno deseé y aunque la terminal de Rx no sea necesaria para el LCD, se ha dejado a libre elección, esto puede modificarse después. Algunas de las funciones importantes de esta librería son:

- Inicialización.
- Imprimir texto en el renglón deseado.
- Imprimir texto en alguna posición del renglón deseado.
- Borrar los caracteres del LCD.
- Cambiar el brillo.

Un ejemplo de su uso se muestra en la figura 3.28:

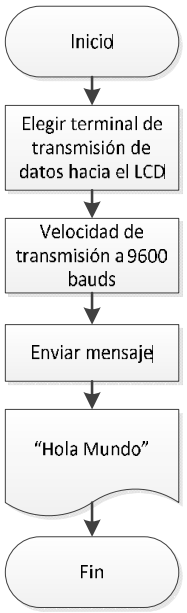


Figura 3.28. Diagrama de flujo para imprimir mensajes en el LCD.

En este capítulo se dieron a conocer de forma general el diseño y desarrollo del sistema, incluyendo sus componentes de *hardware* y de *software.* Una parte muy importante en el desarrollo de un sistema son las pruebas hechas al mismo, por lo que al sistema adquisidor de datos vehiculares se le hizo lo propio y algunas de sus pruebas están descritas en el siguiente capítulo.

Capítulo 4
Pruebas Al Sistema

En este capítulo se describen las pruebas que se realizaron a cada uno de los dispositivos empleados en el sistema al que llamaremos UNAM-CAN, así como las pruebas a los dispositivos en conjunto y se finaliza describiendo las pruebas hechas al UNAM-CAN en vehículos automotores.

4.1. Pruebas a los dispositivos

Con el fin de verificar el funcionamiento de cada uno de los dispositivos que conforman al UNAM-CAN, se les hicieron pruebas por separado, cada prueba nos dio información acerca de su funcionamiento y permitió comprobar que el dispositivo probado funcionara correctamente.

4.1.1. Display de cristal líquido

Para realizar la prueba al LCD, lo primero que se hizo fue energizarlo, por defecto enciende su iluminación y muestra un mensaje por 500ms; esto verifica que la unidad se ha energizado, trabaja correctamente y la conexión del LCD es correcta. El mensaje que muestra es: 'Sparkfun.com SerLCD v2' y puede ser deshabilitado o cambiado; para el caso del UNAM-CAN el mansaje se cambió por el de: 'Instituto Ingenieria LSC-Miguel I.'.

La hoja de especificaciones del LCD serial dice que su velocidad de comunicación por defecto es a 9,600bps, con 8 bits de datos. Un aspecto muy importante, también mencionado en las especificaciones, es que el LCD serial se controla con caracteres ASCII, lo que significa es que si le enviamos al módulo cualquier carácter de texto en ASCII se mostrará en el LCD, a excepción de los caracteres 254 (0xFE) y 124 (0x7C) que se usan como comandos. Para probar su operación se verificó, con ayuda de un microcontrolador, que el LCD funcionara como lo describe su hoja de especificaciones, para ese propósito se le enviaron caracteres en ASCII, el resultado de una de estas pruebas se muestra en la figura 4.1.

Figura 4.1. Primer prueba del LCD.

4.1.2. Receptor GPS

Para probar el receptor GPS lo primero que se hizo fue alimentarlo y esperar a que el LED que tiene integrado emitiera su luz intermitentemente, lo que sucede cuando se sincroniza con los satélites, eso nos indicaba que el receptor funcionaba correctamente. Esto es una manera indirecta de conocer el correcto funcionamiento del receptor, lo que implicaba tener que esperar el tiempo que necesita el receptor para sincronizarse, esto sucede cuando ha estado durante mucho tiempo apagado. La sincronización por primera vez, toma en ocasiones mucho tiempo, y mientras no se sincronice no sabremos si hay una mala recepción de la señal o que el receptor está dañado; de esta manera tampoco se sabe si el receptor envía los datos por su terminal de transmisión de datos.

Debido a que se necesitaba probar varios receptores GPS, se tuvo que plantear otra manera de comprobar su correcta operación. Por lo anterior se instaló y ejecutó un monitor serial en una PC, para observar si el receptor GPS enviaba datos. La velocidad de recepción de datos de ese monitor serial se estableció según la velocidad de transmisión por defecto del receptor a 4,800bps. En la figura 4.2 se pueden observar las cadenas NMEA 0183 que el receptor está enviando.

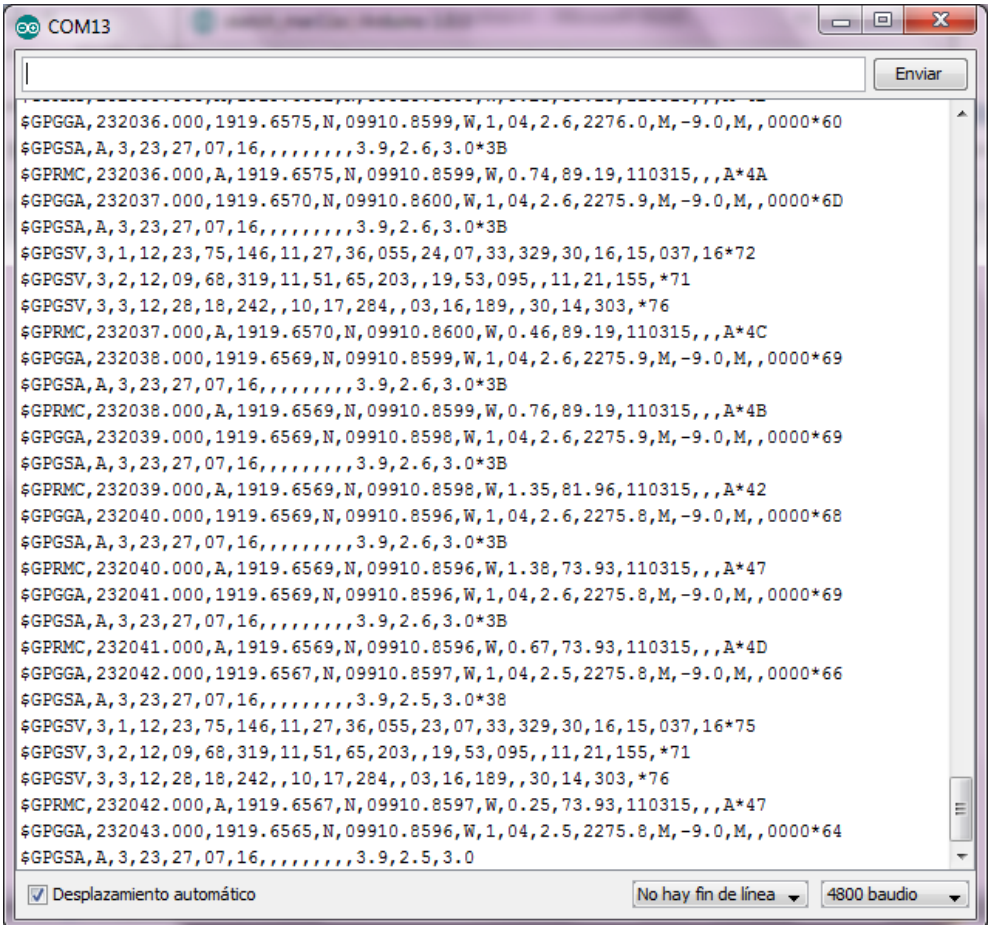


Figura 4.2. Obtención de datos del receptor GPS en un monitor serial.

Cabe comentar que en la página web del fabricante del receptor GPS se encuentra disponible un programa llamado *SiRFDemo,* el cual también se utilizó para verificar el funcionamiento de los receptores GPS, de esta manera la verificación fue más eficiente, ya que sólo se necesita ejecutar el programa mencionado y seleccionar tanto el puerto en que se conectó el dispositivo a la PC, como la velocidad de transmisión de datos con el que opera el dispositivo (4,800 bps). Como resultado de esta operación obtuvimos información más detallada, como: datos referentes a la posición, el tiempo UTC, un esquema en donde se ilustra gráficamente el número de satélites que en ese momento detecta el receptor GPS, además de que muestra las cadenas del protocolo NMEA 0183 soportadas por el dispositivo.

Los receptores GPS utilizados utilizan comunicación serial con voltajes TTL, así que para comunicarlos con la PC se armó un circuito utilizando un integrado que transforma la señal serial a USB; las terminales Tx y Rx del receptor GPS se conectaron al integrado mencionado, y del integrado al puerto USB de la PC.

Durante la realización de las pruebas a los receptores GPS se contaba con dos versiones de éstos, V1 y V2, los dos son del mismo fabricante. Las pruebas fueron determinantes para escoger el dispositivo que se utiliza en el UNAM-CAN. En el programa *"SiRFDemo"* podemos ver las ventanas *"Signal View", "Radar View", "Map View", y "Debug View - NMEA"*, por mencionar algunas de las más importantes, donde la información de cada una es:

Sigal View
- SV.- Número del satélite.
- St.- Estatus de 18 posibles de cada satélite que está siendo rastreado.
- Az.- Acimut en grados del satélite.
- El.- Elevación en grados del satélite, siendo cero grados en el horizonte y noventa grados estando exactamente por encima del receptor.
- C/No.- Nivel de la señal en dB-Hz.
- -5 cycl.- Historial de cinco segundos de medición de la fuerza de la señal.

Radar View.- Ésta ventana muestra gráficamente la ubicación de cada uno de los satélites rastreados, utilizados y disponibles en forma de un gráfico polar. Los colores mostrados en esta ventana corresponden a los de la ventana *"Signal View"* y significan lo siguiente:

-Rojo.- La localización del satélite se conoce de la información del almanaque; sin embargo todavía no ha sido rastreado.
-Azul.- El satélite está siendo rastreado, pero no se está utilizando para calcular la posición actual en coordenadas.
-Verde.- El satélite está siendo rastreado y se está utilizando para calcular la posición actual.

Map View.- En *Map View* podemos encontrar información de hora UTC, y cuando el receptor está sincronizado con al menos tres receptores vemos también las coordenadas de nuestra ubicación, la altitud y el número de satélites con el que está sincronizado.

Debug View – NMEA.- En esta ventana se muestran las cadenas del protocolo NMEA 0183 enviadas por el receptor GPS.

La figura 4.3 muestra una captura de pantalla del programa *SiRFDemo* cuando se tiene conectado el receptor GPS V1 antes de sincronizarse. Se observa que tiene seguimiento de sólo 2 satélites, por lo tanto no nos muestra las coordenadas, sólo nos muestra la hora UTC; observamos también las cadenas del protocolo NMEA 0183 que este receptor puede enviarnos. La presencia de todos estos datos nos demuestran que el GPS está funcionando, por lo que podemos esperar a que se sincronice en algún momento.

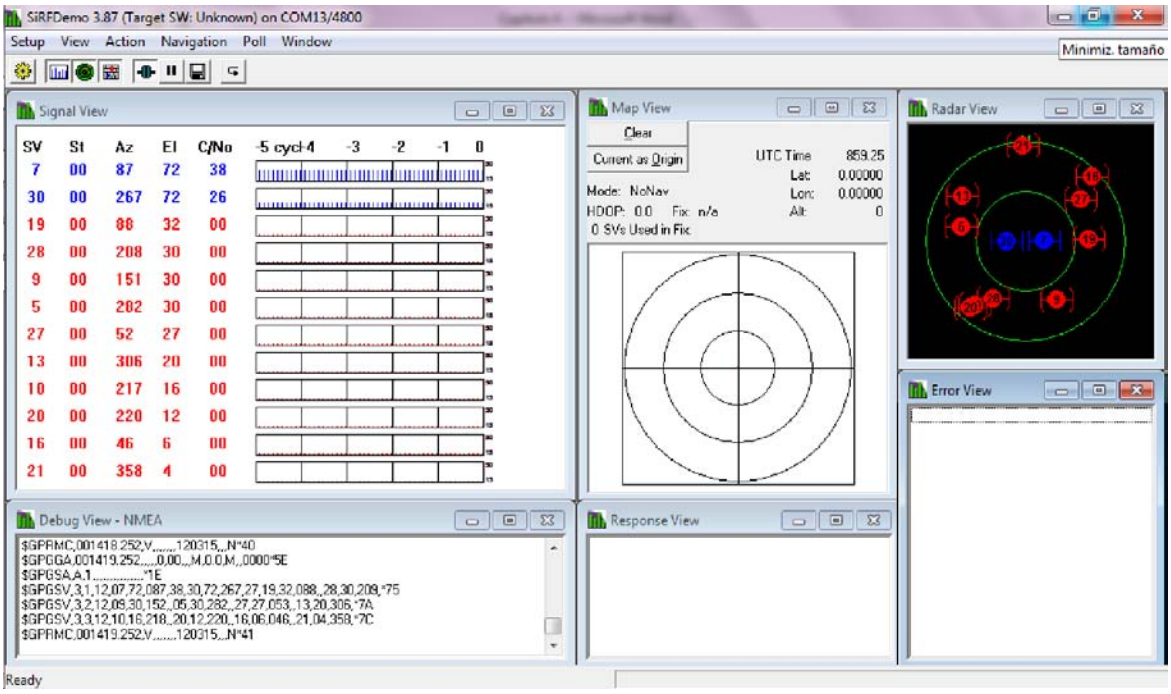


Figura 4.3. Verificación del receptor GPS V1 antes de sincronizarse.

Después de esperar el tiempo necesario para que el receptor V1 se sincronizara se tomó otra captura de pantalla y el resultado lo vemos en la figura 4.4. Ahora el receptor tiene seguimiento de tres satélites, que también están siendo utilizados para calcular la posición del receptor, por lo que ahora nos muestra la posición con la latitud y la longitud.

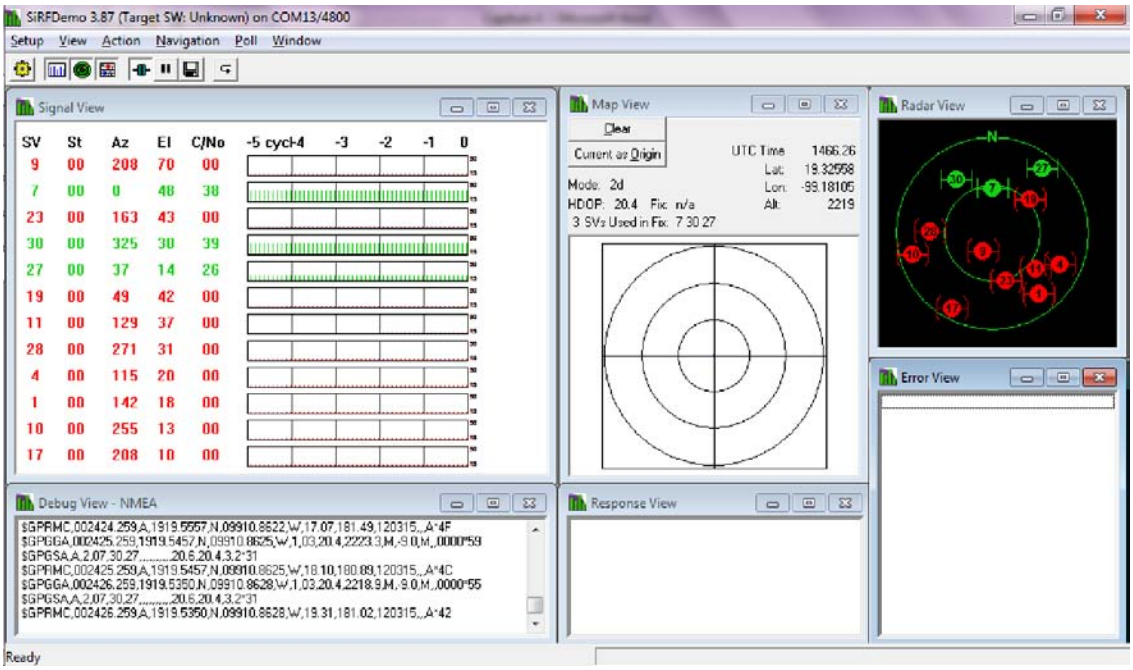


Figura 4.4. Receptor GPS V1 sincronizado.

En la figura 4.5 se muestra la captura de pantalla del programa ya mencionado, con un receptor GPS V2, y en ella observamos que este receptor tiene seguimiento de 6 satélites, de los cuales utiliza 4 para obtener las coordenadas. También observamos las cadenas del protocolo NMEA 0183 que el receptor maneja.

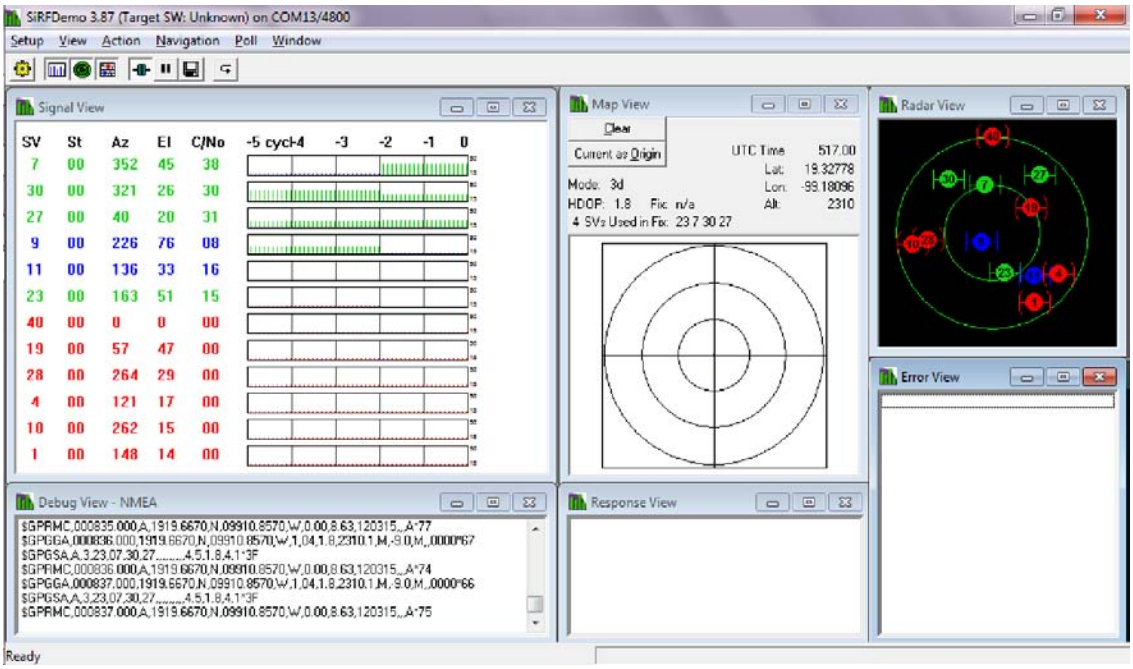


Figura 4.5. Verificación de un receptor GPS V2.

Cabe mencionar que los dos receptores se probaron dentro del laboratorio; el GPS V1 demoró más en sincronizarse que el GPS V2. Como los dos se probaron en el mismo lugar, a juzgar por la intensidad de la señal que recibe de cada satélite mostrado por el programa y por el tiempo que les llevó sincronizarse a cada uno, decimos con certeza que el GPS V2 tiene mayor sensibilidad que el GPS V1, por lo que éste se escogió para operar con el UNAM-CAN.

4.1.3. Memoria micro SD

Para el registro físico de datos, UNAM-CAN requiere de una tarjeta de memoria micro SD, con sistema de archivos FAT16 o FAT32, para realizar la operación de guardado; en general estas memorias ya tienen dichos sistemas de archivos al momento de adquirirlas (las memorias con capacidad de almacenamiento de hasta 2 GB tienen el sistema de archivos FAT16, las de capacidad mayor a 2 GB utilizan FAT32). Sabiendo eso, la manera más fácil y rápida de verificar que funcionen las tarjetas de memoria es insertándola en la ranura para memorias SD de la PC, con ayuda de un adaptador micro SD a SD, y observando en el explorador de archivos del sistema operativo que la tarjeta sea detectada, podemos ver la información que nos interesa, como: la capacidad máxima de almacenamiento de la memoria, la capacidad restante y el sistema de archivos con el que cuenta. En la figura 4.6 se muestra que la PC ha detectado a la memoria SD en la unidad 'D:', y en la parte inferior se observan los datos de esa memoria.

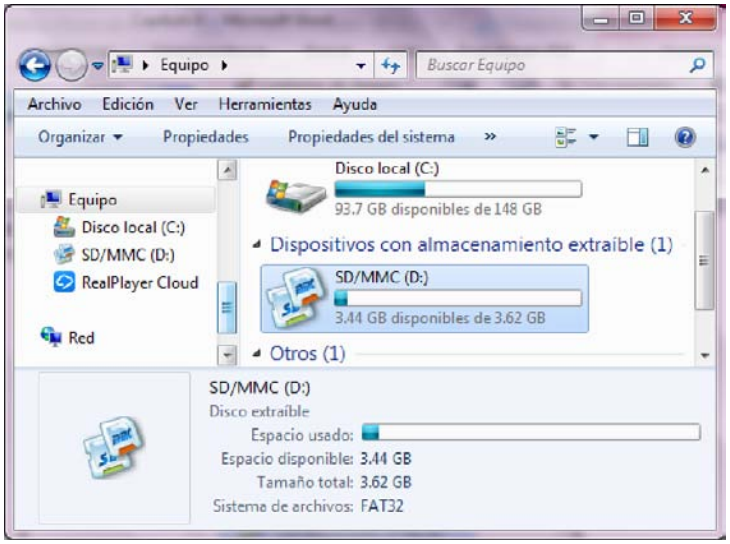


Figura 4.6. Detección de una memoria SD.

La asociación de memorias SD (SD Association) da la fuerte recomendación de utilizar el software *SDFormatter* para formatear las memorias SD, esto en lugar de cualquier utilidad de formateo con el que viene cualquier sistema operativo; dicho software está disponible en su página *web* y fue creado específicamente para las memorias de los estándares SD, SDHC y SDXC y les provee de un rápido y fácil

acceso a todas la capacidades, las cuales se ven limitadas con cualquier otra utilidad de formateo.

Una vez ejecutada la operación mencionada en el párrafo anterior, se hizo una prueba de guardado utilizando la librería para el manejo de la SD. La prueba generó un archivo de texto con la información mostrada en la figura 4.7.

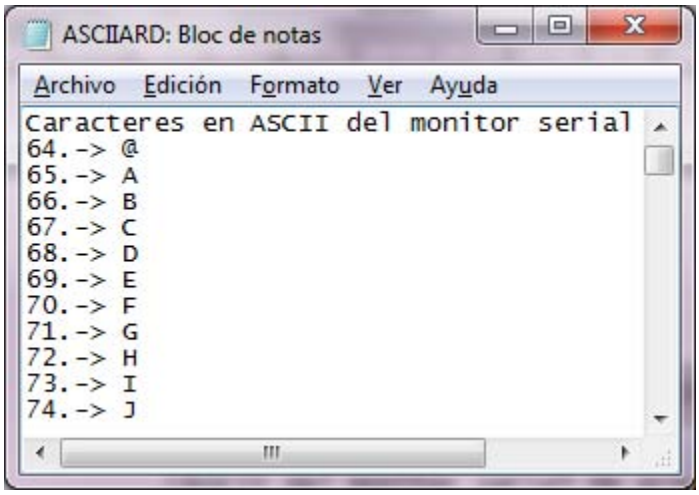


Figura 4.7. Archivo de texto almacenado en la tarjeta de memoria SD.

4.1.4. Joystick

Las pruebas a este dispositivo fueron muy sencillas, ya que en esencia se trata de varios contactos contenidos en el mismo encapsulado; se verificó su funcionamiento con pruebas de continuidad por medio de un multímetro. Para realizar la verificación primero se identificó a la terminal común con ayuda de la hoja de datos del *joystick*, cuyo diagrama se muestra en la figura 4.8. Después se construyó un pequeño circuito impreso para montar al *joystick*, figura 4.9; en la prueba la terminal común se puso una punta del multímetro, la otra punta se fue recorriendo en cada una de las demás terminales, correspondiendo con la posición pulsada en el *joystick* y se observaba la continuidad.

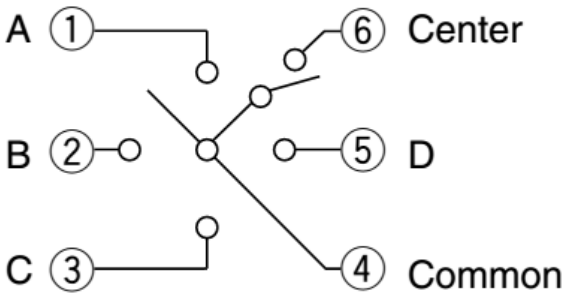


Figura 4.8. Diagrama de conexión del *joystick*.

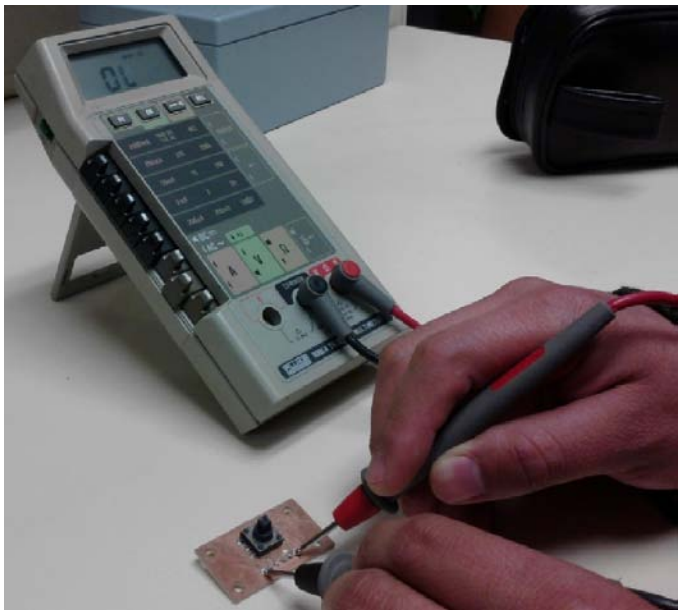


Figura 4.9. Pruebas de continuidad a los contactos del *joystick*.

Las pruebas efectuadas a cada uno de los elementos mencionados en los diferentes apartados ya descritos, fueron realizadas utilizando dos tarjetas de desarrollo, una de ellas tiene un microcontrolador y conectores para acceder a cada una de sus terminales; la otra tarjeta tiene conectores en los que se pueden conectar cada uno de los componentes a los que se realizaron pruebas. Estas tarjetas sirvieron de ayuda para crear la pieza de *software* asociado a cada uno de dichos componentes, que después de unirlas se crearon versiones completas del *software* del UNAM-CAN. Las tarjetas y el software implementado en ellas sirvieron como precedente para el diseño del circuito impreso y *software* del prototipo final del UNAM-CAN vistos más adelante.

4.2. Integración del sistema UNAM-CAN

Para la integrar al UNAM-CAN se diseñó una placa de circuito impreso como lo muestra la figura 4.10. En esa placa se integraron cada uno de los componentes por “módulos” y de manera ordenada para ir verificando el funcionamiento de cada etapa.

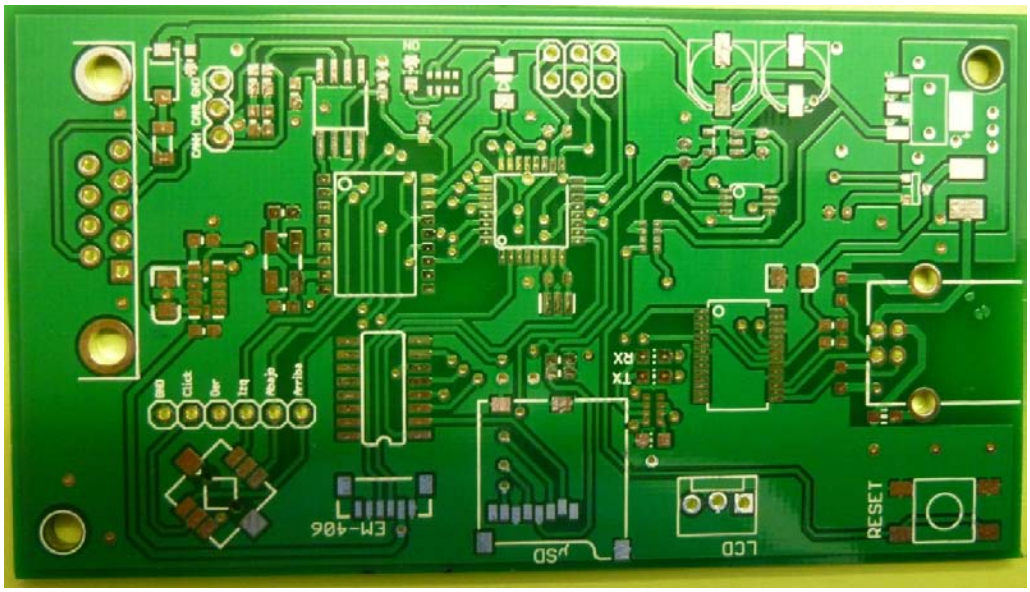


Figura 4.10. Placa de circuito impreso del UNAM-CAN.

4.2.1. Pruebas al hardware

Las primeras pruebas al *hardware* del UNAM-CAN se hicieron a medida que se integraban los componentes en la placa de circuito impreso, estas pruebas fueron hechas a cada "módulo funcional", entre los cuales podemos mencionar algunos como: el módulo de alimentación, el módulo de comunicación por USB, el microcontrolador y sus componentes externos (capacitores, oscilador, regulador, etc.), y el módulo para la comunicación por bus CAN.

El primer módulo que se integró fue el de la comunicación por USB, por medio de éste se podrá programar al microcontrolador del UNAM-CAN y proveerle el voltaje de alimentación. El componente principal de este módulo es un FTDI, que para disminuir la probabilidad de alguna falla se le hicieron pruebas de continuidad con un multímetro, con esta acción se comprobó que las terminales del circuito integrado no estuvieran en corto no previsto, y tuvieran continuidad con las pistas correspondientes. Después de soldar el resto de los componentes y ya hechas las pruebas de esta primera parte, se conectó con seguridad con un cable USB a un puerto USB de una PC.

La figura 4.11 muestra a la tarjeta y a sus componentes para la conexión por USB, el LED rojo de la parte superior indica "encendido".

Figura 4.11. Componentes de la conexión USB.

Por medio de un mensaje en la PC se indicaba, “Nuevo hardware encontrado”, con lo que nos dio la primera señal de que estaban correctos los componentes de la comunicación por USB. El FTDI que se utiliza para la conversión serial-USB requiere que la PC tenga instalado su controlador, el fabricante de este integrado lo proporciona en su página oficial; la figura 4.12 muestra el resultado después de la instalación exitosa de dicho controlador.

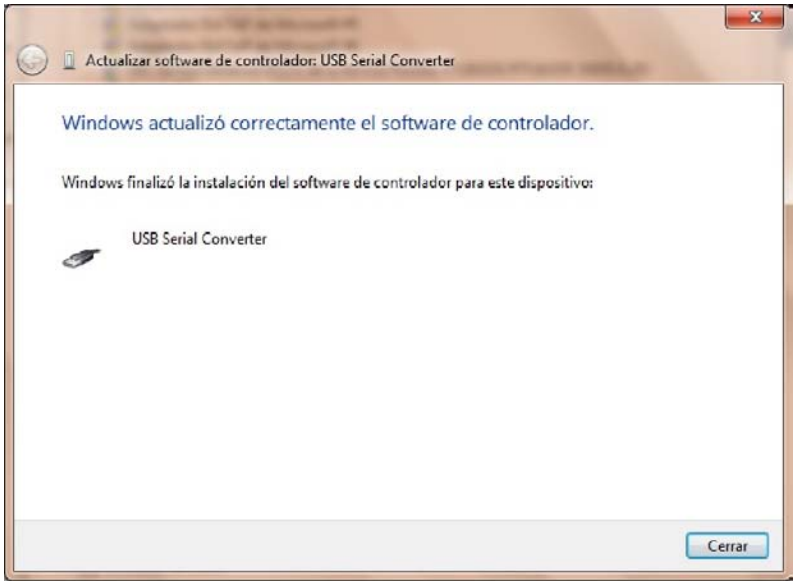


Figura 4.12. Instalación exitosa del controlador del FTDI.

Con la instalación del controlador y el mensaje de confirmación queda comprobado que la comunicación por USB funciona y está lista para usarse.

Los siguientes componentes a ser integrados fueron los correspondientes a la alimentación “alternativa” de la tarjeta, el microcontrolador y sus componentes externos necesarios. Dentro de los componentes para la alimentación “alternativa”

se encuentra un MOSFET, que junto con otro componente se encargan de seleccionar automáticamente la fuente con la cual se alimentará la tarjeta (por USB o por el DLC del automóvil). Esta segunda alimentación utiliza dos reguladores, uno de 5 V y otro de 3.3 V; una de las pruebas para esta parte fue verificar con un multímetro que esos voltajes estuvieran presentes donde corresponde. En el caso del microcontrolador se verificó que no hubiera cortos, que hubiera continuidad en sus terminales con las pistas y que recibiera su alimentación de voltaje.

Otro elemento importante asociado al microcontrolador es un conector de dos filas con tres postes cada una, con el que se tiene otra forma de programar al microcontrolador con un programador externo. El programador externo con el que cuenta el Instituto de Ingeniería es el AVRISP mkII de la empresa Atmel, aprovechando éste se realizó la última prueba que definió si el microcontrolador estaba montado y funcionando correctamente. El programador tiene integrado un LED que indica el estado de conexión; cuando es rojo significa que no hay conexión y si es verde significa que la conexión es correcta. La figura 4.13 ilustra lo anterior; primero se ve al programador con su LED de estado en rojo, después, cuando se conecta al UNAM-CAN el LED se vuelve verde, a partir de este momento se pueden cargar programas en el microcontrolador.

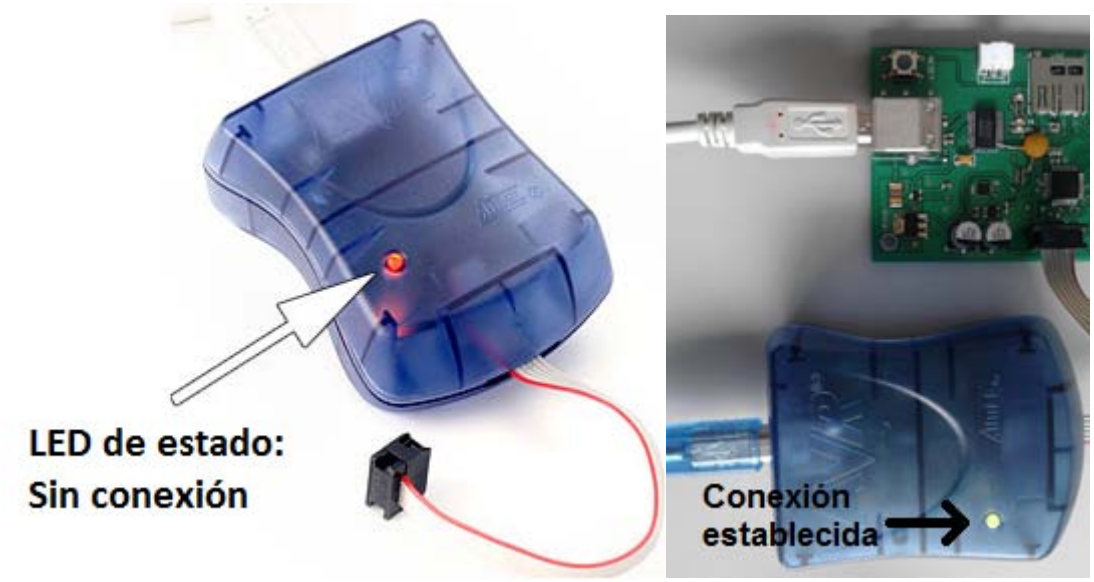


Figura 4.13. Verificación del microcontrolador con un programador externo.

A partir de esta etapa se tiene integrada la parte principal del UNAM-CAN. Se cargó un programa sencillo para probar la programación por el USB y con el programador externo. La ejecución exitosa del programa cargado nos indicó también que el oscilador del microcontrolador está montado satisfactoriamente.

El resto de los componentes fueron montados y probados de manera similar a lo descrito anteriormente, la figura 4.14 muestra a la tarjeta del UNAM-CAN con todos sus componentes.

Figura 4.14. Integración completa de los componentes del UNAM-CAN.

4.2.2. Pruebas al software

Con todos los componentes instalados se procedió a cargar el software para hacerle pruebas al mismo. La primera prueba que se hizo fue cargar el programa del menú, éste manda letreros al LCD, esta prueba se muestra en la figura 4.15.

Figura 4.15. Software del menú principal.

Componentes como el receptor GPS, el LCD y el joystick se conectan al UNAM-CAN por medio de sus conectores correspondientes, y así, con el *hardware* completo, se hicieron pruebas al *software* para los módulos restantes. En la figura 4.16 se muestra el montaje provisional del UNAM-CAN en conjunto.

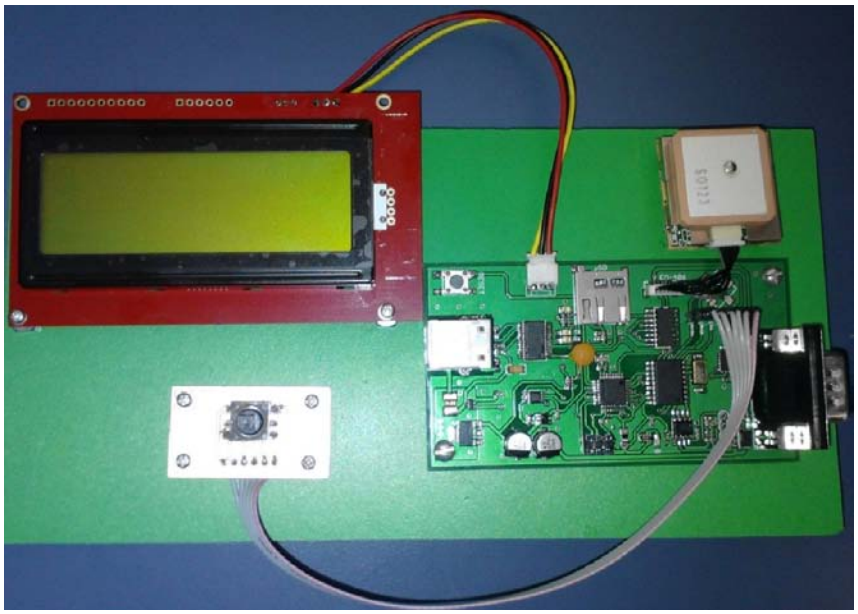


Figura 4.16. Montaje provisional del UNAM-CAN para pruebas.

Después de probar el *software* para cada uno de los elementos del UNAM-CAN se programaron versiones del *software* completo. La figura 4.17 muestra la prueba de una de las funciones que forma parte de las tareas del UNAM-CAN; aquí se obtienen y visualizan datos del GPS.

Figura 4.17. Visualización de datos del GPS.

Con la primera prueba también se prueba el software del LCD y del *josystic*, ya que se utilizó el *joystick* para navegar a través del menú y seleccionar la función y se despliegan los datos del GPS en el LCD.

Software para la petición de parámetros vehiculares sobre bus CAN

Las peticiones y respuestas a los PID ocurren sobre el bus CAN estándar de los vehículos (identificador de 11 bits). Las peticiones y respuestas en el OBD estándar utilizan identificadores especiales, así que el UNAM-CAN fue programado para iniciar peticiones de datos utilizando una trama CAN con el valor 7DFh en su identificador, dicho valor actúa como indicador de petición; el UNAM-CAN debe aceptar como respuesta tramas con el valor 7E8h en su identificador.

La petición de un PID se envía al automóvil por medio de una trama CAN con identificador igual a 7DFh, usando 8 bytes en el campo de datos.

En el caso de un bus CAN extendido (29 bits en el identificador), el identificador tiene el valor 18DB33F1h y usa 8 bytes en el campo de datos.

El campo de datos tiene el mismo formato en ambos tipos de trama CAN (11 y 29 bits) como lo muestra la tabla 4.1.

Número de byte							
0	1	2	3	4	5	6	7
Número de bytes de datos adicionales: 2	Modo del OBD-II: 01	Número del PID	No se usan (se ignoran) pueden tener 55h				

Tabla 4.1. Bytes para la petición de parámetros automotrices.

El vehículo responde a una petición de PID con una trama CAN, cuyo identificador corresponde a la ECU principal, con un identificador igual a 7E8h en CAN estándar y 18DAF1xx en CAN extendido; el número de bytes de datos con el que responde es variable y depende del parámetro solicitado. La tabla 4.2 muestra dichos bytes y es igual en ambos tipos de trama CAN.

Número de byte							
0	1	2	3	4	5	6	7
Número de bytes de datos adicionales: 3 a 6	Modo del OBD-II: 01	Número del PID (el mismo que se pidió)	Valor del parámetro especificado, byte 0	Valor del byte 1 (opcional)	Valor del byte 2 (opcional)	Valor del byte 3 (opcional)	No se usa (55h)

Tabla 4.2. Bytes de respuesta al parámetro pedido.

Verificación de tramas de CAN estándar

Para verificar las tramas CAN del UNAM-CAN se navegó por el menú y se seleccionó la segunda opción, la cual consiste en leer parámetros del vehículo. Con esta función se pueden observar los datos en tiempo real del vehículo. Estos datos pueden ser: valores reportados por los sensores y sus voltajes así como valores de los actuadores. Si determinado parámetro no puede ser reportado por el vehículo se muestra “PID no soportado o cable no conectado”. Las primeras pruebas de esta parte del software no se realizaron en vehículos porque lo que se pretendía en esta prueba fue enviar y verificar las tramas en un osciloscopio. La figura 4.18 muestra el resultado en el LCD al ejecutar dicha función.

Figura 4.18. Prueba a la función que muestra parámetros del automóvil.

La función de lectura de parámetros del vehículo realiza la comunicación por medio del protocolo CAN, por lo tanto incluye el funcionamiento del controlador y transceptor CAN. Para comprobar que el protocolo estuviera presente en el UNAM-CAN y que se enviaran correctamente las tramas con sus mensajes correspondientes, se realizaron las pruebas conectando las puntas “A” y “B” del osciloscopio, en las salidas CAN-H y CAN-L respectivamente, de la tarjeta programada. Los resultados obtenidos de las primeras pruebas se muestran en las capturas de pantalla del osciloscopio en la figura 4.19. Al analizar los resultados observamos que no corresponden a una trama CAN.

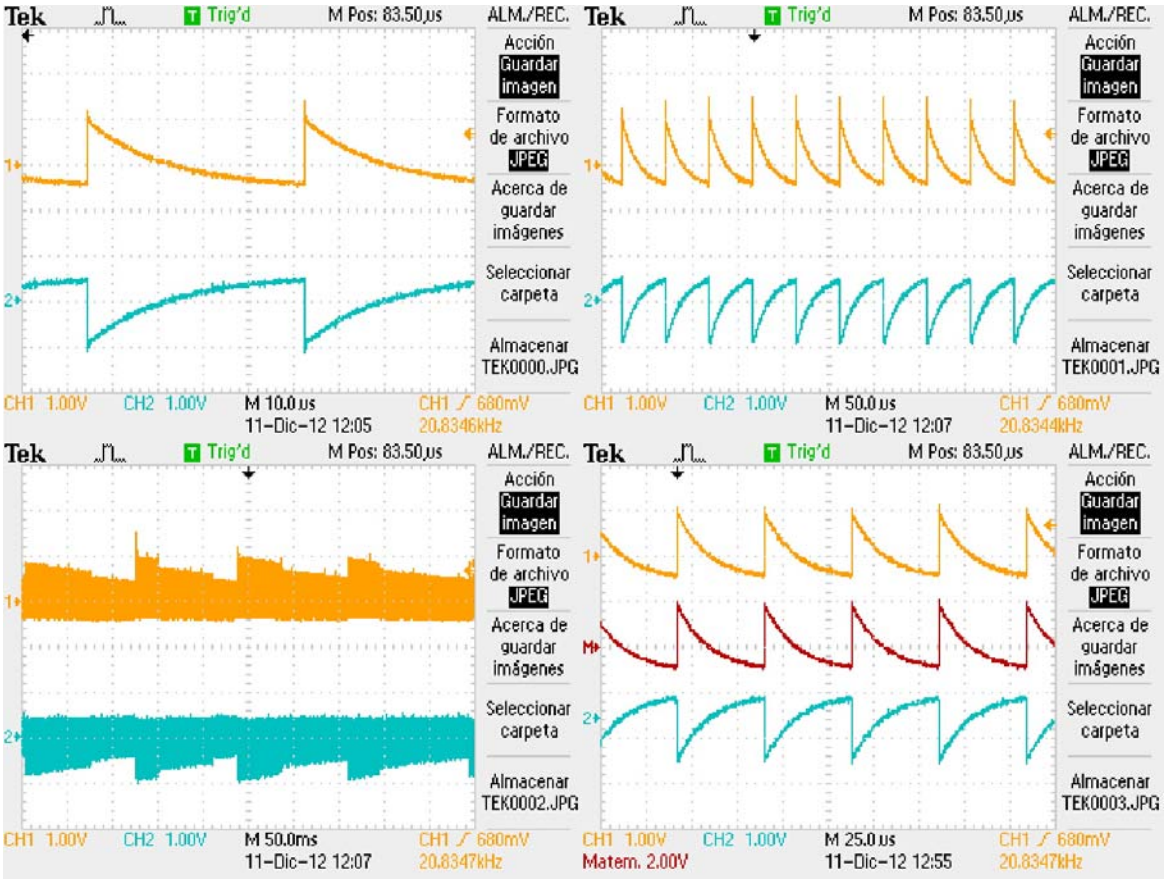


Figura 4.19. Señales de la terminal CAN del UNAM-CAN.

Dentro de las características de CAN mencionadas en el capítulo dos hay una que indica que el bus CAN necesita resistencias de terminación en sus extremos con un valor igual a la impedancia característica del bus. El problema de las señales de la figura anterior se resolvió entonces cuando se colocó una resistencia entre las terminales CAN-H y CAN-L de la tarjeta. Con esto las tramas se mostraron como se observa en las capturas de pantalla de la figura 4.20, en donde la imagen izquierda corresponde a la señal cuando está en reposo, con un nivel de voltaje en sus terminales de 2.47 [V] para CAN-H y de 2.49 [V] para CAN-L. La imagen de la derecha corresponde a la señal con las tramas CAN, con un nivel de voltaje máximo de 3.64 [V] para CAN-H y un voltaje mínimo de 1.28 [V] para CAN-L, los cuales corresponden con los valores esperados.

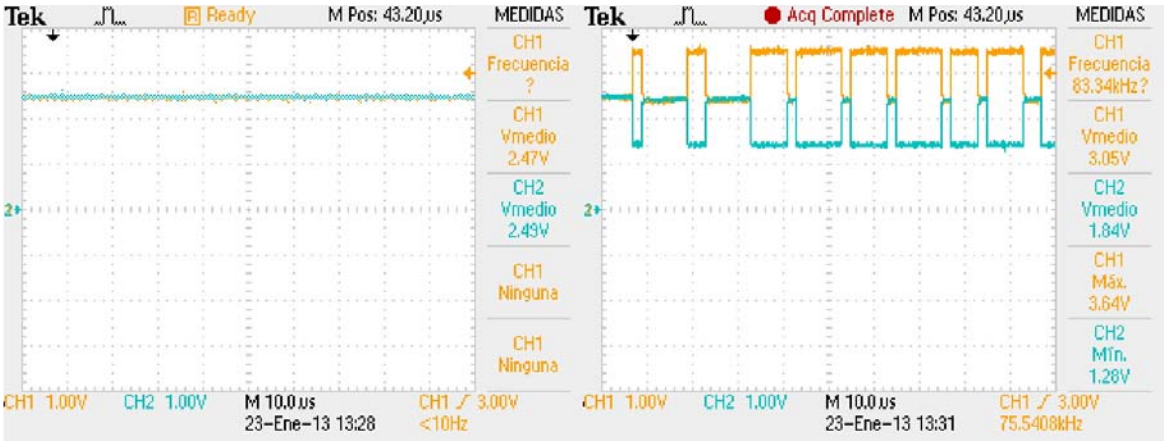


Figura 4.20. Señal con la resistencia de terminación.

Para realizar la identificación y verificación de las tramas, se ajustaron las escalas de voltaje y de tiempo del osciloscopio de tal forma que la identificación de la señal sea clara. Debido a tal ajuste, una trama completa de CAN se visualiza en diferentes capturas de pantalla del osciloscopio. La figura 4.21 es una captura que muestra la primera parte de una trama. El ajuste de tiempo realizado permite que un bit ocupe una de las cinco divisiones por cuadro.

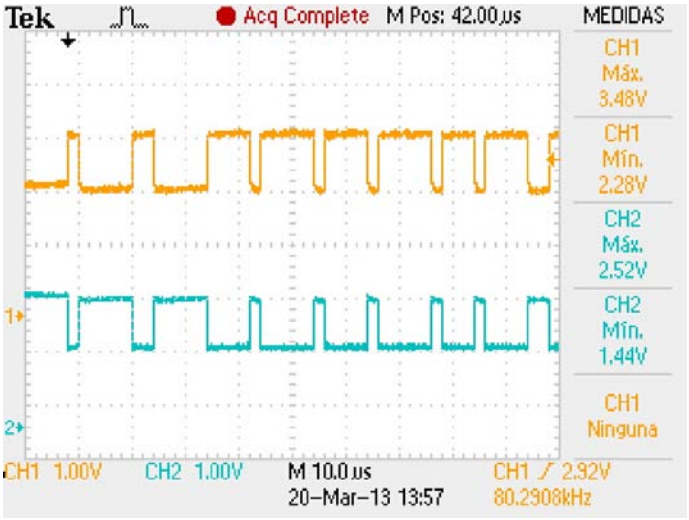


Figura 4.21. Captura de pantalla del inicio de una trama CAN.

Al analizar las tramas de la figura 4.21, se pudo verificar que se enviaran las tramas conforme lo especifica el protocolo (a saber: SOF, los campos de arbitraje, de control, de datos, de CRC, EOF y entre-tramas) y que en cada campo estén los datos que se requieren mandar por el bus CAN.

Por ejemplo, al solicitar las revoluciones por minuto del motor del vehículo figura 2.8, se deberá incluir la siguiente información en la trama CAN.

- **SOF:** **0x0**, que indica el inicio de trama.
- **Arbitraje:** **0x7DF** como identificador en este campo, este valor actúa como una dirección de difusión, el dispositivo de diagnóstico lo manda para señalar que quiere pedirle datos a alguna ECU (Engine Control Unit) en particular y un **0x0** en RTR al final del mismo campo.
- **Control:** **0x0** en IDE y RB0, **0x08** en la parte del código de la longitud de datos (DLC3-DLC0) que indica que en el campo de datos se enviarán 8 bytes.
- **Datos:** Como lo muestra la tabla 4.1 los datos en el campo de datos son:
    - En el byte 0 hay **0x02,** que indica que se enviarán 2 bytes más.
    - En el byte 1 hay **0x01**, que corresponde al modo 1 de OBD-II para la petición de parámetros en tiempo real.
    - En el byte 2 se indica el código PID del parámetro que queremos obtener, en este caso es **0x0C** para las RPM.
    - Los últimos 5 bytes no se usan, sin embargo los enviaremos, y como dice la tabla 4.1, pueden tener el valor 0x55, en nuestro caso los enviamos como **0x00**.

Para verificar lo anteriormente dicho, se tiene la figura 4.22, que corresponde a una trama CAN enviada por el UNAM-CAN, en ella podemos observar el SOF seguido del identificador b11111011111=0x7DF, el campo de control también coincide con lo esperado. Los siguientes bytes corresponden al campo de dato, donde el primer byte (0) tiene el valor 0x02, el segundo byte (1) tiene 0x01, el tercer byte (2) tiene el PID de las RPM del motor que es b00001100=0x0C.

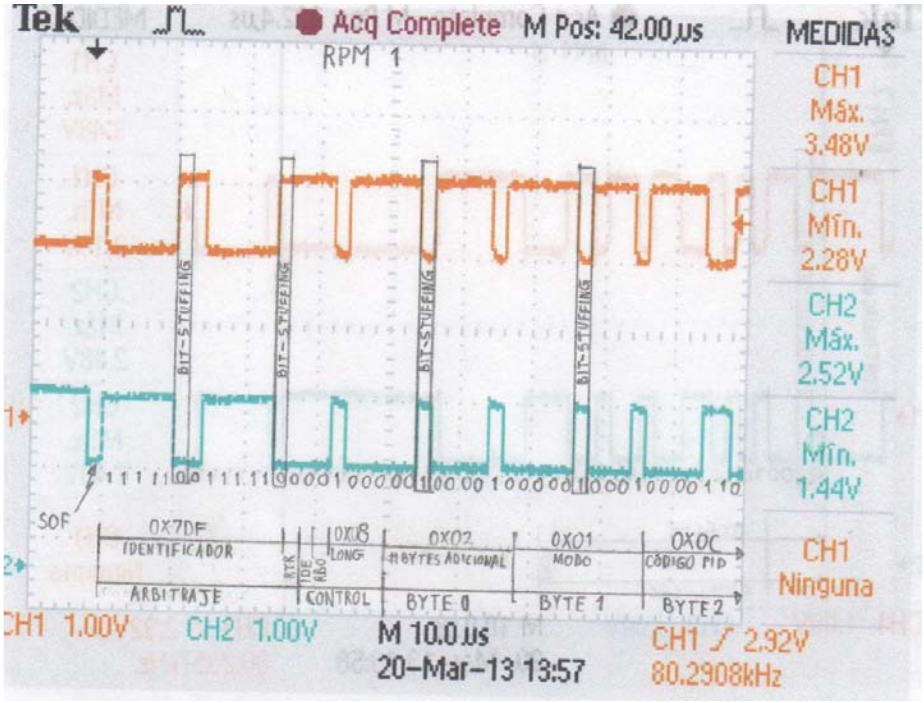


Figura 4.22. Identificación de la trama de petición de las RPM.

La figura 4.23 muestra los 5 bytes restantes del campo de datos con el valor 0x00 cada uno y la figura 4.24 vemos la última parte de la trama, aquí está el CRC con su delimitador, y luego la trama se repite.

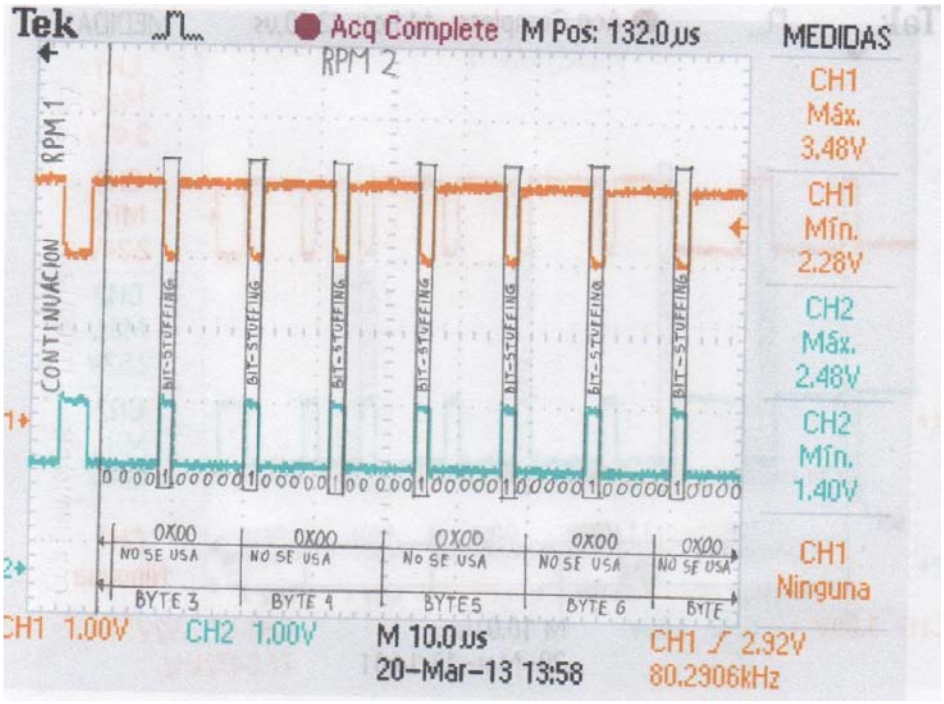


Figura 4.23. Identificación de la trama de petición de las RPM (cont.).

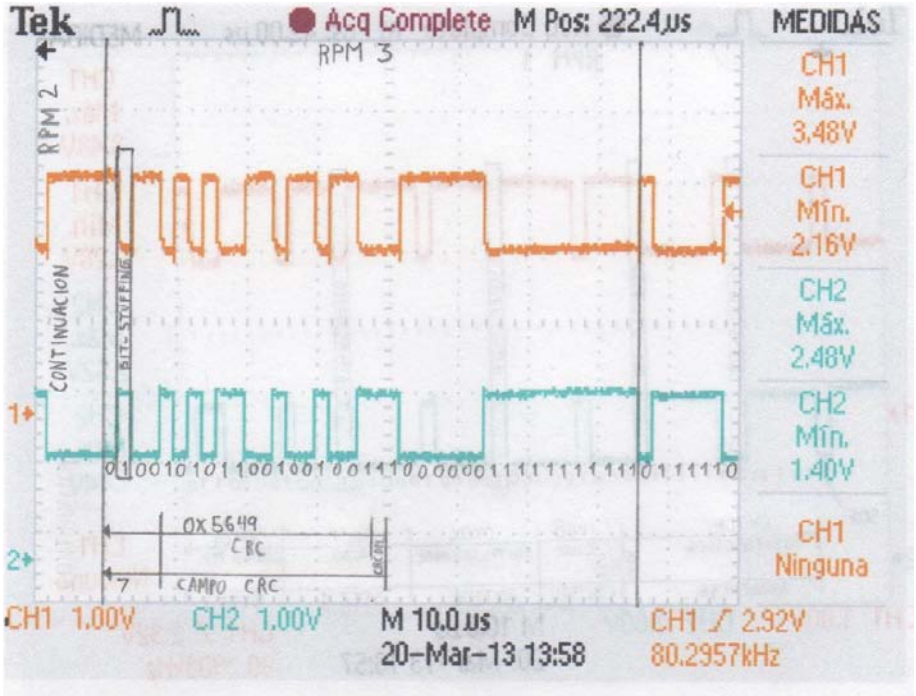


Figura 4.24. Identificación de la trama de petición de las RPM (cont.).

En estas imágenes analizadas se observan unas barras verticales con la leyenda "*bit-Stuffing*", éstas indican donde está el bit de relleno, las cuales, según la especificación CAN, aparecen cuando hay cinco bits seguidos con el mismo nivel lógico de voltaje, el *bit-Stuffing* es de un nivel contrario a estos cinco. La trama analizada, después del campo del CRC, tiene una trama de error debida a un error de reconocimiento, esto se debe a que no hay ningún vehículo que responda la petición del UNAM-CAN.

Para comprobar el campo de CRC se empleó una aplicación que calcula el CRC en línea. La figura 4.25 es la captura de pantalla de esta calculadora con el cálculo del CRC de la trama anterior. En esta calculadora se tiene que ingresar el polinomio del CRC como secuencia de bits y la secuencia en hexadecimal del mensaje al cual se le va a realizar el cálculo. En el caso de CAN el polinomio CRC es: $P(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$.

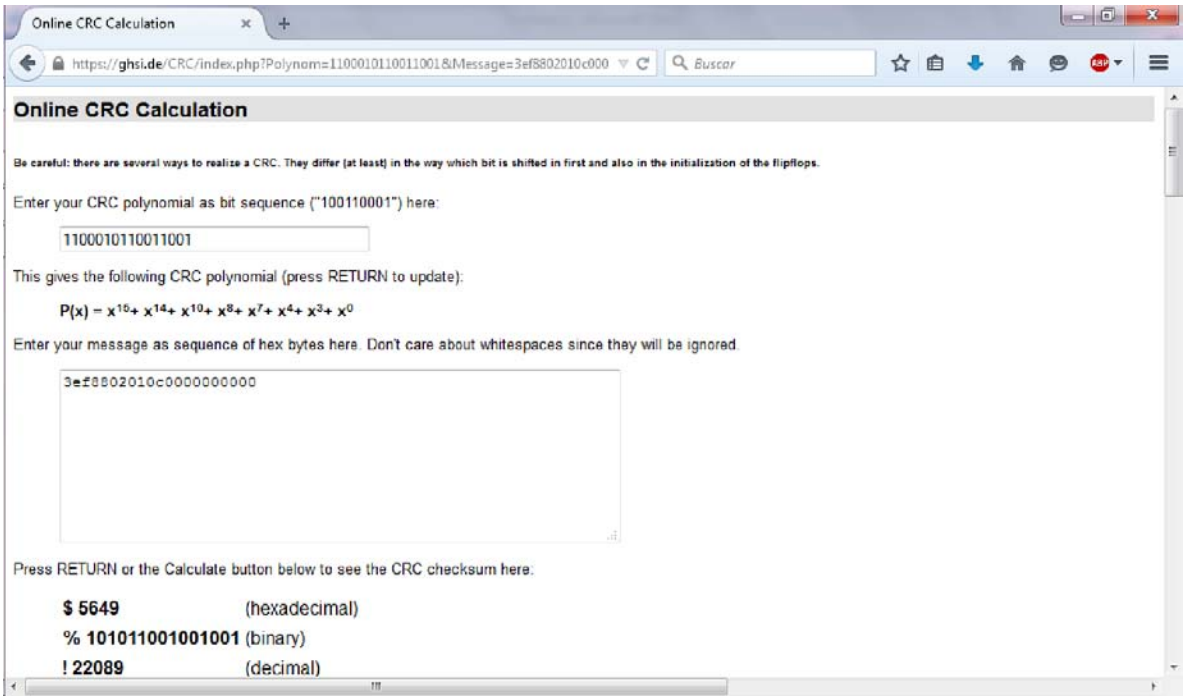


Figura 4.25. Cálculo en línea del CRC.

El valor calculado 5549h corresponde con el valor visto en el osciloscopio, con esto se ha terminado de comprobar que la trama CAN enviada por el UNAM-CAN es correcta.

Verificación de tramas de CAN extendido

En otra versión del software se implementó el uso de tramas CAN extendido (29 bits en el identificador) para vehículos que lo utilizan, se verificó que este tipo de tramas fueran enviadas por el UNAM-CAN, el osciloscopio que se utilizó trabaja

con el protocolo CAN, la visualización de las tramas y sus campos con este osciloscopio es directa. Como ejemplo, enviamos una trama CAN correspondiente a la petición de la temperatura del anticongelante del motor, esta trama en sus campos debe llevar la siguiente información:

- **Arbitraje:** 0x18DB33F1 como identificador.
- **Control:** 08h.
- **Datos:** 0x02, 0x01, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00.
- **CRC:** 2769h (según la calculadora CRC).

La figura 4.26 es una captura de pantalla del osciloscopio y sus valores coinciden con los requeridos, esta ocasión la comprobación se realiza directamente porque el osciloscopio divide cada campo y muestra sus valores correspondientes.

Después del campo del CRC se muestra un campo en rojo con el signo “!” debido a un error de reconocimiento porque no hay un vehículo que responda.

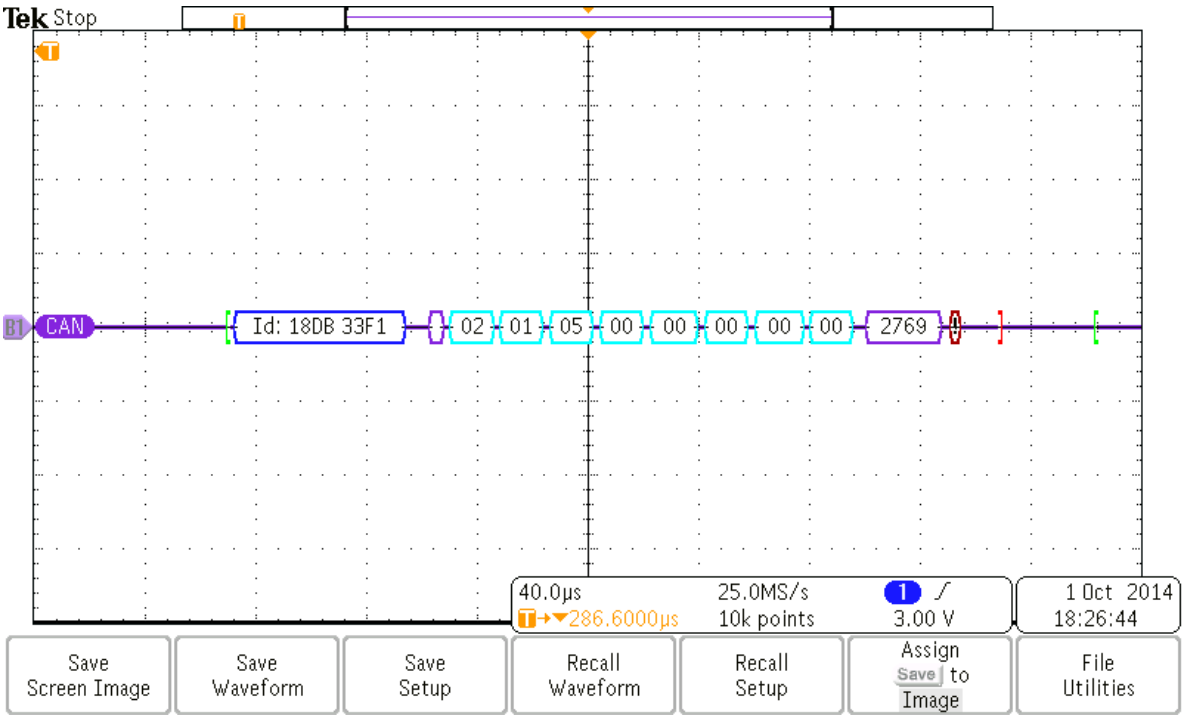


Figura 4.26. Trama con identificador de 29 bits.

Cálculo del consumo de combustible (Rendimiento)

El software del UNAM-CAN incluye el cálculo del rendimiento del motor del vehículo, ya que éste no es un parámetro estándar, por lo que no se puede obtener directamente el valor del consumo de combustible. Para realizar el cálculo del rendimiento se tiene que hacer uso de los parámetros que sí entregan los

vehículos. El método que se implementa en el UNAM-CAN utiliza el dato del sensor MAF y el de la velocidad del vehículo, con unidades de gramos/s de aire y km/h respectivamente. Los pasos para calcular el consumo de combustible son:

1. Se divide el dato obtenido del MAF entre la relación estequiométrica para obtener gramos de gasolina por segundo, ecuación 4.1. La relación estequiométrica para la gasolina es de 14.7 gramos de aire por cada gramo de gasolina.
2. Se divide el resultado anterior entre la densidad de la gasolina (720 g/L) para obtener litros de gasolina por segundo, ecuación 4.2. La densidad de la gasolina fue obtenida del "Diccionario de Términos de Pemex Refinación" p. 100.
3. De la ecuación 4.3, se multiplica el resultado por 3600 segundos para obtener litros de gasolina por hora.
4. Se divide la velocidad del vehículo entre el resultado del paso anterior para obtener km recorridos por litro de gasolina, ecuación 4.4.

$$\frac{MAF}{14.7} = \frac{g_{gasolina}}{s} \tag{4.1}$$

$$\frac{g_{gasolina}}{s \times 720} = \frac{L_{gasolina}}{s} \tag{4.2}$$

$$\frac{L_{gasolina} \times 3600\ s}{s} = \frac{L_{gasolina}}{h} \tag{4.3}$$

$$\frac{VSS\,[km/h]}{\frac{L_{gasolina}}{h}} = \frac{km}{litro} \tag{4.4}$$

La expresión final queda expresada en la ecuación 4.5.

$$\frac{VSS \times 2.94}{MAF} = \frac{km}{litro} \tag{4.5}$$

En donde:

- VSS: Velocidad del vehículo
- MAF: Valor del sensor MAF en g(aire)/s

Manejo de la tarjeta micro SD

La última subrutina de software que se probó fue la del manejo de la tarjeta micro SD, esta subrutina también envía mensajes en el LCD. Esta subrutina primero debe verificar si el UNAM-CAN tiene insertada una tarjeta de memoria, si no la tiene insertada o hay error, regresa al menú. La figura 4.27 muestra el resultado a la prueba en la que el UNAM-CAN no tiene insertada una tarjeta de memoria.

Figura 4.27. Tarjeta micro SD no encontrada.

El siguiente paso fue insertar una tarjeta de memoria micro SD para verificar que el software realizara lo correspondiente. Si se detecta una memoria micro SD, el UNAM-CAN muestra "Esperando satélites, CLICK para ignorar". La figura 4.28 muestra el resultado a la prueba con una tarjeta micro SD insertada.

Figura 4.28. Tarjeta micro SD detectada.

Otra prueba con la tarjeta SD fue comprobar la creación de un archivo de texto y que contuviera los datos que se planea guardar. Cuando se sincroniza el GPS, el UNAM-CAN empieza a guardar los datos del vehículo en la micro SD y se muestra la leyenda "Escribiendo en: *nombre de archivo.TXT"* en el que el nombre de archivo es la fecha en la que se está realizando la prueba, como lo muestra la figura 4.29. Si el UNAM-CAN no se encuentra conectado en algún vehículo no registra ningún dato, por lo tanto, para comprobar la creación de un archivo se modificó el código del UNAM-CAN para que existieran datos para registrar, por lo que las imágenes muestran datos "basura", que son los valores de inicialización de algunas variables del código.

Figura 4.29. Mensaje con el nombre del archivo que se está escribiendo.

Después de darle la orden al UNAM-CAN de terminar de registrar los datos, se extrajo la tarjeta de memoria para verificar su contenido en una PC. La figura 4.30 es una captura de pantalla de dicha verificación, en la que observamos que se generó el archivo "28-11-14.txt".

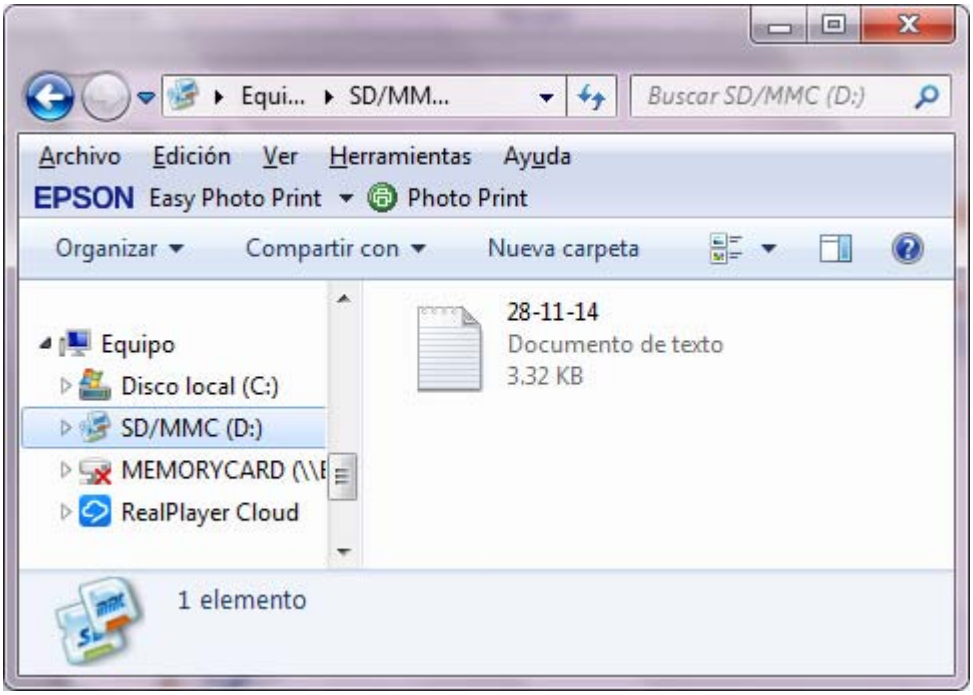


Figura 4.30. Archivo generado en la memoria micro SD.

El formato con el que están acomodados los datos que contiene el archivo de texto es tal que al llevarlos a una hoja de cálculo, en este caso Excel, figura 4.31, éstos están legibles, pueden ser fácilmente manipulados e identificados.

	A	B	C	D	E	F	G	H	I
1	23:30:54 UTC								
2	coordenadas	Rendimiento [km/L]	Temp. Anticongelante [gradC]	RPM	Velocidad [km/h]	Sensor MAF [g/s]	sensor O2 [V]	% Pos. Mariposa	
3	19.3275,-99.1808	1.14	215	16383.75	255	655.35	1.27	100	
4	19.3275,-99.1808	1.14	215	16383.75	255	655.35	1.27	100	
5	19.3275,-99.1808	1.14	215	16383.75	255	655.35	1.27	100	
6	19.3275,-99.1808	1.14	215	16383.75	255	655.35	1.27	100	
7	19.3275,-99.1808	1.14	215	16383.75	255	655.35	1.27	100	
8	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
9	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
10	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
11	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
12	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
13	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
14	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
15	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
16	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
17	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
18	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
19	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
20	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
21	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
22	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	
23	19.3275,-99.1809	1.14	215	16383.75	255	655.35	1.27	100	

Figura 4.31. Datos de la memoria en una hoja de cálculo.

4.3. Pruebas del UNAM-CAN en vehículos

Las pruebas concluyentes y definitivas del UNAM-CAN fueron las pruebas hechas en vehículos. Estas pruebas también siguieron una secuencia y dado que para hacer pruebas con diferentes vehículos se tiene que seguir el mismo proceso que incluye decodificar al PID 0x00, se creó un “decodificador” Excel utilizando sus funciones. De manera general, la secuencia de pruebas consiste en:

1. Verificar que el UNAM-CAN se comunique con el vehículo bajo prueba, para ello se identifica la ubicación del conector DLC del automóvil.
2. Después, por medio del cable OBD2-DB9, se conecta el UNAM-CAN. Generalmente los vehículos en su conector tienen habilitadas las terminales que están conectadas a la batería, por lo que el UNAM-CAN enciende inmediatamente al conectarlo.
3. Una vez hecha la conexión, encender el *switch* del vehículo y en el menú del UNAM-CAN se elige la opción “Datos del vehiculo”, esta función tiene como principal objetivo pedirle datos al vehículo utilizando los PID, el primero es el 0x00.
4. Por medio del PID 0x00 se solicita a la ECU del vehículo acerca de los parámetros que se pueden obtener de él; podríamos decir que es el PID más importante, porque funciona en cualquier vehículo que cuente con OBD-II y la respuesta a éste es una lista de los otros PID que el vehículo soporta, figura 4.32, lo que significa que nos devuelve una lista (codificada) de la información que podemos obtener del vehículo.

Figura 4.32. Respuesta al PID 0x00.

5. En esta primera versión, el UNAM-CAN no hace un reconocimiento automático (basándose en la respuesta al PID 0x00) de la información que puede pedirle al vehículo, por lo que se requiere de este paso. La respuesta al PID 0x00 consta de 4 bytes, en los que cada bit (del más significativo al menos significativo) representa a uno de los siguientes 32 PIDs, donde un "1" lógico en la celda de bit indica que el PID está soportado por el vehículo. Cada digito de la respuesta se introdujo en cada celda resaltada del decodificador hecho en Excel, figura 4.33, el resultado es la lista de los parámetros decodificada.

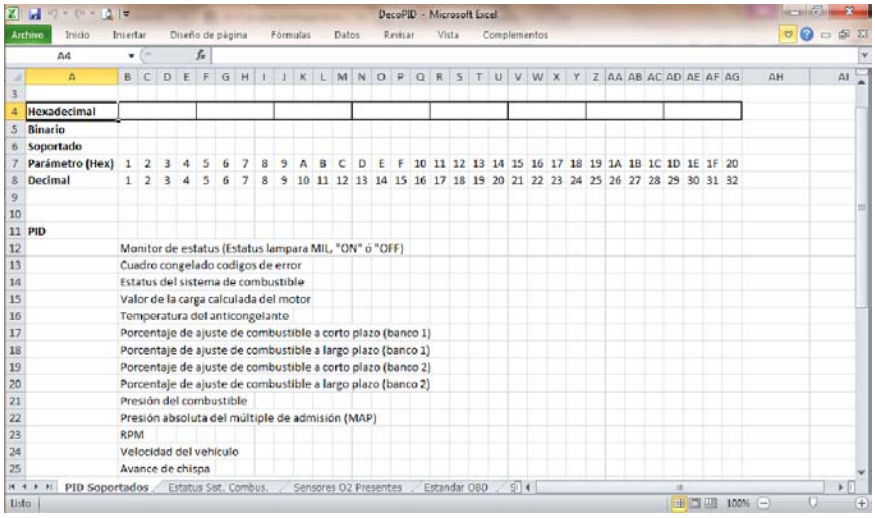


Figura 4.33. Decodificador del PID 0x00.

6. El resultado que muestra este decodificador, figura 4.34, es la lista de los parámetros o PID que el vehículo soporta, estos aparecen resaltados en color verde y en rojo los que no están soportados (recuadro rojo), debajo tienen su respectivo valor en hexadecimal y en decimal (recuadro azul), y en la parte inferior aparece la descripción de los mismos. Los parámetros de la lista decodificada se programan posteriormente en el UNAM-CAN.

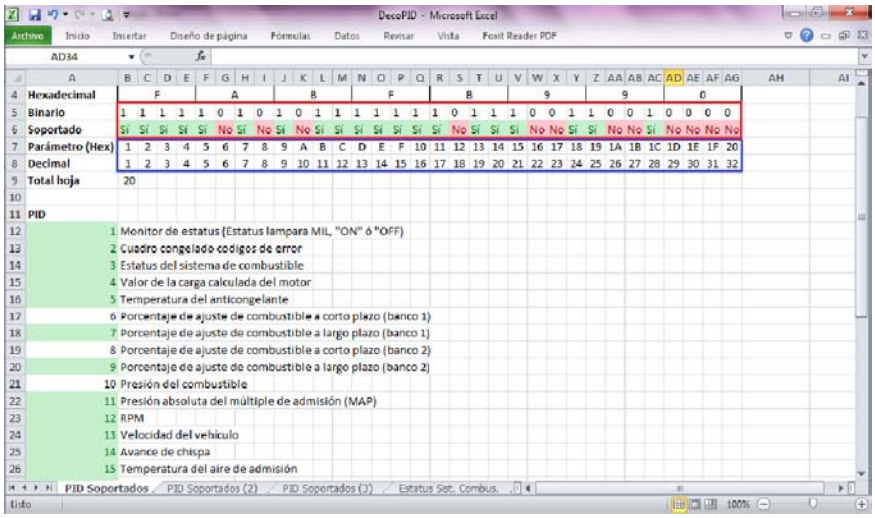


Figura 4.34. Resultado del decodificador.

7. Por último el UNAM-CAN se vuelve a conectar al vehículo y se hacen los registros de los datos en la memoria micro SD.

La secuencia de prueba mencionada se realizó en diferentes vehículos, entre los que cabe destacar a los de gasolina, con CAN de 11 bits, a los de gasolina, con CAN de 29 bits, y de motor a Diesel, con CAN de 29 bits.

4.3.1. Pruebas en vehículo de gasolina con CAN estándar

En la prueba de vehículo de gasolina con CAN estándar, la figura 4.35 muestra la respuesta en hexadecimal al PID 0x00.

Figura 4.35. Respuesta del vehículo a gasolina bajo prueba.

El siguiente paso, según el procedimiento, es decodificar la respuesta “BE 3E B0 10”, figura 4.36, esto para identificar los PID soportados por el vehículo, para que por último esos PID se programen en el UNAM-CAN.

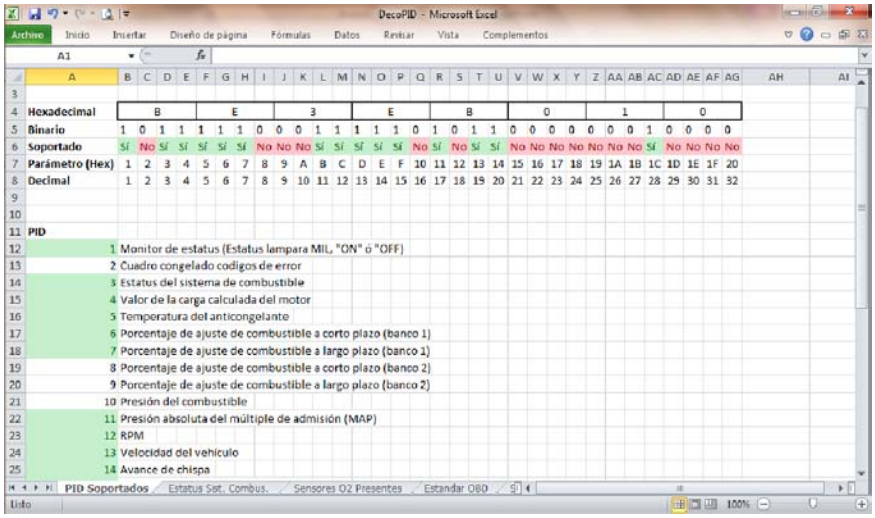
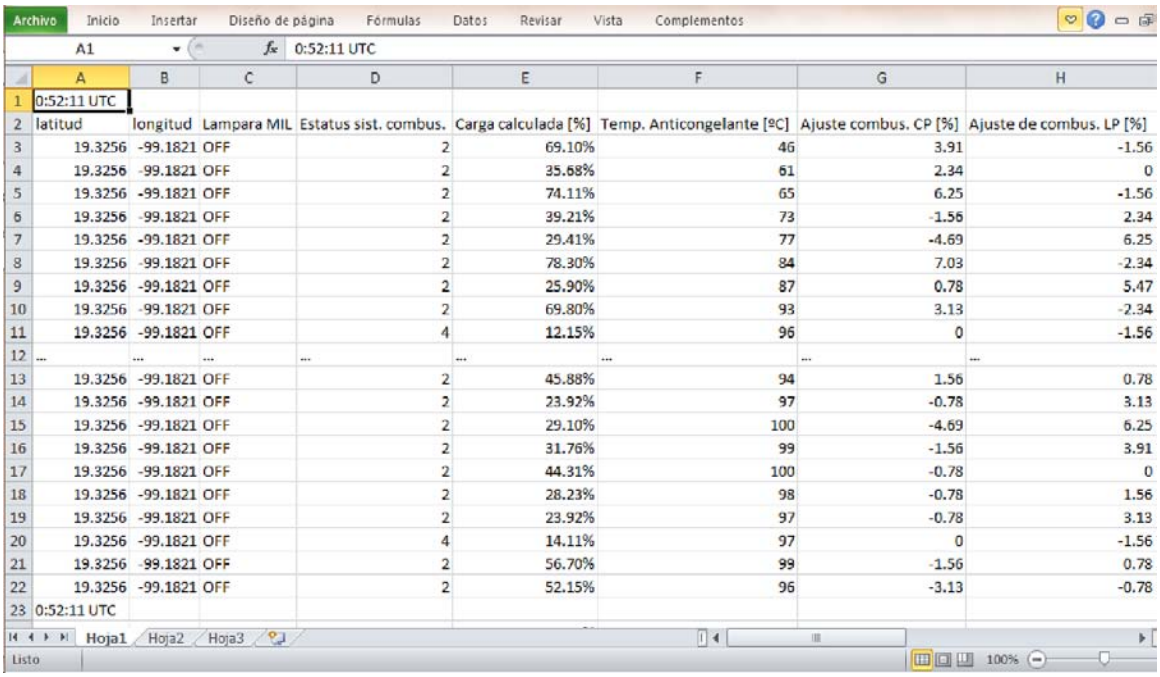


Figura 4.36. Decodificación de la respuesta al PID 0x00.

Una vez que el UNAM-CAN fue programado se hicieron las peticiones y guardado de los datos del vehículo. La figura 4.37 muestra el resultado de uno de los parámetros pedidos, el cual corresponde a la temperatura del aire de admisión.

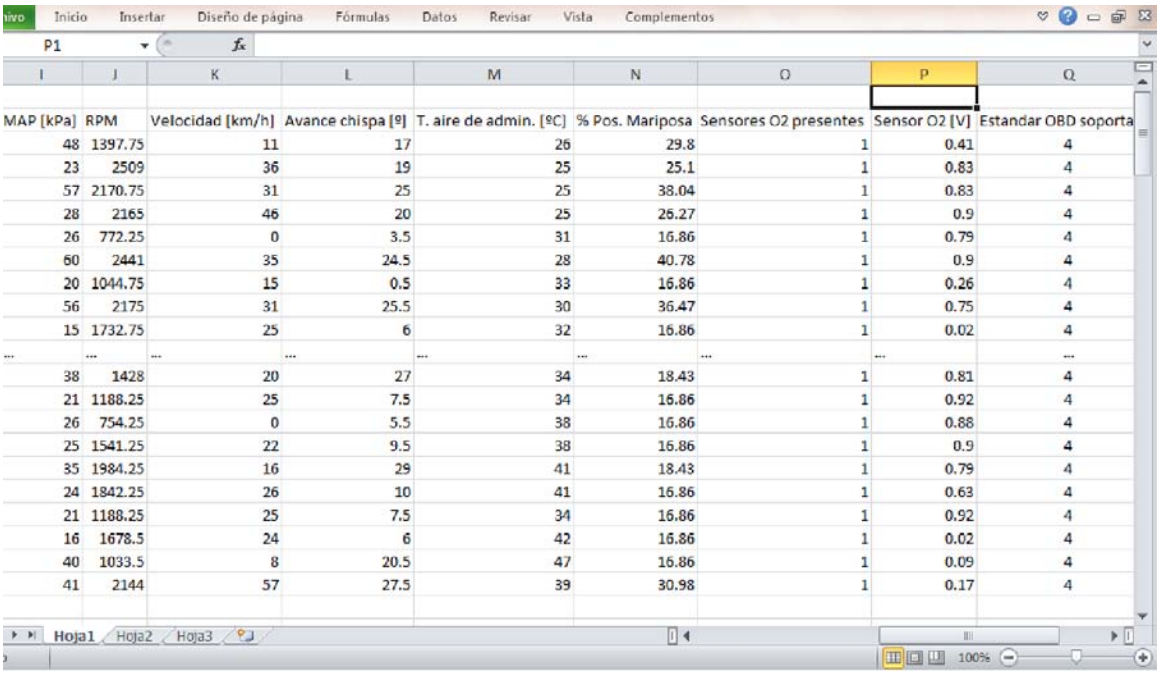
Figura 4.37. Temperatura del aire de admisión.

Por último, la prueba más importante es la de registrar los datos del vehículo en la tarjeta de memoria micro SD; también es la prueba más completa porque ésta hace uso de casi todos los componentes del UNAM-CAN y del *software* que los controla. El archivo generado con esta prueba, figura 4.38, tiene la hora UTC del comienzo de la prueba en la celda A1, dos columnas (latitud-longitud) que contienen las coordenadas de la posición del vehículo y columnas con el valor de cada dato pedido al vehículo.



	A	B	C	D	E	F	G	H
1	0:52:11 UTC							
2	latitud	longitud	Lampara MIL	Estatus sist. combus.	Carga calculada [%]	Temp. Anticongelante [ºC]	Ajuste combus. CP [%]	Ajuste de combus. LP [%]
3	19.3256	-99.1821	OFF	2	69.10%	46	3.91	-1.56
4	19.3256	-99.1821	OFF	2	35.68%	61	2.34	0
5	19.3256	-99.1821	OFF	2	74.11%	65	6.25	-1.56
6	19.3256	-99.1821	OFF	2	39.21%	73	-1.56	2.34
7	19.3256	-99.1821	OFF	2	29.41%	77	-4.69	6.25
8	19.3256	-99.1821	OFF	2	78.30%	84	7.03	-2.34
9	19.3256	-99.1821	OFF	2	25.90%	87	0.78	5.47
10	19.3256	-99.1821	OFF	2	69.80%	93	3.13	-2.34
11	19.3256	-99.1821	OFF	4	12.15%	96	0	-1.56
12	...	...	...	...	...	...	...	...
13	19.3256	-99.1821	OFF	2	45.88%	94	1.56	0.78
14	19.3256	-99.1821	OFF	2	23.92%	97	-0.78	3.13
15	19.3256	-99.1821	OFF	2	29.10%	100	-4.69	6.25
16	19.3256	-99.1821	OFF	2	31.76%	99	-1.56	3.91
17	19.3256	-99.1821	OFF	2	44.31%	100	-0.78	0
18	19.3256	-99.1821	OFF	2	28.23%	98	-0.78	1.56
19	19.3256	-99.1821	OFF	2	23.92%	97	-0.78	3.13
20	19.3256	-99.1821	OFF	4	14.11%	97	0	-1.56
21	19.3256	-99.1821	OFF	2	56.70%	99	-1.56	0.78
22	19.3256	-99.1821	OFF	2	52.15%	96	-3.13	-0.78
23	0:52:11 UTC							

Figura 4.38. Datos del vehículo registrados en la memoria (continúa).



I	J	K	L	M	N	O	P	Q
MAP [kPa]	RPM	Velocidad [km/h]	Avance chispa [º]	T. aire de admin. [ºC]	% Pos. Mariposa	Sensores O2 presentes	Sensor O2 [V]	Estandar OBD soporta
48	1397.75	11	17	26	29.8	1	0.41	4
23	2509	36	19	25	25.1	1	0.83	4
57	2170.75	31	25	25	38.04	1	0.83	4
28	2165	46	20	25	26.27	1	0.9	4
26	772.25	0	3.5	31	16.86	1	0.79	4
60	2441	35	24.5	28	40.78	1	0.9	4
20	1044.75	15	0.5	33	16.86	1	0.26	4
56	2175	31	25.5	30	36.47	1	0.75	4
15	1732.75	25	6	32	16.86	1	0.02	4
...	...	...	...	...	...	...	...	...
38	1428	20	27	34	18.43	1	0.81	4
21	1188.25	25	7.5	34	16.86	1	0.92	4
26	754.25	0	5.5	38	16.86	1	0.88	4
25	1541.25	22	9.5	38	16.86	1	0.9	4
35	1984.25	16	29	41	18.43	1	0.79	4
24	1842.25	26	10	41	16.86	1	0.63	4
21	1188.25	25	7.5	34	16.86	1	0.92	4
16	1678.5	24	6	42	16.86	1	0.02	4
40	1033.5	8	20.5	47	16.86	1	0.09	4
41	2144	57	27.5	39	30.98	1	0.17	4

Figura 4.38. Datos del vehículo registrados en la memoria.

4.3.2. Prueba en vehículo Diesel con CAN estándar

En vehículos Diesel que tengan el sistema OBD-II, el procedimiento es exactamente el mismo que el descrito para vehículos de gasolina, entonces el

primer paso es enviarle al vehículo el PID 0x00. El vehículo en prueba fue una camioneta FORD XL Super Duty, los parámetros que soporta se muestran codificados en la figura 4.39.

Figura 4.39. Parámetros soportados por el vehículo Diesel.

Como podemos observar, el valor hexadecimal obtenido termina en "7", por lo tanto el último bit de ese último nibble es "1", este bit indica si existen PIDs de los siguientes 32 soportados por el vehículo. Para saber cuál de los siguientes 32 PIDs (0x21-0x40) son soportados, se requiere enviar un parámetro similar a 0x00, mostrado como "PID 2" en la figura 4.40, que en realidad es el 0x20.

Figura 4.40. PID soportados del 0x21 al 0x40.

En el resultado de la solicitud mencionada, el último byte tiene el valor "0x01", por lo que también el último bit es "1", lo que significa que existen más PIDs soportados, de los siguientes 32. El tercer PID similar al 0x00 es el 0x40, para saber cuál de los PID del 0x41 al 0x60 están soportados. En la figura 4.41 se muestra la respuesta a éste, observamos que el último valor es "0x00", lo cual nos indica que ya no hay otros PID soportados más allá del 0x60 y terminamos de hacer las peticiones.

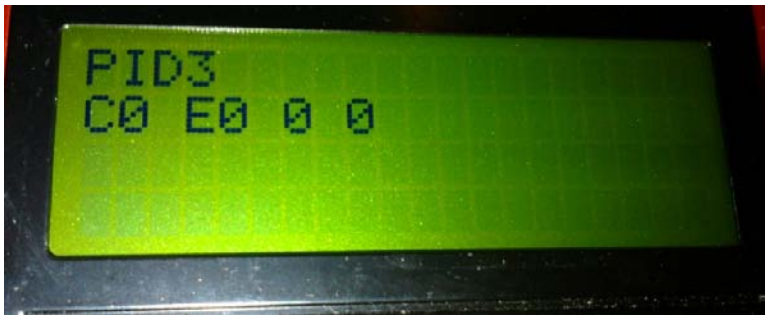


Figura 4.41. PID soportados del 0x41 al 0x60.

Se utiliza nuevamente el decodificador de Excel para escribir estas tres respuestas y saber qué parámetro pedirle al vehículo, figura 4.42.

4	**Hexadecimal**	9				8				3				B				0				0				1				7	
5	**Binario**	1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1
6	**Soportado**	Sí	No	No	Sí	Sí	No	No	No	No	No	Sí	Sí	Sí	No	Sí	Sí	No	No	No	No	No	No	No	No	No	No	Sí	No	Sí	Sí
7	**Parámetro (Hex)**	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20
8	**Decimal**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
9	**Total hoja**	12																															
10	Total	31																															
11	**PID**																																
12	1	Monitor de estatus (Estatus lampara MIL, "ON" ó "OFF")																															
13	2	Cuadro congelado codigos de error																															
14	3	Estatus del sistema de combustible																															
15	4	Valor de la carga calculada del motor																															
16	5	Temperatura del anticongelante																															
17	6	Porcentaje de ajuste de combustible a corto plazo (banco 1)																															
18	7	Porcentaje de ajuste de combustible a largo plazo (banco 1)																															
19	8	Porcentaje de ajuste de combustible a corto plazo (banco 2)																															
20	9	Porcentaje de ajuste de combustible a largo plazo (banco 2)																															
21	10	Presión del combustible																															
22	11	Presión absoluta del múltiple de admisión (MAP)																															
23	12	RPM																															

Último bit

PID Soportados | PID Soportados (2) | PID Soportados (3) | Estatus Sist. Combus.

Listo 100%

39	28	Estandar OBD con el cual cumple el vehículo
40	29	Sensores de oxígeno presentes (similar a PID 13)
41	30	Estatus de la entrada auxiliar (Toma de Potencia)
42	31	Tiempo transcurrido desde el encendido del motor
43	32	PID soportados [21-40], similar al PID 00

Último bit

Figura 4.42. PIDs soportados por el vehículo Diesel.

Después de programar al UNAM-CAN con los PIDs que indica el decodificador, se pidieron y guardaron sus datos, entre ellos la temperatura del anticongelante en grados Celsius, figura 4.43.

Figura 4.43. Temperatura del anticongelante del vehículo Diesel.

Uno de los parámetros soportados por este vehículo es su nivel de combustible, la figura 4.44 es un comparativo del nivel de combustible en tanque reportado en su tablero y el reportado a través de su PID, en la figura se observa que los valores corresponden entre sí.

Figura 4.44. Nivel de combustible visto en tablero y en el UNAM-CAN.

El vehículo también proporciona las RPM de su motor, dicho valor se obtuvo con el motor apagado y con el motor en ralentí, la figura 4.45 muestra los dos valores respectivamente.

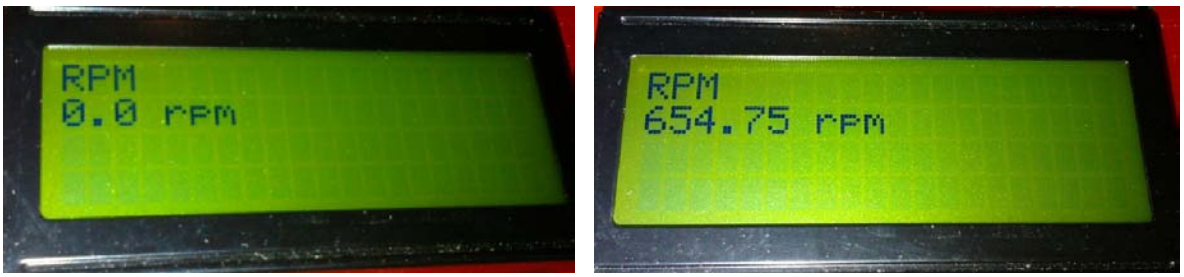
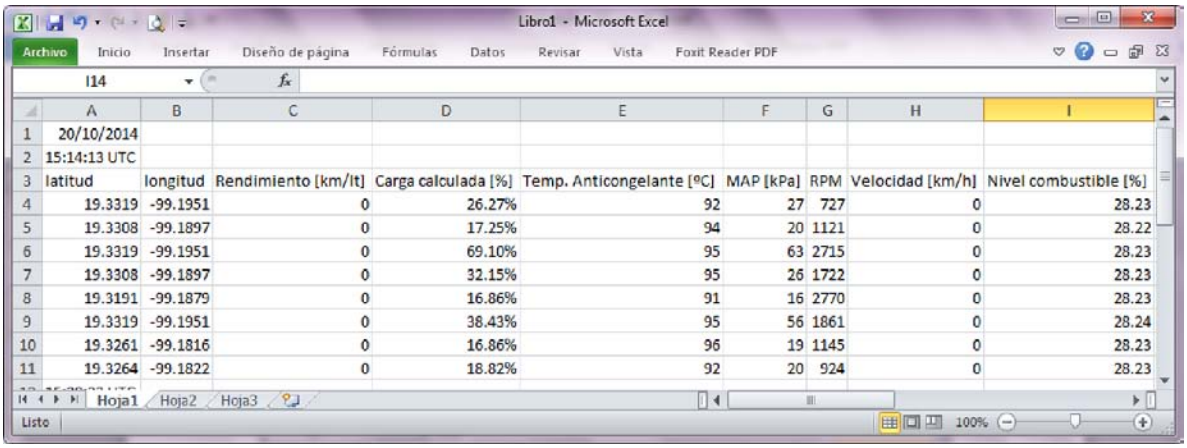


Figura 4.45. RPM con el motor apagado y en ralentí.

La última prueba a este vehículo fue registrar sus datos en la memoria, figura 4.46.



Libro1 - Microsoft Excel

	A	B	C	D	E	F	G	H	I
1	20/10/2014								
2	15:14:13 UTC								
3	latitud	longitud	Rendimiento [km/lt]	Carga calculada [%]	Temp. Anticongelante [ºC]	MAP [kPa]	RPM	Velocidad [km/h]	Nivel combustible [%]
4	19.3319	-99.1951	0	26.27%	92	27	727	0	28.23
5	19.3308	-99.1897	0	17.25%	94	20	1121	0	28.22
6	19.3319	-99.1951	0	69.10%	95	63	2715	0	28.23
7	19.3308	-99.1897	0	32.15%	95	26	1722	0	28.23
8	19.3191	-99.1879	0	16.86%	91	16	2770	0	28.23
9	19.3319	-99.1951	0	38.43%	95	56	1861	0	28.24
10	19.3261	-99.1816	0	16.86%	96	19	1145	0	28.23
11	19.3264	-99.1822	0	18.82%	92	20	924	0	28.23

Figura 4.46. Datos registrados del vehículo Diesel.

4.3.3. Pruebas en vehículo de gasolina con CAN extendido

En el transcurso del proyecto se planteó hacer pruebas con un vehículo con equipamiento mayor a los de las pruebas anteriores, el vehículo elegido fue un Acura TL 2010. Cuando se le hizo la petición del primer parámetro no hubo respuesta. Para encontrar la causa de la falta de comunicación, primero se verificó, mediante las terminales del conector OBD-II, que contara con el protocolo CAN. El resultado fue que, efectivamente tenía CAN y OBD-II. Antes de buscar alguna avería en el vehículo, se realizó el cambio a CAN extendido en la comunicación del UNAM-CAN, lo que resultó en una comunicación exitosa. Al cuestionar al vehículo bajo prueba sobre sus parámetros soportados, la respuesta fue en tres partes, como en el caso del apartado anterior, la figura 4.47 muestra las respuestas.

Figura 4.47. Información de los PID soportados por Acura TL.

El resultado que arrojó el decodificador nos lo muestra la figura 4.48, en un recuadro rojo se resalta al bit que indica que existen otros PID soportados por el vehículo.

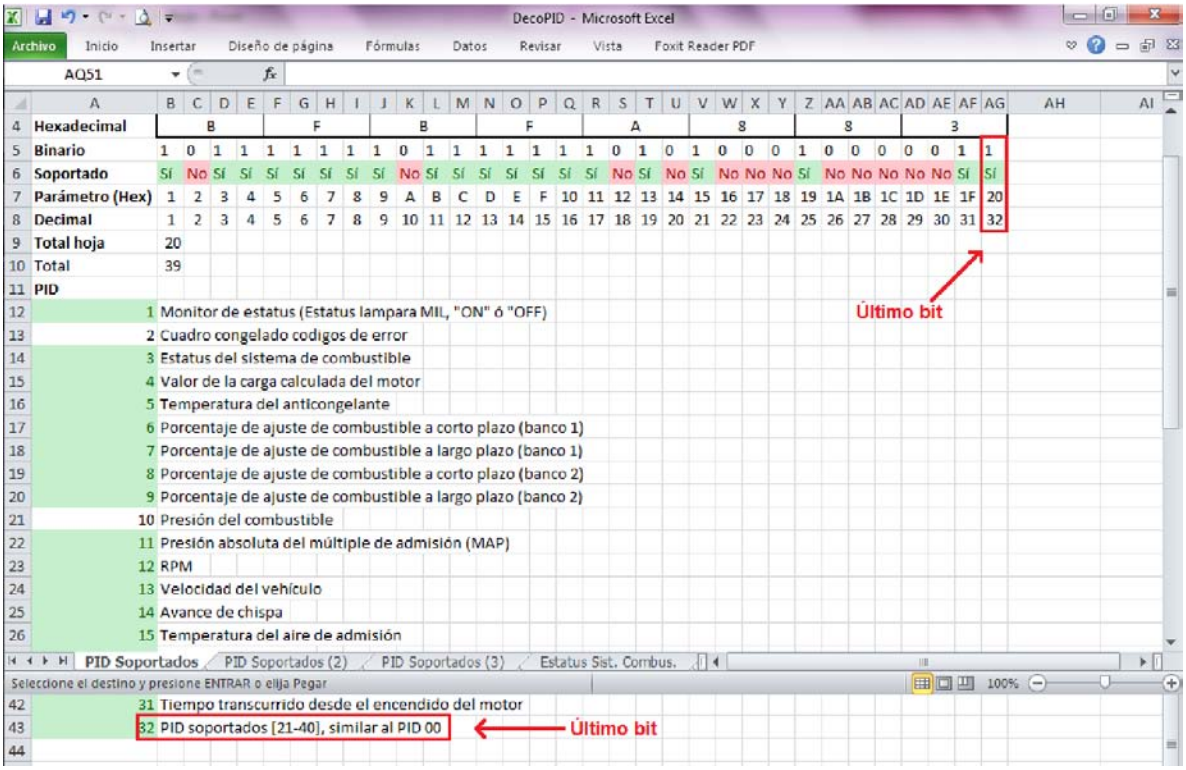


Figura 4.48. PIDs soportados por el vehículo Acura TL.

Después de programar al UNAM-CAN, el último paso fue el registro de los parámetros del Acura TL. La figura 4.49 muestra una parte de los datos registrados en la memoria micro SD.

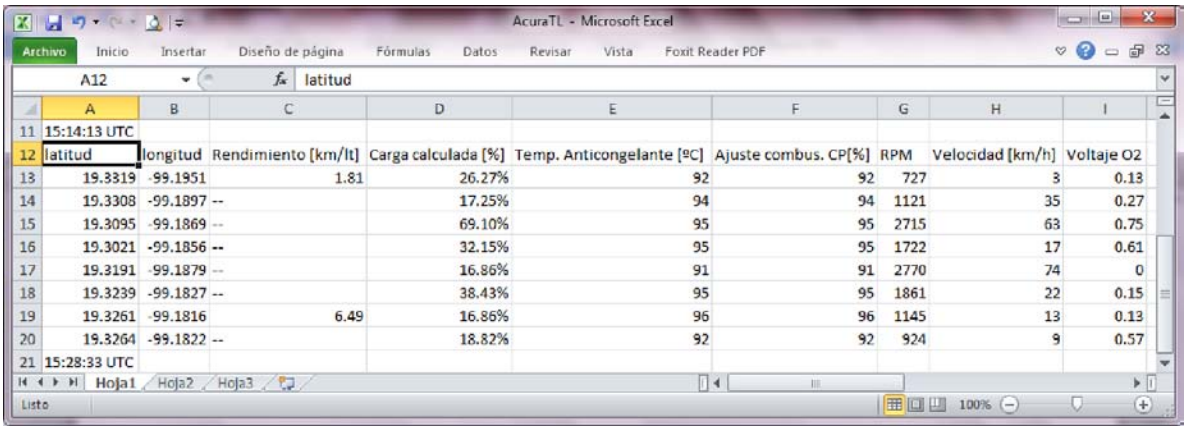


Figura 4.49. Parámetros registrados en memoria del vehículo Acura TL.

4.3.4. Pruebas adicionales

En las pruebas anteriormente descritas se encontraron algunas fallas, entre las cuales se pueden destacar dos importantes, como: pérdida de datos en la columna del rendimiento, figura 4.50, y en el tiempo de adquisición para el registro de los datos

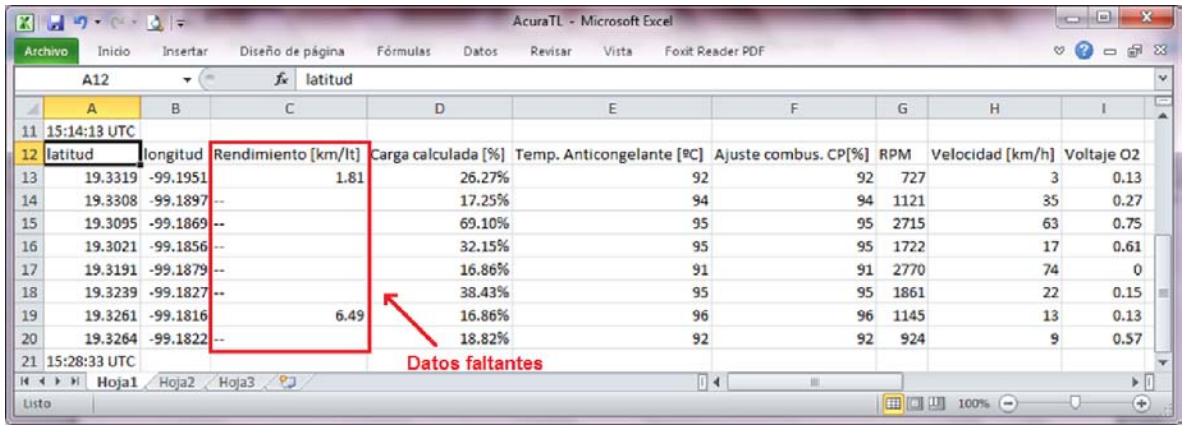
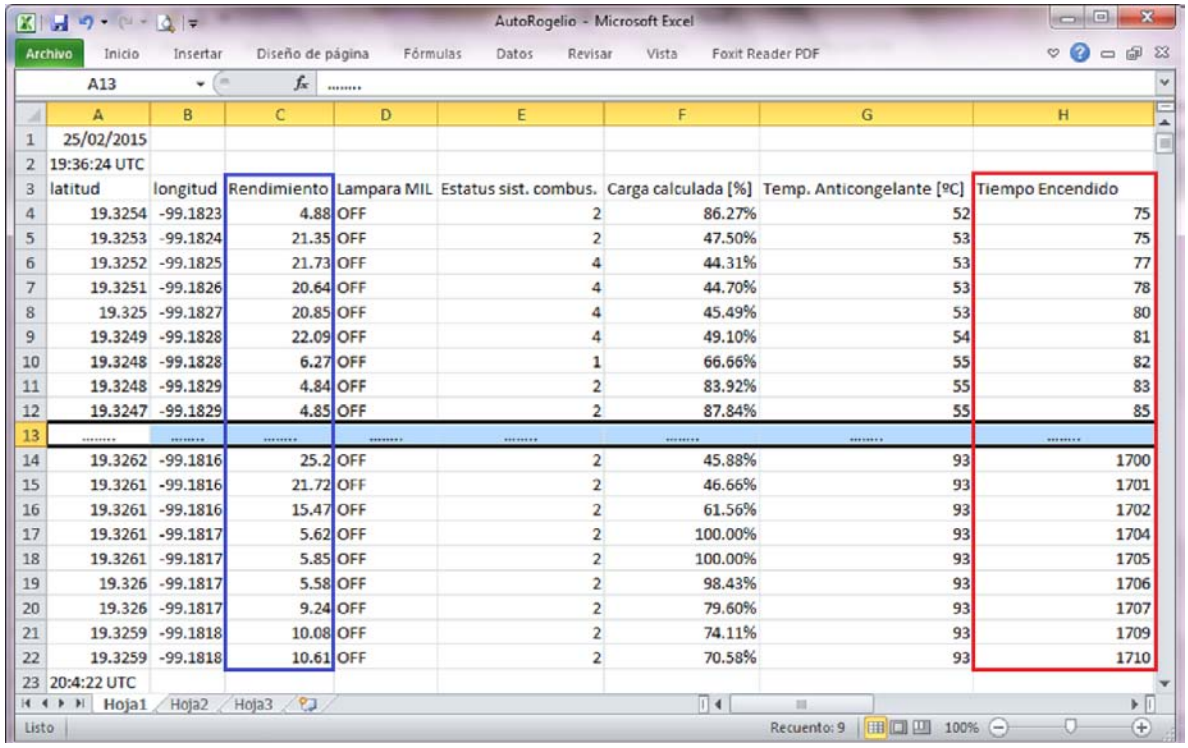


Figura 4.50. Pérdida de datos del rendimiento.

Como hemos mencionado, el dato del rendimiento se obtiene a partir de dos valores reportados por el vehículo. El error consistía en que no siempre había tiempo suficiente para obtener el valor de los dos parámetros, por lo que no se calculaba el rendimiento, la solución fue modificar el código para dar tiempo para obtener los datos requeridos.

El segundo error consistía en que el tiempo para el registro de los datos no era homogéneo, había una variación de este tiempo, prácticamente aleatoria. Para tener un control preciso sobre este tiempo, por software se ajustó el retardo para que el registro tuviera un segundo de diferencia entre cada bloque de valores.

Después de realizar las correcciones pertinentes, se realizaron una serie de pruebas adicionales, incluyendo a los vehículos de las primeras pruebas y a otros más, de los cuales pondremos como ejemplo a un vehículo Corolla de la marca Toyota, las pruebas en este vehículo siguieron la misma secuencia que todos los mencionados. Las correcciones se observan en figura 4.51; con un recuadro azul se resaltan los datos del rendimiento sin pérdidas de datos y con un recuadro rojo se resalta el parámetro “Tiempo de encendido”, que indica el tiempo en segundos que ha estado encendido el motor del vehículo, que nos puede dar una idea aproximada del tiempo de registro.



	A	B	C	D	E	F	G	H
1	25/02/2015							
2	19:36:24 UTC							
3	latitud	longitud	Rendimiento	Lampara MIL	Estatus sist. combus.	Carga calculada [%]	Temp. Anticongelante [ºC]	Tiempo Encendido
4	19.3254	-99.1823	4.88	OFF	2	86.27%	52	75
5	19.3253	-99.1824	21.35	OFF	2	47.50%	53	75
6	19.3252	-99.1825	21.73	OFF	4	44.31%	53	77
7	19.3251	-99.1826	20.64	OFF	4	44.70%	53	78
8	19.325	-99.1827	20.85	OFF	4	45.49%	53	80
9	19.3249	-99.1828	22.09	OFF	4	49.10%	54	81
10	19.3248	-99.1828	6.27	OFF	1	66.66%	55	82
11	19.3248	-99.1829	4.84	OFF	2	83.92%	55	83
12	19.3247	-99.1829	4.85	OFF	2	87.84%	55	85
13	.....	.....	.....	.....	.....	.....	.....	.....
14	19.3262	-99.1816	25.2	OFF	2	45.88%	93	1700
15	19.3261	-99.1816	21.72	OFF	2	46.66%	93	1701
16	19.3261	-99.1816	15.47	OFF	2	61.56%	93	1702
17	19.3261	-99.1817	5.62	OFF	2	100.00%	93	1704
18	19.3261	-99.1817	5.85	OFF	2	100.00%	93	1705
19	19.326	-99.1817	5.58	OFF	2	98.43%	93	1706
20	19.326	-99.1817	9.24	OFF	2	79.60%	93	1707
21	19.3259	-99.1818	10.08	OFF	2	74.11%	93	1709
22	19.3259	-99.1818	10.61	OFF	2	70.58%	93	1710
23	20:4:22 UTC							

Figura 4.51. Datos registrados del Toyota con las mejoras incluidas.

4.3.5. Prototipo final

Una vez que las pruebas se realizaron sin problemas, y los errores de *software* fueron corregidos, se montó el *hardware* del UNAM-CAN dentro de un gabinete habilitado para tal fin, como lo muestra la figura 4.52.

Figura 4.52. Montaje de la tarjeta en el gabinete.

Las imágenes de la figura 4.53 muestran vistas en diferentes ángulos del gabinete que contiene a la tarjeta del UNAM-CAN.

Figura 4.53. Vistas del gabinete y del sistema.

Como se ha mencionado, el UNAM-CAN se conecta con el conector OBD del vehículo utilizando un cable con conector DB9 y OBD, cada uno en cada extremo. Debido a lo anterior, al gabinete se le hizo una abertura por la cual se tiene acceso al conector DB9 del UNAM-CAN. Las imágenes de la figura 4.54 muestran al conector sobresaliendo de él y el cable conectado respectivamente.

Figura 4.54. Conexión del UNAM-CAN.

A modo de verificaciones finales, con el UNAM-CAN montado se hicieron pruebas en los mismos vehículos, las cuales resultaron exitosas. La figura 4.55 muestra la conexión del UNAM-CAN en uno de los vehículos bajo prueba, en ella podemos ver la ubicación de su conector, dicha ubicación, según las especificaciones, es estándar.

Figura 4.55. UNAM-CAN conectado a un vehículo.

A lo largo del capítulo se habló de las diferentes pruebas hechas al UNAM-CAN en sus diferentes etapas a lo largo del proyecto, en el siguiente capítulo se hablará de algunos de los resultados logrados y las conclusiones del proyecto, adicionalmente se darán algunas sugerencias de las mejoras que podrían efectuarse.

Capítulo 5 Resultados y Conclusiones

En este capítulo se abordan los resultados que se obtuvieron del trabajo y de la integración del sistema para la obtención de datos de vehículos automotores con OBD-II utilizando protocolo CAN. También se proporcionarán conclusiones acerca del mismo y se mencionarán algunas recomendaciones para la mejora y actualización del sistema en cuestión.

5.1. Resultados

Se logró integrar un sistema capaz de comunicarse con el sistema OBD-II de vehículos automotores, el cual es un sistema de monitoreo a bordo estandarizado. La comunicación de nuestro sistema con OBD-II se realiza utilizando el protocolo CAN, ya sea estándar (de 11 bits en el identificador) o extendido (29 bits en el identificador).

Se cumplió con las especificaciones iniciales dadas para el sistema que se desarrolló, además de algunas modificaciones y nuevas especificaciones que surgieron a lo largo del desarrollo del proyecto. Los resultados obtenidos son:

- Software completamente intuitivo, sólo existe un *Jostick* para interactuar con un menú muy sencillo y fácil de interpretar.
- El UNAM-CAN es completamente portable.
- Datos del vehículo desplegables en el display de cristal líquido.
- Registro de datos con formato legible de parámetros vehiculares con fecha, hora y localización, en una tarjeta de memoria micro SD.
- El UNAM-CAN no requiere de una batería de alimentación interna, la energía la toma de la batería del vehículo.
- Los datos se registran en archivos de texto, que pueden ser leídos fácilmente en una computadora. El sistema registra los datos en la memoria con sistema de archivos FAT32.
- Adicionalmente, el UNAM-CAN cuenta con un módulo USB mediante el cual se puede reprogramar y hacer ajustes con una computadora personal.
- Se implementó un algoritmo para calcular el consumo de combustible del motor.
- UNAM-CAN funciona en todos los automóviles que cuenten con OBD-II y con bus CAN.

5.2. Conclusiones

El objetivo del presente trabajo ha sido el desarrollo e integración de un sistema de obtención y registro de datos de parámetros vehiculares. Para lograr este objetivo, se propuso el diseño y desarrollo de un sistema basado en microcontrolador, que fuera capaz de enviar y recibir datos a través del protocolo CAN. Los datos enviados hacia la unidad de control del vehículo le indican que se trata de una petición de datos y le indica también de que dato se trata, las respuestas pueden visualizarse en un LCD, además el sistema tiene la capacidad de registrar la información proveniente de dicha unidad en tarjetas de memoria flash SD, utilizando el sistema de archivos FAT32.

Los vehículos actuales cuentan con una o varias unidades de control electrónicas con el propósito de gestionar ciertos parámetros del vehículo y así asegurar un correcto funcionamiento, también realizan funciones de diagnóstico del motor así como de los componentes asociados y almacenan los errores que se detecten, esto es la esencia a lo que se le llama Diagnóstico a Bordo o mejor conocido como OBD. La segunda generación de este sistema a bordo, OBD-II, una de las características que ofrece es la estandarización de parámetros y tipo de conector de acceso al mismo. El sistema aprovecha este hecho, dando como resultado una compatibilidad con la mayoría de los automóviles, tanto con su conector como con la forma obtener los datos.

Como se sabe, prácticamente todo sistema requiere de mejoras, en nuestro caso el sistema desarrollado corresponde a la primera versión y necesita de varias mejoras. Cuando se realizaba la integración y pruebas posteriores al sistema se observaron ciertos detalles a los que se les podrían hacer mejoras, algunas de ellas son:

- Añadir un capacitor entre el microcontrolador y el FTDI para proporcionar un *reset* cuando se utiliza la programación por USB.
- Girar el conector del LCD para que quede hacia afuera de la tarjeta y no hacia adentro como actualmente.
- Redistribución de los conectores DB9, USB y el socket de la memoria micro SD, para que queden disponibles fuera del gabinete y tener un mejor acceso a ellos.
- Añadirle al sistema, además del protocolo CAN; el protocolo PWM, el VPW, el ISO9141-2 y el KWP2000, y así expandir la posibilidad de analizar mayor cantidad de vehículos.
- El software debe ser capaz de realizar el reconocimiento automático de los parámetros que nos puede reportar el vehículo en prueba, con esto se podrá realizar pruebas en un solo paso, en lugar de dos, como hasta el momento se hace.
- Identificación automática de la trama CAN, ya sea estándar o extendida, que el vehículo bajo prueba utiliza.

- Debido a estas y posteriores mejoras se requerirá de un microcontrolador con mayor capacidad de memoria de programa.

La implementación de CAN por medio de un controlador para tal propósito ha sido muy sencilla, su hoja de especificaciones indica cuales son los pasos a realizar para la cada función que necesitemos lograr, desde iniciar la inicialización, hasta la forma de enviar y recibir datos con CAN. El Instituto de Ingeniería tiene poca participación con respecto a CAN bus, por las modificaciones y mejoras que se tendrán que hacer al presente trabajo, el instituto seguirá en la exploración de este protocolo robusto. El siguiente paso sería que en el Instituto de Ingeniería se escalara hacia el desarrollo de las capas del modelo ISO/OSI que CAN no define (recordando que define la capa de enlace de datos y la capa física) para adaptarlo a las necesidades que se requieran en proyectos del instituto.

GLOSARIO

ABS. *Anti-lock Break Sistem.* Sistema de frenos que evita el bloqueo de las ruedas del vehículo.

ALU. *Arithmetic and Logic Unit*. Componente de la unidad central de procesamiento encargada de realizar las operaciones aritméticas y también de las operaciones lógicas.

BPB. *BIOS Parameter Block*. El BPB contiene la información sobre el sistema de archivos, que nos servirá para poder leer y escribir en él medio de almacenamiento o volumen.

CAN. *Controller Area Network.* Es un protocolo de comunicación en serie desarrollado por Bosch para el intercambio de información entre unidades de control electrónicas del automóvil.

CARB. *California Air Resources Board*. Es el órgano público dentro del gobierno de California responsable por la vigilar y mantener la calidad del aire en niveles saludables en todo el estado.

CPU. *Central Processing Unit.* Es la unidad central de procesamiento de un sistema de cómputo.

CRC. *Cyclic Redundancy Code.* Código de detección de errores usado en redes digitales y dispositivos de almacenamiento de datos, usado para detectar cambios accidentales en los datos.

Display. Dispositivo de ciertos aparatos electrónicos, como los teléfonos y las calculadoras, destinado a la representación visual de información.

DLC. *Data Link Connector.* Es el conector de diagnóstico automotriz estandarizado por OBD-II.

DTC. *Diagnostic Trouble Code.* Son códigos que indican una probable falla y ubicación.

ECU. *Electrónic Control Unit.* Es una Unidad de Control Electrónico que gestiona las funciones del motor.

EGR. *Exhaust Gas Recirculation.* En vehículos, gases de escape que regresan a la cámara de combustión.

EOBD. Sistema de diagnóstico a Bordo Europeo.

EPA. *Enviroment Protection Agency.* Es una agencia del gobierno federal de Estados Unidos encargada de proteger la salud humana y proteger el medio ambiente: aire, agua y suelo.

EVAP. También llamado cánister, es el sistema que recupera el vapor del combustible.

FAT. *File Allocation Table*. Nombre de los sistemas de archivos de los sistemas operativos Windows que utilizan la tabla de asignación de archivos.

FTDI. *Future Technology Devices International.* Dispositivo electrónico en circuito integrado que transforma las señales del USB a señales entendibles por el microcontrolador y viceversa

GPS. *Global Positioning System.* Sistema utilizado en la navegación del cual se pueden obtener datos como fecha, hora, posición entre otros.

JOBD. Sistema de diagnóstico a Bordo Japonés.

Joystick. Dispositivo de control utilizado para la navegación en un sistema electrónico.

KWP2000. *Keyword Protocol 2000.* Es el protocolo estándar de la norma europea EOBD. En México lo vemos manifiesto en autos de tal procedencia, como: Renault, Peugeot, Daewoo y descendientes Opel (vendido por GM).

LCE. Laboratorio de Control de Emisiones de la facultad de ingeniería de la UNAM.

MBR. *Master Boot Record*. Primer sector localizado en unidades de almacenamiento, tales como discos duros. Contiene código para inicializar al equipo de cómputo anfitrión.

MIL. *Malfunction Indicator Lamp.* Es una luz testigo que le indica al conductor que el vehículo necesita revisión.

NMEA. Es un protocolo de comunicación que permite comunicar dispositivos de navegación tales como el GPS.

NRZ. *Non Return to Zero.* Codificación digital en la que el nivel de voltaje no vuelve a cero entre bits consecutivos de valor alto.

OBD. *On Board Diagnostic*. Es un sistema de diagnóstico a bordo en coches y camines.

OBD-II. Es el sistema de diagnóstico a bordo mejorado o la segunda versión.

OSI. *Open System Interconection.* Es un marco de referencia para la definición de arquitecturas en la interconexión de los sistemas de comunicaciones.

PCM. *Powertrain Control Module.* Módulo de Control Electrónico dedicado al motor.

PWM. *Pulse Width Modulation*. Protocolo de comunicación utilizado en automóviles Ford exclusivamente.

SD. *Secure Digital.* Formato de tarjeta de memoria inventado por Panasonic, usado en dispositivos portátiles como cámaras digitales, teléfonos móviles y computadoras personales.

SPI. *Serial Peripheral Interface.* Protocolo de comunicación serie entre un dispositivo maestro y uno o varios esclavos que utilizan cuatro terminales para transmitir información entre ellos.

UTC. *Universal Time Coordinated.* Es un estándar de la referencia de tiempo internacional mediante la cual se calculan todos los husos horarios.

VIN. *Vehicle Identification Number.* Este número es una secuencia de dígitos que identifica los vehículos de motor de cualquier tipo, y los remolques a partir de un cierto peso, es un código específico y único para cada unidad fabricada.

VPW. *Variable Width Modulation.* Protocolo de comunicación utilizado en automóviles, primordialmente por GM en prácticamente todos sus autos OBD-II, también los usa Chrysler en vehículos 2000 y posteriores.

Bibliografía y Referencias

Bibliografía:

[1].Mazidi, Muhamad., Naimi, S., *The AVR microcontroller and embedded systems, using assembly and C,* primera edición, Pearson Prentice Hall, Estados Unidos, 2011.

[2].Huerta, E., Mangiaterra, A., Noguera, G., *GPS: Posicionamiento satelital,* primera edición, UNR Editora, Argentina, 2005.

[3].POZAR, D. M., *Microwave Engineering*, Addison-Wesley, U.S.A., 1990.

Referencias:

[1].Hernández H., *Diseño e implementación de una interfaz de registro de datos para la unidad SR04*., Tesis de Licenciatura., UNAM, 2010.

[2].Hernández A., *Registro de datos en tarjetas de memoria SD CARD implementando los sistemas de archivos FAT16 y FAT32*, Tesis de licenciatura, UNAM, 2009.

[3].*AVR microcontroller datasheet.*

[4].*NMEA reference manual*. Technical paper, Document revision 1.3 January 2005, SiRF Technology Inc.

[5].*Microsoft Extensible Firmware Initiative FAT32 File System Specification FAT: General Overview of On-Disk Format*. Version 1.03, December 6, 2000 Microsoft Corporation.

[6].*SD Specifications, Part 1 Physical Layer Simplification.* Version 4.10, January 22, 2013, SD Group.

[7].*Introduction to CAN,* Renesas, 2010.

[8].Valasek, Chris., Miller, Charlie., *Adventures in Automotive Networks and Control Units,* Technical white paper, IOActive, 2014.

[9].Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics, ISO 15031, 2005

[10].Road vehicles — Diagnostics on Controller Area Networks (CAN), ISO 15765, 2004.

[11].”Diccionario de términos de Pemex refinación”, p. 100.

Enlaces de Internet:

[1].http://www.sparkfun.com/datasheets/LCD/SerLCD_V2_5.PDF
[2].https://ghsi.de/CRC/index.php?
[3].http://www.can-cia.org/
[4].http://www.outilsobdfacile.com/obd-mode-pid.html
[5].http://en.wikipedia.org/wiki/OBD-II_PIDs
[6].https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf