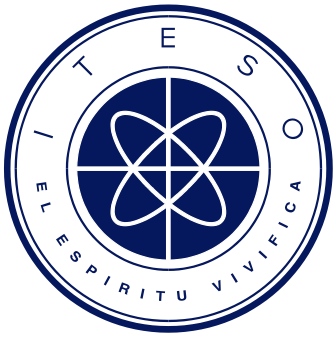
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



DISEÑO PARA UN SISTEMA EMBEBIDO DE PROTOCOLO UDS SOBRE CAN

Trabajo final que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Julio Cesar Cuevas Cedillo

Asesor: Héctor A. Rivas Silva

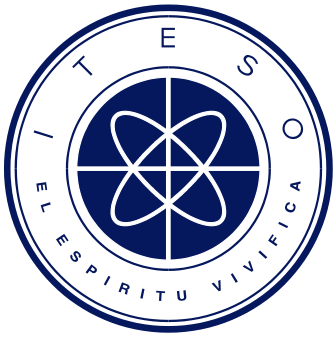
Tlaquepaque, Jalisco, Junio de 2022.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



DISEÑO PARA UN SISTEMA EMBEBIDO DE PROTOCOLO UDS SOBRE CAN

Trabajo final que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Julio Cesar Cuevas Cedillo
Becario CONACYT No. 373460

Asesor: Héctor A. Rivas Silva

Tlaquepaque, Jalisco, Junio de 2022.

Agradecimientos

Agradezco a mi familia por el apoyo que me han brindado a lo largo de la elaboración de este proyecto, en especial a mis padres que me dieron la oportunidad de estudiar esta especialidad y expandir así mis oportunidades en el ámbito laboral y académico.

Agradezco al Mtro. Héctor Rivas que me guío y asesoró en el desarrollo de este trabajo, y además por todas las aportaciones que hizo para el mejor desempeño de este sistema Embebido de Comunicación en el contexto de diagnósticos para la industria automotriz.

Agradezco al Dr. Raul Campos Rodríguez por el apoyo y la asesoría, así como la revisión y la conclusión de este trabajo.

De igual manera quiero agradecer a mis compañeros de la especialidad en sistemas embebidos por sus consejos y ayuda y al ITESO.

Agradezco a CONACYT por la Beca No. 373460 otorgada para el estudio de este programa de posgrado.

RESUMEN

La Unidades de Control Electrónicas (ECU) son de los componentes más críticos en la industria automotriz en la actualidad. En la fase de desarrollo de los ECU's las tecnologías empleadas para el diseño de los mismos tienen grandes variaciones dependiendo del fabricante, es por esta razón que es necesario un protocolo de comunicación que provea una estructura estándar sin que las diferencias entre los ECU's sean un impedimento para la correcta comunicación entre estos.

Un sistema de diagnóstico debe de contener un protocolo capaz de conectarse con las herramientas de desarrolladores, probadores y reparadores para poder revisar la información de diagnóstico (DTC's) de cada ECU. Sin embargo, hay varios protocolos de diagnóstico definidos por ISO y SAE dependiendo del tipo de vehículos o sistemas para los que se fabrica el ECU. Para tratar de unificar varios protocolos surge el protocolo Servicios de Diagnostico Unificados (Unified Diagnostics Service). El protocolo UDS fue definido por ISO con el fin de soportar diferentes estándares de diagnóstico.

Este trabajo se enfoca en el diseño de un sistema que soporte nueve servicios del protocolo UDS, que permitan la activación de sesiones de diagnóstico y a la reprogramación de la unidad de control electrónica (ECU). El diseño propuesto es compatible con el protocolo UDS, que tiene como medio de comunicación un canal de CAN y se implementa utilizando los ID de formato estándar (CAN-A) según el protocolo definido en el estándar de ISO.

ABSTRACT

Electronic Control Units (ECUs) are among the most critical components in the automotive industry today. In the development phase where the ECUs are involved, the technology used for the design of this devices have great variations depending on the manufacturer facilities. Is mainly for this reason that a communication protocol is necessary to provide a standard structure to address the differences between the ECU's, and to allow an efficient and correct communication between them.

A diagnostic system must contain a protocol capable of connecting with the tools of developers, testers and repairing stations, in order to be able to review the diagnostic information (DTCs) of each ECU. However, there are several diagnostic protocols defined by ISO and SAE depending on the type of vehicles or systems for which the ECU is manufactured. In order to unify several protocols, the Unified Diagnostics Service arises. The UDS protocol was defined by ISO to support different diagnostic standards.

This work focuses on the design of a system that supports the most important services of the UDS protocol. This design will support the activation of diagnostic sessions and the reprogramming of the electronic control unit (ECU). This design shall to be compliant with the UDS protocol, that has a CAN channel as the communication medium and is implemented using the standard format (CAN-A) IDs according to the protocol defined in the ISO standard.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

ACRONIMOS Y SIMBOLOS

ECM	Engine Control Module
TCM	Transmission Control Module
SF	Single Frame
FF	First Frame
CF	Control Frame
FC	Flow Control Frame
PCI	Protocol Control Information
SID	Servide Identificator
ACK	Ackonaldge
CRC	Cyclic Redundac Check
ECU	Electronic Control Unit
UDS	Unified Diagnostics Services
CAN	Controller Area Network
ISO	International Organization for Standardization
SAE	Society of Automotive Engineers
DTC	Diagnostic Trouble Codes
AUTOSAR	Automotive Open System Architecture
UART	Universal Asynchronous Receiver-Transmitter)
LLC	Logical Link Control
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
MAC	Media Access Control
OSI	Open System Interconnection

LIN	Local Interconnect Network

INTRODUCCION

Resumen: *En este capítulo se presenta la introducción a este trabajo.*

1.1. Introducción

En este documento se pretende describir las características fundamentales del diseño de un sistema embebido para diagnostico en el bus CAN, sus alcances y sus limitaciones, y los aspectos fundamentales. Se presentará la arquitectura propuesta del sistema así como los análisis que se llevaron a cabo para dicho diseño.

Las siguientes líneas describen la se organiza el resto de este documento:

- El este capítulo se describen los principales elementos del problema, los antecedentes y objetivos del trabajo, ofrece una descripción funcional del sistema y la fundamentación teórico/técnica sobre Protocolos de comunicación en contexto de Diagnostico.
- El Capítulo 2 detalla aspectos del diseño de la solución, los cuales son utilizados para la comunicación entre el dispositivo de diagnósticos “Cliente” y la unidad local ECU “Servidor”.
- El Capítulo 3 contiene las conclusiones y un panorama general del trabajo y futuros desarrollos de este proyecto.
- El Capítulo final incluye las referencias técnicas y bibliográficas consultadas para el desarrollo de este trabajo.

1.2. Planteamiento del Problema

El problema abordado en este trabajo de grado consiste en la propuesta de diseño de un sistema de comunicación basado en el protocolo UDS sobre un canal de CAN (Controller Area Network), para poder tener acceso a distintos servicios como: la reprogramación o actualización de Firmware que contienen el ECU de la unidad, o el acceso a información dentro del mismo ECU todo con su respectivo nivel de seguridad.

1.2.1. Antecedentes

Históricamente las comunicaciones entre los ECU's eran implementadas en un arreglo punto a punto entre las señales en el vehículo eran trasmitidas y recibidas a través de cables, dando como resultado un voluminoso, costoso y complicado sistema de cableado, como se ilustra en la Figura 1 a). En un intento por facilitar el sistema de cableado surgieron algunas soluciones basadas en UART (**U**niversal **A**synchronous **R**eceiver-**T**ransmitter). Como los requerimientos para el control del vehículo crecieron rápidamente poco después surgió de la mano de la empresa BOSCH el protocolo de comunicación *CAN* con la que se pudo simplificar el cableado de la red dentro del vehículo como se muestra en la Figura 1 b) [1].

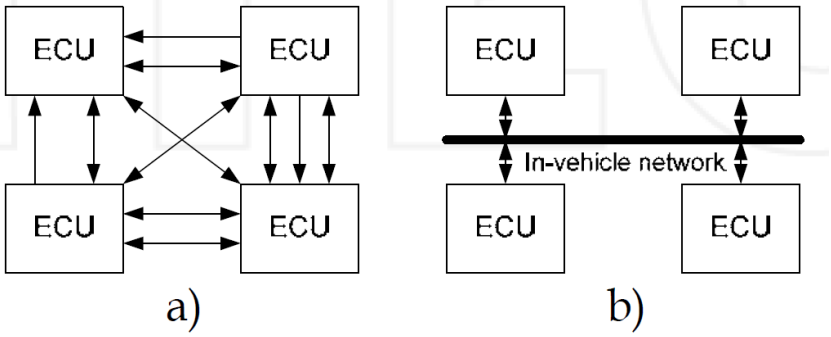


Figura 1. **Evolución de las redes en los vehículos [1]**

1.2.2. CAN

CAN (Controller Area Network) es un protocolo de comunicación serial con una configuración de BUS desarrollado por la empresa Robert Bosch GmbH, su publicación oficial fue en 1986. El uso de *CAN* tuvo un rápido crecimiento desde principios de la década de los 90's debido al incremento de ECU's usados en los automóviles y otros vehículos [2].

El protocolo de *CAN* cubre dos capas del modelo OSI, Capa física y capa de ligado de datos, las capas superiores se definen conforme a la aplicación específica. La capa de ligado de datos se divide en dos subcapas la primera es el Control lógico de ligado (LLC) que se encarga por ejemplo

de decidir cuáles son los mensajes que son recibidos y aceptados, la segunda capa es la de Control de Acceso al medio (MAC) que se encarga del arbitraje, comprobación de errores, señalización de errores, etc. La capa física controla la trasferencia de bits entre los nodos con respecto a todas las propiedades eléctricas [3].

Una red de *CAN* consiste en nodos los cuales están identificados por un número de identificación (CAN ID) dentro del campo de arbitración, este campo es usado para dar prioridad a los mensajes cuando múltiples nodos intentan mandar mensajes al mismo tiempo debito al protocolo que utiliza la capa MAC el cual es CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) cuyo funcionamiento a grandes rasgos es el siguiente: todos los nodos censan el nivel eléctrico del canal todo el tiempo; cada nodo puede iniciar una trasmisión en cualquier momento si el canal está libre; cuando varios canales hacen la trasmisión al mismo tiempo, solo el canal con mayor número de bits dominantes en el campo de arbitración es el que gana el acceso y puede seguir con la trasmisión dejando a los demás en espera. Típicamente las redes tienen entre 3 y 40 nodos, el número de nodos depende de las capacidades de los transceptores de CAN.

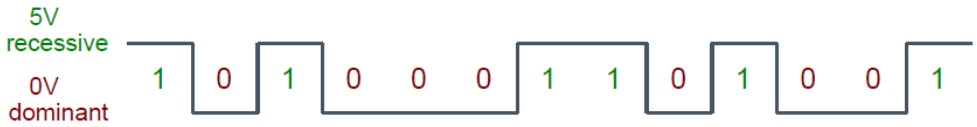


Figura 2. **Bits dominantes y recesivos [4].**

CAN está especificado para transmitir datos con un bit-rate de hasta 1Mbit/s en una red de menos de 40m de longitud, la velocidad tienen que disminuir conforme se aumenta la distancia de la red. [3]

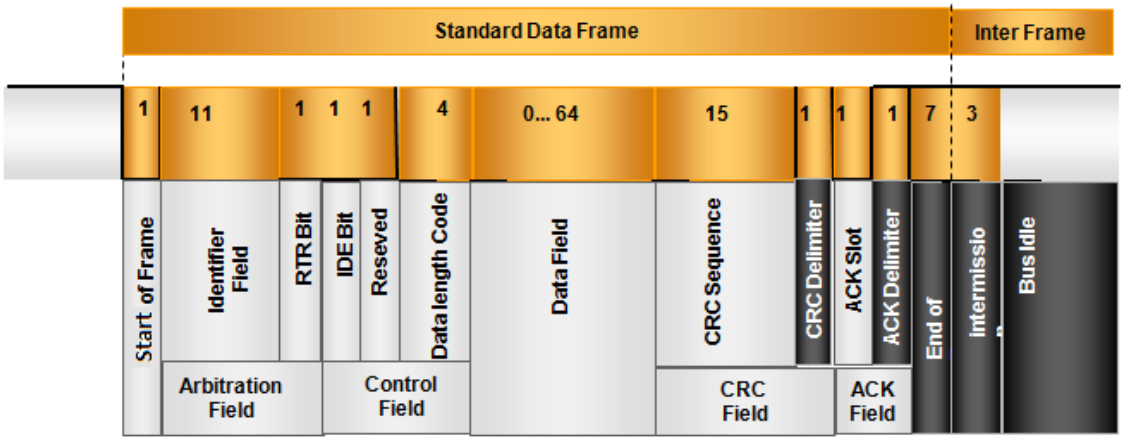


Figura 3. **Trama estándar de CAN A (11 bits ID) [4].**

Como se puede ver en la figura 3 mensaje normal de CAN consiste en 1 bit de inicio de paquete, 11 o 29 bits de ID en el campo de arbitración, 6 bits en el campo de control, 0 a 8 bytes en el campo de datos, 16 bits en el campo de CRC, 2 bits en el campo de confirmación (ACK) y por ultimo 7 bits recesivos de fin de paquete. CAN también cuenta con mensajes Remotos, los cuales pueden ser usados para pedir información de algún nodo, masajes de Error los cuales son usados para indicar errores en el BUS y mensajes de Sobrecarga que son usados para insertar un retardo entre mensajes.

Con el crecimiento y aceptación del protocolo CAN para las comunicaciones en los vehículos surgen varios protocolos para el diagnóstico o autodiagnóstico de la unidades ECU's montadas en los automóviles entre ellos el protocolo de "Servicios de Diagnostico Unificados" "UDS" por sus siglas en inglés, uno de los protocolos más nuevos y por lo tanto se está estandarizando en la industria.

1.2.3. UDS

Definido por la Organización Internacional de Normalización ISO pos sus siglas en ingles en el documento ISO-14229-1, UDS es un estándar de comunicaciones automotrices que especifica de manera independiente al ligado de datos los requerimientos para los servicios de diagnóstico, de acuerdo con el ISO 14229-1 un servicio de diagnóstico es en detalle:" un intercambio de información

iniciada por un cliente con el fin de requerir información de diagnóstico de un servidor y/o modificar el funcionamiento del mismo con propósitos de diagnóstico [5].

Estos servicios permiten la "cliente" controlar funciones de diagnóstico en un ECU aplicado por ejemplo en la inyección electrónica de combustible, trasmisión automática, sistema de frenos, etc., conectado en el BUS embebido en el vehículo. In el contexto de diagnósticos, los siguientes conceptos son útiles para una mejor comprensión de la semántica del estándar UDS:

- Códigos de Diagnóstico de problemas (DTC): identificador numérico para una condición de falla identificada por el sistema de diagnóstico.
- Datos de diagnóstico: Datos localizados en la memoria del ECU la cual puede ser revisada o posiblemente modificada por el "cliente". Ejemplo: velocidad del vehículo, posición de espejos, estatus de sistema, etc.
- Sesión de diagnóstico: Es el modo en el que se encuentra el "servidor" que afecta el nivel de la funcionalidad del sistema de diagnóstico.
- Rutina de diagnóstico: Rutina que esta embebida en el ECU que puede ser iniciada por el "servidor" hasta que se tienen una requisición por el "cliente".
- Cliente: sistema que controla la prueba, inspección o monitoreo del sistema de autodiagnóstico del vehículo.

Como se mencionó antes el protocolo UDS reside en la papa de aplicación del modelo OSI y es independiente de las capas inferiores, por lo tanto los servicios de diagnóstico de un vehículo pueden ser ejecutados utilizando UDS empleando redes de CAN, o algún otro protocolo de comunicación como LIN, FlexRay o Ethernet para las capas de ligado de datos y capa física. Cuando el protocolo UDS funciona sobre una red de CAN se especifica en el ISO-15765-3 [6].

La empresa Vector dedicada a desarrollar herramientas de desarrollo para la industria automotriz dice que el protocolo UDS gradualmente remplazara a los protocolos más antiguos [7]. Esto debido a la complejidad que tienen los nuevos vehículos. La figura 4 es un ejemplo del incremento de ECU en la red de trabajo automotriz.

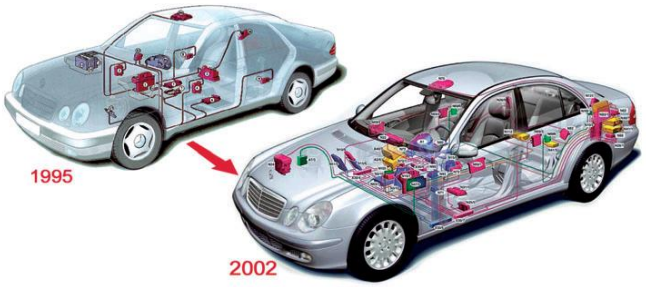


Figura 4. **Desarrollo de redes electrónicas basado en modelo clase E W210 y W211 [7].**

1.2.4. Objetivos

El objetivo general de este trabajo es proponer el diseño de un sistema embebido que funcione con servicios básicos del protocolo UDS los cuales permitan la reprogramación del *firmware* que contiene el ECU.

Para una versión beta del sistema, los objetivos específicos a cumplir son:

- Configuración y manejo de un canal de CAN.
- Soportar el direccionamiento especificado en el documento ISO 15765-4.
- Implementación de 9 servicios UDS necesarios para dar soporte a un *Bootloader*.

1.2.5. Descripción Funcional

Para la solución del problema planteado en este trabajo, se requiere implementar y probar un sistema embebido basado en un microcontrolador comercial que cumpla con las siguientes funciones:

- El sistema soporta entradas de CAN en velocidades de hasta 500kb/s.
- El sistema se puede comunicar con un sistema externo “cliente” el cual requerirá los servicios de diagnóstico.

1.2.6. Fundamentación teórica

Una unidad de control de motor es un componente que controla un motor de combustión interna, así como otras funciones que se implementan como servicios de diagnóstico, relación aire combustible, control de combustión, control de par, etc. Pero estas funciones se implementan de manera muy diferente en cada fabricante, debido a esto es necesario la implementación de un de un protocolo estándar que sea capaz de realizar los servicios de diagnóstico sin importar como el fabricante realice la implementación final.

El software de aplicación dentro de un ECU se puede simplificar en dos componentes principales como se muestra en la figura 5 que son el sistema de diagnóstico y el sistema de control. El sistema de control tiene como tarea principal el control del motor de combustión interna, es capaz de generar códigos de diagnóstico DTC's (Diagnostic Trouble Codes) si este detecta un mal funcionamiento. Estos DTC's son usados por los reparadores o técnicos para dar mantenimiento al motor de combustión interna o equipo en cuestión. Sin embargo los talleres de mantenimiento no deben de tener libre acceso al sistema de control es por eso que se crea el sistema de diagnóstico para limitar los derechos de acceso. El sistema provee de la información completa de los DTC's y acceso a algunas partes del sistema de control confines de calibración [9].

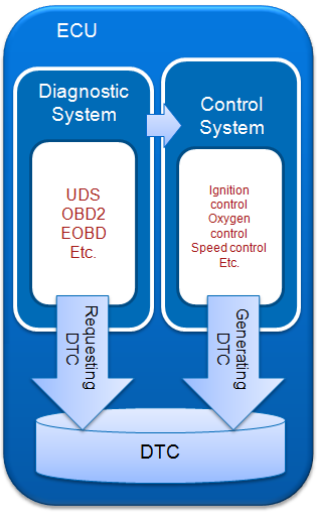


Figura 5. **Visión General del sistema [8].**

El protocolo UDS se puede representar en el modelo OSI como se muestra en la figura 6 [9].

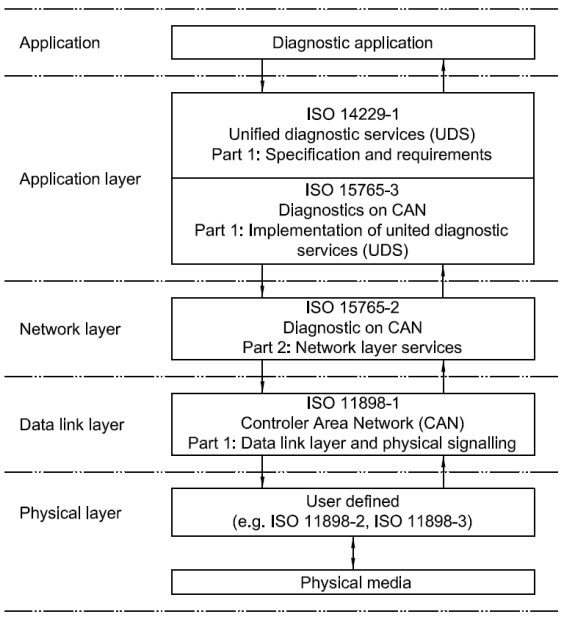


Figura 6. **Modelo OSI para protocolo UDS [9].**

Los comandos UDS son divididos en 6 grupos de acuerdo a sus funcionalidades:

1. *Diagnóstico y Gestión de la comunicación de la unidad Funcional*.
2. Tr*ansmisión de datos de la unidad funcional*.
3. *Almacenamiento de datos de la unidad funcional.*
4. *Control de entradas y salidas de la unidad funcional.*
5. *Activación remota de rutinas de la unidad funcional.*
6. *Subir o descargar datos en la unidad funcional.*

DISEÑO DEL SISTEMA

Resumen: *Este capítulo presenta de manera detallada el diseño propuesto en este trabajo.*

Para la solución al problema planteado en este trabajo se realizó el diseño de un sistema con capacidad de entender comandos básicos de UDS que se mencionaron en la entapa anterior.

2.1. Direccionamiento protocolo UDS (Red)

El Protocolo UDS implementado en CAN soporta el direccionamiento con los dos tipo de ID de CAN A y CAN B en este trabajo se utilizara el direccionamiento de CAN A el cual se especifica en el ISO 15765 parte 4 [9].

Tabla 1**. Direccionamiento de ECU's en protocolo UDS**

CAN ID (HEX)	DESCRIPTION
7DF	Identificador de CAN para direccionamiento funcional requerido por un equipo externo
7E0	Identificador de CAN para requisición física de ECU#1 desde equipo externo "Cliente"
7E8	Identificador de CAN para la respuesta física de ECU#1 al equipo externo "Cliente"
7E1	Identificador de CAN para requisición física de ECU#2 desde equipo externo "Cliente"
7E9	Identificador de CAN para la respuesta física de ECU#2 al equipo externo "Cliente"

7E2	Identificador de CAN para requisición física de ECU#3 desde equipo externo “Cliente”
7EA	Identificador de CAN para la respuesta física de ECU#3 al equipo externo “Cliente”
7E3	Identificador de CAN para requisición física de ECU#4 desde equipo externo “Cliente”
7EB	Identificador de CAN para la respuesta física de ECU#4 al equipo externo “Cliente”
7E4	Identificador de CAN para requisición física de ECU#5 desde equipo externo “Cliente”
7EC	Identificador de CAN para la respuesta física de ECU#5 al equipo externo “Cliente”
7E5	Identificador de CAN para requisición física de ECU#6 desde equipo externo “Cliente”
7ED	Identificador de CAN para la respuesta física de ECU#6 al equipo externo “Cliente”
7E6	Identificador de CAN para requisición física de ECU#7 desde equipo externo “Cliente”
7EE	Identificador de CAN para la respuesta física de ECU#7 al equipo externo “Cliente”
7E7	Identificador de CAN para requisición física de ECU#8 desde equipo externo “Cliente”
7EF	Identificador de CAN para la respuesta física de ECU#8 al equipo externo “Cliente”

Mientras no sea requerido por implementaciones actuales, es altamente recomendable (tal vez requerido por la legislación aplicable) que las implementaciones futuras las siguientes asignaciones de los CAD-ID de 11–bits:

- 7E0/7E8 para ECM (Engine Control Module) módulo de control de motor.
- 7E1/7E9 para TCM (Transmission Control Module) módulo de control de transmisión.

2.2. Capa de transporte

Todos los mensajes de UDS incluyen por lo manos un byte con la Información de Control de Protocolo (PCI) y un Identificador de Servicio (SID). El campo de PCI nos dice cual tipo de paquete es y cuantos bytes de datos hay en el mismo, los 3 tipos de paquete son:

- Paquete simple (SF)
- Primer paquete (FF)
- Paquete consecutivo (CF)

El paquete simple (SF) es usando cuando el mensaje es más corto o igual a 7 bytes sin incluir el byte de PCI. Si el mansaje contiene más de 7 bytes, entonces el mensaje tiene que ser dividido y enviado con múltiples paquetes.

Cuando se utiliza la fragmentación de mansaje se envía como su nombre lo indica el Primer paquete (FF). El primer paquete también incluye la información acerca de la longitud del mensaje (DL) así como los primeros 6 bytes de datos, el resto del mansaje es trasmitido mediante los paquetes consecutivos (CF), cada paquete consecutivo consiste en un campo con un número se secuencia (SN) + 7 bytes de datos. El número de secuencia es usado por el receptor para re ensamblar el mansaje en el orden correcto.

Message Type Description	CAN Message Frame														
	Address Field	PCI Field								Data Field (7 bytes)						
	CAN ID	Byte #1								Byte#2	Byte#3	Byte#4	Byte#5	Byte#6	Byte#7	Byte#8
		b7	b6	b5	b4	b3	b2	b1	b0							
Single Frame (SF)	AI	**0**	**0**	**0**	**0**	**DL**				Data	Data	Data	Data	Data	Data	Data
First Frame (FF)	AI	**0**	**0**	**0**	**1**	**XDL**				**DL**	Data	Data	Data	Data	Data	Data
Consecutive Frame (CF)	AI	**0**	**0**	**1**	**0**	**SN**				Data	(Data)	(Data)	(Data)	(Data)	(Data)	(Data)
Flow Control Frame (FC)	AI	**0**	**0**	**1**	**1**	**FS**				**BS_{max}**	**ST_{min}**	---	---	---	---	---
(Reserved)	---	$40 through $FF								---	---	---	---	---	---	---

Figura 7. **Codificación de paquetes [6]**

El protocolo UDS específica que los paquetes de *CAN* siempre deben contener 8 bytes [6] entonces en caso de no necesitar todos los bytes de datos estos deben de rellenarse en este caso 0x55 para las requisiciones y 0xAA para las respuestas.

Con los 3 tipos de paquetes se pueden tener 4 tipos de comunicación que SF → SF, SF→MF, MF→SF, MF→MF. Esto se ilustra en la Figura 8.

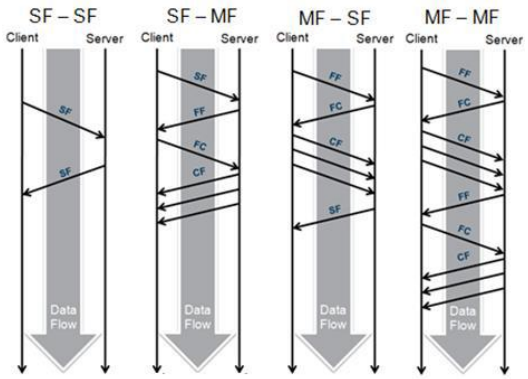


Figura 8. **Capa de trasporte- 4 tipos de comunicación [8].**

La identificación de los paquetes simples y los multi-paquetes se realiza basado en el primer byte del campo de datos del paquete de CAN en la tabla se puede ver cuál es la codificación de estos. El proceso que sigue la capa de transporte de datos para la identificación de los distintos tipos de paquetes que se puede ejemplificar con el diagrama de flujo de la figura [8].

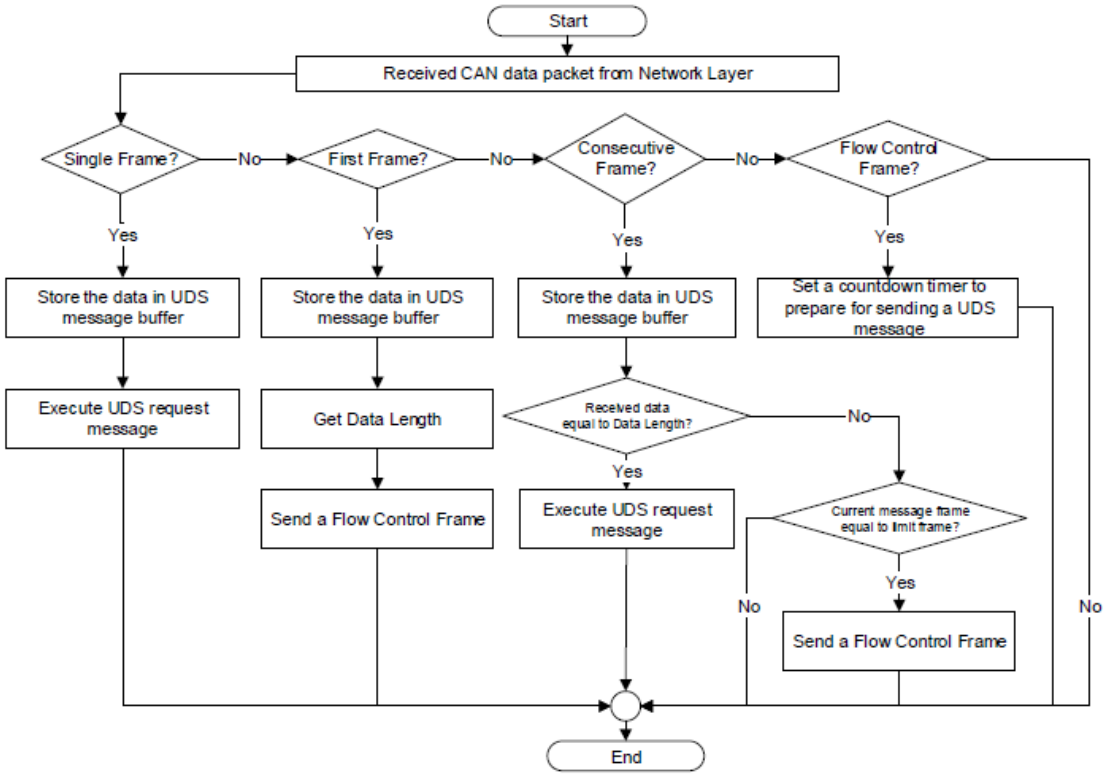


Figura 9. **Diagrama de flujo de la capa de transporte [8].**

2.3. Comandos de UDS

Como fue anteriormente mencionado en los objetivos solo se menciona que serán implementados 9 servicios de los 25 que conforman el protocolo UDS, esto debido a que son los necesarios para dar soporte a la reprogramación del ECU mediante un *bootloader*.

La “X” en el CAN ID de la requisición de servicio puede tomar cualquier valor entre 0~7.

La respuesta positiva a los comandos siempre se agrega un 0x40 al ID del servicio para regresar este como afirmativo, la “X” en el CAN ID para la respuesta se debe a que puede ser cualquier valor entre 8~F pero se debe respetar la tabla de direccionamiento.

2.3.1. Control de Sesión de diagnóstico (0x10)

La funcionalidad del protocolo UDS gira en torno al comando de control de sesión (0x10H) el cual puede estar solamente en una sesión al mismo tiempo.

Cada tipo de sesión tiene diferentes privilegios, esto es para evitar que personas no autorizadas tengan acceso a funciones que pueden cambiar el comportamiento del ECU y legar a provocar fallas. Se tienen dos diferentes tipos de sesión de default (0x01H) y no-default (0x02H y 0x03H). La sesión de default como su nombre lo dice es la sesión que se ejecuta sin necesidad de solicitud de cambio, esta está limitada a los servicios de diagnóstico que puede soportar pero por lo menos cuenta con el servicio de reset y el servicio de control de sesión.

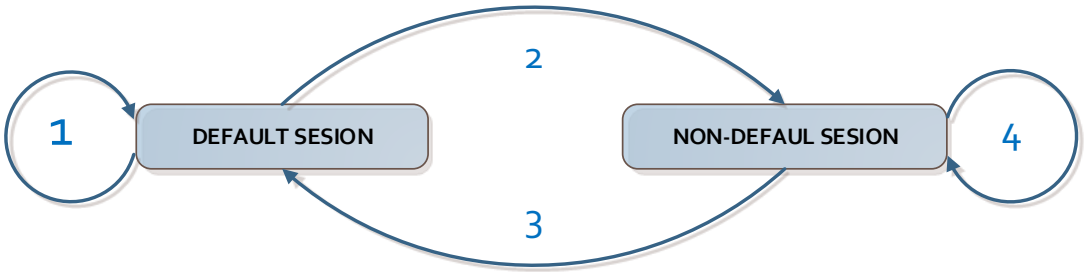


Figura 10. **Diagrama simplificado de transiciones de sesión.**

2.3.1.1. Construcción de paquete para servicio:

Requisición de servicio, este servicio soporta direccionamiento físico y funcional.

CAN ID	**DLC**	**DATOS**							
0x7DF ó 0xEX	8	0x02	0x10	0x03	0x55	0x55	0x55	0x55	0x55

El byte remarcado en color amarillo indica cual es el estado al que se quiere pasar con el servicio de control de sesión y pueden ser los siguientes:

- Sesión de default (0x01) en esta sesión no son soportados la mayoría de los servicios solo el control de sesión (0x10) y el reset (0x11).
- Sesión extendida (0x03) los servicios de diagnóstico son habilitados, depende de la aplicación cuales son los servicios habilitados.
- Sesión de programación (0x02) todos los servicios son activados para dar soporte a la reprogramación del ECU.

Respuesta positiva al servicio.

CAN ID	**DLC**	**DATOS**

0xEX	8	0x02	0x50	0x03	0xAA	0xAA	0xAA	0xAA	0xAA

El servicio soporta códigos de respuesta negativa para informar al “cliente” por qué razón no se puede dar el servicio.

Código	**Descripción**	**Mnemónico**
0x12	Sub función no soportada	SFNS
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x22	Condiciones no correctas	CNC

2.3.2. ECU RESET (0x11)

Este servicio es usado para solicitar el reinicio del ECU con el que se están trabajando los servicios, es capaz de soportar diferentes tipos de reinicio, pero en este trabajo solo se habilita la opción de “hard reset”.

Construcción de paquete, este servicio soporta direccionamiento físico y funcional.

CAN ID	**DLC**	**DATOS**							
0x7DF ó 0xEX	8	0x02	0x11	0x01	0x55	0x55	0x55	0x55	0x55

Respuesta positiva al servicio.

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x02	0x51	0x01	0xAA	0xAA	0xAA	0xAA	0xAA

El servicio soporta códigos de respuesta negativa para informar al “cliente” por qué razón no se puede dar el servicio.

Código	**Descripción**	**Mnemónico**
0x12	Sub-función no soportada	SFNS
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x22	Condiciones no correctas	CNC
0x33	Acceso de seguridad denegado	SAD

2.3.3. Acceso de seguridad (0x27)

Para prevenir que personas no autorizadas realicen cambios en el ECU la mayoría de los servicios que generan cambios permanentes en la memoria estos se encuentran bloqueados. Solo cuando el servicio de acceso a sido superado los servicios pueden ser usados. Este servicio tiene dos variantes principales, requerir semilla y verificar llave.

Construcción de paquete para requerir semilla, Este servicio solo soporta direccionamiento físico.

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x02	0x27	0x01	0x55	0x55	0x55	0x55	0x55

Respuesta positiva al servicio

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x04	0x67	0x01	Semilla	Semilla	0xAA	0xAA	0xAA

Construcción de paquete para verificar llave, este servicio solo soporta direccionamiento físico.

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x04	0x67	0x02	Llave	Llave	0x55	0x55	0x55

Respuesta positiva al servicio

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x02	0x67	0x02	0xAA	0xAA	0xAA	0xAA	0xAA

Códigos de respuesta negativa soportados por el servicio

Código	**Descripción**	**Mnemónico**
0x12	Sub-función no soportada	SFNS
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x22	Condiciones no correctas	CNC
0x24	Error de secuencia	RSE
0x31	Requisición fuera de rango	ROOR
0x35	Llave invalida	IK
0x36	Máximo número de intentos	ENOA
0x37	Requisición tempo de retardo no expirado	RTDNE

2.3.4. Probador Presente “Tester Present” (0x3E)

Este servicio tiene como función mantener la sesión activa cuando se está ejecutando alguna sesión no-default, mediante el refrescado del tiempo límite de un reloj que tienen como función si alcanza el tiempo límite regresar la sesión a la de default limitando así el acceso a los servicios, si la sesión que se ejecuta es la de default no es necesario.

Construcción de paquete

CAN ID	**DLC**	**DATOS**							
0x7DF ó 0xEX	8	0x02	0x3E	0x00	0x55	0x55	0x55	0x55	0x55

Respuesta positiva.

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x02	0x3E	0x00	0xAA	0xAA	0xAA	0xAA	0xAA

Códigos de respuesta negativa soportados.

Código	**Descripción**	**Mnemónico**
0x12	Sub-función no soportada	SFNS
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF

2.3.5. Leer memoria por dirección (0x23)

Con este servicio se le indica al ECU que se quiere conocer el contenido de una sección de memoria, el servicio puede solicitar la lectura de un byte en específico o una porción mayor de la memoria a este servicio esta limitados por la capacidad de repuesta del multi- paquete de ECU.

Construcción de paquete

CAN ID	DLC	DATOS							
0xEX	8	0x07	0x23	0x23	0x00	0x1C	0xFF	0x00	0x64

El dato inmediato después del ID de servicio indica el formato y longitud de la dirección y datos a leer, el 2 indica que son dos bytes para la cantidad de bytes a leer y el 3 son los bytes que conforman la dirección de donde se comienza a leer, los datos remarcados en color amarillo son la dirección y los datos en verde son la cantidad de datos a leer.

Respuesta positiva

CAN ID	DLC	DATOS							
0xEX	8	0x10	0x65	0x67	0x01	0x02	0x03	0x04	0x05
0xEX	8	0x21	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0D
0xEX	8	…							
0xEX	8	…							
0xEX	8	0x2E	0x61	0x62	0x63	0x64	0xAA	0xAA	0xAA

Códigos de respuesta negativa soportados.

Código	Descripción	Mnemónico
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x22	condiciones no correctas	CNC
0x31	Requisición fuera de rango	ROOR
0x33	Acceso de seguridad denegado	SAD

2.3.6. Escribir memoria por dirección (0x3D)

Con este servicio se le indica al ECU que se quiere escribir información en una sección de memoria, el servicio puede solicitar la escritura de un byte en específico o una porción mayor de la memoria. Este servicio está limitado por la capacidad que el ECU le asigna a esta función.

Construcción de paquete

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x10	0x14	0x3D	0x13	0x00	0x1C	0xFF	0x0F
0xEX	8	0x21	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0xEX	8	0x22	0x08	0x09	0x0A	0x0B	0x0C	0x0B	0x0C
0xEX	8	0x23	0x0D	0x0E	0x0F	0x55	0x55	0x55	0x55

Los bytes remarcados en color cian son los bytes para Fragmentación de multi-paquete, el byte remarcado en gris es la longitud del paquete sin incluir los bytes que se usan para la fragmentación, el byte remarcado con dos colores indica el formato y la longitud da la dirección y datos a escribir, Bytes remarcados en amarillo son la dirección donde se comienza la escritura y el byte remarcado en verde es la cantidad de bytes a escribir.

Respuesta positiva

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x06	0x7D	0x13	0x00	0x1C	0xFF	0x0F	0xAA

Códigos de respuesta negativa soportados.

Código	Descripción	Mnemónico
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x22	condiciones no correctas	CNC
0x31	Requisición fuera de rango	ROOR
0x33	Acceso de seguridad denegado	SAD
0x72	Falla de programación general	GPF

2.3.7. Solicitud de descarga (0x34)

El servicio de solicitud de descarga se usa cuan se van a transferir datos hacia el ECU. Dicha solicitud debe de contener la cantidad de dados a trasferir y la dirección donde se alojaran. El ECU responde con un mensaje en el que indica la cantidad máxima de datos que puede recibir para que el cliente pueda dividir el paquete entre este máximo y enviar uno a la vez.

Construcción de paquete

CAN ID	DLC	DATOS							
0xEX	8	0x10	0x08	0x34	0x00	0x23	0x00	0x1F	0x65
0xEX	8	0x21	0x00	0x60	0x55	0x55	0x55	0x55	0x55

CAN ID	DLC	DATOS							
0xEX	8	0x07	0x34	0x00	0x13	0x00	0x1F	0x65	0x60

Los dos paquetes anteriores tienen exactamente el mismo significado, depende del formato que se le asigne a los campos de dirección y de datos se pueden tener variantes.

El byte remarcado en gris en ambos casos indica que los datos no tienen encriptación

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x04	0x74	0x20	0x00	0x64	0xAA	0xAA	0xAA

El byte remarcado en amarrillo indica el máximo número de bytes que puedes mandar en un solo bloque, si la capacidad del sistema no es suficiente para el paquete que se desea mandar es necesario seguir un proceso para segmentar en bloque el paquete que se desea enviar.

Códigos de respuesta negativa soportados por el servicio.

Código	**Descripción**	**Mnemónico**
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x22	Condiciones no correctas	CNC
0x31	Requisición fuera de rango	ROOR
0x33	Acceso de seguridad denegado	SAD
0x70	Descarga o subida no aceptada	UDNA

2.3.8. Solicitud de trasferencia (0x36)

El servicio de transferencia de datos se encarga de recibir los bloques de datos y verificar que se recibieron en el orden correcto. Si el bloque recibido fue el correcto entonces se escribe en la locación de memoria y se envía una respuesta positiva.

Construcción de paquete

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x10	0x20	0x36	0x01	0x02	0x03	0x04	0x05
0xEX	8	0x21	0x06	0x07	0x08	0x09	0x0A	0x0B	0X0C
0xEX	8	0x22	0X0D	0X0E	0X0F	0X10	0X11	0X12	0X13
0xEX	8	0X23	0X14	0X15	0X16	0X17	0X18	0X19	0X1A
0xEX	8	0X24	0X1B	0X1C	0X1D	0X1E	0x1F	0xAA	0xAA

El byte remarcado en gris es el contador de bloques es necesario cuando se tienen que mandar varios bloques esto para evitar duplicidad de datos y reconstruir el paquete original.

Códigos de respuesta negativa soportados por el servicio.

Código	**Descripción**	**Mnemónico**
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF
0x24	Error de secuencia	RSE
0x71	transferencia de datos Suspendida	TDS
0x31	Requisición fuera de rango	ROOR
0x72	Falla de programación general	GPF
0x73	Contador de secuencia de bloques erróneo	WBSC
0x92	Voltaje muy Alto	VTH
0x93	Voltaje muy Bajo	VTL

2.3.9. Solicitud de salir de transferencia (0x37)

Este servicio se utiliza cuando se han terminado de enviar la cantidad de datos que se solicitaron en el servicio de solicitud de descarga, una vez recibido se sale del modo de transferencia y se envía la respuesta positiva.

Construcción de paquete

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x01	0x37	0x55	0x55	0x55	0x55	0x55	0x55

Respuesta positiva

CAN ID	**DLC**	**DATOS**							
0xEX	8	0x01	0x77	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA

Códigos de respuesta negativa soportados por el servicio.

Código	**Descripción**	**Mnemónico**
0x24	Error de secuencia	RSE
0x13	Incorrecta longitud de mensaje o formato invalido	IMLOIF

2.4. Diagrama de estados de servicio 0x10

El servicio 0x10 es mediante el cual se pueden habilitar las diferentes funcionalidades y capacidades del sistema de diagnóstico.

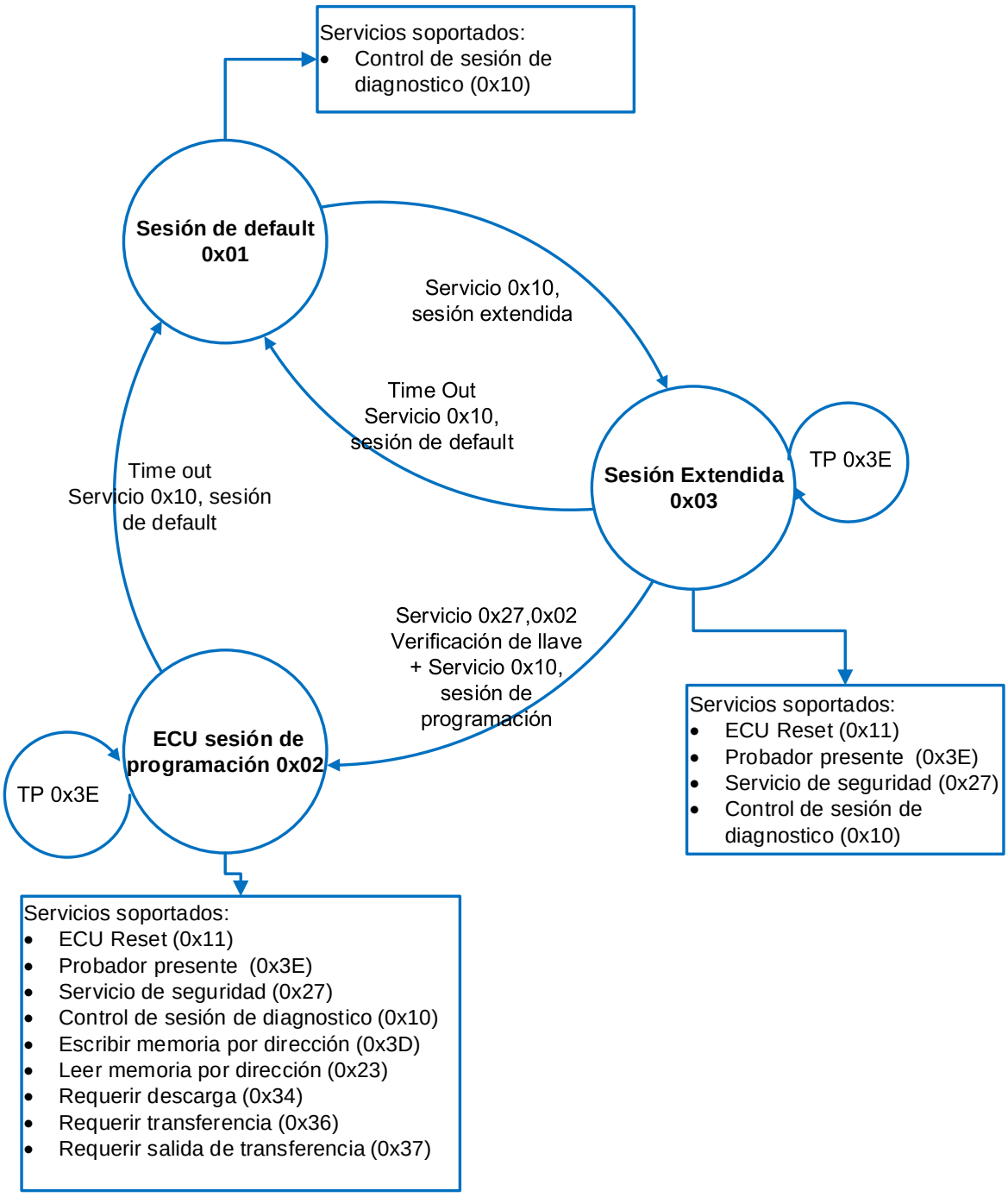


Figura 11. **Estados de sesión de diagnóstico y servicio soportados.**

2.5. Arquitectura de software del Sistema

El firmware para el dispositivo es uno de los elementos más importantes en la solución, ya que se encarga de toda la configuración y personalización del dispositivo, a la vez que realiza las tareas de tratamiento de señales. La Figura 5 muestra una vista de la arquitectura del software del sistema [11].

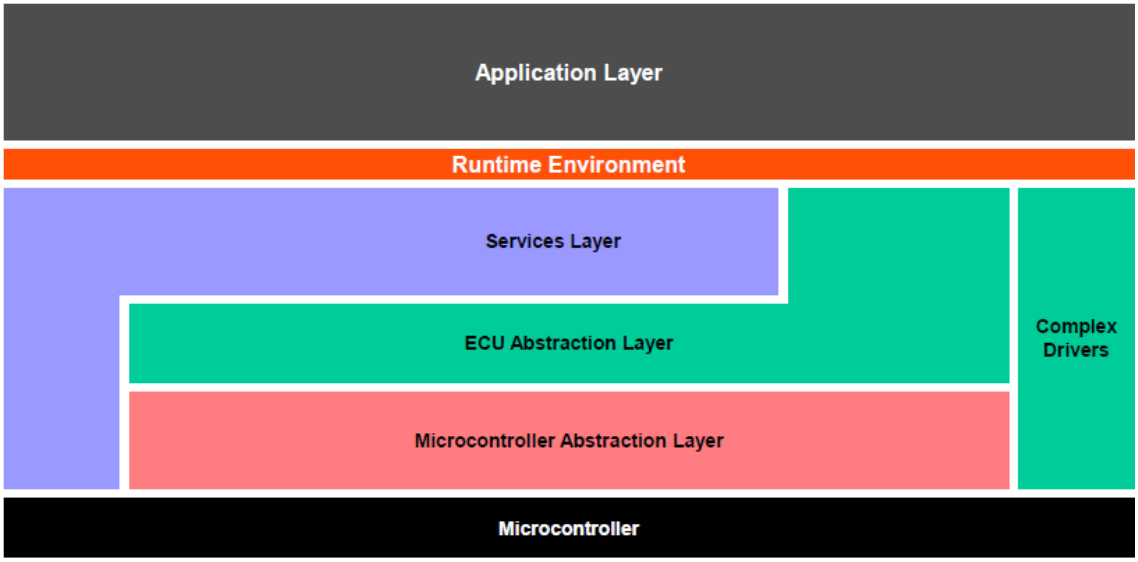


Figura 12. **Arquitectura de capas de AUTOSAR [11].**

2.5.1. Capa de Abstracción de Hardware (HAL):

En esta capa se definen las características del Hardware, se especifica el dispositivo que va a ser utilizado, además de administrar los periféricos del sistema, como puertos, pines de I/O, temporizadores, etc.

2.5.2. Capa de Comunicación:

En esta capa se hace la implementación de los protocolos de comunicación de los diferentes componentes de hardware del sistema. También implementa las tablas de comandos para la comunicación entre los componentes y con el dispositivo anfitrión al cual se conecta el sistema.

2.5.3. Capa de Abstracción:

En esta capa se definen las funciones necesarias para interactuar la capa de comunicación está encargada de armar los paquetes conforme al protocolo UDS y mandar los paquetes para la capa de a aplicación.

2.5.4. Capa de Aplicación:

En esta capa se implementa la aplicación de UDS y se mandan llamar a las funciones necesarias para dar soporte a los servicios de diagnóstico requeridos. En esta capa se procesa la información y se le da un formato adecuado para ser enviada al dispositivo cliente el cual se conecta al sistema.

CONCLUSIONES

1 Conclusiones y Trabajo Futuro

En este trabajo se diseñó un sistema compatible con el protocolo UDS para el servicio de diagnóstico en un ECU con los siguientes servicios:

1. Control de sesión de Diagnostico DSC (0x10).
2. ECU reset ER (0x11).
3. Acceso de seguridad SA (0x27).
4. Probador presente TP (0x3E).
5. Leer Memoria por dirección RMBA (0x23).
6. Escribir memoria por dirección WMBA (0x3D).
7. Requisición de descarga RD (0x34).
8. Transferencia de datos TD (0x36).
9. Salida de trasferencia de datos RTE (0x37)

Las principales tareas para el trabajo futuro incluyen la implementación de los servicios restantes del protocolo UDS. Mejorar y dar soporte a todo el manejo de errores así como sus respuestas. Mejorar la capa de red para que esta sea capaz de distinguir entre diferentes estándares por ejemplo XCP y CCP los cuales son protocolos para calibración de sistema. Mejorar la compatibilidad del protocolo UDS con los identificadores de 29-bits (CAN-B).

REFERENCIAS

[1] Fault detection and diagnosis for in-vehicle networks. Jittiwut Suwatthikul, PhD. National Electronics and Computer Technology Center (NECTEC) Thailand.

[2] "Federal Register Environmental Documents," 2005. [Online]. Available: http://www.epa.gov/fedrgstr/EPA-AIR/2005/December/Day-20/a23669.htm. [Accessed 10 08 2014].

[3] "CAN Specification version 2.0," ROBERT BOSCH GmbH, Stuttgart, 1991.

[4] Embedded Networks Controller Area Network 2.0 A/B, Abram Tezmol, 2014

[5] ISO-14229 Road vehicles — Unified diagnostic services (UDS) — Specification and requirements. Second Edition. 2006

[6] ISO 15765-3 Road vehicles Diagnostics on CAN – Part 3 UDS on CAN. First edition. 2004.

[7] Diagnostics_Congress_ElektronikAutomotive_200703_PressArticle_EN.https://www.vector.com/portal/medien/cmc/press/PDG/Diagnostics_Congress_ElektronikAutomotive_200703_PressArticle_EN.pdf

[8] Unified Diagnostic Services Protocol Implementation in an Engine Control Unit. Pennwalt Assawinjaipetch y various. Consult date August 2014.

[9] ISO 15765-4 Road vehicles Diagnostics on CAN – Requirements for emissions-related systems. First edition. 2005

[10] ISO 15765-2 Road vehicles Diagnostics on CAN – Part 2 Network layer Services. First edition. 2004

[11] AUTOSAR, "Layered Software Architecture 4.1," AUTOSAR, 2014

APENDICE

```
/*****************************************************************************
****************************/
/**
\file       UDS.h
\brief
\author     Julio Cuevas
\version    1.0
\date       18/03/2014
*/
/*****************************************************************************
****************************/

/*****************************************************************************
****************************
* Include files
*****************************************************************************
****************************/

#include "typedefs.h"
#include "hal_can.h"

/*-- Defines ------------------------------------------------------------
-----*/

#define DIAGNOSTIC_SESSION_DEFAULT    0u
#define DIAGNOSTIC_SESSION_EXTENDED   1u
#define DIAGNOSTIC_SESSION_PROGRAMING 2u
#define DIAGNOSTIC_RECEIVING          3u
#define DIAGNOSTIC_SESSION_BUSY       4u

#define UDS_MaxBytesResponse          100u
#define UDS_MaxBytesReceive           100u
/* UDS Service Definitions */

#define SERVICE_DSC                   0x10        /*
DiagnosticSesionControl */
#define SERVICE_TP                    0x3E        /* TESTER PRESENT */
#define SERVICE_RDBI                  0x22        /* ReadDataByIdentifier
*/
#define SERVICE_RDTCI                 0x19        /* ReadDTCInformation
*/
#define SERVICE_ER                    0x11        /* ECUReset */
#define SERVICE_WDBI                  0x2E        /* WriteDataByIdentifie
*/
#define SERVICE_IOCBI                 0x2F        /*
InputOutputControlByIdentifier */
#define SERVICE_RMBA                  0x23        /* ReadMemoryByAddress
*/
#define SERVICE_SA                    0x27        /* SecurityAcess*/
#define SERVICE_WMBA                  0x3D        /* WriteMemoryByAddress
*/
```

```
#define SERVICE_RD                      0x34        /* RequestDownload */
#define SERVICE_TD                      0x36        /* TransferData */
#define SERVICE_RTE                     0x37        /* RequestTransferExit
*/



/* UDS Negative Response Codes DEFINITIONS  */
#define UDS_FILL_BYTE                0xAA            /* Data to fill
incomplete frames */
#define UDS_NR                       0x7F            /* Negative response
*/
#define NRC_IMLOIF                   0x13            /*
IncorrectMessageLengthOrInvalidFormat*/
#define NCR_SNS                      0x11            /* ServiceNotSuported
*/
#define NCR_SFNS                     0x12            /*
SubFunctionNotSuported */
#define NCR_CNC                      0x22            /*
ConditionNotCorrect */
#define NCR_SAD                      0x33            /*
SecurutyAccessDenied */
#define NCR_ROOR                     0x31            /* RequestOutOfRange
*/
#define NCR_GPF                      0x72            /*
GeneralProgramingFailure */

 /* UDS SEED Security Access */

#define UDS_SA_MSB                   0x55
#define UDS_SA_LSB                   0xAA


/*-- Macros ------------------------------------------------------------
-----*/


/*-- Function Prototypes -----------------------------------------------
-----*/


void vfnPrepareResponse(UINT8 u8ResponsePositive, UINT8 u8Service,UINT8
u8NCR,UINT32 u32Type);
void vfnUDSservices(UINT8 *ptrdata,UINT32 u32IdType);
```

```
/*****************************************************************************
*****************************/
/**
\file       UDS.c
\brief
\author     Julio Cuevas
\version    1.0
\date       18/03/2014
*/
/*****************************************************************************
*****************************/

/*****************************************************************************
*****************************
* Include files
*****************************************************************************
*****************************/

#include "UDS.h"

/*****************************************************************************
*****************************
* Definition of  VARIABLEs -
*****************************************************************************
*****************************/


UINT8  u8UDS_Session_Status = DIAGNOSTIC_SESSION_DEFAULT;
UINT8  u8Time_Out_Counter1;
UINT8  u8UDSResponse[UDS_MaxBytesResponse];
UINT8  *ptrDataResponse = &u8UDSResponse[0];

/*****************************************************************************
*******/
/**
* \brief    UDS services Suported
* \author   Julio Cuevas
* \param    void
* \return   void
* \todo     void
*/

void vfnUDSservices(UINT8 *ptrdata,UINT32 u32IdType)
{
  UINT8   u8Index;
  UINT8   u8Service;
  UINT8   u8SubService;
  UINT8   u8Lenght;
  UINT8   u8NCR;
  UINT8   *ptrUDSdata;
  UINT8   *ptrdataread;
  UINT16  u16BytesToRead;

  UINT32  u32AddressMemtoRead;

  ptrUDSdata = ptrdata;
```

```
  u8Lenght= *ptrUDSdata;  ptrUDSdata++;
  u8Service= *ptrUDSdata;  ptrUDSdata++;



  switch  (u8Service)
  {
          /* Service 10H DiagnosticSesionControl */
    case SERVICE_DSC:
           if(u8Lenght==2)          /* Verify the format of the UDS
frame expected length for the command*/
              {
                u8SubService =  *ptrUDSdata;  ptrUDSdata++;
                if((u8SubService&0x7F) == 0x03)
                {

vfnPrepareResponse(TRUE,u8Service,u8SubService,u32IdType);
                  u8UDS_Session_Status = DIAGNOSTIC_SESSION_EXTENDED;
                  u8Time_Out_Counter1=0;
/*Refresh TimeOut for Active sesion*/
                }
                else
                {
                  if((u8SubService&0x7F) == 0x02||(u8SubService&0x7F) ==
0x01)
                  {
                   u8NCR = NCR_CNC;
                   vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                  }
                }
              }
              else
              {
                  /*Prepering Negative response with error format code
IMLOIF*/
                u8NCR = NRC_IMLOIF;
                vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
              }
         break;

          /* Service 3EH Tester Present */
    case SERVICE_TP:
             if(u8Lenght ==2)       /* Verify the format of the UDS
frame expected length for the command*/
              {
                u8SubService =  *ptrUDSdata;  ptrUDSdata++;
                if((u8SubService&0x7F) == 0x00)
                {

vfnPrepareResponse(TRUE,u8Service,u8SubService,u32IdType);
                  u8Time_Out_Counter1=0;                        /*Refresh
TimeOut for Active sesion*/
                }
                else
                {

                   u8NCR = NCR_SFNS;
```

```
                    vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                  }
                }
                else
                {
                      /*Prepering Negative response with error format code
IMLOIF*/
                  u8NCR = NRC_IMLOIF;
                  vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                }


          break;
          /* Service 11H ECU RESET */
     case SERVICE_ER:
                if(u8Lenght == 2)           /* Verify the format of the UDS
frame expected length for the command*/
                {
                  u8SubService =  *ptrUDSdata;  ptrUDSdata++;
                  if((u8SubService&0x7F) == 0x01)
                  {

vfnPrepareResponse(TRUE,u8Service,u8SubService,u32IdType);
                    /*Reset ECU*/
                  }
                  else
                  {

                     u8NCR = NCR_SFNS;
                     vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                  }
                }
                else
                {
                        /*Prepering Negative response with error format
code IMLOIF*/
                  u8NCR = NRC_IMLOIF;
                  vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                }
          break;

          /* Service 27Hex SecurityAcess*/
     case SERVICE_SA:
               if(u8UDS_Session_Status == DIAGNOSTIC_SESSION_PROGRAMING ||
u8UDS_Session_Status == DIAGNOSTIC_SESSION_EXTENDED)
                {
                    if(u8Lenght==2 ||u8Lenght==4)
                   {
                     u8SubService =  *ptrUDSdata;  ptrUDSdata++;
                     if((u8SubService&0x7F) == 0x01)
                        {
                          u8UDSResponse[0]= 0x04;
                          u8UDSResponse[1]= u8Service + 0x40;
                          u8UDSResponse[2]= u8SubService;
                          u8UDSResponse[3]= UDS_SA_MSB;
                          u8UDSResponse[4]= UDS_SA_LSB;
                          vfn_Response_CAN_Frame (ptrDataResponse);
```

```
                        u8Time_Out_Counter1=0;
/*Refresh TimeOut for Active sesion*/
                    }
                  else  if ((u8SubService&0x7F) == 0x02)
                    {
                        /* se debe verificar que la semilla que se recibe
sea la correcta*/

vfnPrepareResponse(TRUE,u8Service,u8SubService,u32IdType);
                        u8UDS_Session_Status =
DIAGNOSTIC_SESSION_PROGRAMING;

                        u8Time_Out_Counter1=0;
/*Refresh TimeOut for Active sesion*/
                    }
                  else
                    {

                        u8NCR = NCR_SFNS;

vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                    }
                 }
                else
                {
                  /*Prepering Negative response with error format code
IMLOIF*/
                  u8NCR = NRC_IMLOIF;
                  vfn_NegativeResponse_Sub(u8Service, u8NCR, u32IdType);
                }
              }
              else
              {
                /*Prepering Negative response with error format code
CNC*/
                  u8NCR = NCR_CNC;
                  vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
              }

         break;

         /* Service 23Hex  ReadMemoryByAddress */
    case SERVICE_RMBA:
         if(u8UDS_Session_Status == DIAGNOSTIC_SESSION_PROGRAMING ||
u8UDS_Session_Status == DIAGNOSTIC_SESSION_EXTENDED)
         {
           if(u8Lenght==2 ||u8Lenght==4)
            {
               u8SubService =  *ptrUDSdata;  ptrUDSdata++;
               if(u8SubService == 0x23 || u8SubService == 0x13 )
               {
                u32AddressMemtoRead |= *ptrUDSdata;  ptrUDSdata++;
                u32AddressMemtoRead = u32AddressMemtoRead<<8;
                u32AddressMemtoRead |= *ptrUDSdata;  ptrUDSdata++;
                u32AddressMemtoRead = u32AddressMemtoRead<<8;
                u32AddressMemtoRead |= *ptrUDSdata;  ptrUDSdata++;
```

```
                if  (u8SubService == 0x23)
                {
                 u16BytesToRead |= *ptrUDSdata;  ptrUDSdata++;
                 u16BytesToRead = u16BytesToRead <<8;
                 u16BytesToRead |= *ptrUDSdata;  ptrUDSdata++;
                }
                else
                {
                 u16BytesToRead |= *ptrUDSdata;  ptrUDSdata++;
                 u16BytesToRead = u16BytesToRead&0x00FF;
                }

                if (u16BytesToRead<UDS_MaxBytesResponse)
                {
                   ptrdataread = (UINT8*)u32AddressMemtoRead;
                   u8Time_Out_Counter1=0;                    /*Refresh
TimeOut for Active sesion*/
                   u8UDSResponse[0] = u16BytesToRead+1;
                  for(u8Index=1;u8Index<u16BytesToRead;u8Index++)
                  {
                    u8UDSResponse[u8Index]= *ptrdataread;
                    ptrdataread++;
                  }
                  vfn_Response_CAN_Frame (ptrDataResponse);
                }
                else
                {
                  u8NCR = NCR_ROOR;
                  vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
                }
               }
            }
            else
            {
              /*Prepering Negative response with error format code
IMLOIF*/
              u8NCR = NRC_IMLOIF;
              vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
            }
         }
         else
          {
            /*Prepering Negative response with error format code CNC*/
              u8NCR = NCR_CNC;
              vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
          }


         break;
          /*  Service 0x3DHex   WriteMemoryByAddress */
    case SERVICE_WMBA:
         if(u8UDS_Session_Status == DIAGNOSTIC_SESSION_PROGRAMING ||
u8UDS_Session_Status == DIAGNOSTIC_SESSION_EXTENDED)
         {
           u8SubService =  *ptrUDSdata;  ptrUDSdata++;
           if(u8SubService == 0x23 || u8SubService == 0x13 )
           {
```

```
                u32AddressMemtoRead |= *ptrUDSdata;  ptrUDSdata++;
                u32AddressMemtoRead = u32AddressMemtoRead<<8;
                u32AddressMemtoRead |= *ptrUDSdata;  ptrUDSdata++;
                u32AddressMemtoRead = u32AddressMemtoRead<<8;
                u32AddressMemtoRead |= *ptrUDSdata;  ptrUDSdata++;

                if  (u8SubService == 0x23)
                {
                 u16BytesToRead |= *ptrUDSdata;  ptrUDSdata++;
                 u16BytesToRead = u16BytesToRead <<8;
                 u16BytesToRead |= *ptrUDSdata;  ptrUDSdata++;
                }
              else
              {
               u16BytesToRead |= *ptrUDSdata;  ptrUDSdata++;
               u16BytesToRead = u16BytesToRead&0x00FF;
              }
                /* Mandar grabar con la funcion de flash, se tiene
direccion, cantidad de datos + apuntador*/
             }
           }
           else
            {
              /*Prepering Negative response with error format code CNC*/
                u8NCR = NCR_CNC;
                vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
            }


          break;
           /*  Service 0x34Hex RequestDownload */
      case SERVICE_RD:
          if(u8UDS_Session_Status == DIAGNOSTIC_SESSION_PROGRAMING )
           {
            u8SubService =  *ptrUDSdata;  ptrUDSdata++;
            if(u8SubService == 0x23 || u8SubService == 0x13 )
             {
              u8UDSResponse[0]= 0x04;
              u8UDSResponse[1]= u8Service + 0x40;
              u8UDSResponse[2]= 0x20;
              u8UDSResponse[3]= 0x00;
              u8UDSResponse[4]= UDS_MaxBytesReceive;
              vfn_Response_CAN_Frame (ptrDataResponse);
             }
           }
           else
            {
              /*Prepering Negative response with error format code CNC*/
                u8NCR = NCR_CNC;
                vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
            }

          break;
           /*  Service 0x36Hex  TransferData */
      case SERVICE_TD:
          if(u8UDS_Session_Status == DIAGNOSTIC_SESSION_PROGRAMING )
          {
```

```
            u8SubService =  *ptrUDSdata;  ptrUDSdata++;
            /* Mandar grabar con la funcion de flash, se tiene direccion,
cantidad de datos + apuntador* a los datos */
            vfnPrepareResponse(TRUE,u8Service,u8SubService,u32IdType);
          }
          else
           {
            /*Prepering Negative response with error format code CNC*/
              u8NCR = NCR_CNC;
              vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);
           }

          break;
           /*  Service 0x37Hex RequestTransferExit */
     case SERVICE_RTE:
          if(u8UDS_Session_Status == DIAGNOSTIC_SESSION_PROGRAMING )
          {
            u8UDSResponse[0]= 0x01;
            u8UDSResponse[1]= u8Service + 0x40;
            vfn_Response_CAN_Frame (ptrDataResponse);
          }
          else
           {
            /*Prepering Negative response with error format code CNC*/
              u8NCR = NCR_CNC;
              vfnPrepareResponse(FALSE,u8Service,u8NCR,u32IdType);

           }


     default: break;
   }
}

/*************************************************************************
*******/
/**
* \brief    Response for UDS protocol
* \author   Julio Cuevas
* \param    void
* \return   void
* \todo     void
*/

void vfnPrepareResponse(UINT8 u8ResponsePositive, UINT8 u8Service,UINT8
u8Data,UINT32 u32Type)
{
   if (u8ResponsePositive)
  {
   u8UDSResponse[0]= 0x02;
   u8UDSResponse[1]= u8Service + 0x40;
   u8UDSResponse[2]= u8Data;
   if((u8Data & 0x80)!=0x80)
   {
   vfn_Response_CAN_Frame (ptrDataResponse);
   }
```

```
  }
  else
  {
   u8UDSResponse[0]= 0x03;
   u8UDSResponse[1]= UDS_NR;
   u8UDSResponse[2]= u8Service;
   u8UDSResponse[3]= u8Data;
   if(u32Type==0x07df && (u8Data==NCR_SNS || u8Data==NCR_SFNS
||u8Data==NCR_ROOR ))
   {
    ;
   }
   else
   {
   vfn_Response_CAN_Frame (ptrDataResponse);
   }
  }
}
/*************************************************************************
*******/
/**
* \brief    TimeOut for UDS active Session
* \author   Julio Cuevas
* \param    void
* \return   void
* \todo     void
*/

void vfnCan_Time_Out(void)
{
      if (u8UDS_Session_Status ==DIAGNOSTIC_SESSION_EXTENDED
||u8UDS_Session_Status ==DIAGNOSTIC_SESSION_PROGRAMING)

      {
         u8Time_Out_Counter1++;
      }

      if(u8Time_Out_Counter1==50)
      {
        u8UDS_Session_Status =DIAGNOSTIC_SESSION_DEFAULT;
        u8Time_Out_Counter1=0;
      }
}
```

```
/*****************************************************************************
*******/
/**
\file       hal_can.h
\brief      Intermediate-layer CAN communication services
\author     Abraham Tezmol
\version    0.1
\date       06/Aug/2007
*/
/*****************************************************************************
*******/

#ifndef _CAN_COMMUNICATIONS        /*prevent duplicated includes*/
#define _CAN_COMMUNICATIONS

/*-- Includes ------------------------------------------------------------
-----*/
/** Variable types */
#include "typedefs.h"
#include "UDS.h"

/*-- Types Definitions ---------------------------------------------------
-----*/


/*-- Defines -------------------------------------------------------------
-----*/



/* UDS Service Definitions */
 #define   SingleFrame        0u
 #define   FirstFrame         1u
 #define   SecondFrame        2u
 #define   FlowControl        3u


/*-- Macros --------------------------------------------------------------
-----*/

/*-- Function Prototypes -------------------------------------------------
-----*/

/* CAN Periodic frame transmission tasks */
void vfnCAN_Periodic_Tx_Queuing(void);

/* CAN Periorid frame reception tasks */
void vfnCAN_Periodic_Rx(void);
void vfnCan_Time_Out(void) ;
void vfn_TesterPresent(UINT8 u8SubService);
void vfn_PositiveResponse_Sub(UINT8 u8Service,UINT8 u8SubService);
void vfn_NegativeResponse_Sub(UINT8 u8Service,UINT8 u8NCR,UINT32
u32Type);
void vfn_Response_CAN_Frame (UINT8 *ptrdata);

void vfnDiagnostic_StateMachine(UINT8 service);
#endif /* _CAN_COMMUNICATIONS */
```

```
/*****************************************************************************
*******/
/**
\file       hal_can.c
\brief      Intermediate - layer CAN communication services
\author     Abraham Tezmol
\version    0.1
\date       06/Aug/2007
*/
/*****************************************************************************
*******/

/** Intermediate layer CAN Communication Functions **/
#include "hal_can.h"
/** Low-level layer CAN Communication Functions **/
#include "mscan.h"
#include "cnf_mscan.h"

/* -- Variables ---------------------------------------------------------
------*/

/** Error flag for CAN transmission*/
UINT8 u8ErrorFlag;
UINT8 u8UDSbuffer[100];
UINT8 *ptrUDSbuffer = &u8UDSbuffer[0];

//tMSCAN_Message aMSCAN_Rx_Buffers[CAN_RX_BUFFER_DEPTH];
tMSCAN_Message aMSCAN_Tx_Buffers[CAN_TX_QUEUE_DEPTH];

//UINT8 u8RxFifoDepth;
UINT8 u8counter;


/*****************************************************************************
*******/
/**
* \brief    CAN Periodic Message Tx Queuing. Execute @ 2ms
* \author   Abraham Tezmol
* \param    void
* \return   void
* \todo     void
* \warning  void
*/
void vfnCAN_Periodic_Tx_Queuing(void)
{
    /* Control index for Transmission logic */
    static UINT16 u16Tx_Loop_Counter;
   // static tMSCAN_Message aMSCAN_Tx_Buffers[2];
    /* Index for data to be transmitted */
    UINT8 u8Index;

    /* Update control variable */
    u16Tx_Loop_Counter++;

    /* Queue periodic messages accordingly */
    if(u16Tx_Loop_Counter %(CAN_TX_1_PERIOD) == 0)
    {
```

```
        aMSCAN_Tx_Buffers[0].ID = CAN_STD_ID_0x01a5;
        aMSCAN_Tx_Buffers[0].Length = 8;
        for (u8Index = 0; u8Index < aMSCAN_Tx_Buffers[0].Length;
u8Index++)
        {
            aMSCAN_Tx_Buffers[0].Data[u8Index] =
(UINT8)u16Tx_Loop_Counter;
        }
        u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[0].ID, aMSCAN_Tx_Buffers[0].Length,
                        aMSCAN_Tx_Buffers[0].Data, STANDARD_ID);

    }
    /* Queue periodic messages accordingly */
    if(u16Tx_Loop_Counter %(CAN_TX_2_PERIOD) == 0)
    {
        aMSCAN_Tx_Buffers[1].ID = CAN_EXT_ID_0x1abcdef7;
        aMSCAN_Tx_Buffers[1].Length = 8;
        for (u8Index = 0; u8Index < aMSCAN_Tx_Buffers[1].Length;
u8Index++)
        {
            aMSCAN_Tx_Buffers[1].Data[u8Index] =
(UINT8)u16Tx_Loop_Counter;
        }
        u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[1].ID, aMSCAN_Tx_Buffers[1].Length,
                        aMSCAN_Tx_Buffers[1].Data, EXTENDED_ID);
    }

    /* Add similar code to enable Tx of other periodic messages */

    /* Check if maximum allowed periodicity has been reached, if so then
reset control variable */
    if (u16Tx_Loop_Counter == MAX_CAN_PERIODICITY)
    {
        u16Tx_Loop_Counter = 0x00;
    }
}

/**************************************************************************
*******/
/**
* \brief    CAN Periodic Message Rx . Execute @ 2ms or 10ms
* \author   Abraham Tezmol
* \param    void
* \return   void
* \todo     void
* \warning  void
*/
void vfnCAN_Periodic_Rx(void)
{
   static tMSCAN_Message aMSCAN_Rx_Buffers[CAN_RX_BUFFER_DEPTH];
   /*Index for data to be transmitted */
   UINT8 u8RxFifoDepth;
   //tMSCAN_Message * CAN_Frame;
   UINT32 u32IdTemp;
   UINT8 u8Index;
```

```
   UINT8 u8Index2;
   UINT8 u8mult;
   static UINT8 u8DataCount;
   UINT8 u8Lenght;
   static UINT8 u8LenghtFrame;
   UINT8 u8FrameType;



    u8RxFifoDepth = u8CAN_RxFifoDepth(MSCAN_A);

    if (u8RxFifoDepth)
    {
        vfnCAN_GetRxMessages(MSCAN_A, &aMSCAN_Rx_Buffers[0],
u8RxFifoDepth);


       for(u8Index = 0; u8Index < u8RxFifoDepth; u8Index++)
       {
          if(aMSCAN_Rx_Buffers[u8Index].ID == 0x07df ||
aMSCAN_Rx_Buffers[u8Index].ID == 0x07e0 )
          {
             u32IdTemp = aMSCAN_Rx_Buffers[u8Index].ID;
             u8Lenght = aMSCAN_Rx_Buffers[u8Index].Data[0];
/* Verify the service of the UDS message */

             if(u8Lenght<8)
             {
               u8FrameType = SingleFrame;
             }

             else if ((u8Lenght&0xF0)==0x10)
             {
               u8FrameType = FirstFrame;
             }
             else if ((u8Lenght&0xF0)==0x20)
             {
               u8FrameType = SecondFrame;
             }
             else if ((u8Lenght&0xF0)==0x30)
             {
               u8FrameType = FlowControl;
             }

            switch (u8FrameType)
            {
             case SingleFrame:

                u8LenghtFrame = u8Lenght;
                for(u8Index2=0;u8Index2<=u8LenghtFrame;u8Index2++)
                {
                  u8UDSbuffer[u8Index2]=
aMSCAN_Rx_Buffers[u8Index].Data[u8Index2];
                }
                vfnUDSservices(ptrUDSbuffer, u32IdTemp);/*mandar llamar
UDS*/
```

```
            break;

            case FirstFrame:
                u8LenghtFrame = aMSCAN_Rx_Buffers[u8Index].Data[1];
                u8DataCount=0;
                for(u8Index2=0;u8Index2<7;u8Index2++)
               {
                 u8UDSbuffer[u8Index2]=
aMSCAN_Rx_Buffers[u8Index].Data[u8Index2+1];
                 u8DataCount++;
               }
               /*send the control flow response*/
                aMSCAN_Tx_Buffers[0].ID = CAN_STD_ID_0x07e8;         /*
ECU ID for Functional and  Physical  response*/
                aMSCAN_Tx_Buffers[0].Length = 8;
                aMSCAN_Tx_Buffers[0].Data[0] = 0x30;                 /*
First Frame segmentation*/
                aMSCAN_Tx_Buffers[0].Data[1] = 0x00;
                aMSCAN_Tx_Buffers[0].Data[2] = 0x00;
                for (u8Index = 3;u8Index<8;u8Index++)
                {

                aMSCAN_Tx_Buffers[0].Data[u8Index]= UDS_FILL_BYTE;
                }
                u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[0].ID,
aMSCAN_Tx_Buffers[0].Length,aMSCAN_Tx_Buffers[0].Data, STANDARD_ID);
            break;

            case SecondFrame:
                 u8mult= (u8Lenght&0x0F);
                 u8mult= (u8mult*7);

for(u8Index2=0;(u8DataCount<=u8LenghtFrame)&&u8Index2<7;u8Index2++)
                 {
                 u8UDSbuffer[u8Index2+u8mult]=
aMSCAN_Rx_Buffers[u8Index].Data[u8Index2+1];
                 u8DataCount++;
                 }

                 if (u8DataCount>u8LenghtFrame)
                 {
                 vfnUDSservices(ptrUDSbuffer, u32IdTemp);/*mandar llamar
UDS*/
                 }

            break;

            case FlowControl:

            break;

            default: break;
           }
         }
       }
    }
```

```
}

/*************************************************************************
*******/

/*************************************************************************
**************************************/
/**
* \brief    UDS_generic_ response
* \author   Julio Cuevas
* \param    u8Service, u8subservice
* \return   void
*/

void vfn_Response_CAN_Frame (UINT8 *ptrdata)
{
  UINT8 u8Lenght;
  UINT8 u8Index;
  UINT8 u8Index2;
  UINT8 u8BytesWait;
  u8Lenght = *ptrdata;

  if( u8Lenght<7)
  {
    aMSCAN_Tx_Buffers[0].ID = CAN_STD_ID_0x07e8;         /* ECU ID for
Functional and  Physical  response*/
    aMSCAN_Tx_Buffers[0].Length = 8;
    for (u8Index =0;u8Index<8;u8Index++)
    {
       if(u8Index <= u8Lenght)
       {
         aMSCAN_Tx_Buffers[0].Data[u8Index]= *ptrdata;
         ptrdata++;
       }
       else
       {
          aMSCAN_Tx_Buffers[0].Data[u8Index]=UDS_FILL_BYTE;
       }

    }
    u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[0].ID,
aMSCAN_Tx_Buffers[0].Length,aMSCAN_Tx_Buffers[0].Data, STANDARD_ID);
  }
  else
  {
    u8BytesWait = u8Lenght;
    aMSCAN_Tx_Buffers[0].ID = CAN_STD_ID_0x07e8;         /* ECU ID for
Functional and  Physical  response*/
    aMSCAN_Tx_Buffers[0].Length = 8;
    aMSCAN_Tx_Buffers[0].Data[0] = 0x10;                 /* First Frame
segmentation*/
    aMSCAN_Tx_Buffers[0].Data[1] = u8Lenght;
      for (u8Index = 2;u8Index<8;u8Index++)
      {
         ptrdata++;
```

```
          aMSCAN_Tx_Buffers[0].Data[u8Index]= *ptrdata;
          u8BytesWait--;
       }
    u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[0].ID,
aMSCAN_Tx_Buffers[0].Length,aMSCAN_Tx_Buffers[0].Data, STANDARD_ID);
      for(u8Index2=1;u8BytesWait>0;u8Index2++)
      {
       aMSCAN_Tx_Buffers[u8Index2].ID = CAN_STD_ID_0x07e8;         /* ECU
ID for Functional and  Physical  response*/
       aMSCAN_Tx_Buffers[u8Index2].Length = 8;
       aMSCAN_Tx_Buffers[u8Index2].Data[0] = (u8Index2+0x20);
       for (u8Index = 1;u8Index<8;u8Index++)
        {
         if(u8BytesWait>0)
         {
          ptrdata++;
          aMSCAN_Tx_Buffers[u8Index2].Data[u8Index]= *ptrdata;
          u8BytesWait--;
         }
         else
         {
          aMSCAN_Tx_Buffers[u8Index2].Data[u8Index]=UDS_FILL_BYTE ;
         }

        }
        u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[u8Index2].ID,
aMSCAN_Tx_Buffers[u8Index2].Length,aMSCAN_Tx_Buffers[u8Index2].Data,
STANDARD_ID);
      }
  }

}
/*************************************************************************
****************************************/
/**
* \brief    UDS_generic_Positive response
* \author   Julio Cuevas
* \param    u8Service, u8subservice
* \return   void
*/
void vfn_PositiveResponse_Sub(UINT8 u8Service,UINT8 u8SubService)
{
  UINT8 u8Index;
  aMSCAN_Tx_Buffers[0].ID = CAN_STD_ID_0x07e8;         /* ECU ID for
Functional and  Physical  response*/
  aMSCAN_Tx_Buffers[0].Length = 8;                     /* On UDS protocol
the length of the frame always is 8 */
  aMSCAN_Tx_Buffers[0].Data[0]= 0x02;                  /* Length of valid
data; the rest of the bytes not used must be filled */
  aMSCAN_Tx_Buffers[0].Data[1]= u8Service + 0x40;      /* Positive
response original Service ID + 40H */
  aMSCAN_Tx_Buffers[0].Data[2]= u8SubService;          /* Sunservice Eco
from original UDS frame */
  if(u8Service == 0x27 && u8SubService == 0x01)
```

```
  {                                                    /* Special case
for request Seed */
    aMSCAN_Tx_Buffers[0].Data[3]= UDS_SA_MSB;
    aMSCAN_Tx_Buffers[0].Data[4]= UDS_SA_LSB;
    aMSCAN_Tx_Buffers[0].Data[5]= UDS_FILL_BYTE;
    aMSCAN_Tx_Buffers[0].Data[6]= UDS_FILL_BYTE;
    aMSCAN_Tx_Buffers[0].Data[7]= UDS_FILL_BYTE;
  }
  else
  {
   for( u8Index= 3;u8Index<8;u8Index++)                /* Fill the free
data bytes withany value in this case UDS_FILL_BITE "0xAA" */
    {
      aMSCAN_Tx_Buffers[0].Data[u8Index] = UDS_FILL_BYTE ;
    }
  }


  if((u8SubService&0x80) == 0x00)                          /* If the
condition not true: Positive Response must be omitted */
  {
    u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[0].ID,
aMSCAN_Tx_Buffers[0].Length,aMSCAN_Tx_Buffers[0].Data, STANDARD_ID);
  }
}


/*************************************************************************
****************************************/
/**
* \brief    UDS_generic_Negative Response
* \author   Julio Cuevas
* \param    u8Service, u8NCR, u32Type
* \return   void
*/

void vfn_NegativeResponse_Sub(UINT8 u8Service,UINT8 u8NCR,UINT32 u32Type)
{
  UINT8 u8Index;
  aMSCAN_Tx_Buffers[0].ID = CAN_STD_ID_0x07e8;       /* ECU ID for
Functional and  Physical  response*/
  aMSCAN_Tx_Buffers[0].Length = 8;                   /* On UDS protocol
the length of the frame always is 8 */
  aMSCAN_Tx_Buffers[0].Data[0]= 0x03;                /* Length of valid
data; the rest of the bytes not used must be filled */
  aMSCAN_Tx_Buffers[0].Data[1]= UDS_NR;              /* Negative
response*/
  aMSCAN_Tx_Buffers[0].Data[2]= u8Service;           /* Eco from the
request */
  aMSCAN_Tx_Buffers[0].Data[3]= u8NCR;               /* Negative Respons
code */
  for( u8Index= 4;u8Index<8;u8Index++)
  {
    aMSCAN_Tx_Buffers[0].Data[u8Index] = UDS_FILL_BYTE ;
  }
```

```
  if(u32Type==0x07df && (u8NCR==NCR_SNS || u8NCR==NCR_SFNS
||u8NCR==NCR_ROOR ))
  {
    ;                                                    /* If the Type of
addressed is functional and the NCR = SNS, SFNS and ROOR the negative
response must be omited */
  }
  else
  {
     u8ErrorFlag = u8CAN_enqueueFrameforTx(MSCAN_A,
aMSCAN_Tx_Buffers[0].ID,
aMSCAN_Tx_Buffers[0].Length,aMSCAN_Tx_Buffers[0].Data, STANDARD_ID);
  }
}
```

```
/*****************************************************************************
*****************************/
/**
\file       mem.h
\brief
\author     Oscar Valdes / Julio Cuevas / Francisco Covarrubias
\version    1.0
\date       18/03/2014
*/
/*****************************************************************************
*****************************/

#ifndef __MEM_H
#define    __MEM_H

/*****************************************************************************
*****************************
* Include files
*****************************************************************************
*****************************/

#include "typedefs.h"

/*****************************************************************************
*****************************
* Definition of module wide VARIABLES
*****************************************************************************
*****************************/

/*****************************************************************************
*****************************
* Declaration of module wide TYPES
*****************************************************************************
*****************************/

/*****************************************************************************
*****************************
* Definition of module wide MACROS / #DEFINE-CONSTANTS
*****************************************************************************
*****************************/

/*****************************************************************************
*****************************
* Declaration of module wide FUNCTIONS
*****************************************************************************
*****************************/

void vfnRead_Mermory(UINT32 * u32MemAddres, UINT8 u8Length);

#endif /*__MEM_H */
```

```
/*****************************************************************************
****************************/
/**
\file       mem.c
\brief
\author     Oscar Valdes / Julio Cuevas / Francisco Covarrubias
\version    1.0
\date       18/03/2014
*/
/*****************************************************************************
****************************/

/*****************************************************************************
****************************
* Include files
*****************************************************************************
****************************/

#include "mem.h"



/*****************************************************************************
****************************
* Definition of  VARIABLEs -
*****************************************************************************
****************************/

UINT16 u8data[16];

UINT8 * StartAdd;
/*****************************************************************************
****************************
* Definition of module wide (CONST-) CONSTANTs
*****************************************************************************
****************************/

void vfnRead_Mermory(UINT32 * u32MemAddres, UINT8 u8Length)
{
   UINT8  u8Index;
   StartAdd = (UINT8*)u32MemAddres;

   for (u8Index = 0; u8Index < u8Length; u8Index++)
   {
     u8data[u8Index] =*StartAdd;
     StartAdd++;
   }
}
/*****************************************************************************
****************************
* Code of module wide FUNCTIONS
*****************************************************************************
****************************/
```