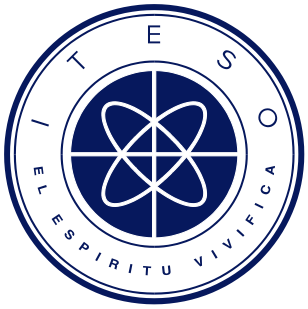
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

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



SOFTWARE PARA DESARROLLO DE PRUEBAS PARA SISTEMAS EMBEBIDOS UTILIZANDO EL PROTOCOLO CAN

Tesina para obtener el grado de:

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Diego Antonio Mejía Sánchez

Director: Dr. Iván Esteban Villalón Turrubiates

San Pedro Tlaquepaque, Jalisco. Diciembre de 2017.

Agradecimientos

Agradezco a la universidad ITESO por proporcionar la infraestructura necesaria para el desarrollo de la especialidad en sistemas embebidos.

De igual manera, agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la asignación de la beca No. 823997 para la realización de mis estudios.

Doy gracias también al Dr. Iván Esteban Villalón Turrubiates y a la Dra. Lorena Michele Brennan Bourdon, quienes con sus recomendaciones ayudaron a guiar este trabajo.

Por último, a los profesores Abraham Tezmol y Francisco Martínez, les agradezco especialmente por la dedicación y calidad con que transmitieron su conocimiento durante el programa de especialidad.

Abstract

This document describes the implementation of a software solution to develop and execute functional tests using the CAN protocol, which is a communication protocol widely used in automotive embedded systems. This software was entirely developed using C# programming language over the .NET framework. Since the CAN protocol is not part of the standard protocols of a computer equipment, the program makes use of commercial devices that serves as an interface between CAN and USB protocols in order to function properly. Program features include the generation of execution reports, a database to store the test results, and a hardware abstraction layer that uses different USB to CAN interfaces. As a result, the developed software executed the testcases using three different hardware interfaces, while preserving exactly the same functionality. Although there might be commercial software available with a greater set of features, they have the drawback of being considerably expensive compared to the software solution proposed in this work.. This is because a license fee has to be paid to the software vendor for every computer that uses its software, while the program described in this document has been developed using only free software tools. Furthermore, commercial software is generally created by the same vendors of USB interfaces, therefore, they only work using their own products. Therefore, the software proposed in this document is a viable alternative for functional testing development, particularly in scenarios where depending on the number of computers needed, the use of a commercial solution would have a significant cost.

.

Resumen

Este trabajo describe la creación de una solución de software para el desarrollo y ejecución de pruebas funcionales utilizando el protocolo CAN, presente en los sistemas embebidos de la industria automotriz. Este software fue construido en su totalidad empleando el lenguaje de programación C# sobre el marco de desarrollo .NET. Debido a que CAN no forma parte de los protocolos estándar de un equipo de cómputo, el programa hace uso de dispositivos comerciales que sirven de interfaz entre CAN y el protocolo USB para funcionar. Las prestaciones del programa incluyen la generación de reportes de ejecución, una base de datos con el resultado de las pruebas funcionales y una capa de abstracción de hardware que permite utilizar distintas interfaces USB a CAN. El resultado del trabajo es un software para el desarrollo y ejecución de casos de prueba utilizando el protocolo CAN que permite la interoperabilidad de interfaces USB a CAN de varios fabricantes. Aunque existen programas informáticos comerciales de prestaciones superiores, estos tienen la desventaja de ser considerablemente más costosos que la solución propuesta en este proyecto. Lo anterior en virtud de que debe pagarse una licencia para cada equipo en el que se instale un programa comercial. Mientras que el software de este trabajo ha sido desarrollado utilizando herramientas de distribución gratuita. Además, los programas comerciales están diseñados para trabajar con una interfaz USB a CAN específica, generalmente distribuida por el mismo fabricante del programa. Por ello, la solución propuesta en este trabajo es una alternativa viable para el desarrollo de pruebas funcionales. Especialmente en escenarios en los que, debido a la cantidad de equipos de cómputo requeridos, utilizar una solución comercial tendría un costo significativo.

Lista de Figuras

Listado de abreviaturas y acrónimos

ARINC	Del inglés, *Aeronautical Radio Inc.*
ASCII	*American Standard Code for Information Interchange*
CA	*Collision Avoidance*
CAN	*Controller Area Network*
CPU	*Central Processing Unit*
CSMA	*Carrier Sense Multiple Access*
DMIPS	*Dhrystone Million Instructions Per Second*
ECU	*Electronic Control Unit*
EEPROM	*Electrically Erasable Programmable Read Only Memory*
FPGA	*Field Programmable Gate Array*
GRASP	*Object-oriented design General Responsibility Assignment Software Patterns*
IEC	*International Electrotechnical Commission*
ISA	*International Society of Automation*
ISO	*International Organization for Standardization*
MAC	*Media Access Control*
OBD	*On Board Diagnostics*
OSI	*Open Systems Interconnection*
PCI	*Peripheral Component Interconnect*
PCMCIA	*Personal Computer Memory Card International Association*
RAM	*Random Access Memory*
RS232	*Recommended Standard 232*
SRAM	*Static Random-Access Memory*
SSI	*Synchronous Serial Interface*
UART	*Universal Asynchronous Receiver Transmitter*
UDS	*Unified Diagnostic Services*
USB	*Universal Serial Bus*
USD	*United States Dollars*

Contenido

Introducción

En la actualidad, los automóviles utilizan un promedio de 35 unidades electrónicas de control [1] conocidas como ECU (del inglés, *Electronic Control Unit)* para funcionar correctamente. Estas unidades se encargan de controlar funciones específicas de un vehículo, tales como gestión del motor, transmisión, dirección asistida, panel de instrumentos, control de iluminación, control de puertas y monitoreo de presión de llantas, por mencionar algunas. Los módulos se coordinan entre sí compartiendo información por medio de algún protocolo de comunicación. Por ejemplo, haciendo posible que el panel de instrumentos sea capaz de mostrar al usuario las revoluciones por minuto del motor, sin importar que esta información haya sido calculada por otro ECU, como la unidad de administración del motor [2].

Uno de los protocolos de comunicaciones empleados para la comunicación entre ECUs es el protocolo CAN (*Controller Area Network*, por sus siglas en inglés), creado en 1985 por la empresa Robert Bosch GmbH [3] y posteriormente definido como estándar ISO 11898 en el año 2003 [4]. El protocolo CAN es tan importante para la industria automotriz que en los Estados Unidos de América todos los vehículos fabricados después del año 2008 lo implementan. Esto debido a que la ley de dicho país establece que todos los automóviles deben contar con servicios de diagnóstico OBD II [5], que requieren de CAN para funcionar. Derivado de lo anterior, puede afirmarse que el protocolo CAN se encuentra presente en la totalidad de los vehículos modernos.

Durante la fase de desarrollo de un ECU que utiliza el protocolo CAN, es necesario realizar pruebas funcionales [6]. Sin embargo, en ocasiones su operación depende de la información que otros dispositivos le proporcionan por medio de mensajes transmitidos en el bus de CAN. Para solucionar esta dependencia, es común contar con herramientas especializadas de software y hardware con capacidad de comunicarse con la unidad de control bajo prueba y transmitirle la información que necesita, simulando el escenario que en las condiciones de funcionamiento normales existiría. Uno de los principales proveedores de estas herramientas es el fabricante Vector Informatik Group [7], el cual ofrece productos de hardware para que una computadora

pueda comunicarse por medio del protocolo CAN, como el producto VN1630A [8]. así como productos de software, como CANoe [9], que funciona exclusivamente con dicho hardware. CANoe posee una gran cantidad de funciones y capacidades tales como la creación de interfaces de usuario, pruebas automáticas, visualización de señales, consola de diagnóstico, entre otras, por lo que es muy útil para el desarrollo y depuración de unidades de control [9].

No obstante, existen escenarios en los que no es viable utilizar este software, ya sea porque existen necesidades muy específicas que están fuera del alcance del programa, o bien porque dichas necesidades no justifican el precio del mismo. Para estos escenarios, Vector provee una librería gratuita [10] pensada para que los desarrolladores de software puedan construir herramientas que se ajusten a sus necesidades, pagando solamente el costo del hardware, el cual es significativamente menor comparado a la solución conjunta de software y hardware.

El objetivo de este trabajo es realizar una implementación de software que permita hacer uso de esta librería gratuita para la ejecución de casos de prueba, así como demostrar cómo se integra el hardware de Vector con el entorno de programación .Net para obtener funcionalidades similares a las que CANoe ofrece. Por ejemplo, controlar los mensajes que se transmiten en el bus en tiempo real. De igual manera, realizar la implementación en un sistema embebido para sustituir el hardware de Vector mencionado, teniendo como resultado una implementación totalmente independiente de dicho fabricante y sus librerías, evitando así el costo de la licencia.

Es importante mencionar que, aplicando una arquitectura de software adecuada es posible realizar una abstracción de hardware. Por ello, la solución puede operar con el hardware de Vector antes mencionado, así como con hardware desarrollado de manera independiente.

1. Antecedentes

Como parte de este trabajo, se considera realizar la implementación de un sistema embebido con la capacidad de transmitir por medio del protocolo USB *(Universal Serial Bus)* la información que es recibida en el bus de CAN. Dicho sistema embebido, puede considerarse como un dispositivo de adquisición de datos, similar al principio de funcionamiento que Hercog y Gergič describen en el artículo "*A Flexible Microcontroller-Based Data Acquisition Device"* [11]. Los autores integraron una solución basada en el microcontrolador PIC18F27J53 [12], que mediante el protocolo USB transfería la información capturada por medio de sus entradas y salidas digitales, así como la capturada utilizando el conversor analógico-digital de 1,024 bits integrado en el microcontrolador. Mencionan en el documento la integración de su sistema en el software LabView [13], logrando una velocidad de muestro de hasta cien mil muestras por segundo.

Su trabajo concluyó que el dispositivo realizado es una propuesta menos costosa que las tarjetas comerciales de adquisición de datos. Además de ser un dispositivo altamente reconfigurable, pues tiene la capacidad de integrar nuevas funcionalidades sin tener que realizar un rediseño del sistema completo. El artículo citado es un caso de éxito de la integración del protocolo USB para transferir a una computadora de uso general, la información capturada en tiempo real por un microcontrolador. Por ello, sirve como un modelo de referencia para la implementación del sistema embebido que este trabajo refiere.

Además del artículo mencionado, existen diversos productos comerciales que sirven de interfaz entre un bus de CAN y un equipo de cómputo. Tal es el caso de CANUSB, un dispositivo fabricado por LAWICEL AB [14]. El precio obtenido de uno de los distribuidores de LAWICEL AB para CANUSB fue de 151 USD [15] (*United States Dollars*) y es capaz de soportar tasas de bits de hasta 1 Mbps, que es la tasa máxima definida por la especificación del bus CAN automotriz ISO-11898 [16]. Sin embargo, este dispositivo tiene el inconveniente de estar basado en un puerto RS232 virtual, por lo que añade una capa extra de procesamiento al CPU al estar basado en la transmisión de comandos ASCII.

Otro producto comercial que ofrece capacidades similares es ValueCAN3, fabricado por Intrepid Control Systems [17]. La empresa ofrece también un software que utiliza este dispositivo para analizar y monitorear los mensajes del bus de CAN, así como también realizar scripts de automatización y casos de prueba funcionales denominado VehicleSpy [17]. Según los datos de la página web del fabricante, consultados en noviembre de 2017, dicho software se vende por 995 USD en su versión básica [18] y 2,795 USD para la versión profesional [18], mientras que el precio de la interfaz ValueCAN3 con sus respectivas librerías para la integración de otras aplicaciones en Windows [19] es de 295 USD [20].

Según el fabricante, ValueCAN3 ha sido probado y verificado para asegurar que soporta una utilización del 100% del bus de CAN a una tasa de transferencia de 1Mbps. Además, incorpora una interfaz USB que no está basada en un puerto serie virtual, como es el caso de la interfaz CANUSB de LAWICEL AB, por lo que ValueCAN3 es un producto que ofrece prestaciones superiores, a cambio de un costo mayor.

Otra alternativa muy atractiva es el producto PCAN-USB elaborado por Peak System Technik GmbH [21]. Este producto ofrece una interfaz USB por un precio de 180 euros [21]. De manera gratuita se distribuye una pequeña aplicación para monitorear el tráfico de la red, sin embargo, la versión completa del software tiene un precio de 450 euros [22]. Los costos de estos productos fueron consultados mediante la web del fabricante en el mes de noviembre de 2017. Aunque no se especifica si el producto soporta una utilización de bus del 100% como lo hace ValueCAN3, la empresa ofrece una amplia gama de productos para utilizar el protocolo CAN en un equipo de cómputo, utilizando distintas interfaces como USB, PCMCIA, PCI, PCI Express e ISA. Derivado de lo anterior, dicha experiencia sirve de respaldo a la compañía y a la calidad de sus productos, ya que la interfaz de PCAN-USB no depende de un puerto RS232 emulado y es técnicamente superior al dispositivo de LAWICEL, aunque al igual que ValueCAN3 tiene un costo superior. Por otra parte, el software básico de PCAN-USB no tiene un costo adicional, por lo que representa una ventaja respecto a ValueCAN3, pues el usuario final puede hacer uso de la solución de software y hardware a un precio significativamente menor si lo que desea es sólo monitorear el tráfico del bus.

Dichos productos comerciales demuestran que la solución propuesta por este trabajo de obtención de grado es viable, además de establecer un punto de referencia sobre las prestaciones y el costo que un sistema con estas características debe tener para ser considerado competitivo. La Tabla 1-1 muestra una comparativa de los dispositivos mencionado.

TABLA 1-1
COMPARATIVA DE INTERFACES

Nombre	Tecnología empleada	Costo de Hardware	Costo de Software	Costo total
CAN USB	Puerto serie emulado	151 USD	No aplica	151USD
PCAN-USB	USB	180 euros	450 euros	630 euros
ValueCAN3	USB	295 USD	2,795 USD	3,090 USD
VN1630A	FPGA + USB	>800USD	>5,000 USD	>5,800 USD

2. Marco Teórico

2.1 Modelo OSI

OSI es un modelo de referencia para la interconexión de sistemas abiertos (*Open Systems Interconnection)* que se encuentra definido en el estándar ISO/IEC 7498-1 [23], cuya versión vigente data del año 1994, 10 años después de haber sido publicado por primera vez en 1984.

De acuerdo a la especificación del modelo OSI, se define como sistema al conjunto de dispositivos de cómputo, software, métodos de transferencia y generación de información que actúan de una manera coordinada para el procesamiento y/o transferencia de datos [23]. Un sistema abierto, es todo sistema aquel que contempla el intercambio de datos con algún otro sistema, en otras palabras, se encuentra "dispuesto" a compartir información.

Es importante mencionar que OSI por sí mismo no define ninguna implementación específica para lograr la interconexión de los sistemas, sino que sirve como un marco de referencia para que sobre él se desarrollen estrategias que permitan a sistemas de cómputo operar en conjunto. Por ello, es común encontrar protocolos de comunicación basados en este modelo, tal es el caso del protocolo de comunicación CAN, cuya especificación detalla qué partes del modelo OSI son implementadas por el protocolo.

Para comprender mejor el modelo OSI como una descripción genérica de interconexión entre sistemas, se puede recurrir a analizar cómo es que las personas intercambian información y cooperan para lograr una determinada tarea.

Este intercambio puede ser realizado en distintos idiomas, mismos que pueden ser transportados de manera distinta. Por ejemplo, mediante sonido, texto o señales. Cada medio de transporte cuenta con sus características y limitaciones. Sin embargo, el lenguaje en sí mismo posee una estructura que no se modifica, es decir, el significado de una oración es el mismo independientemente de si esta última fue transmitida de manera escrita o verbal. Sin embargo, para

cada medio de transmisión existen mecanismos diferentes para garantizar una correcta transferencia de información. Por ejemplo, las personas procuran no hablar al mismo tiempo para que el mensaje pueda entenderse, y el lenguaje transmitido de manera no verbal debe ser más explícito al no contar con la ayuda de la expresión corporal para describir lo que se transmite. Este conjunto de reglas y características de cada medio de transmisión de información establece un modelo de referencia para la comunicación.

En el caso de la comunicación entre sistemas computacionales, existen también numerosos protocolos y medios de transferencia, por lo que el modelo OSI identifica los componentes de dicha comunicación, agrupados en 7 capas. La Figura 2-1 contiene una representación gráfica de las 7 capas de abstracción del modelo OSI, mismas que son detalladas posteriormente.

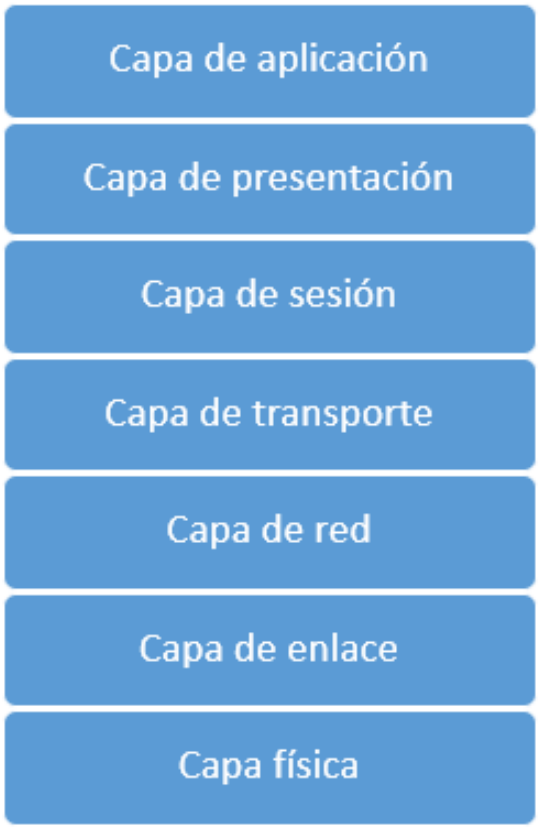


Figura 2-1. Las 7 capas del modelo OSI.

- **Capa física**

  Esta capa del modelo describe el medio físico por el que la señal será transmitida, incluyendo las características eléctricas de esta. En el caso de las comunicaciones alámbricas, se establece en esta capa el tipo de cable a utilizar, así como el tipo de conectores que se requieren, o en su defecto se recomienden. Es importante mencionar que algunos estándares de comunicación emplean frecuencias del orden de los Giga Hertz, por lo que mantener la impedancia de la línea es crítico para garantizar una transmisión adecuada [23].

  Si representáramos la comunicación verbal humana en términos del modelo OSI, esta capa contendría información de los decibeles máximos permitidos para hablar, así como el rango de frecuencia que se producen a través de las cuerdas vocales. También, tendría lugar describir la sensibilidad mínima del receptor, en este caso, el sistema auditivo humano, cuya frecuencia máxima en la que puede percibir el sonido es inferior a los 20KHz.

- **Capa de enlace**

  Esta capa describe principalmente el acceso al medio, es decir, el mecanismo empleado para que todos aquellos nodos que transmiten en una determinada red lo hagan de forma tal que se evite la corrupción de la información. En caso de contar con algún mecanismo de corrección o detección de errores, tendría lugar en esta capa, misma que también describe detalles del direccionamiento, es decir, la manera en la que cada nodo es identificado [23]. En términos de comunicación entre dos personas, se establecería en este nivel del modelo que solamente una persona pueda hablar al mismo tiempo, y en caso de que el receptor no comprenda la información recibida, le pida al emisor retransmitir la información que no fue comprendida.

- **Capa de red**

  Este nivel describe los componentes necesarios para lograr que un nodo pueda comunicarse con otro a pesar de no estar conectados físicamente, siempre y cuando el protocolo lo soporte [23]. Una manera de lograr esto es ofrecer algún tipo de mecanismo que permita que un nodo intermediario se encargue de llevar los mensajes hacia el destinatario final. El concepto para el ser humano es parte de la vida cotidiana, tan simple como pedir a una persona que envié un recado para un tercero que no se encuentra físicamente accesible para el emisor original del mensaje.

- **Capa de transporte**

  Se encarga de ofrecer un método para llevar la información hacia su destino, aunque esta descripción puede ser ambigua, o parecer una tarea trivial, es menester tener en cuenta que, tanto el emisor como el receptor se encuentran limitados en la cantidad de información que pueden procesar en determinado momento. Por lo anterior, es conveniente dividir esta última en segmentos o bloques de información que serán transmitidos uno a uno a medida que el receptor y emisor tengan los recursos disponibles para hacerlo [23].
  Para comprender mejor este concepto, se utiliza la analogía de la comunicación entre personas, las cuales hablan pausadamente al comunicarse ya que tienen la limitante de la cantidad de aire que pueden almacenar en sus pulmones y que es necesario para poder producir el sonido. Además, debe el emisor asegurarse que su interlocutor entienda las ideas que comunica antes de seguir con la transmisión de la comunicación.

- **Capa de sesión**

  En comunicaciones, el concepto de sesión refiere a que una vez establecido un canal de comunicación, es posible establecer una sesión, es decir, tanto emisor y receptor comienzan un intercambio de información dentro de un contexto diferente. Un ejemplo de esto es la transferencia de archivos en la que un usuario requiere autentificarse por medio de una contraseña para tener acceso a los archivos en el servidor [23].

En comunicación entre personas, es posible comparar esta capa del modelo OSI a cuando durante una llamada telefónica el interlocutor pide datos personales para garantizar que la persona que llama es en efecto quien dice ser.

- **Capa de presentación**

Es bien sabido que los sistemas computacionales, como cualquier dispositivo digital utilizan una codificación de datos basada en el sistema binario, por lo que, naturalmente es necesario contar con un método de interpretación y codificación de la información del mundo real [23]. Un ejemplo de esto es el código ASCII, en el que cada letra del abecedario, algunos números y símbolos son representados mediante un valor numérico en el rango del 32 al 128 [24]. En términos de comunicación de seres humanos, traducir de un lenguaje a otro es un buen ejemplo de la capa de presentación de datos del modelo OSI.

- **Capa de aplicación**

La séptima capa del modelo es la de aplicación, básicamente este nivel representa el uso que se dará a la información [23], es decir, si la comunicación sirve para mostrar un video en tiempo real, correo electrónico o un cliente de mensajería instantánea, por mencionar algunos ejemplos. Para los seres humanos la capa de aplicación es el propósito del lenguaje, este puede ser prácticamente cualquiera; relatar un cuento, hacer poesía e informar un suceso a otra persona son algunos ejemplos. Sin embargo, el ser humano por naturaleza no realiza tareas estrictamente definidas como el caso de los sistemas computacionales, por lo que esta capa no puede ser representada fielmente.

Una de las ventajas más importantes de cualquier implementación estructurada en capas, como es el caso del modelo OSI, es que cualquier implementación debe ocuparse solamente de transferir la información requerida a la capa con la que está en contacto. Es decir, el cliente de mensajería instantánea mencionado en el ejemplo del párrafo anterior se limita a transferir el mensaje a la capa de aplicación, este mensaje será procesado por cada uno de los niveles para

terminar como una secuencia de impulsos eléctricos en un cable (capa física). Especialmente relevante es mencionar que el protocolo de comunicación empleado se vuelve completamente irrelevante para la aplicación. Es decir, este cliente de mensajería instantánea transmitirá el mensaje sin necesidad de saber siquiera si la transmisión se realiza de manera alámbrica o inalámbrica, en una red o en un enlace punto a punto, o si es necesario dividir la información en bloques para su transporte.

2.2 Protocolo CAN

El protocolo CAN fue desarrollado por la empresa Bosch en el año 1986 y revisado en el año 1991 con la especificación 2.0 del protocolo [25]. Con base en ella se creó el estándar ISO 11898 en 1993, el cual especifica la capa de datos y parte de la capa física del protocolo CAN de acuerdo al modelo OSI para su utilización en la industria automotriz. La versión más reciente del estándar es la del año 2016. Es importante mencionar que el protocolo CAN no es exclusivo de la industria automotriz, se encuentra también presente en la industria de instrumentación médica, la industria de aviación y la de automatización industrial en la forma de protocolos basados en CAN tales como CANOpen [26], ARINC 429 [27] y DeviceNET [28], entre otros.

CAN es un protocolo que emplea un esquema de transmisión de datos en serie, es decir, utiliza una sola línea de transmisión que secuencialmente cambia sus niveles de voltaje para transmitir datos bit a bit. Esta línea de transmisión única hace que el costo del cableado sea más barato en comparación con otros protocolos que emplean líneas de transmisión en paralelo, pues estos últimos requieren más cables para lograr transmitir más de un bit a la vez.

La topología de CAN es un bus, es decir, todos los dispositivos se encuentran conectados a la misma línea de transmisión diferencial, misma que se compone de dos conductores denominados CANH y CANL. Cuando el bus se encuentra inactivo, no existe diferencia de potencial entre los conductores, pues ambos se encuentran a 2.5V. Por el contrario, cuando alguno de los dispositivos envía un cero lógico al bus, los niveles de voltaje pasan a ser de 1V para la línea CANL y 4V para la línea CANH, teniendo una diferencia de potencial de 3V. En el caso de la

transmisión de un 1 lógico, los niveles de voltaje del bus no cambian, por lo que se dice que el 1 lógico es el valor recesivo, mientras que el 0 lógico es dominante, ya que este último es el único que puede cambiar el voltaje del bus. Es común que se haga referencia al valor de 1 y 0 lógico como bit recesivo y bit dominante, respectivamente, por lo que en adelante se utilizará esta convención.

CAN es multimaestro, es decir, cualquier nodo puede transmitir en cualquier momento, por lo que es necesario emplear un mecanismo para evitar que dos dispositivos transmitan al mismo tiempo, lo que ocasionaría una colisión de datos.

Para lograr esto, se emplea CSMA/CA, acrónimo del inglés "*Carrier Sense Multiple Access with Collision Avoidance*" (Acceso Múltiple mediante Detección de Portadora y Prevención de Colisiones). Esta técnica permite que todos los nodos del bus "escuchen" en todo momento al mismo, pudiendo detectar cuando el bus se encuentra libre para iniciar la transmisión [29]. La implementación de prevención de colisiones permite, asumiendo que el bus se encuentra libre en determinado momento, que todos los nodos del bus comiencen a transmitir, pero sólo uno obtenga el acceso al bus mediante el mecanismo de arbitración del protocolo.

Este mecanismo sucede en el campo que contiene los primeros 11 bits transmitidos en una trama de CAN, conocido como el campo de arbitración. Estos bits contienen el identificador del mensaje que, a su vez, representa la prioridad del mismo. Entre menor sea el valor del identificador del mensaje, mayor es la prioridad, por lo que el dispositivo que se encuentre transmitiendo el mensaje con un identificador menor será el que gane el acceso al bus, mientras los otros dispositivos intentarán comenzar una transmisión cuando el bus vuelva a estar disponible.

En la Figura 2-2, se muestra una captura de osciloscopio de una trama de CAN, así como una breve descripción de cada campo de la misma.

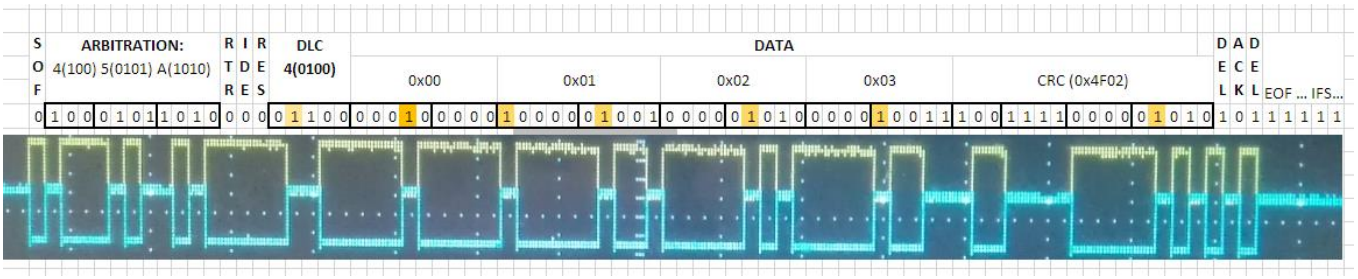


Figura 2-2. Trama de CAN en el osciloscopio.

Los dos componentes más importantes de una trama de CAN son el campo de arbitración y el campo de datos. Ellos contienen el identificador del mensaje y los datos a transmitir, respectivamente. Adicionalmente, existen ciertos bits contenidos en estos campos que poseen información de control. Se detallan estos bits a continuacion.

SOF *(Start of Frame)*

La transmisión comienza cuando el bus pasa del estado inactivo al estado dominante, generando un pulso de sincronización. Mediante este pulso, todos los dispositivos en el bus sincronizan sus relojes internos para recibir los bits de la trama.

Campo de arbitración

Contiene el identificador del mensaje, que sirve para denotar la prioridad del mismo y existen 3 bits que son transmitidos en este campo:

RTR *(Remote Transmision Request)*

Se utiliza cuando un dispositivo envia un mensaje sin datos, esperando que otro nodo del bus lo complete.

IDE *(Identifier Extension bit)*

En el caso de que se utilice el formato extendido de CAN, en el que el identificador del mensaje ocupa 29 bits en lugar de 11, este bit es transmitido como un 1 lógico para señalizarlo.

RES *(Reserved)*

Este bit es reservado para futuras implementaciones o cambios en el protocolo, en CAN 2.0 es transmitido siempre en valor dominante.

Campo de datos

Contiene los datos a ser transmitidos, además de información adicional para su interpretación.

DLC *(Data Length Code)*

Los siguientes 4 bits componen el campo de longitud de datos, el cual contiene la cantidad de bytes que se transmitirán en la trama. (Hasta un máximo de 8 bits pueden ser transferidos por trama de acuerdo a la especificación de CAN 2.0)

DATA *(Data)*

La cantidad de bytes especificada en el campo anterior es transmitida secuencialmente en este campo, el cual puede tener una longitud de hasta 8 bytes.

CRC *(Cyclic Redundancy Check)*

La comprobación de redundancia cíclica utiliza una división entre los datos y un polinomio predefinido, utilizando el residuo como resultado, por lo que sirve de comprobación para asegurar que los datos que obtiene el receptor son válidos.

ACK ***(****Acknowledge)*

Este bit es puesto en estado dominante por el receptor, para señalizar al transmisor que el mensaje se recibió correctamente y no es necesario retransmitirlo.

2.3 Protocolo USB

El protocolo USB, es sin lugar a dudas la opción por excelencia cuando se trata de conectar periféricos a una computadora personal. Basta con prestar atención a la cantidad de dispositivos que son conectados de esta manera por el usuario promedio de una computadora personal para darse cuenta de lo ubicuo del protocolo. Algunos ejemplos de estos periféricos son: teclado, ratón, cámara, memoria, impresora, audífonos y micrófono. Sin embargo, hacer una lista completa de dispositivos compatibles con USB sería una labor imposible, ya que cualquier persona con conocimientos técnicos suficientes puede hacer su propio dispositivo [30].

La versión 1.0 del estándar USB fue lanzada al mercado en 1996, con una velocidad máxima de transferencia de 1.6Mbps, misma que es 3,000 veces inferior a la tasa de transferencia del estándar 3.0 que data del año 2011, la cual es de 5.0 Gbps [31]. Además es 9,000 veces inferior a la velocidad máxima del estándar más actual a la fecha, la versión 3.2 de USB, publicada en el año 2017, con tasas de transferencia de hasta 20Gbps [32]. Es importante destacar que la evolución tecnológica ha permitido y a la vez establecido, la necesidad de velocidades de transferencia miles de veces mayores a las de hace poco más de 20 años. Por mencionar un ejemplo, hoy en día se pueden encontrar cámaras USB que transmiten video en alta definición haciendo uso del estándar 3.0 [33].

El protocolo USB soporta 4 tipos de transferencia [34] que son mencionados a continuación:

- **Transferencia de control**

  Normalmente se utiliza para enviar comandos al dispositivo, así como la recepción del descriptor del mismo, esto sucede normalmente luego de conectar un dispositivo a una computadora.

- **Transferencia bulk:**

  Sirve para enviar cantidades grandes de información, utilizado en dispositivos cuya prioridad es enviar los datos de un punto a otro más que garantizar una latencia determinista, como es el caso de un escáner digital. Esta transferencia tiene corrección de errores integrado.

- **Transferencia por interrupción:**

  Cuando la información que se desee transferir tiene restricciones de tiempo, es recomendable utilizar este método, ya que puede generar interrupciones al CPU para garantizar una latencia constante.

- **Transferencia isócrona:**

Este método es una combinación de la transferencia bulk y la determinada por interrupción. En el modo isócrono los datos son enviados de una manera continua y por medio de interrupciones, garantizando una latencia constante y una tasa de transferencia máxima. Sin embargo, si alguna transferencia falla no se reintenta la transmisión. Por sus características, este método es el utilizado en la transmisión de video y audio en videoconferencias.

2.4 Microcontrolador

Se le conoce como microcontrolador al dispositivo aquel que integra en un solo chip una CPU (del inglés, *Central Processing Unit*), memoria, periféricos como timers, puertos de entrada y salida, conversores analógico digital y periféricos de comunicación, entre otros. A diferencia de las computadoras de propósito general, los microcontroladores típicamente poseen una cantidad muy limitada de memoria y poder de cómputo, por lo que son diseñados para cumplir una aplicación específica que se encuentre dentro de sus capacidades.

Debido a que sus aplicaciones son específicas, existen numerosos fabricantes que ofrecen un amplio catálogo de microcontroladores, los cuales cuentan con prestaciones muy distintas. Como referencia, uno de los dispositivos más básicos que figuran en el catálogo del fabricante Microchip, es el ATTINY4 [35], el cual posee 512 bytes de memoria para almacenar el programa, 6 pines de los cuales 4 pueden emplearse como entradas o salidas, una frecuencia de operación máxima de 12 MHz y está basado en la arquitectura AVR de 8 bits [36]. Por otro lado, uno de los más potentes microcontroladores que ofrece el mismo fabricante, es el SAMA5D3 [37], con una memoria de 160 Kilobytes, 324 pines, de los cuales pueden emplearse 160 como entradas o salidas, una frecuencia máxima de operación de hasta 536MHz y está basado en la arquitectura ARM Cortex-A5 de 32 bits [38].

La CPU integrada en los microcontroladores posee un conjunto de instrucciones bien definidas, las cuales son transferidas al dispositivo para su posterior ejecución. En el caso de que se emplee un lenguaje de alto nivel durante la programación, el código debe primero debe ser

compilado, es decir, traducido a una secuencia de instrucciones que dependen del conjunto de instrucciones que el dispositivo implemente.

2.4.1 Microcontrolador TM4C1294

Para el desarrollo del sistema embebido con capacidad de comunicación CAN – USB se utilizará el microcontrolador TM4C1294, del fabricante Texas Instruments. Este microcontrolador implementa la arquitectura ARM Cortex M4 [39], de tipo Harvard, pues tiene un bus de instrucciones y de datos separados. Incorpora una memoria Flash de 1024KB para el almacenamiento de programa y datos, así como una memoria SRAM de 256KB como memoria volátil. Adicionalmente, posee 6 KB de EEPROM para el almacenamiento de información a largo plazo.

La frecuencia de trabajo máxima es de 120-MHz, otorgando 150 DMIPS de poder de cómputo a esta velocidad. Por su parte, como periféricos integrados a este dispositivo existen 8 UARTs, 2 controladores de CAN, 4 módulos SSI, 1 MAC de Ethernet y 1 MAC + PHY de USB 2.0. El núcleo Cortex-M4 cuenta con un conjunto de instrucciones Thumb-2, de 16 y 32 bits, por lo que el tamaño del programa puede ser optimizado al requerir menor cantidad de bits para almacenar cada instrucción. Es importante mencionar también que este dispositivo cuenta con la capacidad de acceso a memoria desalineada, lo que quiere decir que la dirección de memoria que almacene una instrucción no necesariamente debe ser un múltiplo de 4 bytes (32 bits).

Una de las características más importantes que emplea este microcontrolador es el controlador de acceso directo a memoria integrado, mismo que permite que existan transferencias de información en segundo plano de un periférico a una sección de RAM definida por el usuario y viceversa. Lo anterior, implica liberar la CPU de la tarea de transferencia, resultando en un mayor desempeño del sistema.

3. Metodología

3.1 Definición del enunciado de trabajo

Para realizar este proyecto fue necesario delimitar el alcance del mismo, estableciendo las necesidades a resolver, como ejecutar casos de prueba sobre productos electrónicos que incorporan el protocolo CAN, así como soportar UDS. Además, se contó con la capacidad de conservar un registro del resultado de las pruebas ejecutadas. Derivado de lo anterior, se estableció como enunciado de trabajo la realización de un software para el sistema operativo Windows para la ejecución y desarrollo de pruebas, empleando hardware conectado por medio de USB para el control del bus de CAN.

3.2 Herramientas para el desarrollo del proyecto

3.2.1 Entorno de programación

Se utilizó el software Visual Studio Community Edition [40], en su versión 2017 para el desarrollo de la aplicación en Windows, utilizando el marco de trabajo .Net 4.2 para el lenguaje C#. La Figura 3-1 muestra la interfaz del software.

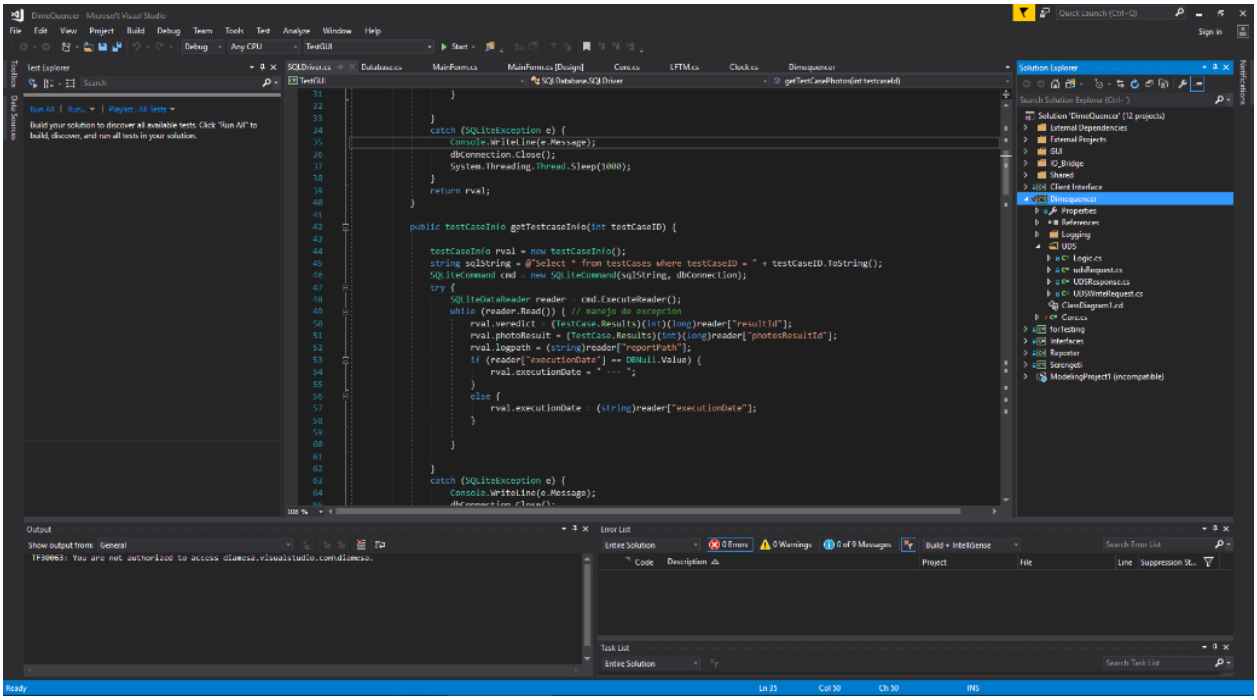


Figura 3-2. Interfaz de usuario de Visual Studio.

Para la programación del microcontrolador contenido en la tarjeta de desarrollo, se utilizó el software de Texas Instruments Code Composer Studio 7.1 [41]. Su interfaz de usuario se muestra en la Figura 3-2.

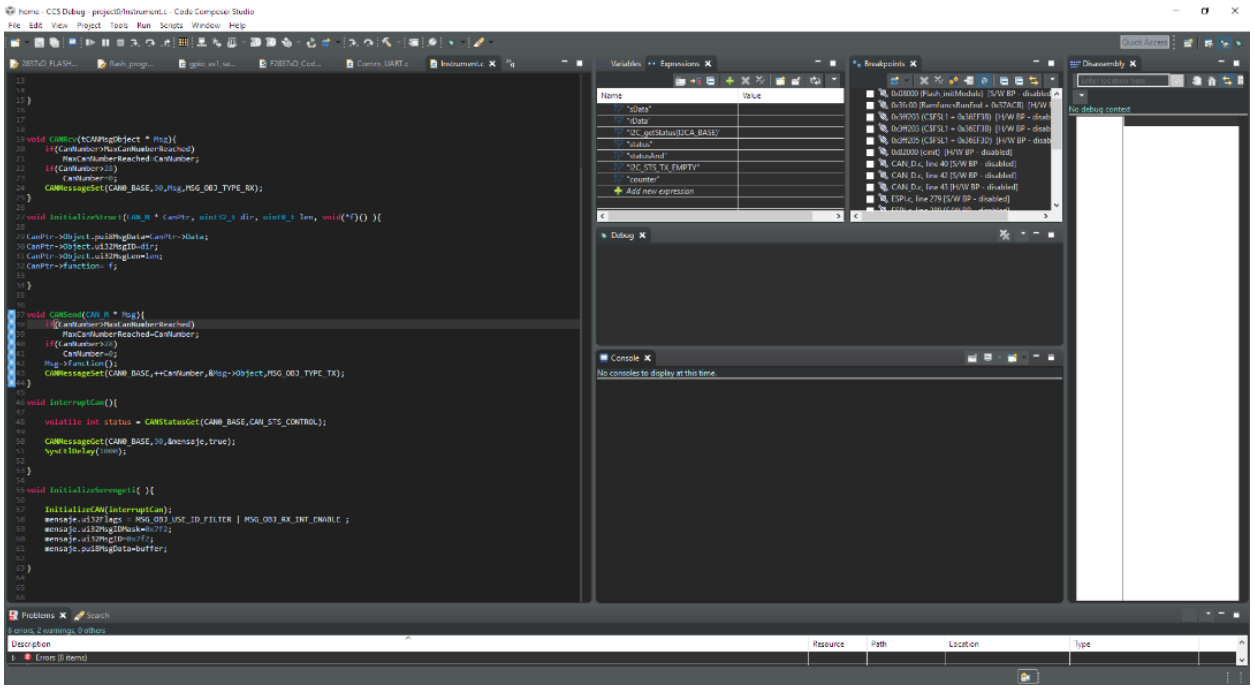


Figura 3-1 Interfaz de Code Composer Studio

3.2.2 Interfaces USB – CAN

Uno de los objetivos principales del proyecto, es lograr una solución que no dependa de ningún fabricante de hardware para la utilización del protocolo CAN. Por ello, se establecieron 3 diferentes interfaces USB a CAN para comprobar que la aplicación de Windows puede utilizarlos indistintamente.

- El hardware VN1630A, basado en tecnología FPGA [42] del fabricante Vector se muestra en la Figura 3-3.

Figura 3-3. VN1630 hardware del fabricante Vector.

- Del fabricante Intrepid Control Systems, el hardware ValueCAN versión 3, una interfaz que incorpora un circuito integrado de aplicación específica para comunicarse a través de USB a la computadora (Figura 3-4).

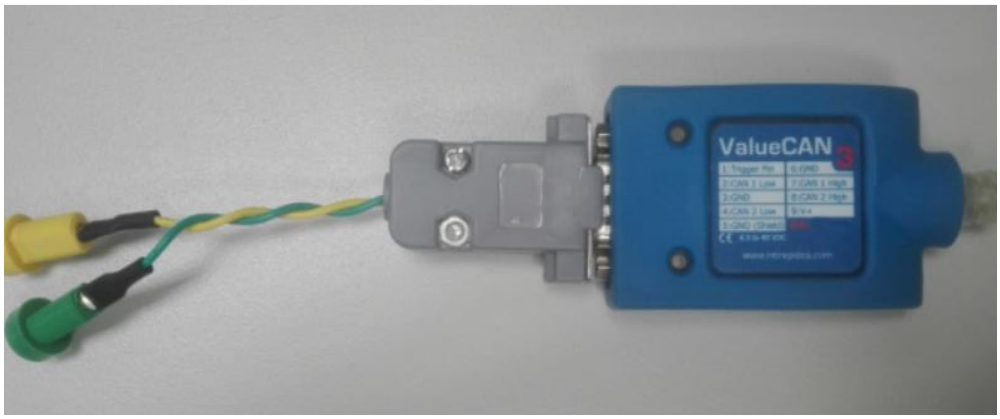


Figura 3-4 Hardware ValueCAN de Intrepid Control Systems

- Tarjeta de desarrollo de Texas Instruments EK-TM4C1294XL [43] que incorpora USB y CAN en el microcontrolador de la tarjeta (Figura 3-5).

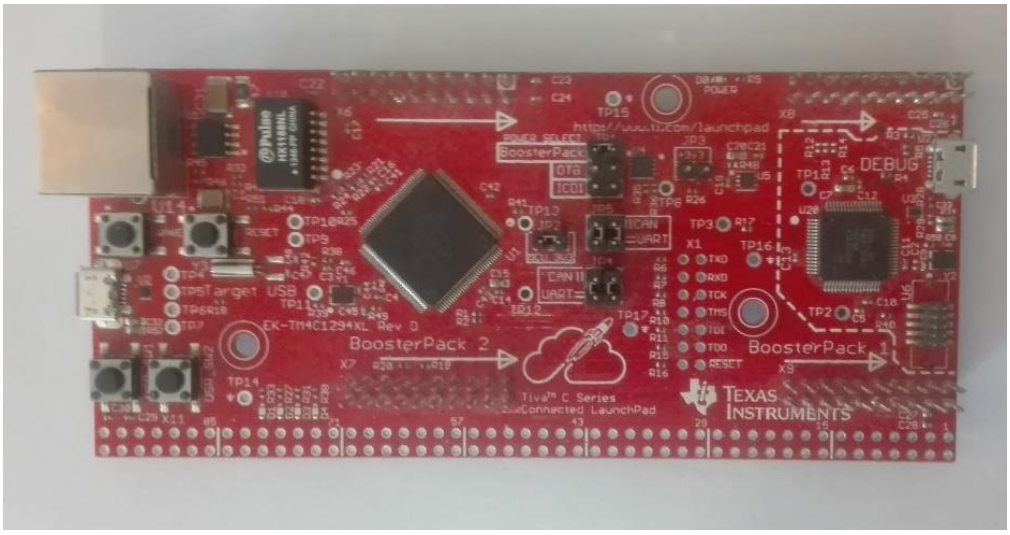


Figura 3-5. Tarjeta de desarrollo de Texas Instruments.

3.3 Arquitectura de Software

Antes de comenzar con la implementación, es necesario establecer la arquitectura del software correspondiente. Se utilizó un enfoque de desarrollo modular, siguiendo las buenas prácticas propuestas por el patrón de diseño GRASP [44] *(Object-oriented design General Responsibility Assignment Software Patterns),* el cual propone un conjunto de buenas prácticas para construir software con un alto nivel de cohesión y bajo acoplamiento.

3.4 Estrategia de implementación

Se decidió implementar el software de abajo hacia arriba, es decir, comenzando por la creación de módulos de más bajo nivel para realizar las pruebas unitarias correspondientes. Una vez que la funcionalidad de cada módulo se verifica, se puede comenzar a programar los módulos de capas superiores. En caso de que los módulos en conjunto formen un subsistema, este se integra y se convierte en un solo módulo para la arquitectura. Esta estrategia permite construir el software con una base sólida, así como acelerar la identificación y corrección de errores, pues al preservar un bajo acoplamiento entre las capas, se asegura que la causa raíz de un defecto de software se contenga en un sólo módulo. En la figura 3-7 se puede observar dicho enfoque; los módulos forman parte de un subsistema, y el conjunto de subsistemas genera el sistema completo.

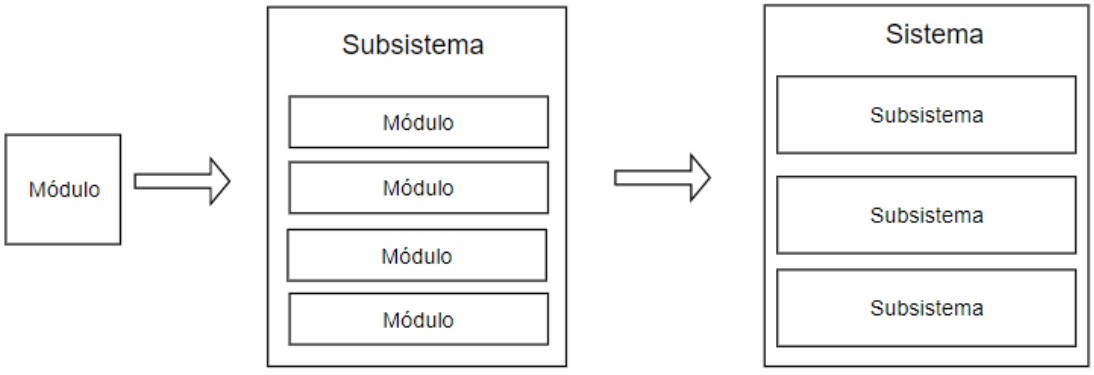


Figura 3-6. Proceso Bottom-Up.

3.5 Desarrollo de código

Respecto a la implementación del software, fue el módulo encargado de la transmisión y recepción de tramas de CAN el primer componente de software creado, pues las capas superiores definidas en la arquitectura de la solución dependen de dicho módulo. Para asegurar el correcto funcionamiento de cada componente, se ejecutaron pruebas unitarias desarrolladas en el entorno de programación de Visual Studio. Como se muestra en la Figura 3-8, una prueba unitaria, también conocida como prueba de módulo, se encarga de verificar que la salida de un módulo corresponda al valor esperado para la entrada del mismo.

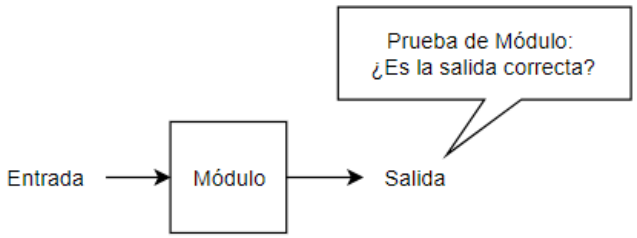


Figura 3-7. Pruebas de módulo.

Las capas superiores fueron creadas de la misma manera, es decir, generando el componente de software y posteriormente las pruebas unitarias correspondientes. Aunque los módulos son probados individualmente, es importante tener en cuenta que pueden ser parte de un subsistema, por lo que además de realizar pruebas unitarias deben probarse dichos subsistemas mediante pruebas de integración, como se muestra en la Figura 3-9.

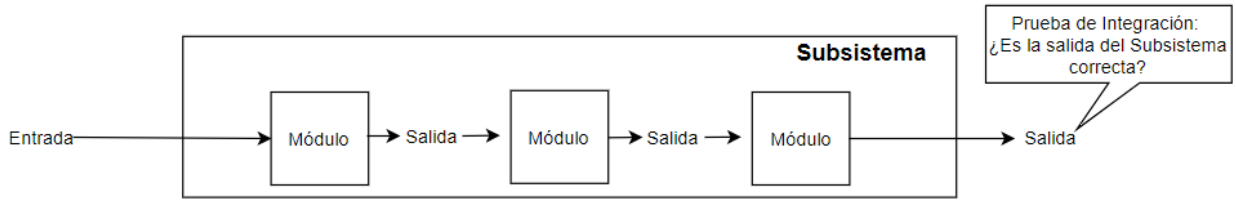


Figura 3-8. Pruebas de Integración.

El software utilizado como entorno de programación, Visual Studio [40], facilita la realización de pruebas unitarias. Ya que el programa posee una herramienta integrada para este propósito. Dichas pruebas son codificadas y compiladas junto con la solución, posteriormente, son

gestionadas mediante un panel expuesto en la interfaz gráfica de usuario. Esto hace que el desarrollador pueda probar una o varias funciones del código sin tener que hacer uso de un software dedicado. La principal diferencia entre una función de prueba y una función que forma parte del programa principal, es el uso del atributo "[TestMethod]" como se muestra en la Figura 3-10.

```
[TestMethod]
public void ServiceReadUDSMultiFrame() {
    Init();
    List<byte[]> L = new List<byte[]>();
    L.Add(Read0158);
    var K = S.UDS_Request(0x7f2, L);
    Assert.IsNotNull(K);
    Assert.AreEqual(Interfaces.CanReturn.Ok, K.retInfo);
    Console.WriteLine("ret info : " + Enum.GetName(typeof(CanReturn), K.retInfo));
    foreach (byte[] V in K.Values)
        foreach (byte b in V) {
            Console.Write(b.ToString("X2") + " ");
        }
}
```

Figura 3-9. Fragmento de código de la solución.

Se muestra también en la Figura 3-11, la interfaz integrada en Visual Studio, que permite ejecutar una o varias pruebas previamente codificadas.

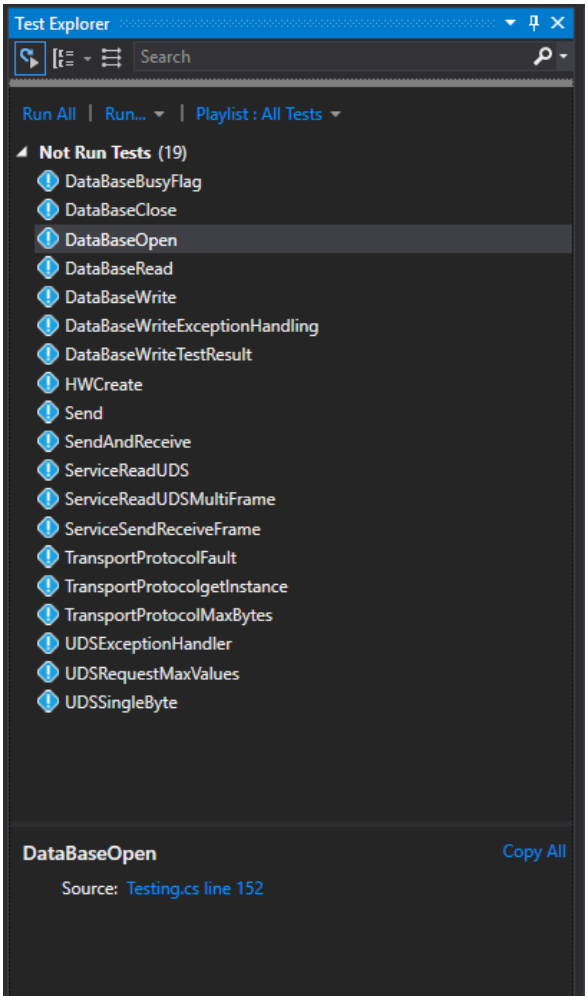


Figura 3-10. Panel de pruebas de Visual Studio.

3.6 Control de versiones

Una funcionalidad importante de Visual Studio, es la incorporación de un sistema de control de versiones denominado *Team Foundation Server* [45] . Este sistema permite almacenar en la nube todos los archivos de código fuente del programa, así como tener versiones incrementales de los mismos a medida que se realizan cambios. De esta manera, se puede regresar a versiones anteriores del software en el caso de que un cambio haya ocasionado que la solución completa dejara de funcionar adecuadamente.

4. Resultados

4.1 Arquitectura de software

Para el desarrollo del proyecto, se realizó la estructura de componentes de software mostrada en la Figura 4.1. Dicha arquitectura permite satisfacer las necesidades establecidas durante el planteamiento del problema, separando las funcionalidades que no tienen una dependencia directa e integrándolas en una capa superior; por ejemplo, los subsistemas de adquisición de imágenes, creación de reportes y comunicación son entidades totalmente independientes, pero la capa de "Test Framework" se encarga de controlarlas y establecer el intercambio de información entre ellas. De esta manera, el componente de creación de reportes puede utilizar las imágenes generadas por la cámara USB, así como las tramas del bus de CAN. Lo anterior, en virtud de que esta información es otorgada por la capa superior y no por los subsistemas mencionados.

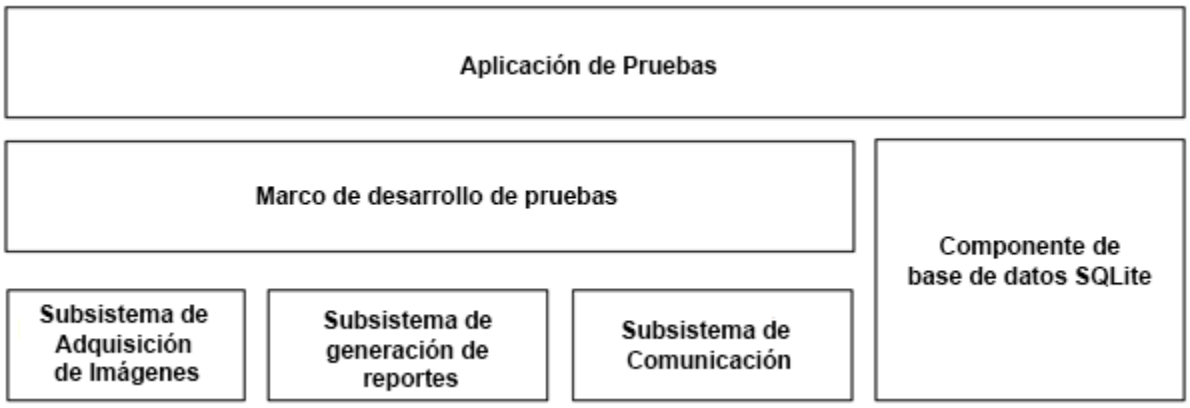


Figura 4-1. Arquitectura de software de la solución.

La funcionalidad de cada subsistema que conforma la arquitectura propuesta es la siguiente:

- **Subsistema de adquisición de imágenes**

  El susbsistema de adquisición de imágenes provee a las capas superiores una interfaz mediante la cual se puede capturar video y fotografías empleando una cámara de video USB.

- **Subsistema de comunicación**

  Este susbsistema se encarga de controlar cualquier aspecto relacionado al bus de comunicaciones CAN, por lo que es el más importante de la solución. Derivado de esto, es discutido a detalle posteriormente en este documento. Su función es otorgar a las capas superiores la capacidad de comunicarse en el bus de CAN.

- **Subsistema de generación de reportes**

  Este componente es capaz de generar reportes sobre la ejecución de las pruebas en formato pdf y xls. Recibe texto e imágenes de las capas superiores que luego integra en un documento.

- **Marco de desarrollo de pruebas**

  Se encarga de utilizar los componentes mencionados anteriormente y de establecer el contexto en el que los casos de prueba se ejecutan. Por ejemplo, puede utilizar el componente de adquisición de imágenes para tomar una fotografía, añadiendo información de tiempo de ejecución de caso de prueba en el que se realizó la captura. Esto para luego transferir la imagen al subsistema de generación de reportes para su documentación.

- **Componente de base de datos SQLite**

  La solución requiere almacenar los resultados de ejecución de las pruebas de manera tal que no sean eliminados al apagar la computadora, por lo que se decidió utilizar una base de datos almacenada en el disco duro para logralo. Este componente se encarga de realizar las operaciones de lectura sobre dicha base de datos utilizando la librería SQLite.

- **Aplicación de pruebas**

  Expone una interfaz gráfica al usuario del programa, mediante la cual se puede realizar la ejecución de casos de prueba, así como la consulta de los resultados en la base de datos.

4.2 Subsistema de comunicación CAN

4.2.1 Descripción general

Como se mencionó anteriormente, el subsistema de comunicaciones de CAN es el componente más importante de la solución, pues es mediante este componente que el programa ejecutado sobre Windows puede interactuar con la unidad bajo prueba. Uno de los objetivos del proyecto es crear una arquitectura de software que permita reutilizar los componentes creados, por lo que los siguientes dos principios fueron muy importantes durante el desarrollo:

- **Modularidad:**

  - Cada capa de software debe realizar un conjunto de tareas bien definidas y con ello, preservando la dependencia con respecto a los detalles de implementación de las capas superiores o inferiores al mínimo. Es decir, la transferencia de información entre capas debe ser mediante interfaces bien definidas. Por lo que los cambios en la implementación de una determinada capa no requieren modificar ningún otro componente mientras las interfaces que las conectan permanezcan iguales.

- **Abstracción de hardware:**

    - Cada hardware con capacidad de comunicación con el bus de CAN posee cualidades distintas y mecanismos de funcionamiento diferentes. Más aún, cada fabricante de hardware crea una interfaz de programación distinta para utilizar con sus productos. Por ello, es necesario crear una interfaz común entre los dispositivos para lograr que las capas superiores interpreten de manera indistinta la información que recibe de un hardware ValueCAN o un VN1630A.

4.2.2 Implementación de abstracción de hardware

Para enfatizar el concepto de abstracción de hardware, que permite la interoperabilidad de dispositivos [46], se analizó uno de los mecanismos de abstracción realizados. La librería proporcionada por el fabricante Vektor GmbH para el dispositivo VN1630A implementa el tipo de dato "*xl_event*". Este tipo de dato contiene, entre otros miembros, un arreglo de bytes para almacenar los datos recibidos en una trama de CAN, así como la marca de tiempo correspondiente a la recepción el mensaje en formato decimal. Por otra parte, la librería proporcionada por el fabricante Intrepid Control Systems para el dispositivo ValueCAN, hace uso del tipo de dato "*IcsSpyMessage*", mismo que almacena cada byte recibido en un miembro diferente, en lugar de agruparlos en un arreglo de bytes como lo hace el tipo "*xl_event*". Además, no almacena la marca de tiempo de recepción de mensaje automáticamente, sino que es responsabilidad del usuario calcularla mediante otra función incluida en la librería. La figura 4-2 representa de una manera gráfica lo anterior.

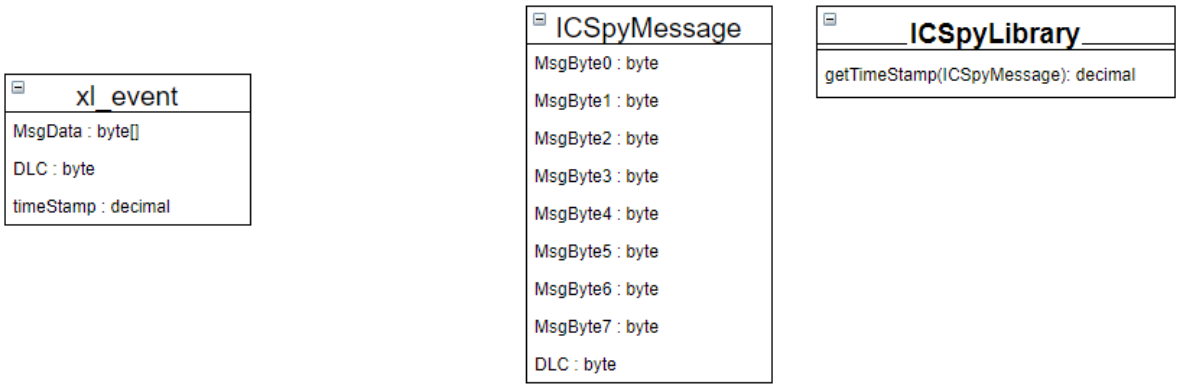


Figura 4-2. Diferencias en la implementación de las distintas interfaces de programación de hardware.

Derivado de lo anterior, la capa de abstracción de hardware hace uso de un nuevo tipo de dato a fin de proporcionar a las capas superiores la misma representación de la información, independientemente del hardware que la genera. Este tipo de dato, de nom bre CAN_Msg_t se muestra en la Figura 4-3.

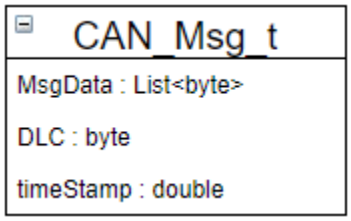


Figura 4-3. Tipo de dato de la capa de interfaz de hardware.

De una manera simplificada, puede decirse que tanto los tipos "*xl_event*" como "*ICSpyMessage*" son convertidos al tipo "*CAN_Msg_t*" para su ingreso al sistema. Lo anterior, implica para el caso de "*ICSpyMessage*", llamar a la función correspondiente para calcular la marca de tiempo y agregarla al campo del nuevo tipo "*CAN_Msg_t*".

En la Figura 4-4, se muestran los módulos que componen al subsistema de comunicaciones. Como se observa en la imagen, la capa inferior es la de abstracción de hardware, por lo que todas las capas superiores funcionan adecuadamente sin importar el tipo de dispositivo que se conecte a la computadora. Cada componente, excluyendo a la ya descrita capa de abstracción, se explica a continuación.

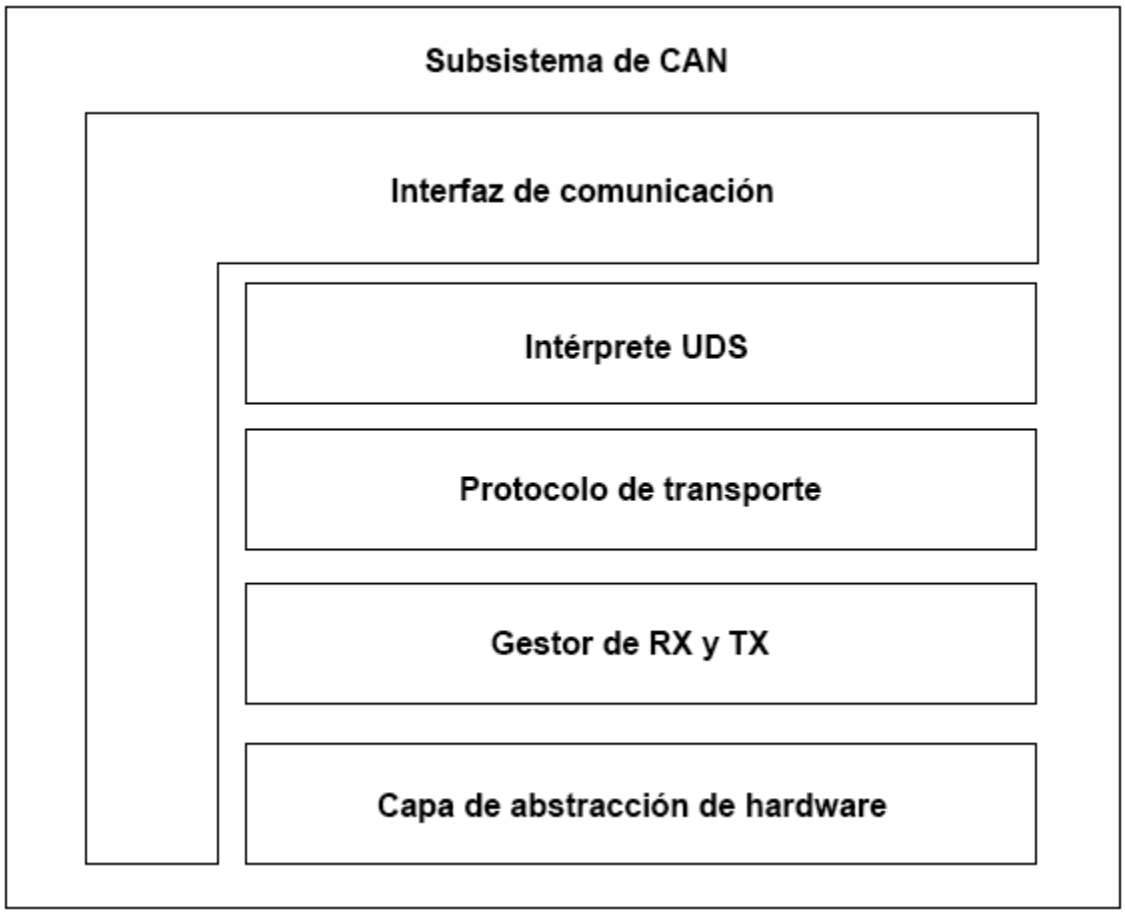


Figura 4-4. Diseño del subsistema de comunicaciones a través de CAN.

- **Gestor de RX y TX de CAN**

  Debido a que el bus de CAN emplea tramas que deben ser transmitidas periódicamente, este componente de software se encarga de implementar un contador de tiempo interno para transmitir los mensajes con la periodicidad correspondiente.

  Se encarga además de tener una tarea dedicada de recepción de tramas, liberando al hardware de almacenarlas en memoria y a las capas superiores de la responsabilidad de capturarlas "a tiempo". Pues, al ser CAN un bus asíncrono, el no implementar una tarea dedicada para la recepción, puede significar la pérdida de información.

- **Protocolo de transporte**

  Cuando se requiere enviar o recibir más de 8 bytes de información es necesario implementar una capa de transporte. Para el caso del protocolo CAN, este se encuentra definido en la especificación ISO 15765-2 [47], aunque para efectos de este proyecto se implementaron solamente las partes necesarias para soportar la capa superior.

- **Intérprete UDS**

  El sistema de diagnósticos unificado utiliza extensivamente la capa de transporte para funcionar, ya que requiere en ocasiones transmitir una cantidad superior a los 8 bytes de información que el protocolo CAN soporta en una sola trama.

  Este módulo se encarga de ofrecer a las capas superiores la interpretación de UDS de los datos recibidos por la capa de transporte, así como el proceso inverso, la generación de datos empleando UDS para ser llevados por la capa de transporte.

- **Interfaz de Comunicación**

  Se encarga de proveer toda la funcionalidad de las capas inferiores a componentes superiores de la arquitectura. Es decir, coordina la transferencia de información entre los módulos independientes, haciéndolos operar como un conjunto.

  Es importante resaltar que, de acuerdo con los principios de la programación orientada a objetos, la única capa del módulo que es visible a las capas superiores es esta interfaz, por lo que se garantiza el encapsulamiento.

4.3 Base de datos

El componente de base de datos se diseñó de manera tal que permita el almacenamiento no volátil de los resultados de la ejecución de prueba, así como la fecha y hora en la que fueron ejecutadas las mismas. Gracias a la librería SQLite compatible con el entorno de ejecución .NET, el desarrollo del componente fue simplificado considerablemente, pues la librería mencionada se encarga del acceso a la base de datos. Por lo tanto, el gestor de base de datos del proyecto se limita solamente a proporcionarle a la librería SQLite, la información a escribir o leer. En la Figura 4-5 se observa el diagrama de diseño de la base de datos realizada durante el desarrollo de este proyecto

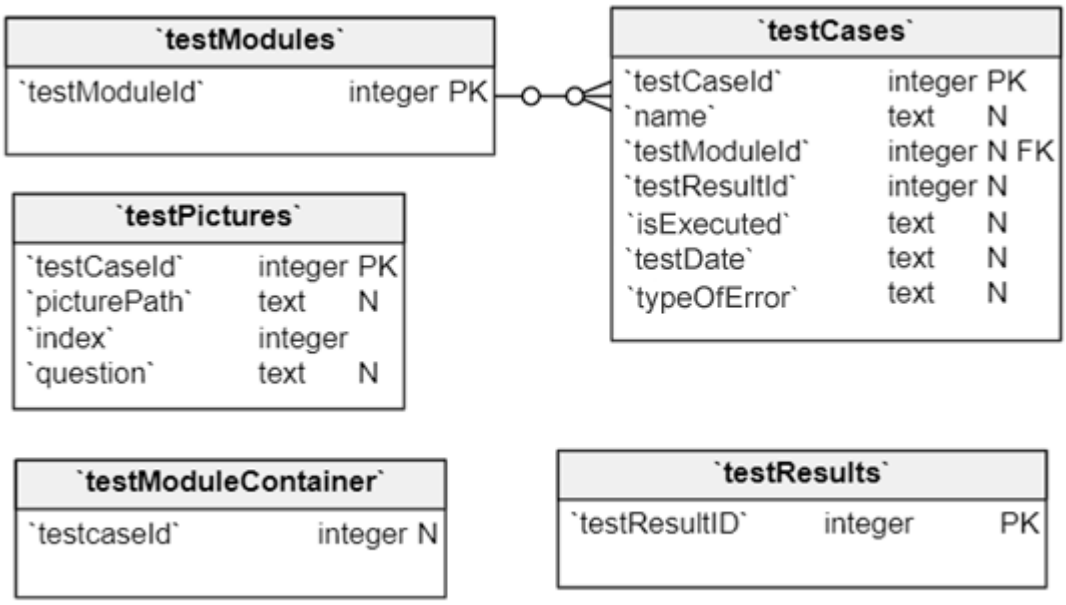


Figura 4-5. Diseño de la base de datos.

Para asistir al desarrollo y la comprobación de la base de datos, el software “DBBrowser for SQLite” [48] fue utilizado. Este programa permite hacer modificaciones en tiempo real a cualquier base de datos con tecnología SQLite. Otorga una interfaz de usuario intuitiva que permite visualizar la información contenida en los registros de la base de datos en un formato de tabla. Lo anterior, permite comprobar rápidamente que los datos escritos sean los correctos. La Figura 4-6 muestra la interfaz gráfica del software mencionado.

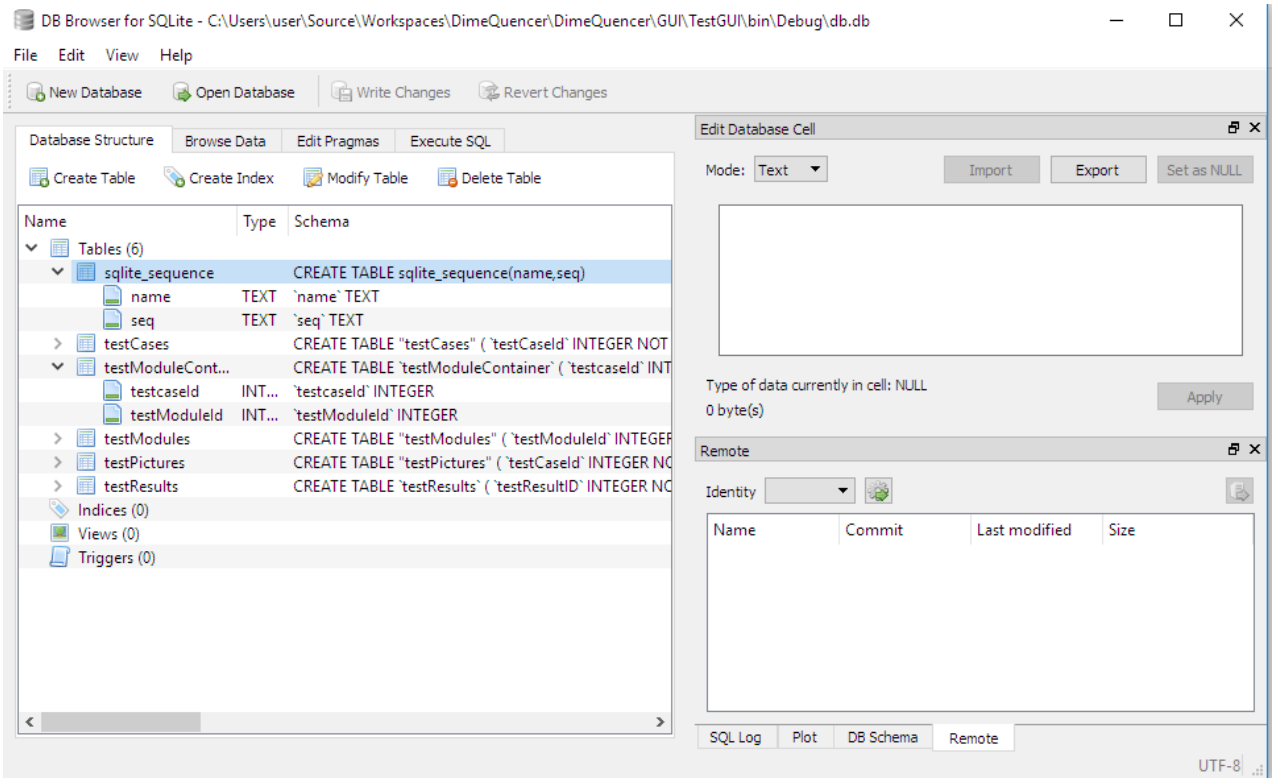


Figura 4-6. Interfaz de Usuario de DBBrowser.

La implementación en código utiliza funciones para envolver el uso de la librería, adaptándolo al nivel de abstracción necesario por las capas superiores de la solución, es decir, el contexto de ejecución de casos de prueba y la información que la base de datos contiene.

En la Figura 4-7 se muestra un método escrito para obtener los resultados de la ejecución de un caso de prueba. Para obtener dicho resultado se necesita solamente conocer el número de identificador único de la prueba, por lo que el acceso a la base de datos queda totalmente encapsulado para las capas superiores.

```
public testCaseInfo getTestcaseInfo(int testCaseID) {

    testCaseInfo rval = new testCaseInfo();
    string sqlString = @"Select * from testCases where testCaseID = " + testCaseID.ToString();
SQLiteCommand cmd = new SQLiteCommand(sqlString, dbConnection);
        SQLiteDataReader reader = cmd.ExecuteReader();
        while (reader.Read()) {
            rval.veredict = (TestCase.Results)(int)(long)reader["resultId"];
            rval.photoResult = (TestCase.Results)(int)(long)reader["photosResultId"];
            rval.logpath = (string)reader["reportPath"];
            if (reader["executionDate"] == DBNull.Value) {
                rval.executionDate = " --- ";
            }
            else {
                rval.executionDate = (string)reader["executionDate"];
            }

        }
    return rval;
}
```

Figura 4-7. Fragmento de código para el acceso a la base de datos.

4.4 Módulo de generación de reportes

La librería NPOI [49] fue utilizada para generar reportes en formato .xls, nuevamente haciendo uso de funciones envoltorio que encapsulan los detalles de implementación de dicha librería. El formato .xls puede ser abierto en Microsoft Excel [50] , como se muestra en la Figura 4-8.

L16

	A	B	C	D	E	F	G	H
1								
2			TEST REPORT					
3								
4						Module	MOD_004	
5	**TestCaseID**	**Description**	**Test Veredict**	**Tester Comment**	**Detail Log**			
6	TC_SG_DISP_001	Park Brake telltale in OFF mode	Passed		TC_SG_DISP_001.pdf			
7	TC_SG_DISP_002	Park Brake telltale in ON mode	Passed		TC_SG_DISP_002.pdf			
8	TC_SG_DISP_003	Park Brake telltale in IDLE mode	Fail		TC_SG_DISP_003.pdf			
9	TC_SG_DISP_004	UDS Override ON - OFF	Passed		TC_SG_DISP_004.pdf			
10	TC_SG_DISP_005	UDS Override OFF - ON	Passed		TC_SG_DISP_005.pdf			
11	TC_SG_DISP_006	Park Brake Blink feature	Passed		TC_SG_DISP_006.pdf			
12	TC_SG_DISP_007	UDS IO control read	Passed		TC_SG_DISP_007.pdf			

Figura 4-8. Reporte generado por el programa.

4.5 Interfaz gráfica

Para mostrar al usuario la información de manera eficiente, se contempló una interfaz con un formato similar a la mayoría de las pantallas de la actualidad, el cual es más ancho que alto. En la Figura 4-9 se aprecian las regiones del programa dedicadas a diferentes funcionalidades.

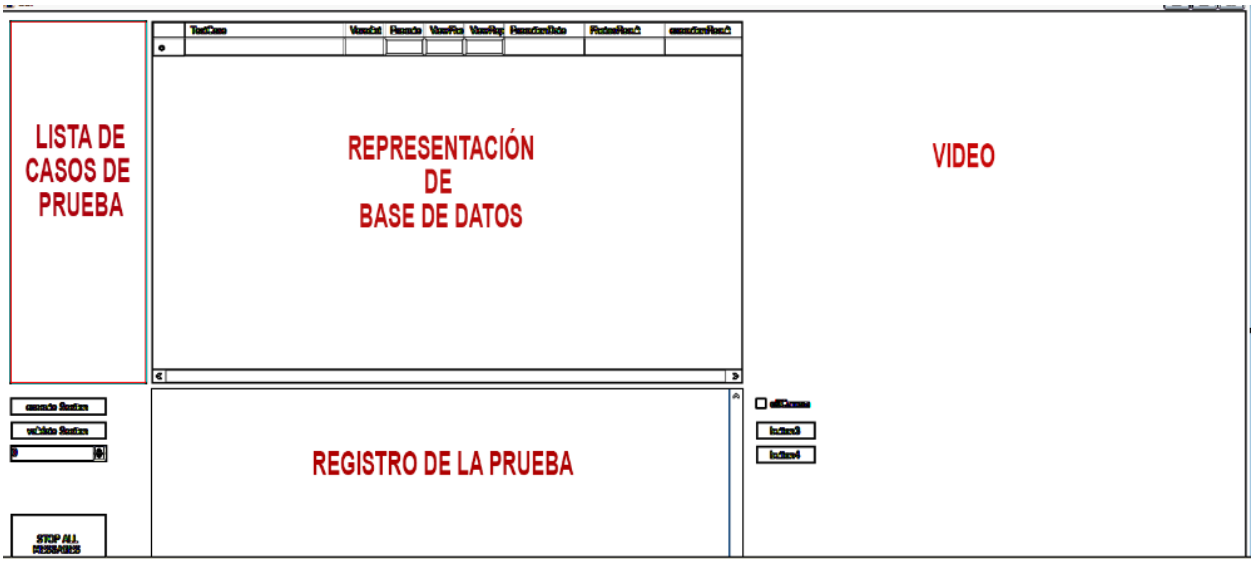


Figura 4-9. Distribución de la interfaz de usuario.

4.6 Pruebas de módulo e Integración

Se establecieron 68 casos de prueba para asegurar que el funcionamiento de los módulos y las interfaces que los conectan sean los correctos. El resultado de las mismas es inherentemente satisfactorio, ya que la metodología de trabajo propuesta impide que se pueda continuar con el desarrollo de la solución, si los casos de prueba que existan en ese momento no tienen un resultado positivo. En la Figura 4-10, se observa que la validación de la solución se llevó a cabo correctamente.

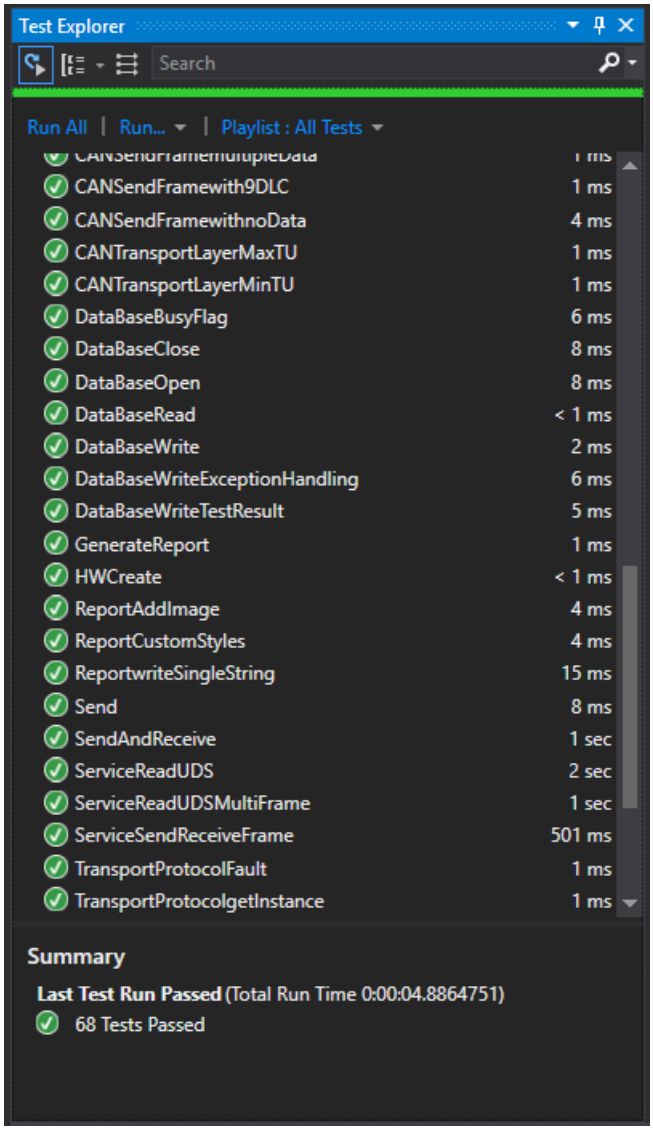


Figura 4-10. Panel de ejecución de pruebas.

4.7 Ejecución de la solución.

La compilación de la solución completa tiene como resultado un archivo ejecutable con la funcionalidad del programa. En la Figura 4-11, se puede apreciar la interfaz gráfica propuesta, esta vez mostrando la información relevante a cada región de la ventana

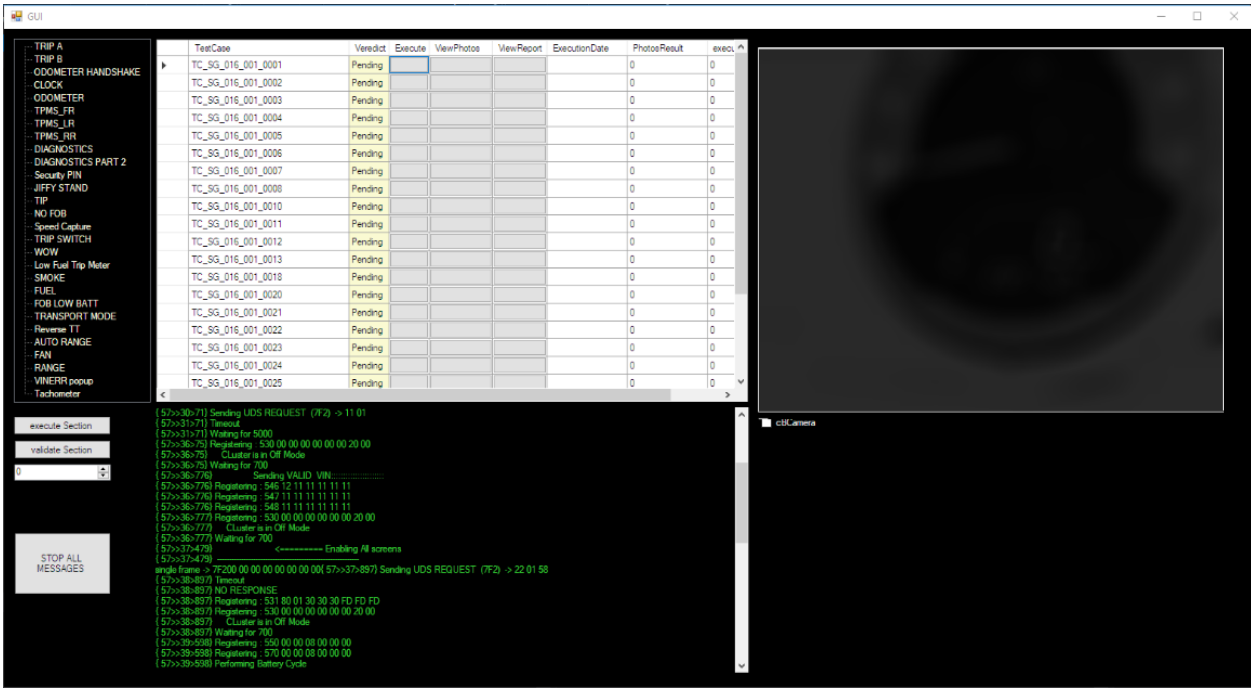


Figura 4-11. Software en ejecución.

En el panel que se encuentra a la izquierda de la ventana se enlistan los módulos de prueba, al seleccionar un módulo, los casos de prueba correspondientes aparecen en el panel central. La información proviene de la base de datos creada para este programa. En el caso de utilizar una cámara, se puede apreciar la imagen obtenida en el panel de la derecha. Y, por último, en la parte inferior del programa se encuentra el registro de la prueba; una salida de texto que se actualiza medida que la ejecución avanza.

Es importante destacar que se utilizaron las 3 interfaces USB-CAN propuestas en la metodología. En todos los casos el software creado operó indistintamente.

5. Discusión

El software descrito en este documento cumple cabalmente con los objetivos planteados en el enunciado de trabajo, ya que tiene la capacidad de utilizar indistintamente diversas interfaces de USB a CAN para ejecutar casos de prueba programados en el lenguaje C#. A su vez, la implementación de una base de datos le brinda la capacidad de almacenar el resultado de cada prueba y el reporte de su ejecución en formato de Excel.

Este programa, comparado con productos como CANoe y VehicleSPY, ofrece un conjunto de funciones reducido. Lo anterior, en virtud de que la finalidad de los productos mencionados incluye el análisis, diagnóstico, simulación, estímulo y pruebas sobre el protocolo CAN [9], mientras que el programa creado como parte de este trabajo, se enfoca solamente en el desarrollo de pruebas. Esto no supone necesariamente una desventaja, sino más bien una diferencia; los programas comerciales citados representan una solución versátil, mientras que el software desarrollado en este trabajo es una solución específica. Por lo tanto, como se muestra en la Figura 5-1, dicha solución implementa funcionalidades orientadas al desarrollo de pruebas que el software comercial no tiene, cómo el soporte nativo para una fuente de video y el soporte de múltiples interfaces USB a CAN.

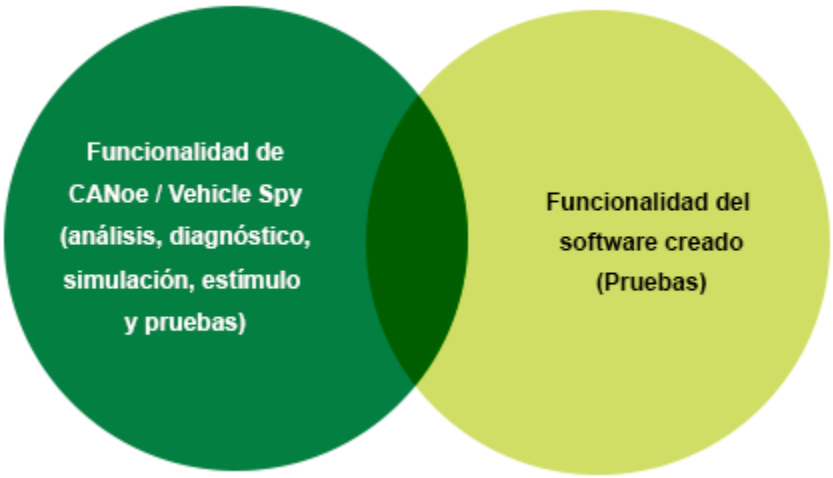


Figura 5-1 Diagrama de Venn comparando las prestaciones ofrecidas por el software creado.

El soporte a distintas interfaces USB a CAN, es la principal ventaja para el usuario final, pues este último cuenta con una mayor libertad de compra. Si el precio del hardware de algún fabricante se eleva, o si el producto se descontinúa, puede el usuario utilizar hardware creado por otra empresa implementando solamente la capa de abstracción correspondiente. Alternativamente, existe la posibilidad de utilizar una interfaz USB a CAN propia, otorgando así una independencia completa de los fabricantes de soluciones comerciales, pues el software y hardware requerido no sería provisto por ellos.

Tomando como referencia la más barata de las soluciones comerciales analizadas en este trabajo, VehicleSPY PRO con un precio de 2,795 USD [17], el costo calculado para provisionar este software a 15 equipos de cómputo es de 41,925 USD. Por lo tanto, puede resultar mucho más económico implementar una solución de software similar a la propuesta en este trabajo, especialmente a medida que la cantidad de equipos aumenta.

Otra de las ventajas importantes de esta solución de software, es el hecho de estar basada en el entorno de programación .NET [51]. Lo anterior, le brinda una gran flexibilidad para el desarrollo de pruebas, al ejecutarse estás últimas en el contexto de .NET y no en el contexto de un software compilado como en el caso de los programas comerciales VehicleSPY o CANoe. Un ejemplo de lo anterior, es la utilización de una librería de procesamiento digital de imágenes como OpenCV [52] en conjunto con una cámara de video USB. La solución de software de este trabajo puede ejecutar un caso de prueba en el que mediante visión artificial se detecte si el comportamiento de la unidad bajo prueba es el esperado. Esta funcionalidad no puede replicarse por el software comercial al no estar diseñado para la interacción con librerías externas.

6. Conclusión

Como resultado de este trabajo, se obtuvo un software para la ejecución y desarrollo de casos de prueba utilizando el protocolo CAN y empleando una capa de abstracción de hardware que permite la interoperabilidad de interfaces USB a CAN de varios fabricantes.

Aunque el software mencionado se enfoca al área de pruebas funcionales, la arquitectura de software modular empleada durante el desarrollo de la solución, facilita la reusabilidad de componentes, el mantenimiento y la escalabilidad del sistema. Por lo que esta estructura puede ser utilizada como una base para trabajos derivados o complementarios. Por ejemplo, utilizando el subsistema de CAN desarrollado, pueden crearse interfaces gráficas para interactuar con los nodos de un bus de CAN en tiempo real. Incluso, es posible cambiar el enfoque completo del software reutilizando gran parte de los componentes ya existentes, para tener una solución de propósito general como CANoe. Implementando de esta manera, componentes de software para graficar señales, consola de diagnóstico, panel de generación de señales, e interfaz gráfica, entre otros.

Preservando el enfoque de pruebas funcionales, trabajos derivados de esta solución incluyen la creación de algoritmos de visión artificial para la utilización del componente de adquisición de imágenes de este programa, así como la elaboración de un sistema capaz de generar estímulo eléctrico a la unidad bajo prueba o adquirir señales provenientes de la misma. Este sistema, por ejemplo, puede reemplazar la entrada de un sensor, o monitorear alguna salida de la unidad bajo prueba en tiempo real, con lo que el estímulo de las pruebas ejecutadas por el software no se limitaría al protocolo CAN.

Bibliografía

[1] Freescale Semiconductor, “Future Advancements in Body Electronics”. [En línea]. Disponible en: http://www.nxp.com/docs/en/white-paper/BODYDELECTRWP.pdf.

[2] Bauser Inc., “Instrument clusters Catalog”. [En línea]. Disponible en: http://www.bauser-control.de/cd_bed_anl/pdf_files/englisch/bauser_kombi-instrumente_1105_en_v1.1.pdf. [Consultado: 09-abr-2017].

[3] “Bosch en México | Bosch México”. [En línea]. Disponible en: http://www.bosch.com.mx/es/mx/startpage_3/country-landingpage.php. [Consultado: 13-nov-2017].

[4] “CAN in Automation (CiA): History of the CAN technology”. [En línea]. Disponible en: https://www.can-cia.org/can-knowledge/can/can-history/. [Consultado: 02-sep-2017].

[5] Federal Register Environmental Documents, “Control of Air Pollution From New Motor Vehicles and New Motor Vehicle Engines”, 10-may-2009. [En línea]. Disponible en: https://web.archive.org/web/20090510202915/http://www.epa.gov/fedrgstr/EPA-AIR/2005/December/Day-20/a23669.htm. [Consultado: 02-sep-2017].

[6] VDA QMC Working Group 13 / Automotive SIG, “Automotive SPICE Process Assessment / Reference Model”. 16-jul-2015.

[7] “Vector: Software + Servicios de Ingeniería Automotriz”. [En línea]. Disponible en: https://vector.com/. [Consultado: 13-nov-2017].

[8] “VN1600 Network Interface for CAN, LIN, K-Line, J1708 and IO”. [En línea]. Disponible en: https://vector.com/vi_vn1600_en.html. [Consultado: 26-nov-2017].

[9] “CANoe - ECU Development & Test”. [En línea]. Disponible en: https://vector.com/vi_canoe_en.html. [Consultado: 02-sep-2017].

[10] “Vector XL Driver Library”. [En línea]. Disponible en: https://vector.com/vi_xl_driver_library_en.html. [Consultado: 02-sep-2017].

[11] D. Hercog y B. Gergič, “A Flexible Microcontroller-Based Data Acquisition Device”, *Sensors*, vol. 14, núm. 6, pp. 9755–9775, jun. 2014.

[12] Microchip Technology Inc., “PIC18F47J53 Family datasheet”. 2016.

[13] “LabVIEW - National Instruments”. [En línea]. Disponible en: http://www.ni.com/es-mx/shop/labview.html. [Consultado: 13-nov-2017].

[14] “CAN Tools”, *CAN Tools*. [En línea]. Disponible en: http://www.can232.com/. [Consultado: 10-sep-2017].

[15] “CANUSB Module”, *Dontronics*. [En línea]. Disponible en: https://www.shop-dontronics.com/canusb-module. [Consultado: 25-nov-2017].

[16] M. Di Natale, H. Zeng, P. Giusto, y A. Ghosal, *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media, 2012.

[17] “Intrepid Control Systems”. [En línea]. Disponible en: https://www.intrepidcs.com/products/software/vehicle-spy/. [Consultado: 13-nov-2017].

[18] “Intrepid Control Systems online store”, *store.intrepidcs.com*. [En línea]. Disponible en: http://store.intrepidcs.com//ProductDetails.asp?ProductCode=VSPY%2D3%2DPRO. [Consultado: 25-nov-2017].

[19] “Windows | Sitio oficial para Microsoft Windows 10 Home, S y Pro. Portátiles, PCs, tabletas y más.” [En línea]. Disponible en: https://www.microsoft.com/es-mx/windows/. [Consultado: 13-nov-2017].

[20] “ValueCAN3 Limited (1x DW CAN)”, *store.intrepidcs.com*. [En línea]. Disponible en: http://store.intrepidcs.com//ProductDetails.asp?ProductCode=VCAN3%2DDWCAN1. [Consultado: 10-sep-2017].

[21] “PCAN-USB: PEAK-System”. [En línea]. Disponible en: http://www.peak-system.com/PCAN-USB.199.0.html?&L=1. [Consultado: 10-sep-2017].

[22] “PCAN-Explorer 5: PEAK-System”. [En línea]. Disponible en: https://www.peak-system.com/PCAN-Explorer-5.249.0.html?&L=1. [Consultado: 25-nov-2017].

[23] “ISO/IEC 7498-1 Open Systems Interconnection - Basic Reference Model”. [En línea]. Disponible en: https://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf.

[24] “ASCII - Cryptography, Information Theory, and Error-Correction - Bruen - Wiley Online Library”. [En línea]. Disponible en: http://onlinelibrary.wiley.com/doi/10.1002/9781118033296.oth/pdf. [Consultado: 08-oct-2017].
[25] C. A. N. Specification, “Bosch”, *Robert Bosch GmbH Postfach*, vol. 50, 1991.
[26] “CAN in Automation (CiA): CANopen”. [En línea]. Disponible en: https://www.can-cia.org/canopen/. [Consultado: 07-oct-2017].
[27] “Aeronautical Radio, Inc (ARINC) Standards | IHS Markit”. [En línea]. Disponible en: https://www.ihs.com/products/arinc-standards.html. [Consultado: 07-oct-2017].
[28] Rockwell Automation, “DeviceNet Media, Design and Installation Guide”. [En línea]. Disponible en: http://literature.rockwellautomation.com/idc/groups/literature/documents/um/dnet-um072_-en-p.pdf. [Consultado: 07-oct-2017].
[29] N. Sidhu, P. Singh, y S. Rani, “Improved Optimal Slotted CSMA/CA Protocol”, *Int. J. Comput. Appl.*, vol. 79, núm. 6, 2013.
[30] E. Williams, “Building your Own USB Devices for AVR with the V-USB Library”. [En línea]. Disponible en: http://www.oreilly.com/pub/e/3082. [Consultado: 22-oct-2017].
[31] USB Implementers Forum, Inc., “Universal Serial Bus Specification”. 2011.
[32] “USB.org - Documents”. [En línea]. Disponible en: http://www.usb.org/developers/docs/. [Consultado: 22-oct-2017].
[33] “IDS USB 3 uEye CP industrial camera, CMOS from e2v, CMOSIS, Aptina, ON Semiconductor, Sony”. [En línea]. Disponible en: https://en.ids-imaging.com/store/products/cameras/usb-3-0-cameras/ueye-cp.html. [Consultado: 22-oct-2017].
[34] J. L. Axelson, *USB complete: the developer’s guide*, 4th ed., [Nachdr.]. Madison, Wis: Lakeview Research, 2009.
[35] Microchip Technology Inc., “ATtiny4 - 8-bit AVR Microcontrollers”. [En línea]. Disponible en: http://www.microchip.com/wwwproducts/en/ATtiny4. [Consultado: 13-nov-2017].
[36] “AVR MCUs - 8-Bit MCUs | 8-Bit MCUs | Microchip Technology Inc.” [En línea]. Disponible en: http://www.microchip.com/design-centers/8-bit/microchip-avr-mcus. [Consultado: 13-nov-2017].
[37] “SAMA5D3 Xplained”. [En línea]. Disponible en: http://www.atmel.com/tools/ATSAMA5D3-XPLD.aspx. [Consultado: 13-nov-2017].
[38] “Cortex-A5 – Arm Developer”. [En línea]. Disponible en: https://developer.arm.com/products/processors/cortex-a/cortex-a5. [Consultado: 13-nov-2017].
[39] “Cortex-M4 – Arm Developer”. [En línea]. Disponible en: https://developer.arm.com/products/processors/cortex-m/cortex-m4. [Consultado: 13-nov-2017].
[40] P. Crutchfield, “IDE de Visual Studio, editor de código, Team Services y Mobile Center”, *Visual Studio*, 16-nov-2016. [En línea]. Disponible en: https://www.visualstudio.com/es/. [Consultado: 25-nov-2017].
[41] “CCSTUDIO Code Composer Studio (CCS) Integrated Development Environment (IDE) | TI.com”. [En línea]. Disponible en: http://www.ti.com/tool/CCSTUDIO. [Consultado: 13-nov-2017].
[42] “What is an FPGA? Field Programmable Gate Array”. [En línea]. Disponible en: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Consultado: 12-nov-2017].
[43] “EK-TM4C1294XL ARM® Cortex®-M4F-Based MCU TM4C1294 Connected LaunchPad™ | TI.com”. [En línea]. Disponible en: http://www.ti.com/tool/ek-tm4c1294xl. [Consultado: 12-nov-2017].
[44] “GRASP patterns”, *Lynda.com - from LinkedIn*. [En línea]. Disponible en: https://www.lynda.com/Programming-Languages-tutorials/GRASP-patterns/471978/502212-4.html. [Consultado: 12-nov-2017].
[45] G. Paolini, “Agile, Git, CI with TFS | Team Foundation Server”, *Visual Studio*, 13-sep-2016. .
[46] K. Yaghmour, Ed., *Building embedded Linux systems*, 2nd ed. Beijing ; Cambridge: O’Reilly, 2008.
[47] “ISO 15765-2:2016 - Road vehicles -- Diagnostic communication over Controller Area Network (DoCAN) -- Part 2: Transport protocol and network layer services”. [En línea]. Disponible en: https://www.iso.org/standard/66574.html. [Consultado: 06-nov-2017].
[48] “DB Browser for SQLite”. [En línea]. Disponible en: http://sqlitebrowser.org/. [Consultado: 12-nov-2017].
[49] “NPOI”, *CodePlex*. [En línea]. Disponible en: https://npoi.codeplex.com/Wikipage?ProjectName=npoi. [Consultado: 13-nov-2017].
[50] “Microsoft Excel 2016: programa de hojas de cálculo - XLS”. [En línea]. Disponible en: https://products.office.com/es-mx/excel. [Consultado: 13-nov-2017].
[51].“NET”, *Microsoft*. [En línea]. Disponible en: https://www.microsoft.com/net/. [Consultado: 20-nov-2017].
[52] “OpenCV library”. [En línea]. Disponible en: https://opencv.org/. [Consultado: 20-nov-2017].