Adquisición de datos distribuidos: desarrollo y análisis de un modelo de caja negra para automóviles en pos de la seguridad vial

Consigli, Carlos Fernando

Gallego, Enrique Javier

Universidad Nacional de La Plata

2010

Índice

Capítulo I – Motivación y Antecedentes

Introducción

1. Breve descripción del trabajo

El siguiente trabajo presenta una breve descripción al sistema de sensado y captura de datos de un dispositivo de seguridad denominado Event Data Recorder (EDR). Un EDR es un dispositivo generalmente cerrado, que recibe datos mediante sensores y los almacena para que, en caso de un "evento", pueda recuperarse la información lindante al mismo.

Los objetivos de los EDR son mayormente orientados a la investigación, puesto que permiten obtener una representación más fiel acerca de los eventos en casos donde se dificultan las pruebas, como es el caso de los accidentes viales.

Nuestro enfoque en este trabajo es el de analizar los estándares principales que tratan la acción de los EDR, desarrollando nuestro propio modelo acorde a las normativas y describiendo los pasos y resultados obtenidos.

Mediante este trabajo esperamos aportar al mejor, más rápido y provechoso desarrollo de cajas negras para automóviles, una tecnología que, tarde o temprano, será de uso cotidiano.

El contenido del trabajo abarca las siguientes actividades:

- Análisis de estándares internacionales
- Estudio de las librerías SystemC
- Estudio del modelado a nivel de transacción (TLM)
- Desarrollo de un modelo de caja negra
- Realización de pruebas exhaustivas de la arquitectura con diversos parámetros
- Descripción de resultados
- Estimación de requerimientos mínimos
- Desarrollo de conclusiones y posibles trabajos a futuro

2. Objetivos a cumplir

2.1. Objetivo primario

Estudiar los estándares internacionales sobre EDRs y utilizar el lenguaje de descripción de hardware SystemC para describir y modelar una arquitectura propuesta sobre cajas negras, adhiriéndonos a los mencionados estándares.

2.2. Objetivo secundario

Realizar simulaciones de la arquitectura descripta probando diferentes escenarios, modificando los parámetros que afectan a la misma (velocidad del procesador, tamaño de la memoria, etc.) para validar su funcionamiento y determinar la exactitud que se obtiene en los resultados basándose en el cálculo de los datos enviados por los sensores.

3. Motivación del trabajo

La descripción del algoritmo para la caja negra trae aparejado una serie de aspectos altamente positivos en el área del diseño de hardware considerado desde una perspectiva informática.

Es necesario realizar un estudio profundo de las técnicas de diseño a nivel de transacción como los modelos System Architectural Model (SAM), logrando la interiorización y comprensión de las metodologías, técnicas y herramientas de software utilizadas comúnmente en ingeniería, pero esta vez dentro del ámbito de la informática.

Este trabajo no se limita sólo a un estudio teórico, sino que el resultado se vea reflejado en una descripción terminada que en un futuro pueda ser implementada en una plataforma específica con cambios adecuados.

4. Organización del informe

El presente informe se dividió en cinco capítulos, cada uno con varias subsecciones, y un anexo.

El *capítulo 1* describe una breve introducción al mundo de las Event Data Recorders, explicando sus orígenes, evolución e incidencias en la seguridad vial a lo largo de los años. El capítulo incluye también algunas de las normativas legales que rigen sobre las cajas negras en el mundo.

El *capítulo 2* trata de profundizar en los conocimientos necesarios para realizar nuestro desarrollo. Aquí se inicia con una descripción de la funcionalidad básica de las cajas negras, se continúa con la adquisición de datos distribuidos, conceptos de simulaciones y metodologías de diseño de hardware mediante TLM. Luego se trata el tema de un lenguaje de descripción de hardware SystemC y el estándar de comunicación CAN, muy utilizado en automóviles. Finalmente se hace un análisis del estándar del Institute of Electrical and Electronics Engineers (IEEE) para EDRs de vehículos automotores.

El *capítulo 3* presenta la arquitectura propuesta con las herramientas utilizadas, para continuar con la descripción detallada cada uno de sus componentes. Se comienza por la descripción funcional pasando finalmente a la descripción de las restricciones particulares del estándar del IEEE. En este capítulo también se introducen detalles de la simulación en la arquitectura descripta.

Las pruebas y resultados obtenidos se mencionan en el *capítulo 4*. La salida de la simulación es utilizada para validar el correcto funcionamiento de las descripciones de hardware y para observar las variaciones que se obtienen en los resultados al alterar los parámetros que afectan al comportamiento del sistema simulado.

En el *capítulo 5*, se exponen las conclusiones obtenidas acerca del uso de SystemC, de la descripción del algoritmo utilizado en las cajas negras y de la exactitud numérica alcanzada durante la simulación. Por último se presentan algunas perspectivas acerca de trabajos futuros.

El *anexo A* describe cada uno de los elementos recomendados del IEEE 1616 que utilizamos en nuestro diseño y simulación de caja negra, contemplando una breve descripción, su resolución, tamaño y formato de datos, tiempo y velocidad de muestreo, y unidad de medida utilizada.

5. Inicios

Desde mediados del siglo XX, los aviones comerciales han sido equipados con Flight Data Recorders (FDR) con el objetivo de grabar ciertos parámetros de vuelo en el caso de un eventual accidente [CAM07].

Ante un incidente, el procedimiento cuenta con tres partes: recolección de evidencia física en el lugar del hecho, extracción de los datos del instrumental de a bordo que fueron grabados por el FDR y finalmente una comparación y combinación de ambos juegos de datos. El resultado es una serie de conclusiones que contribuyen al esclarecimiento de las causas del accidente y por consiguiente facilitan la identificación de acciones de prevención para evitar eventos similares en el futuro.

Para el caso de la reconstrucción de accidentes automovilísticos se han usado técnicas similares, recolectando tanto datos físicos sobre los vehículos como datos humanos (por ejemplo heridas de los involucrados o testimonios de terceros). Sin embargo, en la mayoría de los casos los investigadores no han contado tradicionalmente con datos provenientes de un dispositivo análogo a las FDRs.

En la década del 70, la sociedad comenzó a hacer hincapié en la seguridad vial, y la experiencia positiva con las FDRs fue tomada en cuenta. Una de las primeras experiencias se realizó en el año 1974 cuando la agencia norteamericana National Highway Traffic Safety Administration (NHTSA) implementó un plan experimental instalando dispositivos de grabación en alrededor de 1000 vehículos que permitieron recolectar datos sobre 23 accidentes [HAI01]. En los años subsiguientes hubo proyectos similares, pero ninguno pudo demostrarse como efectivo.

En 1998 se pudo dar un ejemplo de la efectividad del análisis de los datos de accidentes, ya que en base al análisis de la información colectada por dispositivos de grabación sofisticados instalados por General Motors en 1992 en automóviles de la Indycar, los biomecánicos refinaron su conocimiento sobre el daño humano potencial, y, gracias a esto, se implementaron cambios en los sistemas de protección al conductor que redujeron el número de heridas graves en esa temporada [MEL96].

Desde entonces la cantidad de vehículos equipados con EDRs se ha ido incrementando. En 2005, según la NHTSA, en Estados Unidos el porcentaje se encontraba en un 64%.

Por este motivo, el IEEE publicó en 2004 [GRO05a] un estándar destinado a establecer ciertos lineamientos en la construcción de estos dispositivos. Sin embargo, las empresas automotrices no lo han adoptado en la construcción de sus vehículos.

6. Marco legal

En Estados Unidos, la NHTSA publicó en 2006 su propia reglamentación sobre EDRs, con conceptos similares a los del estándar del IEEE. Las regulaciones de esta reglamentación deben ser respetadas en todos los automóviles fabricados a partir del 1 de septiembre de 2010 que estén equipados con una caja negra. Sin embargo, no se establece que sea obligatorio que los vehículos estén equipados con una.

En Argentina, en abril de 2008 se sancionó la Ley 26.363 [LEY08] que establece, entre otras medidas, la instalación de un sistema de desgrabación de registros de operaciones de vehículos ante siniestros para su investigación.

En octubre de 2008 se reglamentó esta Ley estableciendo cuáles son los datos que deben registrar los EDRs en transportes comerciales y se estableció la obligatoriedad de este equipamiento en este tipo de transporte.

Sin embargo, la reglamentación no tiene consideraciones sobre la implementación en automotores particulares; solamente establece que los plazos para su implementación se acuerden con las terminales automotrices con la intervención de la Secretaría de Industria, Comercio y de la Pequeña y Mediana Empresa del Ministerio de Economía y Producción y de la Secretaría de Transporte del Ministerio de Planificación Federal, Ingresos Públicos y Servicios.

7. Misión

Dada la ausencia de lineamientos de diseño para cajas negras en vehículos particulares en Argentina y la inminente obligatoriedad de estos dispositivos,

decidimos realizar un modelo subsanando el vacío legal en Argentina con el estándar IEEE 1616.

Para ello, utilizamos la técnica de Transaction-Level Modelling [OPE09] (TLM) junto con el lenguaje de descripción de hardware SystemC [GRO05b]. De esta manera se obtiene un modelo ejecutable sin necesidad de llegar a una implementación en hardware real, pero dejando abierta la posibilidad de hacer una derivación del modelo a hardware en el futuro.

Con este modelo, es posible realizar simulaciones del funcionamiento del sistema con diferentes configuraciones y verificar los requerimientos y limitaciones de cada una.

Antecedentes

Los EDRs han estado instalados en los automóviles desde hace más de 30 años. La automotriz norteamericana General Motors ha sido la pionera en este sentido cuando en 1974 instaló las primeras en algunos automóviles. Inicialmente la funcionalidad estaba destinada al registro de activación (o no) de los airbags. Debido a que las bolsas de aire recién empezaron sus pruebas en automóviles de línea en la década del 70, se utilizaban los EDRs para verificar el correcto funcionamiento de las mismas.

A pesar de que existen varios nombres, a estos sistemas se los conoce comúnmente como Sensing Diagnostic Module (SDM), Restraint Control Module (RCM) o Airbag Control Module (ACM). La función primaria es la de ejecutar un algoritmo que analiza datos del sensor y activa los sistemas de airbag o cinturones inerciales del automóvil cuando un parámetro clave alcanza un umbral pre especificado. Estos módulos también tienen capacidad de almacenamiento limitada, la cual ha sido utilizada por los Original Equipment Manufacturer (OEM) para almacenar información de eventos con propósitos de investigación.

Hasta no hace mucho tiempo, sólo los OEMs tenían capacidad de descargar y analizar la información almacenada en estos módulos. Alrededor del año 2000 se hizo pública la primera herramienta de descarga de datos de accidentes. Esta herramienta permitió que la policía, los investigadores encargados de reconstrucción de accidentes y el público en general pudiesen conectar un EDR mediante el Diagnostic Link Connector (DLC) de un vehículo o mediante el módulo del airbag. Este paquete de recuperación de datos, de la compañía Vetronix, puede recobrar datos del EDR de cierta cantidad de vehículos de General Motors desde 1996 y de Ford desde 2003. El sistema genera un archivo Crash Data Retrieval (CDR) que almacena y permite visualizar la información descargada del EDR.

Las áreas de transporte estatales pueden obtener beneficios inmediatos y a largo plazo mediante la recolección de datos de EDRs. El beneficio inicial es el hecho de

que los datos de los EDRs se pueden utilizar para investigaciones de accidentes de tránsito como una nueva y poderosa forma de evidencia en procedimientos legales (demandas por accidentes de tránsito). Por otro lado, con un sistema más metódico de recolección de datos de EDRs, los organismos estatales pueden expandir este beneficio para mejorar significativamente la eficiencia de la recolección de base de datos para estadísticas de accidentes.

Las organizaciones gubernamentales deberán afrontar costos de inicialización y los costos operacionales asociados con la recolección de datos de los EDRs. Los costos de inicialización incluyen la compra de unidades lectoras de EDRs como así también el entrenamiento para los investigadores de accidentes o personal de fuerzas que realizarán las descargas de los EDRs (por ejemplo la Policía Científica). Además, la recolección de datos de EDRs de alguna manera agrega tiempo al requerido para la investigación de un accidente.

Sin embargo, se espera que estos costos en la recolección de datos EDR se den sólo en el corto plazo. A medida que los datos de los EDRs se vuelvan más aceptados por la justicia y se extienda su uso en flotas de vehículos de pasajeros, esta práctica será una más de las realizadas por todo el personal de investigación en general.

Los EDRs evolucionan rápidamente y, en muchas formas, son una tecnología inmadura. Tanto la Society of Automotive Engineers (SAE) como el IEEE hicieron públicos estándares o prácticas recomendadas para EDRs. En 2004, la National Highway Traffic Safety Administration (NHTSA) publicó un anuncio con propuestas de creación de reglas para EDRs instalados voluntariamente en vehículos livianos. Asimismo, el National Cooperative Highway Research Program (NCHRP) ha desarrollado varias recomendaciones para la mejora de estos dispositivos para alcanzar las necesidades específicas del análisis de datos de accidentes de tránsito.

Mientras que los problemas tecnológicos mencionados anteriormente son desafiantes, son también resolubles. Más inciertas son las preocupaciones que aparecen alrededor de la legalidad y aceptación del público sobre la recolección de datos del EDR. El problema principal es el de aceptar que el EDR grabe datos previos al accidente, debido a que eso podría implicar al conductor como responsable del mismo.

Capítulo II – Elementos y Herramientas a utilizar

Event Data Recorders

1. Conceptos

El concepto de reconstrucción de accidentes abarca un amplio rango de factores viales, del vehículo y humanos, y datos grabados aplicado a varios aspectos de estas áreas. El movimiento del vehículo se describe principalmente por los parámetros de aceleración y velocidad, y también por otros factores como las acciones sobre el freno o el volante entre otros. Se considera de relevancia además si existen datos sobre la utilización de sistemas de protección en caso de accidentes, como los airbags y cinturones de seguridad.

Un gran conjunto de términos se utiliza para describir los dispositivos que graban datos digitales, tanto como "caja negra", "grabador de accidentes", "desgrabador de eventos", etc. El acrónimo EDR (del inglés Event Data Recorder), que en castellano se denomina *dispositivo desgrabador de eventos,* pareciera haberse establecido como un nombre popularmente aceptado para este propósito. Se entiende por EDR al dispositivo o sistema capaz de capturar datos digitales obtenidos de sensores en los vehículos. Esta es una definición funcional: mientras que el vehículo sea capaz de grabar datos, no se hacen suposiciones específicas acerca de los componentes físicos utilizados en el dispositivo o sistema.

En la práctica, existen dos categorías de EDRs que se pueden distinguir debido a sus diferentes características y capacidades: de producción o de postventa. Un EDR de postventa es un dispositivo independiente diseñado y colocado en un vehículo con el propósito específico de medir el pulso de aceleración y otros datos durante un impacto. VDO [VDO10] y Davis Instruments [DAV10] entre otras empresas son fabricantes de este tipo de EDRs. Por otro lado, cuando un sistema intrínseco del vehículo, como el airbag, es capaz de grabar datos relacionados con los accidentes, se

los llama EDRs de producción. En términos generales, los EDRs de producción han emergido como modificaciones o mejoras de los sistemas del vehículo que los fabricantes han instalado por diferentes razones, tales como la inyección electrónica, frenos Anti-Lock Brake System (ABS), airbags o controles crucero.

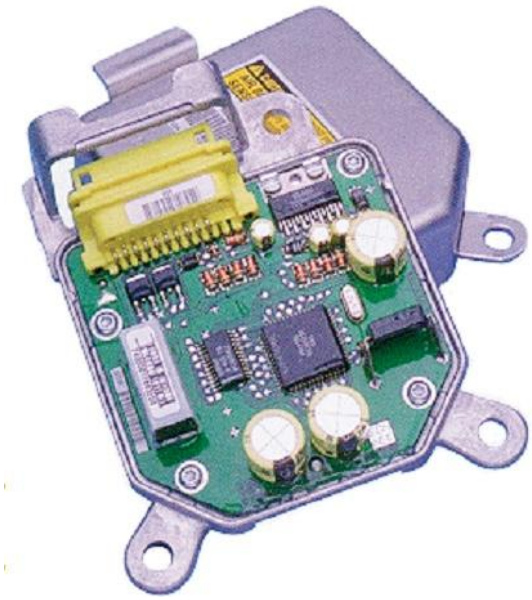

Figura 1: Event Data Recorder

2. Caja negra

El término *caja negra* es un término comúnmente utilizado para referirse a una colección de varios dispositivos de grabación utilizados en transporte: los Flight Recorders (desglosados en Flight Data Recorder y Cockpit Voice Recorder), los Event Recorders (en locomotoras) y los Event Data Recorders en automóviles.

Este término también se utiliza en física y electrónica para describir un mecanismo en el cual la entrada y la salida esperada son comprendidas pero las operaciones internas son deliberada y completamente desconocidas.

3. Origen

Una de las primeras pruebas se realizó a finales de la década del '30, cuando en un centro de pruebas de aviones se tomaron un conjunto de fotografías para grabar un vuelo.

Casi 10 años después se patentó esta invención, conocida como el "hussenógrafo". En 1953, un ingeniero australiano concibió un dispositivo que grabara no solo la lectura de instrumentos sino también las voces de la cabina de mando. Luego de un accidente importante en 1956, las cajas negras comenzaron a instalarse en aviones comerciales. Ya a mediados de 1965, estas se rediseñaron y trasladaron a la parte trasera de los aviones para mejorar la probabilidad de recuperación de datos en caso de accidentes.

4. Rol del EDR

El rol del Motor Vehicle Event Data Recorder (MVEDR) es el de capturar y almacenar datos que no pueden determinarse con precisión utilizando otros medios. Por ejemplo, los datos del MVEDR pueden ser utilizados para determinar elementos críticos específicos a una investigación, tales como el nivel de explosión de un airbag, el que no se puede determinar cuantitativamente de la evidencia del accidente. Puede utilizarse también para mejorar la habilidad de los analistas para determinar la gravedad del accidente y los impactos en los ocupantes del vehículo, agregándole valor al software de reconstrucción de accidentes. Los datos de un EDR pueden llegar a ser invaluables para determinar información sobre el uso y desarrollo de los dispositivos de seguridad y otros sistemas del vehículo, tales como el estado del cinturón de seguridad, información que generalmente se pierde debido a la atención inmediata que requiere un individuo post accidente.

4.1. Perspectivas de usuarios

La información que puede proveer un EDR puede ser aprovechada por diferentes sectores:

1. Área de investigación general de accidentes (Policía Científica)
2. Industria automotriz
3. Servicios de emergencia
4. Área de seguridad y de diseño de infraestructura vial
5. Industria aseguradora

Los datos recuperados se utilizan para corroborar evidencias físicas con los datos de la performance del vehículo, para diagnosticar y resolver fallas del vehículo y de sus sistemas, y como una fuente de datos cuando el conductor no se encuentra disponible para ser interrogado.

Los datos también pueden ser entregados a los ingenieros para que realicen simulaciones del software de los vehículos, e incluso para que logren deducir el movimiento de los ocupantes durante el accidente. El proceso de simulación consiste en hacer equivaler la evidencia física de la escena con los datos del EDR, comenzando con datos conocidos y luego utilizando cálculos de ingeniería y física para determinar otros parámetros críticos. El proceso de simulación comienza al construir una escena tridimensional del accidente, incluyendo el camino hecho en un programa Computer Aided Design (CAD), y luego se construye el vehículo, comenzando con las dimensiones físicas y su motor. Se asignan parámetros de rigidez y se resuelven ecuaciones de movimiento en función del tiempo. Uno de los productos de la simulación es el pulso del accidente (la historia de aceleración en el tiempo del vehículo).

El software puede luego utilizarse para simular a los ocupantes del vehículo. Esto comienza con una definición plana o elipsoidal de las superficies de contacto definiendo el espacio que rodea al ocupante. El ocupante es un modelo humano que comienza en equilibrio con el vehículo. Las interacciones con las superficies de contacto se definen para cada ocupante. El pulso del accidente se utiliza para "conducir" al vehículo a través del accidente. A medida que el vehículo acelera o desacelera, el ocupante tiene una diferencia de velocidad con el vehículo que define las fuerzas de impacto y por tanto los niveles de heridas. Se pueden tomar análisis paramétricos basados en suposiciones de rigidez diferencial de los asientos, cinturones de seguridad, etc. Desafortunadamente, se necesitan muchas suposiciones acerca del vehículo y los ocupantes. Aquí es donde el EDR entra en acción. Trabajar con datos más específicos acerca de la dinámica, aceleraciones y movimientos del vehículo permitiría una evaluación más profunda del movimiento del ocupante y, en el largo o mediano plazo, permitir a los diseñadores desarrollar vehículos más seguros.

Hoy en día, la mayor parte de la protección del ocupante está diseñada en base a los resultados de pruebas de accidentes conocidas como *crash tests*. Ciertas

investigaciones se basan en determinar escenarios del mundo real y patrones de daños resultantes, pero la dinámica de los vehículos no es comúnmente conocida. Los datos del pulso de accidente permitirían a los diseñadores e ingenieros desarrollar mejoras de seguridad basadas en el ambiente del accidente real, en vez de ser sólo una configuración de laboratorio. La corroboración de otra evidencia sería una ventaja clave del uso de los datos del EDR. A pesar de que los datos grabados nunca se usan en forma aislada, la información de un EDR podría afectar el foco de una investigación. Usar datos de EDRs para diagnóstico y resolución de problemas es una tecnología emergente. A medida que los sistemas de los vehículos se vuelven más complejos e interconectados, hay una preocupación incremental de que las condiciones no anticipadas o combinación de eventos puedan tener consecuencias imprevistas.

Los potenciales usuarios de los sistemas de respuesta médica en caso de emergencia que podrían incorporarse a los EDRs son el personal médico que acude a la escena del accidente, que evalúa si las víctimas poseen heridas, estiman la gravedad del accidente, tratan a las víctimas y deciden en el momento dónde y cómo pueden transportar las víctimas. En caso de poseer un EDR, la gravedad del accidente se podría determinar casi instantáneamente, guiando a la respuesta médica para minimizar los tiempos y decisiones importantes a tomarse en la escena del accidente.

5. Matriz de Haddon

En la década del 60, los teóricos de la época comenzaron a tener enfoques más amplios sobre la seguridad vial. Una de las visiones más influyentes fue la de William Haddon Jr., físico y editor de *Accident Research* a mediados de 1960. Este pionero creía que la seguridad vial no se basaba sólo en la educación a los conductores, sino también en la integración de elementos de seguridad activa y pasiva, así como también en leyes viales que la refuercen. Uno de los pilares de esta visión era que las medidas de seguridad vial debían ser sujetas a exámenes científicos.

Haddon desarrolló en la década de 1970 una matriz de 3 x 3 conocida como "Matriz de Haddon" (Fig. 2) que se utiliza generalmente para describir la información acerca de los accidentes automovilísticos y poder entenderlos. Esto proviene de tres fuentes de información: el humano, el vehículo y el ambiente. La matriz de Haddon

propone que los elementos de la tríada epidemiológica deben considerarse en unísono con la secuencia del accidente.

	Humano	Vehículo	Ambiente
Pre-Accidente	recolecciones *subjetivas* por las partes/testigos de observaciones y acciones		evidencia vial post-accidente que indica comportamiento del conductor pre-accidente
Accidente		deformación del vehículo post-accidente que sugiere parámetros generales del accidente	
Post-Accidente	evaluación post-accidente de los daños que *pueden* indicar posición, acciones y utilización de cinturón de seguridad del ocupante	deformación del vehículo post-accidente que *puede* sugerir parámetros generales del accidente	se inspecciona, documenta y analiza evidencia vial post-accidente

Figura 2: Matriz de Haddon

Toda la información recolectada se basa en la evaluación investigativa individual o de un equipo idóneo. En el caso de recolecciones humanas subjetivas, pueden adoptarse como confiables para algunos propósitos.

5.1. Modificación de la matriz

El IEEE en su estándar 1616 propone la utilización de una matriz Haddon modificada. En ésta, se agrega una columna denominada "Invariante en el tiempo" para denotar ítems que no poseen dependencia temporal. La utilización de esta matriz ilustra cómo la secuencia del accidente interactúa con los factores humanos, de ambiente y vehiculares para definir la frecuencia y gravedad de los daños. En este enfoque, el ambiente incluye entidades fuera del par humano-vehículo que pueden haber influenciado tanto las acciones que llevaron al accidente como al resultado del mismo.

5.2. División de datos recolectados

Los datos del accidente que se deben recolectar y almacenar pueden clasificarse según la siguiente división de acuerdo a su frecuencia:

- Datos de alta frecuencia: tienen una tasa de muestreo de 1000 hz o mayor
- Datos de baja frecuencia: tienen una tasa de muestreo de 1 hz o mayor
- Datos estáticos: dado que no tienen resolución temporal y deben almacenarse una vez por evento

5.3. Categorías basadas en la ubicación

La ubicación u origen de la información puede utilizarse para realizar la siguiente clasificación:

- Variables humanas: identificadores descriptivos que se aplican especialmente a humanos involucrados en el incidente. Típicamente incluyen a las víctimas de los accidentes, ocupantes de vehículo y no ocupantes (peatones). Esto puede ser una medida cuantitativa como por ejemplo el peso, o basada en el sistema legal (estado de la licencia de conducir). Las variables humanas también pueden incluir medidas asociadas con la integridad física de las personas luego del accidente, como por ejemplo severidad de las heridas.

- Variables del vehículo: En esta categoría se incluyen identificadores descriptivos de los vehículos involucrados en el accidente (como tipo de carrocería del vehículo, modelo, año de fabricación) y respuesta del vehículo al accidente como información sobre si se disparó el airbag.
- Variables del ambiente: identificadores descriptivos que se aplican específicamente a los alrededores de la ubicación del accidente, tales como el estado del tiempo y la curvatura del camino.

Adquisición de datos distribuidos

1. Definiciones

La adquisición de datos es el proceso mediante el cual fenómenos físicos o magnitudes del mundo real son transformados en señales eléctricas que son digitalizadas para luego ser analizadas, procesadas, y almacenadas por una computadora o sistema.

En la mayoría de las aplicaciones, los sistemas están diseñados no sólo para adquirir datos del entorno sino también para actuar sobre el mismo. Por lo tanto, es útil extender la definición para abarcar los aspectos de control del sistema.

Un sistema de adquisición de datos y control es aquel que monitorea determinados parámetros de su entorno y, en base a su análisis, realiza tareas de control mediante actuadores, relés, motores u otros dispositivos electromecánicos.

Finalmente, un sistema de adquisición de datos distribuidos consiste en una red de sensores y actuadores regida por alguna aplicación de control.

2. Composición de un sistema de adquisición de datos

Los elementos básicos de un sistema de adquisición de datos son:

- Sensores y transductores
- Cableado
- Acondicionadores de señal
- Hardware de adquisición de datos
- Sistema operativo
- Software de adquisición de datos
- Computadora Principal
- E/S distribuida

2.1. Sensores y transductores

Los sensores y transductores son la parte del sistema que está en contacto con el mundo real. Su función es convertir fenómenos físicos en señales eléctricas que el hardware de adquisición de datos o los acondicionadores de señal puedan aceptar [PAR03].

Actualmente existen transductores capaces de medir casi cualquier magnitud física y proveer la correspondiente señal eléctrica. Por ejemplo, una termocupla convierte temperatura en una señal analógica, mientras que un caudalímetro envía pulsos digitales a una frecuencia proporcional a la velocidad del caudal [LEF02].

2.2. Cableado

El cableado representa la conexión física entre los sensores o transductores y los acondicionadores de señal o el hardware de adquisición de datos. Dado que es el componente que más se extiende en el sistema, es el más susceptible a los ruidos e interferencias del entorno. Por esta razón en la mayoría de los sistemas es necesario que los cables posean cierta aislación para disminuir los efectos de agentes externos.

2.3. Acondicionadores de señal

Generalmente, las señales eléctricas generadas por los transductores deben ser llevadas a una forma que sea aceptable para el conversor analógico/digital a utilizarse de la señal a formato digital.

Los acondicionadores de señal son los encargados de realizar el tratamiento de la señal, a través de las tareas de filtrado, amplificación, linealización, aislamiento y excitación.

2.4. Hardware de adquisición de datos

El hardware de adquisición de datos y control se define como aquel componente de un sistema que realiza alguna de las siguientes funciones:

- Recepción, procesamiento y conversión a formato digital a través de Analog to Digital Converter (ADC) de las señales analógicas medidas de un sistema o proceso
- Recepción de señales digitales que contienen información sobre un sistema o proceso

- Procesamiento y conversión a formato analógico a través de Digital to Analog Converter (DAC) de las señales digitales del procesador
- Salida de señales digitales de control

Estos dispositivos pueden presentarse de distintas maneras, desde tarjetas de expansión que se insertan directamente en las ranuras de expansión de una PC, hasta dispositivos inteligentes programables que pueden operar independientemente de una computadora.

2.5. Software de adquisición de datos

El hardware de adquisición de datos no puede trabajar sin software, ya que éste es el que introduce la lógica de todo el sistema.

El software de adquisición comprende todos los algoritmos que, dada la información tomada o medida por el hardware de adquisición, determinan el comportamiento y el output del sistema.

Existen tres opciones para programar un sistema de hardware:

- Programar directamente los registros del hardware de adquisición directamente
- Utilizar controladores (drivers) de bajo nivel provistos con el hardware para desarrollar el software de aplicación para las tareas requeridas
- Utilizar aplicaciones de software provistas con el hardware que realizan tareas para una aplicación particular.

2.6. Computadora Principal

La computadora usada en un sistema de adquisición de datos afecta directamente las velocidades a la que los datos pueden ser continuamente adquiridos, procesados y almacenados para una determinada aplicación.

De acuerdo a los requerimientos y a la semántica de la aplicación, la velocidad del procesador, la velocidad de acceso y la capacidad de los dispositivos de

almacenamiento, y los tipos de transferencia disponibles tendrán cierto impacto en el funcionamiento y performance del sistema.

2.7. E/S Distribuida

Generalmente los sensores de un sistema de adquisición de datos se encuentran en una ubicación remota con respecto a la computadora encargada del procesamiento y almacenamiento de los datos. Consecuentemente, es difícil que las señales de sensores pequeños como termocuplas ubicados a varios metros de la unidad de procesamiento sobrevivan la transmisión sin perder precisión.

Para subsanar este problema, se utiliza E/S distribuida, que consiste en colocar los acondicionadores de señal junto a los sensores cuyas señales se deben transformar. Se requiere un módulo por cada sensor, permitiendo un alto grado de modularización del sistema.

Una de las formas de E/S distribuida más comúnmente implementadas consiste en utilizar un conjunto de transmisores digitales, es decir dispositivos encargados de las funciones de acondicionamiento de señal que poseen un microcontrolador y un ADC. Estos dispositivos están conectados a un bus a través del cual envían la información a la computadora.

De esta forma, además, se reduce la cantidad de cableado ya que todos los transmisores se encuentran conectados al mismo par de cables.

3. Sensores y actuadores

La mayoría de los parámetros monitoreados por un sistema son analógicos, es decir, los valores que pueden tomar están en rangos continuos. Estos parámetros pueden ser por ejemplo temperatura, presión, fuerza, velocidad, etc. Para que puedan ser capturados por un sistema de adquisición de datos, deben ser primeramente convertidos a magnitudes eléctricas tales como voltaje, corriente o impedancia.

Un sensor o transductor es un dispositivo que convierte una magnitud física en otra. En el contexto de los sistemas de adquisición de datos, la conversión de los sensores es hacia señales eléctricas que podrán luego ser tomadas por conversores para adecuarlas al formato digital del hardware del sistema.

Las señales eléctricas producidas por los transductores serán proporcionales al parámetro que está siendo medido de acuerdo con una relación preestablecida.

4. Conversor Analógico/Digital

Dado que los sensores generan señales eléctricas analógicas y que el hardware del sistema trabaja con señales digitales, es necesario realizar una conversión de analógico a digital. Esta tarea es realizada por un ADC.

Un conversor analógico digital toma muestras periódicas de la señal analógica, mide el nivel de cada una y traduce su valor al código binario.

5. Separadores y E/S aislada

La técnica de separación (buffering) permite desacoplar el software de monitoreo del hardware de adquisición de datos.

A través de un almacenamiento temporario para los datos recibidos, se compensan las latencias del software

De esta forma se puede garantizar una determinada tasa de recepción de mensajes

5.1. Hardware aislado

Muchos dispositivos tienen una capacidad propia de aislamiento limitada. Los datos recibidos pueden entonces ser colocados en un buffer de hardware intermedio a una mayor velocidad de la que el dispositivo sería capaz de operar.

Algunos dispositivos permiten el acceso a los buffers de memoria al mismo tiempo que nuevos valores están siendo escritos.

La principal ventaja del hardware buffering es que el sistema no se ve limitado por los tiempos de respuesta de los componentes de software.

5.2. Software aislado

Los componentes de software pueden poseer también sus propios separadores (buffers) de memoria. Estos *buffers* proveen un soporte para el procesamiento de los datos recibidos. Diferentes tareas, hilos o procesos pueden compartir el acceso a ellos.

Los buffers de software más utilizados son los que se basan en estructuras FIFO o LIFO. En los sistemas de adquisición de datos, los buffers FIFO son esenciales para facilitar la interacción entre procesos asincrónicos. Por su semántica, se garantiza que los datos son leídos del buffer en el mismo orden en el que fueron escritos.

5.3. Buses externos

Todo sistema de adquisición de datos posee al menos un bus de comunicación externo a través del cual se interconecta con su entorno.

Estos buses se dividen en buses paralelos y buses seriales.

Los buses paralelos poseen una señal separada para cada bit. De esta forma, un *byte*, *word* o *dword* puede ser transmitida en una sola operación, permitiendo potencialmente una mayor tasa de transferencia. La mayoría de los buses paralelos trabajan sincrónicamente, es decir que una de las señales es utilizada para la sincronización de la transmisión y recepción de datos.

Por esta razón, los buses paralelos podrían potencialmente alcanzar una velocidad mayor a la de los buses seriales. Sin embargo, como están diseñados para ser implementados utilizando cables relativamente cortos y dado que para establecer una interconexión paralela son necesarias varias líneas, en la mayoría de los sistemas de tiempo real se utilizan comunicaciones seriales para tener una mayor flexibilidad con respecto a las distancias entre los dispositivos y para reducir la cantidad de cables en el sistema.

En los buses seriales, por otro lado, los datos son descompuestos en series de bits que son transmitidos uno a uno por una sola línea de comunicación (dos cables). Esto no sólo reduce la cantidad de metros de cable para realizar una conexión sino que también permite que los datos sean transferidos a mayores distancias.

Los buses pueden clasificarse según la dirección en la que pueden transferirse los datos. Si el bus es *simplex*, los datos pueden ser transmitidos sólo en una dirección. Por el contrario, en un bus *half-duplex*, los datos pueden fluir en ambas direcciones, pero no en las dos al mismo tiempo. Finalmente, en un sistema *full-duplex*, los datos pueden fluir en ambas direcciones simultáneamente.

5.4. Velocidad de transmisión de datos

La máxima tasa de transferencia alcanzable depende de un conjunto de factores:

- Tipo y complejidad de la interface de los emisores y receptores
- Tipo de enlace (coaxil, par trenzado, etc)
- Distancia entre emisores y receptores
- Cantidad de datos transferidos
- Sobrecosto (overhead) asociado con la transferencia de datos
- Tasa de error aceptable

La velocidad de transferencia es medida en bits por segundo (bps). Esta medida se refiere sólo a los bits de datos transferidos, es decir que no son tenidos en cuenta los bits de exceso o sobrecosto (overhead) de la comunicación.

Simulaciones

1. Definiciones

Las simulaciones por computadoras se pueden definir en términos generales como: *la utilización de una computadora para imitar las operaciones de los procesos o facilidades del mundo real de acuerdo a asunciones desarrolladas apropiadamente tomando la forma de relaciones lógicas, estadísticas o matemáticas que se hayan desarrollado y formado en un modelo* [MCH09].

Estas técnicas para imitar las operaciones de procesos o instalaciones del mundo real generalmente son llamados *sistemas*, y para estudiarlos científicamente generalmente se realizan suposiciones sobre su funcionamiento. Estas suposiciones, las que normalmente toman forma de relaciones lógicas o matemáticas, constituyen el *modelo* utilizado para intentar comprender cómo se comporta el sistema [LAW06].

Si las relaciones que componen el modelo son lo suficientemente simples, puede ser posible la utilización de métodos matemáticos para obtener información *exacta* sobre preguntas de interés; esto es lo que se denomina solución *analítica*. Sin embargo, la mayoría de los sistemas del mundo real son demasiado complejos como para permitir que los modelos realísticos sean evaluados analíticamente, y estos modelos deben ser estudiados mediante simulaciones. En una *simulación* se utilizan computadoras para evaluar un modelo *numérico*, y los datos son recolectados para estimar las características deseadas del modelo.

Un *sistema* se define como una colección de entidades (como personas o máquinas), que actúan e interactúan en conjunto para lograr un final lógico. En la práctica, la definición de "el sistema" depende de los objetivos de un estudio particular. La colección de entidades que comprenden un sistema para un estudio puede ser sólo un subconjunto de la totalidad de otro sistema. El *estado* de un sistema

se define como una colección de variables necesarias para describir un sistema en un momento particular, relativo a los objetivos de un estudio.

Los sistemas se categorizan en dos tipos, discretos y continuos. Un sistema *discreto* es uno para el cual las variables de estado cambian instantáneamente en puntos separados del tiempo. Un sistema *continuo* es aquel para el cual las variables de estado cambian continuamente respecto al tiempo. En la práctica, pocos sistemas son enteramente discretos o enteramente continuos; pero dado que un tipo de cambio predomina la mayoría de los sistemas, generalmente es posible clasificar a un sistema como discreto o continuo.

En nuestro caso, se puede definir como continuas las variables de velocidad del automóvil y como discreta al estado del cinturón de seguridad (abrochado o desabrochado).

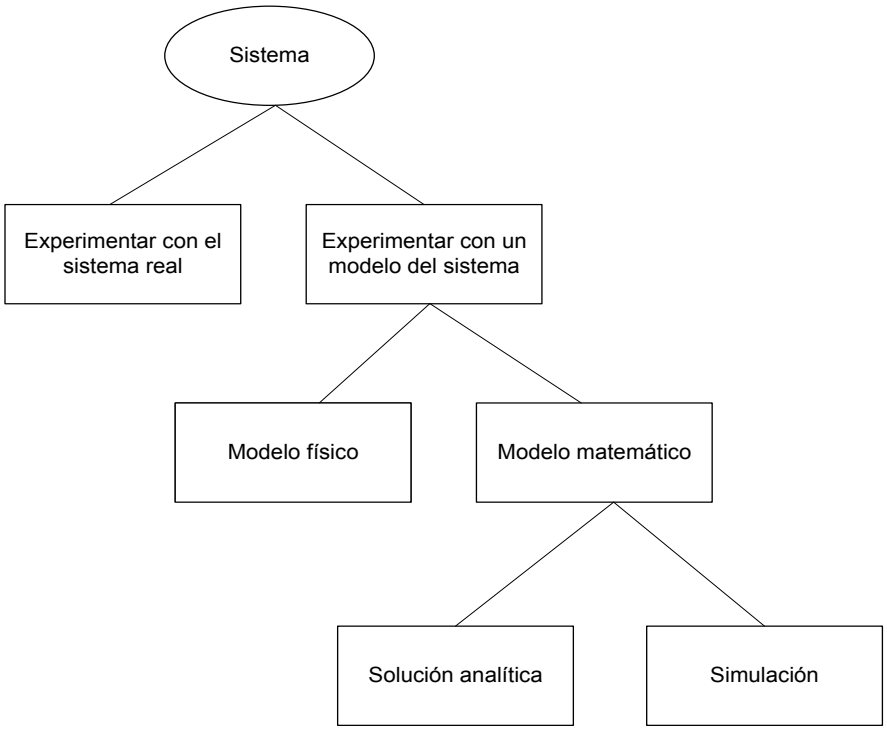

Figura 3: Formas de estudiar un sistema

Entre la elección de soluciones, podemos encontrar las diferentes decisiones [LAW06]:

- Experimentar con el sistema real vs Experimentar con un Modelo del Sistema. Si es posible alterar el sistema físicamente y luego dejarlo operar bajo nuevas condiciones, es deseable realizarlo. Sin embargo, esto raramente es factible, debido a que tal experimento será demasiado costoso o perjudicial para el sistema. También se debe considerar el caso en el que el sistema no exista, y se quieran plantear alternativas de configuración para ver como debería construirse en primera instancia. En estos casos, generalmente es necesario construir un modelo como una representación del sistema y estudiarlo como un sustituto del sistema real. Al utilizar un modelo, siempre existe el interrogante de qué tan precisamente refleja el sistema para los propósitos de las decisiones a realizarse; esto se lo refiere como *validez* del sistema.
- Modelo físico vs Modelo Matemático. Un modelo físico (también llamado icónico) es aquel que representa un sistema utilizando objetos (como las pruebas aerodinámicas de los automóviles) y un modelo matemático representa un sistema en términos de relaciones lógicas y cuantitativas que son manipuladas y cambiadas para ver cómo reacciona el modelo, y además cómo reaccionaría -siempre y cuando el modelo sea válido.
- Solución Analítica vs Simulación. Una vez construido el modelo matemático, debe examinarse cómo puede ser utilizado para responder los interrogantes de interés sobre el sistema que se supone representa. Si el modelo es lo suficientemente simple, puede ser posible trabajar con las relaciones y cantidades para obtener una solución analítica exacta. Algunas soluciones analíticas pueden volverse muy complejas, tornando casi como única opción la simulación.

Los modelos de simulación pueden clasificarse comúnmente de tres maneras:

- Modelos de Simulación Estáticos vs Dinámicos. Un modelo de simulación estático es una representación de un sistema en un momento determinado, o uno que puede utilizarse para representar un sistema en el cual el tiempo simplemente no juega un rol. Por otro lado, un modelo de simulación dinámico representa un sistema a medida que evoluciona en el tiempo.
- Modelos de Simulación Determinísticos vs Estocásticos. Si un modelo no contiene ningún componente probabilístico (aleatorio), se lo denomina determinístico. En los modelos determinísticos, la salida se "determina" una vez que el conjunto de cantidades de entrada y las relaciones del modelo hayan sido especificadas. Sin embargo, muchos sistemas deben ser modelados teniendo algunos componentes de entrada aleatorios, y esto da lugar a los modelos de simulación estocásticos. Los modelos de simulación estocásticos producen salidas en si misma aleatorias, y deben entonces tratarse como un estimado de las caractacterísticas reales del modelo; esta es una de las desventajas principales de la simulación.
- Modelos de Simulación Continuo vs Discreto. Un modelo discreto no siempre es usado para modelar un sistema discreto, y viceversa. La decisión de usar un modelo discreto o continuo para un sistema particular depende de los objetivos específicos de estudio.

Nuestro modelo puede considerarse como discreto, dinámico y estocástico y por ende se lo denomina *modelo de simulación de eventos discretos.*

2. Simulaciones de eventos discretos

Las *simulaciones de eventos discretos* abarcan el modelado de un sistema a medida que evoluciona en el tiempo mediante una representación en la cual las variables de estado cambian instantáneamente en distintos instantes de tiempo. Estos puntos de tiempo son aquellos en los que ocurre un evento. Un evento se define como

una ocurrencia instantánea que puede cambiar el estado del sistema. A pesar de que la simulación discreta de eventos puede hacerse conceptualmente por cálculos a mano, la cantidad de datos que debe ser almacenada y manipulada para la mayoría de los sistemas reales establece que las simulaciones de eventos discretos sea en una computadora.

2.1. Mecanismos de avance de tiempo

Debido a la naturaleza dinámica de los modelos de simulación de eventos discretos, debemos estar al tanto de los valores actuales del tiempo simulado a medida que la simulación continúa su camino, y también necesitamos un mecanismo para avanzar el tiempo simulado de un valor a otro. La variable en el modelo de simulación que entrega el valor actual del tiempo simulado se la denomina *reloj de simulación.* La unidad de tiempo para el reloj de simulación nunca se declara explícitamente cuando un modelo se escribe en un lenguaje de propósito general, y se asume que está en las mismas unidades que los parámetros de entrada. Además, generalmente no hay relación entre el tiempo simulado y el tiempo necesario para correr una simulación en la computadora.

Históricamente, dos enfoques principales han sido sugeridos para avanzar el reloj de simulación: *avance de tiempo del próximo evento* y *avance del tiempo de incremento fijo*. El primer enfoque es utilizado por la gran mayoría del software de simulación y la mayor cantidad de personas que codifican el modelo en un lenguaje de propósito general, y el segundo es un caso especial del primero.

Con el enfoque del avance de tiempo al próximo evento, el reloj de simulación es inicializado en cero y se determinan los tiempos de ocurrencia de futuros eventos. El reloj de simulación entonces avanza al tiempo de la ocurrencia *más inminente* (en este caso, la primera) de los eventos futuros, tal que el estado del sistema se actualiza a cuenta del hecho que un evento haya ocurrido y que el conocimiento de los tiempos de ocurrencia de futuros eventos también se haya actualizado. Luego, el reloj de simulación avanza al tiempo del nuevo evento más inminente, el estado del sistema se actualiza y se determinan los tiempos de eventos futuros, etc. El proceso de avanzar el reloj de un tiempo de evento a otro continúa hasta la eventual condición de finalización pre-especificada. Dado que todos los cambios de estado ocurren sólo en

tiempos de eventos para un modelo de simulación de eventos discretos, los periodos de inactividad se saltean avanzando el reloj de evento en evento (a diferencia de los avances de tiempo fijo, que pueden consumir mucho tiempo de procesamiento). Los saltos sucesivos del reloj de simulación generalmente son variables (no son iguales) en tamaño.

2.2. Componentes y organización de un modelo de simulación de eventos discretos

Los siguientes componentes se encuentran en la mayoría de los modelos de simulación de eventos discretos utilizando el enfoque de avance de tiempo en un lenguaje de propósito general:

- *Estado del sistema*: colección de variables de estado necesarias para describir el sistema en un momento en particular
- *Reloj de simulación*: una variable que contiene el valor actual del tiempo simulado
- *Contadores estadísticos:* variables utilizadas para almacenar información estadística sobre la performance del sistema
- *Rutina de inicialización:* subprograma que inicializa el modelo de simulación en el tiempo 0
- *Rutina de evento:* subprograma que actualiza el estado del sistema cuando un tipo particular de evento ocurre (existe una rutina de evento para cada tipo de evento)
- *Rutinas de librería:* conjunto de subprogramas utilizados para generar observaciones aleatorias de distribuciones de probabilidad que fueron determinadas como parte del modelo de simulación
- *Generador de reportes:* subprograma que computa estimados de las medidas deseadas de performance y producen un reporte cuando la simulación finaliza
- *Programa principal:* subprograma que invoca la rutina de tiempo para determinar el próximo evento y luego transfiere el control a la rutina del evento correspondiente para actualizar el estado del

sistema apropiadamente. El programa principal puede también comprobar finalización e invocar al generador de reportes cuando la simulación finalice.

El enfoque de modelado de simulación con avance por eventos se denomina programación de eventos, dado que los tiempos de los futuros eventos son codificados explícitamente en el modelo y planificados a ocurrir en el futuro simulado.

3. Construcción del modelo

Uno puede distinguir los siguientes pasos en la construcción de un modelo de simulación:

1. *Análisis del problema y recolección de información*. El primer paso en la construcción de un modelo de simulación es el de analizar el problema. Para facilitar una solución, el análisis primero debe recolectar información estructural que afecta al problema, y representarla convenientemente. Esta actividad incluye la identificación de parámetros de entrada, medidas de performance de interés, relaciones entre parámetros y variables, etc.
2. *Recolección de datos*: esta actividad es necesaria para estimar parámetros de entrada del modelo. Pueden formularse suposiciones sobre la distribución de variables aleatorias en el modelo. En caso de ausencia de datos, pueden designarse rangos de parámetros y simular el modelo con rangos acotados. Esta tarea también es necesaria para la validación del modelo (la salida del sistema se compara con las reales).
3. *Construcción del modelo*. Una vez que el problema se encuentra estudiado y los datos sean recolectados, se procede a la construcción de un modelo e implementación del mismo en un programa de computadora.
4. *Verificación del modelo*. El propósito de esta actividad es el de asegurarse que el modelo esté correctamente construido. Esto significa que el modelo conforma las especificaciones y realiza lo que se supone

que debe realizar. La verificación se realiza mediante inspecciones de código y comparándolo con la especificación del modelo.

5. *Validación del modelo*. Cada modelo debe verse inicialmente como una propuesta, sujeta a validación. La validación del modelo examina si el modelo se adecua a los datos empíricos (medidas de sistemas reales). Este tipo de validación sólo puede lograrse si existe algún sistema real comparable.
6. *Diseño y ejecución de experimentos de simulación*. Una vez que el modelo sea válido, se procede a diseñar un conjunto de experimentos de simulación para estimar la performance del modelo y ayudar a resolver el problema del proyecto. Una forma de obtener medidas estadísticas confiables, es replicar cada escenario y promediar los resultados
7. *Análisis de datos de salida*. Las medidas de la performance estimada están sujetas a un análisis lógico y estadístico.
8. *Recomendaciones finales*. Se utilizan los análisis de los datos de salida para formular las recomendaciones finales para los problemas del sistema subyacente. Generalmente esto toma forma de reporte.

4. Sistemas de tiempo real

4.1. Definición

Un sistema de tiempo real es un sistema de software que debe producir el resultado esperado en un determinado punto en el tiempo físico. Este punto es determinado por la naturaleza del problema para el cual el sistema fue construido [KOP00].

Un sistema de tiempo real, entonces, debe proveer no sólo la salida esperada de acuerdo a la entrada, sino que también debe respetar determinadas restricciones temporales para satisfacer los requerimientos del problema.

4.2. Características

Temporizado (*Timing*). Como se ha mencionado en la definición, los sistemas de tiempo real tienen requisitos específicos con respecto al comportamiento temporal.

Estas restricciones no implican necesariamente que las respuestas del sistema deban ser inmediatas. Por ejemplo, un semáforo que encienda la luz verde antes de tiempo representaría una respuesta incorrecta del sistema. Las limitaciones de tiempo pueden clasificarse como de prestaciones (performance) o de comportamiento. Las limitaciones de performance son aquellas impuestas sobre la respuesta del sistema. Las limitaciones de comportamiento son aquellas impuestas sobre los estímulos generados por el ambiente. Cada limitación puede clasificarse como:

- Límite de retraso: captura el tiempo mínimo que debe transcurrir entre la ocurrencia de dos eventos arbitrarios
- Límite de tiempo límite: captura la separación máxima permitida de tiempo que debe ocurrir entre dos eventos arbitrarios
- Limitación de duración: especifica el período de tiempo sobre el que actúa un evento

Respuesta ante eventos. Un evento es algo que ocurre en un determinado punto en el tiempo. Los sistemas de tiempo real conducen su flujo de ejecución en base a la ocurrencia de eventos. Estos se manifiestan, por ejemplo, como interrupciones disparadas por la llegada de datos a un puerto de entrada o como los tics de un reloj de hardware.

Hardware ad-hoc. La mayoría de los sistemas de tiempo real interactúan con hardware específico del dominio del problema, es decir dispositivos eléctricos o electromecánicos. Esto dificulta el proceso de desarrollo debido a que muchas veces estos componentes son prototipos o diseños experimentales con poca certeza acerca de la fiabilidad o funcionalidad de los mismos. Los programadores generalmente deben aprender detalles o particularidades del dominio para poder codificar algunos algoritmos. Por ejemplo, en el caso de un dispositivo que monitoree la temperatura de un contenedor con líquido y regule la fuente de calor en consecuencia, se debe tener en cuenta la inercia térmica al programar el controlador.

Volatilidad de los datos. Muchos de los datos consumidos por un sistema de tiempo real provienen de agentes o dispositivos externos, que se diferencian, por ejemplo, de los que provienen de un archivo en el disco rígido en los sistemas ordinarios. En consecuencia, el software debe estar adecuadamente construido para

recuperar los datos correctamente, respetando velocidades de muestreo y otros parámetros inherentes a la recolección de datos externos.

Concurrencia. Un sistema de tiempo real generalmente debe responder a varios eventos independientes con límites de tiempo estrictos y acotados. Un concepto relacionado con la concurrencia es el término "multitarea", que provee una técnica que permite asistir a los programadores a particionar sus sistemas en componentes razonables con responsabilidades delegadas para llevar a cabo ciertas partes de una actividad completa.

Planificación en tiempo de ejecución. A diferencia de los sistemas de software de procesamiento de datos tradicionales, los sistemas de tiempo real realizan la planificación de tareas en tiempo de ejecución. Esto se logra por medio del sistema operativo o núcleo (kernel) de simulación, que utilizan un planificador para ordenar los eventos. Posiblemente esta sea la característica más interesante de los sistemas de tiempo real.

Imprevisibilidad. Como los sistemas de tiempo real se basan en eventos, están expuestos a cambios impredecibles en el entorno que lo rodea. En la práctica, no es viable anticiparse a todas las posibles permutaciones de situaciones que puedan surgir.

Manejo de excepciones. La mayor parte de los sistemas de tiempo real debe poder trabajar sin intervención de operadores humanos y muchas veces en forma permanente. Por lo tanto, cuando ocurre una falla en ausencia de operadores humanos, el sistema puede intentar tomar acciones correctivas. Sin embargo, dada la naturaleza imprevisible de este tipo de sistemas, se torna dificultoso prever todas las soluciones posibles para los casos de fallo que puedan ocurrir. En el caso de que no se puedan ejecutar acciones inmediatas, es deseable que el sistema sea capaz de detectar las fallas y continuar su funcionamiento en un modo reducido en vez de detenerse abruptamente.

Estabilidad. Los sistemas de tiempo real deben ser capaces de satisfacer las restricciones temporales de las tareas críticas en situaciones de sobrecarga del sistema, aunque para esto deban desatenderse las tareas no críticas. Esto contrasta con el principio de equidad (*fairness*) de los sistemas tradicionales.

Reactividad. Los sistemas de tiempo real usualmente son reactivos. Un sistema reactivo es uno en el cual una interacción siendo llevada a cabo entre la computadora y el ambiente es mantenida. Ante una determinada entrada no se produce una salida, sino que el sistema actúa sobre el entorno y se retroalimenta con el muestreo de la reacción del mismo.

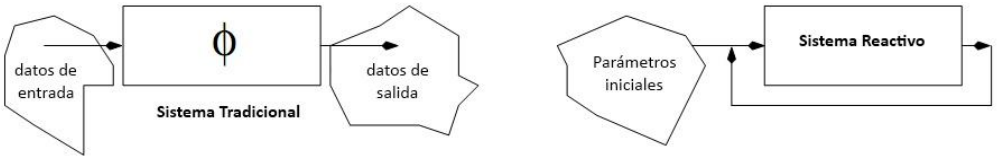

Figura 4: Sistema Tradicional vs Sistema Reactivo

4.3. Sistemas de tiempo real

Se pueden distinguir dos grandes tipos de sistemas de tiempo real: *hard* (o duros) y *soft* (o blandos), dependiendo de cuáles son los requisitos para considerar que el resultado provisto por el sistema es correcto o incorrecto.

En los sistemas de tiempo real *soft*, si no se logra satisfacer la restricción temporal especificada se reduce la utilidad del resultado pero no se genera un perjuicio significativo. Siempre que sea provista la mayoría de los resultados dentro del límite de tiempo, el sistema funcionará correctamente.

Por otro lado, en los sistemas *hard*, si el resultado no es provisto respetando los límites de tiempo establecidos, el mismo es considerado incorrecto y generalmente tiene graves consecuencias en el dominio del problema.

Además del orden de ejecución de los algoritmos, el tiempo de respuesta de un sistema de software depende ampliamente del hardware subyacente, y si bien los avances de la tecnología permiten recursos físicos cada vez más rápidos, esto no debe restar importancia a los demás factores del desarrollo de un sistema. En numerosos casos existen diversos factores económicos, físicos, energéticos, etc. que generan restricciones en la elección del hardware.

Por esto, la ingeniería de software para sistemas de tiempo real debe tener en cuenta un nuevo factor que es la capacidad del hardware sobre el cual se implantará el software.

5. Conceptos de TLM

TLM se basa en el concepto de modelar sólo en nivel de detalle necesario por los equipos desarrolladores de los componentes del sistema y subsistema para una tarea particular [GHE05].

La utilización del TLM junto con el lenguaje de descripción de hardware como SystemC, permite la utilización de un modelo temprano de ejecución del sistema, mejorando tareas de diseño y facilitando trabajos de pruebas con la utilización de simulaciones, sin la necesidad de implementaciones de hardware físico.

Una de las características principales de TLM es la desagregación entre los componentes funcionales y la implementación de hardware, lo que permite un refinamiento independiente de componentes.

Básicamente, podemos distinguir entre tres niveles de refinamiento. *Sin tiempo:* sólo se representa la funcionalidad. Un modelo con comunicación sin tiempo y funcionalidad generalmente se lo refiere como un modelo arquitectónico del sistema. *De tiempo aproximado:* se incluye cierta información básica sobre el tiempo. La duración real es aproximada. *A nivel de ciclos:* los detalles de implementación permiten una simulación con precisión a nivel de ciclos de reloj [BLA04].

Comenzando con un SAM, considerando que este es sin tiempo tanto en funcionalidad como en comunicación, puede derivarse hacia un TLM con comunicación sin tiempo y funcionalidad de tiempo aproximado. Subsecuentemente, los componentes se refinan separadamente hasta que alcancen un nivel de ciclos (un enfoque mucho más preciso).

Un modelo que es de tiempo de ciclos presenta precisión de comunicación y de funcionalidad. Generalmente se lo refiere como modelo de Register Transfer Level (RTL). Este tipo de modelo comúnmente requiere grandes cantidades de tiempo para simularse.

La comunicación en TLM se modela por medio de canales. Los componentes funcionales se comunican entre ellos con transacciones, las que se llevan a cabo mediante la utilización de funciones de interfaces implementadas por los canales.

Esta metodología permite una mayor reutilización de componentes de diseño, no sólo dentro de un proyecto en específico, sino también en proyectos futuros, ocultando detalles de implementación más finos.

SystemC

1. Introducción

SystemC es un lenguaje de descripción de hardware cuyo objetivo es proveer a los diseñadores y arquitectos de hardware un estándar basado en C++ para el desarrollo de sistemas híbridos software-hardware.

Consiste en una librería implementada en C++ que provee los constructores necesarios para modelado de hardware y que permite representar conceptos de tiempo, tipos de datos de hardware, comunicación y concurrencia.

Esta librería permite al usuario escribir un conjunto de funciones C++ que son ejecutadas bajo el control de un scheduler en un orden que imita el paso de tiempo simulado y que se sincronizan y comunican emulando sistemas electrónicos que contienen hardware y software embebido.

Los procesos están encapsulados en una jerarquía de módulos que capturan las relaciones estructurales y la conectividad del sistema. La comunicación entre procesos se lleva a cabo mediante un mecanismo que facilita la abstracción y el refinamiento independiente de interfaces a nivel del sistema.

La utilización de SystemC permite afrontar la complejidad de los diseños de sistemas modernos mediante la utilización de técnicas como abstracción, reutilización de diseño, reutilización de proyecto, etc.

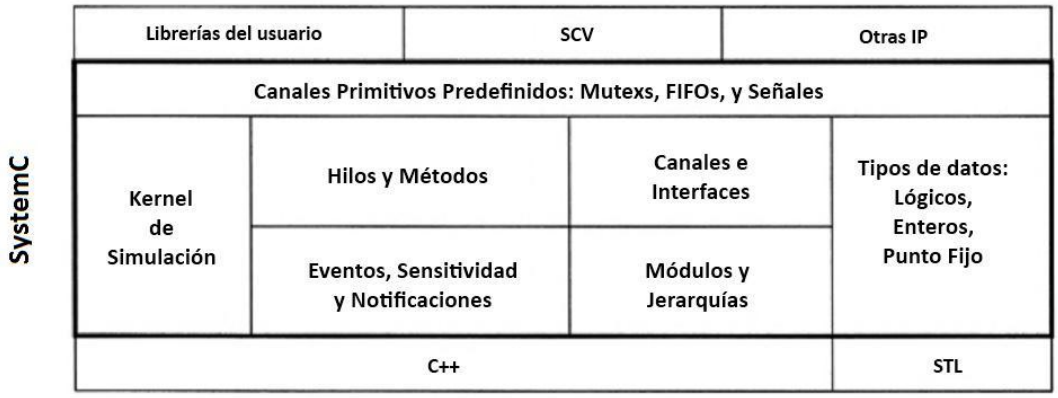

Figura 5: Arquitectura de SystemC

2. Estructura

Un sistema de SystemC consiste en una jerarquía de módulos. Los módulos contienen procesos que se ejecutan concurrentemente. Cada proceso describe funcionalidad, se comunica con otros procesos a través de canales y se sincroniza con otros procesos mediante eventos. La comunicación entre módulos es mediante canales.

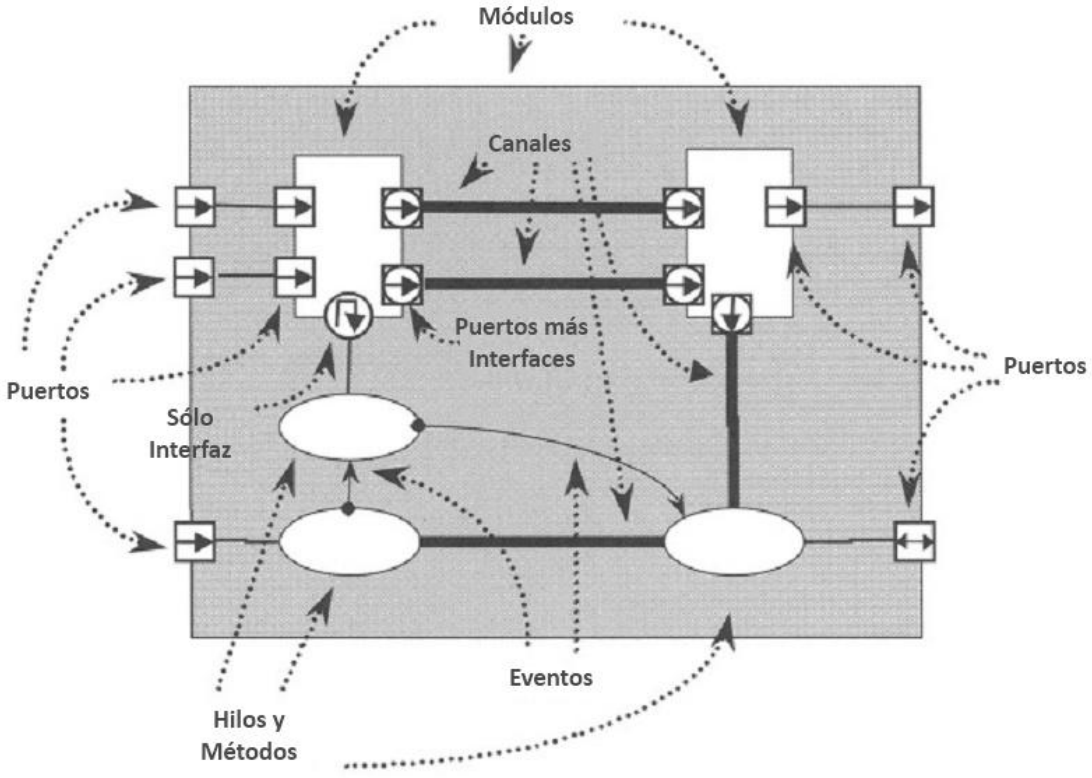

Figura 6: Estructura de SystemC

2.1. Módulos

Los sistemas complejos están compuestos por un conjunto de componentes que funcionan en forma independiente. Estos componentes pueden representar software, hardware u otra entidad física.

En SystemC, el elemento estructural básico para representar estos componentes es el módulo (*sc_module*). Es el mínimo contenedor de funcionalidad con estado, comportamiento y estructura. Puede representar hardware, software o cualquier entidad física.

Los diseñadores pueden de esta manera dividir sistemas complejos en partes más pequeñas. Un módulo puede contener:

- Puertos
- Canales
- Procesos
- Eventos
- Instancias de otros módulos
- Otros datos
- Otras funciones

Los módulos, así como también los puertos, los canales, las interfaces y los eventos están implementados como clases de C++.

Un módulo puede representar un sistema entero, un bloque, un tablero, un chip, etc.

2.2. Procesos

Los procesos son la unidad básica de ejecución en SystemC. Durante la simulación, todo el código que se ejecuta es iniciado a partir de uno o más procesos, que son ejecutados en forma concurrente.

Un proceso SystemC es una función miembro o método de clase de un módulo que es invocado por el programador (o scheduler) del núcleo (o kernel) de simulación. No tiene argumentos ni devuelve ningún valor.

Existen dos tipos de procesos: hilos (*sc_thread*) y métodos (*sc_method*).

Un hilo es igual a un hilo o 'thread' de software tradicional. El kernel de simulación de SystemC es el que permite que varios hilos se ejecuten en paralelo.

Los hilos de SystemC son iniciados sólo una vez por el simulador. Una vez que empieza a ejecutarse tiene el control de la simulación hasta que se lo devuelva al simulador ya sea terminando o bien pasando a estado de espera mediante una sentencia *wait*.

El hilo, para concluir, debe invocar la sentencia *return*. Una vez que lo hace, no vuelve a ejecutarse durante el resto de la simulación. Por eso, generalmente el código contiene un lazo (loop) infinito con al menos una sentencia *wait*.

La invocación a la sentencia *wait* es otra forma de devolver el control al simulador. En ese momento, el proceso se suspende y pasa a estado de espera. Además, el método *wait* también puede ser invocado indirectamente, por ejemplo, al invocar el método *read* o *write* de una cola (*sc_fifo)* si la cola está vacía o llena respectivamente.

Existe además una variación de los hilos sensibles al reloj (sc_*cthread*). Estos hilos facilitan la codificación para los procesos cuya ejecución depende estrictamente de un reloj (*sc_clock*).

Por otra parte, los métodos son similares a los hilos con la excepción de que no pueden ser suspendidos, es decir, se ejecuta todo su cuerpo y una vez finalizado se retorna el control al kernel de simulación.

Cada método se declara sensible a uno o más eventos, y es invocado ante la ocurrencia de uno de esos eventos. La sensibilidad puede ser definida estáticamente, es decir en tiempo de compilación, y modificada dinámicamente en tiempo de ejecución mediante la sentencia *next_trigger (anEvent)*.

2.3. Canales

Generalmente los sistemas cuentan con algún sistema de comunicación entre sus componentes.

En SystemC, la comunicación entre procesos dentro de los módulos y entre módulos se realiza a través de canales. Cada canal debe implementar una interfaz (*sc_interface*) es decir un conjunto de métodos que se utilizarán para accederlo.

Los módulos contienen puertos que se asocian a canales. Cuando un proceso necesita enviar o recibir datos por un canal, realiza una lectura o escritura (generalmente *read* o *write (data)*) sobre el puerto asociado.

Los canales pueden ser jerárquicos (*sc_channel*) o primitivos (*sc_prim_channel*). Los canales primitivos no tienen ninguna estructura visible y no pueden tener acceso directo a otros canales primitivos. Están concebidos para proveer una comunicación rápida y simple.

En contraste, los canales jerárquicos son módulos, por lo que pueden contener procesos, instancias de módulos, puertos y además pueden tener acceso directamente a otros canales jerárquicos.

El propósito de los canales jerárquicos es permitir implementar canales complejos como PCI, CAN Bus, FlexRay, etc.

2.4. Interfaces

Una interfaz es un conjunto de métodos de acceso. No proporciona la implementación de los métodos sino que es puramente declarativa.

Las interfaces se ligan a los puertos y luego cada puerto debe conectarse a un canal que implemente la interfaz asociada antes de que comience la simulación. De lo contrario se generará un error en tiempo de ejecución durante la etapa de *elaboración*.

Un canal que hereda una interfaz debe implementar todos los métodos definidos en ésta o se producirá una falla en tiempo de compilación.

2.5. Puertos

Un puerto es el mecanismo principal para permitir la comunicación a través de las fronteras de un módulo.

En tiempo de elaboración, todos los puertos se conectan a un canal.

Durante la ejecución de la simulación, cada puerto transfiere las llamadas a métodos provenientes de un proceso dentro del módulo al canal al cual el puerto fue conectado en la etapa de *elaboración*. El puerto direcciona llamadas a métodos hacia afuera de la instancia de un módulo.

2.6. Programador (Scheduler)

El programador (scheduler) es la parte del kernel que controla la simulación ocupándose del avance del tiempo simulado haciendo que los procesos se vuelvan ejecutables a medida que se notifican eventos, ejecutando procesos y actualizando canales primitivos.

El propósito primario del scheduler es el de provocar o reanudar la ejecución de los procesos que el usuario provee como parte de la aplicación. Es orientado a eventos, lo que significa que los procesos se ejecutan en respuesta a la ocurrencia de ciertos eventos que ocurren en puntos específicos del tiempo de simulación. El scheduler no es apropiativo.

2.7. Eventos

El kernel de simulación de SystemC es orientado a eventos, por lo que es importante definir el concepto de *evento*.

Un evento es algo que ocurre en un determinado punto en el tiempo. No tiene duración y tampoco tiene ningún valor.

En SystemC, se usa la clase *sc_event* para modelar eventos. Esta clase permite disparar un evento mediante una llamada al método *notify*.

La ocurrencia de un evento solamente tiene incidencia en los procesos que están esperando por ese evento o que son sensibles a él. En SystemC, los procesos pueden esperar la ocurrencia de un determinado evento mediante sensibilidad dinámica utilizando la sentencia *wait (evento)* en el caso de los hilos y *next_trigger (evento)* en el caso de los métodos, o bien mediante sensibilidad estática declarando el proceso sensible a un determinado evento.

2.8. Relojes

Un reloj es un canal primitivo predefinido que modela el comportamiento de una señal de reloj digital periódica. Los procesos pueden declararse sensibles a un reloj y de esa manera reaccionar ante sus pulsos.

3. Dinámica de la simulación

Las simulaciones en SystemC se dividen en tres etapas principales: *elaboración*, *ejecución* y *limpieza*.

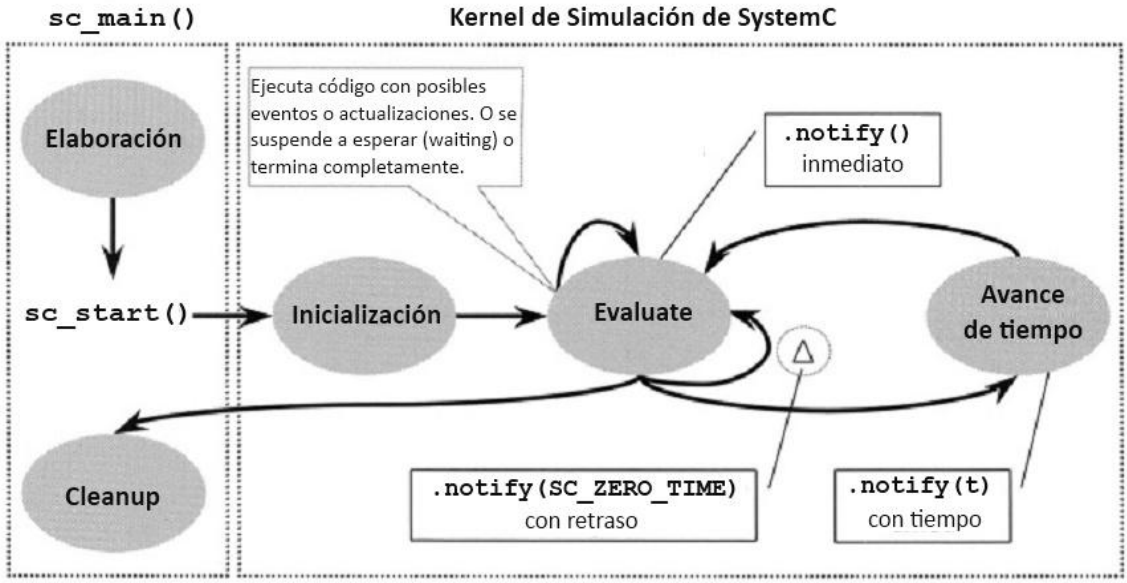

Figura 7: Kernel de Simulación de SystemC

Inicialmente, durante la etapa de *elaboración,* se instancian los módulos, se realizan las conexiones de los puertos a los canales y se establecen los parámetros de la simulación.

A continuación, se realiza un llamado a *sc_start* dando inicio a la etapa de *ejecución.* Esta etapa realiza la *inicialización* en la que se colocan los procesos definidos en la fase anterior en estado listo (*ready*) y comienza la ejecución del kernel de simulación.

Se dice que un proceso está en estado de *ready* siempre que está listo para ser ejecutado.

Seguidamente, comienza la etapa de evaluación (*evaluate*). En esta etapa, cada uno de los procesos que se encuentran en estado de *ready* va siendo elegido

aleatoriamente para pasar al estado de corriendo (*running*) y ser ejecutado. Cada uno se ejecuta hasta que termine a través de la sentencia *return* o bien hasta que se suspenda por ejemplo mediante la sentencia *wait*.

Los procesos que finalizaron son descartados y los que se suspendieron pasan al estado de esperando (*waiting*).

De este modo, el kernel de simulación va pasando el control a los diferentes procesos hasta que no quede ninguno en estado *ready*. En ese momento avanza el tiempo hasta el próximo instante en el que un proceso pase a estado ready, es decir el tiempo en el que caduca el tiempo establecido en la sentencia *wait (time)*.

La simulación es ejecutada durante un tiempo de simulación preestablecido y finalmente se ejecuta, opcionalmente, una etapa de limpieza (*cleanup*) en la que se pueden analizar los datos generados por la simulación y crear reportes.

3.1. Uso de *wait*

El método *wait* es invocado por los hilos para suspender su ejecución. Cuando es invocado, el estado del hilo es guardado y el kernel de simulación toma el control y lo otorga a otro proceso que esté en estado de *ready*. Cuando el hilo suspendido es reactivado, el scheduler restaura su contexto y éste reanuda la ejecución a partir de la sentencia siguiente del *wait*.

El método *wait* está sobrecargado, lo que significa que puede ser invocada con distinto tipo y cantidad de parámetros. Cada una de las formas de invocación dejará el hilo suspendido esperando por distintas condiciones, como un plazo de tiempo, un evento, una invocación a un *notify* o combinaciones entre ellos. La sintaxis es la siguiente:

```
Wait(time);

wait(event);

wait(event1 | event2 | … | eventn); // cualquiera de
estos

wait(event1 & event2 & … & eventn); // todos estos
eventos
```

```
wait(timeout, event);
wait(timeout, event1 | event2 | … | eventn);
wait(timeout, event1 & event2 & … & eventn);
wait();
```

La primera invocación demora al hilo por un período de tiempo. De esta forma, se pueden simular retardos (delays) de acciones reales, como el movimiento de un brazo mecánico, una reacción química o la propagación de una señal. Cuando finaliza el tiempo, el hilo pasa al estado *ready* y su ejecución es reanudada por el scheduler.

Las otras formas de invocar *wait* suspenden el hilo hasta la ocurrencia de uno o más eventos. El operador | denota "cualquiera de estos". La ocurrencia de cualquiera de los eventos listados sacará de suspensión al hilo. El operador &, por el contrario, indica que deben ocurrir todos los eventos para que el hilo pueda salir del estado de suspensión.

Además, también se puede agregar el parámetro *timeout* para reanudar el proceso si no ocurren los eventos por un determinado período de tiempo. Cuando se utiliza un timeout, puede invocarse posteriormente la función booleana *timed_out* para comprobar si fue esa la causa por la que se reanudó el hilo, aunque el uso de esta función se considera no recomendado (deprecated) a partir de la versión 2.2 de SystemC [DOU10].

Por otro lado, al utilizar *wait* con eventos separados por el operador |, cuando el hilo se reanude, no es posible saber cuál o cuáles fueron los eventos que ocurrieron.

4. Open SystemC Initiative

Varias de las organizaciones que contribuyeron fuertemente en los esfuerzos de desarrollo del lenguaje, notaron en una etapa temprana que cualquier nuevo lenguaje de diseño debería ser abierto a la comunidad y no propietario. Como resultado de esto, en 1999 se formó la Open SystemC Initiative (OSCI) cuyos objetivos fueron [OPE10]:

- Evolucionar y estandarizar el lenguaje

- Facilitar la comunicación entre los usuarios del lenguaje y las herramientas de los fabricantes
- Permitir la adopción del lenguaje
- Proveer mecanismos para desarrollos y mantenimiento de código abierto

Especificación CAN

Controller Area Network (CAN) es un protocolo de comunicación serial que soporta control distribuido en tiempo real con un alto grado de seguridad.

El dominio de aplicaciones abarca desde redes de alta velocidad hasta cableado multiplexado de bajo costo.

1. Introducción

Un automóvil moderno puede poseer alrededor de 70 Electronic Control Unit (ECU) para varios subsistemas. Típicamente, el mayor y más complejo de estos procesadores es la unidad de control del motor (también llamada ECU); pero también existen otras ECUs para la transmisión, airbags, ABS, control crucero, audio, etc. Algunos de estos son subsistemas independientes, pero otros consideran la comunicación como una parte esencial de su operación normal. Un subsistema podría requerir controlar actuadores o recibir información de sensores. El estándar CAN se construyó específicamente para satisfacer este propósito [COR02].

En la electrónica automotriz, las ECUs, sensores, sistemas de control de tracción, etc. están conectados utilizando CAN con tasas de transferencia de hasta 1 Mbit/s.

CAN se presenta también como una atractiva opción para utilizar en el cableado del automóvil, debido a su relativo bajo costo.

Para lograr transparencia en el diseño y flexibilidad en la implementación, CAN ha sido subdivida en tres capas:

- Capa de objetos
- Capa de transferencia
- Capa física

La capa de *objeto* y de *transferencia* abarcan todos los servicios y funciones de la capa de enlace definida por el modelo ISO/OSI. El alcance de la capa de objetos incluye:

- Encontrar los mensajes a transmitir
- Decidir qué mensajes recibidos por la capa de transferencia serán utilizados
- Proveer una interfaz al hardware en el que se basa la capa de aplicación

Hay mucha libertad al definir el manejo de objetos. El alcance de la capa de transferencia principalmente es el protocolo de transferencia, lo que significa controlar el entramado, realizar arbitración, señalización de errores y captura de fallas.

Dentro de la capa de transferencia se decide si el bus está libre para comenzar una nueva transmisión o si la recepción recién está comenzando. Además algunas características del tiempo de bits se consideran como parte de la capa de transporte. Es importante mencionar que la capa de transporte no está abierta a modificaciones.

El alcance de la capa *física* es la transferencia real de bits entre los diferentes nodos con respecto a todas las propiedades eléctricas. Dentro de una red, la capa física tiene que ser la misma para todos los nodos. Sin embargo, existe más de una implementación de capa física para elegir.

2. Conceptos básicos

CAN posee las siguientes propiedades:

- Priorización de mensajes
- Garantía de tiempos de latencia
- Flexibilidad de configuración
- Recepción multicast con sincronización de tiempo
- Consistencia de datos a en el sistema
- Multimaster
- Detección de errores y señalización

- Retransmisión automática de mensajes corruptos tan pronto como el bus esté disponible nuevamente
- Distinción entre errores temporales y fallas permanentes de nodos y desconexiones autónomas de nodos defectuosos

3. Estructura en capas de un nodo CAN

Capa de aplicación
Capa de objetos • Filtrado de mensaje • Manejo de mensaje y estado
Capa de transferencia • Manejo de falla • Detección y señalización de errores • Validación de mensaje • ACK (Reconocimiento/Aceptación) • Arbitración • Tramado del mensaje • Tasa de transferencia y temporizado
Capa física • Nivel de señal y representación de bits • Medio de transmisión

Figura 8: Estructura en capas de nodo CAN

La capa física define cómo son realmente transmitidas las señales. Dentro de esta especificación, es lo suficientemente laxa como para permitir que las implementaciones del medio de transferencia y el nivel de señal sean optimizadas por sus aplicaciones.

La capa de transferencia representa el kernel del protocolo CAN. Presenta mensajes recibidos de la capa de objeto y acepta mensajes para ser transmitidos desde la capa de objetos. La capa de transferencia es responsable del timing y la sincronización de bits, entramado de mensajes, arbitración, ack, detección y señalización de errores y manejo de fallas.

La capa de objetos se encarga del filtrado de mensaje así como también del manejo de estado y mensajes.

4. Características

4.1. Mensajes

La información en el bus se envía en mensajes de formato fijo de distinta longitud pero siempre delimitada. Cuando el bus está libre, cualquier unidad conectada puede comenzar a transmitir un nuevo mensaje.

4.2. Ruteado de información

En los sistemas CAN, los nodos CAN no hacen uso de información sobre la configuración del sistema. Esto tiene consecuencias importantes:

- *Flexibilidad del sistema*: los nodos pueden ser agregados a la red CAN sin requerir ningún cambio en el hardware o software de cualquier nodo o capa de aplicación
- *Ruteado de mensajes*: el contenido del mensaje se define por un identificador. El identificador no indica el destino del mensaje, pero describe el significado de los datos, de este modo todos los nodos en la red son capaces de decidir mediante el filtrado de mensajes si deben actuar sobre los datos o no.
- *Multicast*: Como una consecuencia del concepto de filtrado de mensajes cualquier cantidad de nodos puede recibir y simultáneamente actuar sobre el mismo mensaje.
- *Consistencia de datos*: Dentro de una red CAN se garantiza que un mensaje es simultáneamente aceptado por todos los nodos o simultáneamente por ningún nodo. Además la consistencia de

datos de un sistema se logra bajo los conceptos de multicast y manejo de errores.

4.3. Tasa de bits

La velocidad de CAN puede ser diferente en diferentes sistemas. Sin embargo, en un sistema dado la tasa de bits es uniforme y fija.

4.4. Prioridades

El identificador define una prioridad de mensaje estática durante el acceso al bus.

4.5. Petición de datos remota

Al enviar una trama remota un nodo que requiere el dato puede pedir a otro nodo que le envía la trama de datos correspondiente. La trama de datos y su trama remota correspondiente se las llama por el mismo identificador.

4.6. Multimaster

Cuando el bus está libre cualquier unidad puede comenzar a transmitir un mensaje. La unidad con el mensaje de mayor prioridad a ser transmitida gana el acceso al bus.

4.7. Arbitración

Cuando un bus está libre, cualquier unidad puede comenzar a transmitir un mensaje. Si 2 o más unidades comienzan a transmitir un mensaje en el mismo momento, el conflicto de acceso al bus se resuelve por arbitración de diferencia de bits utilizando el identificador. El mecanismo de arbitración garantiza que no se pierde ni información ni tiempo. Si una trama de datos y una trama remota con el mismo identificador se inicializan al mismo tiempo, la trama de datos prevalece por sobre la trama remota. Durante la arbitración cada transmisor compara el nivel del bit transmitido con el que es monitoreado en el bus. Si estos niveles son iguales la unidad puede continuar transmitiendo. Cuando un nivel recesivo se envía y un nivel dominante se monitorea, la unidad ha perdido la arbitración y debe dejar de transmitir.

4.8. Seguridad

Para lograr la seguridad de la transferencia de datos, se implementan en cada nodo CAN medidas para la detección, señalización y auto-comprobación de errores.

4.9. Detección de errores

Para detectar errores se han tomado las siguientes medidas:

- Monitoreo (los transmisores comparan el nivel de bits a ser transmitidos con el nivel de bits detectado en el bus)
- Chequeo de Redundancia Cíclica (CRC)
- Bit Stuffing
- Chequeo de Trama de Mensajes (MFC)

Performance de la detección de errores

- Los mecanismos de detección de errores poseen las siguientes propiedades:
- Se detectan todos los errores globales
- Se detectan todos los errores locales en los transmisores
- Hasta 5 errores aleatoriamente distribuidos en un mensaje son detectados
- Errores en ráfaga de longitud menor que 15 en un mensaje son detectados
- Errores de cualquier número impar en un mensaje son detectados

La probabilidad de error residual total para mensajes corruptos sin detectar: menor que

$$\text{Tasa de error de mensaje} * 4.7 * 10^{-11}$$

4.10. Error de señal y tiempo de recuperación

Los mensajes corruptos son marcados por cualquier nodo que detecte un error. Tales mensajes son abortados y serán retransmitidos automáticamente. El tiempo de recuperación desde que se detecta un error hasta el comienzo del próximo mensaje es de a lo sumo el tiempo en que toma pasar 29 bits por la red, si no hay otro error.

4.11. Manejo de fallos

Los nodos CAN son capaces de distinguir pequeñas discrepancias de fallas permanentes. Los nodos defectuosos se apagan.

4.12. Conexiones

El enlace de comunicación serial CAN es un bus al cual una cantidad de unidades puede conectarse. Este número no posee un límite teórico. Prácticamente todas las unidades pueden ser limitadas por tiempos de retraso y/o cargas eléctricas en la línea del bus.

4.13. Canal único

El bus consiste en un canal único que transporta bits. La información de resincronización puede derivarse de estos datos.

4.14. Valores del bus

El bus puede tener uno o dos valores lógicos complementarios: dominante o recesivo. Durante la transmisión simultánea de bits dominantes y recesivos, el valor del bus resultante será dominante.

4.15. Comprobación de consistencia

Todos los receptores comprueban la consistencia de un mensaje recibido y reconocerán un mensaje consistente y marcarán uno que no lo sea.

5. Estructura de mensajes

5.1. Tipos de trama

La transferencia de mensajes se manifiesta y controla por cuatro tipos de trama diferente:

Una *trama de datos* transporta datos desde un transmisor a los receptores.

Una *trama remota* se transmite por una unidad de bus para pedir por la transmisión de una *trama de datos* con el mismo identificador.

Una *trama de error* se transmite por cualquier unidad que detecte un error en el bus.

Una *trama de sobrecarga* se utiliza para proveer retrasos extra entre las sucesivas tramas de dato o remotas.

Las tramas de datos y las tramas remotas se separan de sus predecesoras mediante un *espacio intertrama*.

5.2. Trama de datos

Una trama de datos se compone de siete campos de bit diferentes: comienzo de trama, campo de arbitración, campo de control, campo de datos, campo CRC, campo ACK y fin de trama. El campo de datos puede ser de longitud cero.

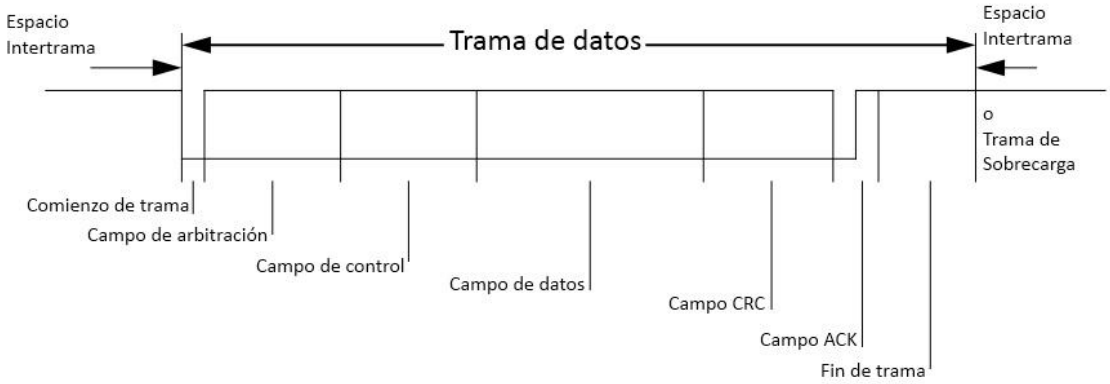

Figura 9: Trama de datos CAN

Comienzo de trama: marca el comienzo de la trama de datos y la trama remota. Consiste de un solo bit dominante. Una estación sólo tiene permitido el comienzo de la transmisión cuando el bus se encuentra ocioso. Todas las estaciones tienen que sincronizar a la punta líder causada por el comienzo de la trama de la estación que comenzó la transmisión primero.

Campo de arbitración: consiste en el identificador y el bit de petición de transmisión remota (Remote Transmission Request). La longitud del identificador es de 11 bits. Estos bits se transmiten en orden desde ID-10 hasta ID-0. El bit menos significativo es ID-0. Los 7 bits más significativos (ID-10 – ID-4) no deben ser todos recesivos.

En las tramas de datos el bit RTR debe ser dominante. Dentro de la trama remota el bit RTR debe ser recesivo.

Campo de control: consiste en seis bits. Incluye el código de longitud de datos y dos bits reservados para expansión futura. Los bits reservados tienen que enviarse dominantes, aunque los receptores aceptan bits dominantes y recesivos en todas las combinaciones.

El código de longitud de datos representa el número de bytes en el campo de datos. Este código es de 4 bits.

El campo de datos es el que incluye los datos a ser transferidos dentro de la trama de datos. Puede contener desde 0 hasta 8 bytes, cuyos bits se transmiten desde el más significativo.

Campo CRC: Contiene la secuencia CRC seguida de un delimitador CRC (un sólo bit recesivo).

Campo ACK: es de dos bits de longitud. Uno corresponde al slot ACK y el otro al delimitador ACK.

Todas las estaciones que hayan recibido la secuencia CRC correspondiente reportan esto dentro del slot ACK sobrescribiendo el bit recesivo del transmisor por uno dominante.

El segundo bit es el delimitador ACK y debe ser recesivo. Como consecuencia, el slot ACK está rodeado de dos bits recesivos.

Fin de trama: Cada trama de datos y trama remota está delimitada por una secuencia de marcas que consiste en siete bits recesivos.

IEEE Standard para EDRs

1. Propósito

El estándar 1616-2004 fue creado con el propósito de proveer un diccionario de datos con todos sus atributos, con el objetivo de unificar los formatos de los elementos y los protocolos de extracción de datos para facilitar el análisis y promover la compatibilidad entre EDRs.

La adopción del estándar debería hacer los datos más accesibles y útiles a los usuarios finales.

2. Roles y objetivos

El Instituto de Ingenieros Eléctricos y Electrónicos es una asociación internacional sin fines de lucro dedicada mayormente a la estandarización. Sus orígenes datan de hace más de 100 años, aunque se lo denomina desde principios de 1960.

En la creación del estándar 1616-2004, se busca en el mediano plazo lograr que la utilización masiva de las cajas negras sea estandarizada, con los beneficios mencionados anteriormente.

La utilización de EDRs puede beneficiar mucho a los investigadores y al equipo dedicado a la seguridad vial. A pesar de que la metodología puede variar significativamente dependiendo de las metas pretendidas, la mayoría de los estudios en esta área incluye una identificación y cuantificación del problema, desarrollo de relaciones, creación y pruebas de posibles soluciones, implementación de las mismas, y finalmente una evaluación de campo.

Los problemas en las rutas y áreas de investigación de infraestructura provienen de una carencia de comprensión completa del problema. Típicamente, esto es el resultado de la dicotomía de los datos: o existe una cantidad importante de datos con un bajo nivel de detalle o hay detalles suficientes pero en poca cantidad. Otro

desafío es la ventana de tiempo para la recolección de datos. Dado que la actual recolección de datos de los accidentes se basa en la información juntada o estimada luego del incidente, los investigadores tienden a carecer de confianza acerca de la información recolectada.

El éxito de la investigación de las rutas e infraestructuras depende críticamente de la validez y consistencia de los datos recolectados del accidente. Hoy en día, la mayoría de los elementos de datos se recolectan o son inferidos posteriormente al accidente, y se almacenan en bases de datos (en ciertos países). El análisis ha sido históricamente dificultoso por los errores en la precisión de los datos recolectados y la carencia de consistencia entre la información capturada de una base de datos a otra. La tecnología MVEDR tiene el potencial de aumentar la colección de estos elementos de datos así como también iniciar la recolección de elementos previamente imposibles de recolectar.

Aunque la tecnología MVEDR no reemplazará todos los aspectos de la investigación de accidentes, tiene el potencial de cambiar la ventana de tiempo de la recolección de datos de accidentes para hacerla más cercana al intervalo en el cual el evento ocurre. Datos tales como la velocidad inicial del vehículo, el cambio total de velocidad y el estado de los cinturones de seguridad pueden ser obtenidos de estos dispositivos mientras el evento ocurre. Esto asiste a los investigadores para proveer una representación más fiel de las condiciones del impacto y las características vehiculares durante un evento. Con una mejor comprensión del problema y los mecanismos involucrados, los investigadores estarán mejor equipados para desarrollar soluciones costo-efectivas.

3. Definición de evento

El estándar IEEE-1616 define un evento como perteneciente a los accidentes de los vehículos a motor. Generalmente, existe un acuerdo de que el tipo de evento total es un accidente, pero los EDRs pueden recolectar datos de diferentes tipos de eventos, tales como aplicar una alta presión de frenado y aceleraciones de gran magnitud. Un evento puede también ser definido como un desvío del camino.

Los eventos relacionado a accidentes han sido las funciones disparadoras típicas utilizadas por los OEMs. Muchos OEMs utilizan el sensor del airbag para determinar si se debe iniciar la función del MVEDR. Algunos sistemas se encuentran incorporados dentro del Sensing Diagnostic Module (SDM), también conocido como el sensor del airbag. En un escenario típico, el procesador comienza a recolectar datos luego de que el acelerómetro lee una desaceleración de 2 gn. La iniciación de la recolección de datos también se conoce como "activación del algoritmo". Luego de que el evento finaliza, los datos se guardan en un área de almacenamiento de memoria permanente (no volátil).

Para los eventos donde el airbag no explota, los datos se almacenan un archivo de "casi utilización/deploy". Estos datos pueden ser sobrescritos por un nuevo evento si tiene un delta-v que excede aquel del evento almacenado. Si es menor, el archivo no se almacena. Para eventos que resultan en la utilización del airbag, los datos se guardan en el área de almacenamiento de memoria de "utilización/deployment". Por esto, es posible que varios vehículos OEM tengan típicamente un archivo almacenado de "casi utilización" y un segundo archivo almacenado de "utilización", si hubiese ocurrido un accidente con acción de los airbags. Otros OEMs almacenan los datos en el módulo de control de los cinturones de seguridad (RCM en inglés) que utiliza un umbral similar para la activación del algoritmo. El RCM almacena un archivo, que es actualizado cuando el delta-v excede el delta-v del archivo almacenado. La acción de un airbag siempre será grabada, sin importar el delta-v.

Capítulo III – Propuesta de EDR

Diseño de la caja negra

1. Introducción

En el presente trabajo se desarrolló un modelo de dispositivo de desgrabación de eventos para automóviles, también conocido como caja negra.

El diseño se realizó tomando como referencia las implementaciones de este tipo de dispositivos por parte de las automotrices. Sin embargo, a diferencia de éstas, se implementó siguiendo las pautas establecidas por el estándar IEEE-1616 para dispositivos de grabación de eventos para vehículos motorizados.

Esto significa que se respetaron todos los tipos de datos de los elementos recolectados, los tiempos y velocidades de muestreo, las resoluciones de los datos, las unidades de medida y las definiciones de datos y convenciones de signo establecidas en dicho documento.

Además, el comportamiento de la caja negra se ajusta a lo establecido por el estándar explicado en el punto 7.7 y los elementos de datos monitoreados y almacenados por la caja negra son los recomendados por este estándar.

El vehículo simulado para el cuál se implementó la caja negra posee un equipamiento de seguridad típico, compuesto por frenos ABS, bolsas de aire frontales para conductor y acompañante, control de estabilidad y sensor de cinturón de seguridad.

2. Propósito

El propósito de la caja negra es almacenar diversas variables sobre el vehículo y sus ocupantes en los momentos previos, durante y posteriores a un evento o accidente.

De este modo, la información almacenada puede ser posteriormente analizada y procesada por expertos junto con las demás evidencias para reconstruir el accidente y entender sus causas.

Con todo conocimiento obtenido se puede mejorar la seguridad vial tanto en el aspecto tecnológico como en el legal y el social.

3. Composición

Dado que se pueden utilizar diferentes tecnologías para construir un MVEDR, el Estándar IEEE 1616 no especifica que se deba utilizar una en particular, pero cita cuáles deben ser sus componentes primarios:

- Un procesador para administrar la organización, adquisición, retención y entrega de los datos
- Una memoria no volátil para retener datos claves que ocurriera antes y posteriormente del evento. Esto mantiene datos críticos relevantes a un evento, en un medio que no requiera una fuente de poder, para retener su contenido
- Un buffer de memoria, para recolectar datos previos a un evento, que pueden ser actualizados, y volcar su contenido a una memoria no volátil cuando ocurre un "evento" o "accidente"
- Un reloj interno o externo, para mantener valores relativos o de tiempo real cuando un evento ocurre y así determinar cuando cada elemento de datos ocurrió relativo a un evento

La caja negra está compuesta por un microprocesador, una memoria RAM, una memoria no volátil y un reloj de tiempo real. Además, cuenta con una interfaz para la extracción de la información recolectada.

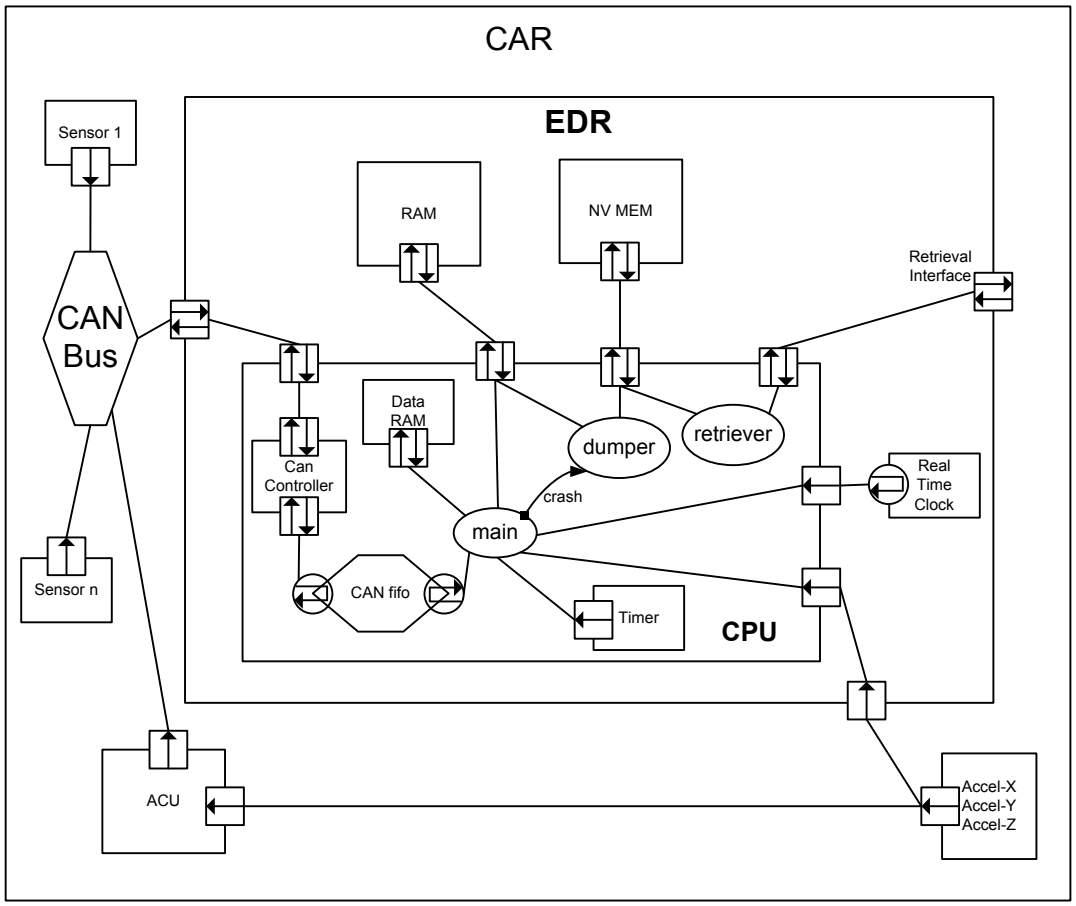

Figura 10: Diseño de la caja negra

El microprocesador ejecuta el programa que se encarga de administrar los datos que se reciben. Esto consiste en decidir cuáles se almacenan, seleccionar la cantidad correcta de muestras de cada elemento de acuerdo con la frecuencia establecida en el estándar, adaptar el formato de almacenamiento de los datos a los establecidos en el estándar, administrar el volcado de la información a memoria no volátil cuando ocurre un accidente y finalmente permitir la extracción de los datos.

La memoria RAM es utilizada para almacenar continuamente la información recibida por la caja negra desde las diferentes fuentes de información del vehículo.

La memoria no volátil es la utilizada para almacenar los datos en el momento en que ocurre un evento y conservarlos para su posterior extracción.

El reloj de tiempo real provee los datos necesarios para que la caja negra pueda registrar la fecha y hora exacta del momento en que ocurre un evento.

4. Restricciones

4.1. Memoria

El estándar IEEE-1616 establece, para cada elemento de datos, cuántas muestras deben tomarse y durante qué período de tiempo en relación al accidente. Por ejemplo, exige que la velocidad de cada rueda deba guardarse durante los 8 segundos previos a un accidente, con una frecuencia de 100 muestras por segundo. Por lo tanto, es necesario tener un historial de la información recibida en los últimos segundos para poder satisfacer esa exigencia en caso de que ocurra un evento.

Para lograr esto, todos los datos recibidos son permanentemente volcados en memoria, lo que implica una alta tasa de escrituras por segundo. Dado que un vehículo durante su período de vida útil está en marcha durante miles de horas, la cantidad total de escrituras es muy alta.

En consecuencia, no es posible utilizar una memoria EEPROM. Este tipo de memorias no sólo que no está preparado para soportar altas tasas de escritura, sino que su velocidad de acceso no es suficiente para trabajar acorde a la caja negra.

Por ese motivo, debe utilizarse una memoria RAM durante el tiempo de operación del vehículo y ante un accidente se deberá efectivamente volcar la información a una memoria no volátil.

Debido a que el volcado debe hacerse lo más rápido posible por la probable destrucción del vehículo y a que además se necesita escribir datos posteriores al accidente, el funcionamiento de la caja debe ser independiente de los demás componentes del automóvil y debe tener una fuente de energía propia que persista el tiempo suficiente.

4.2. Comunicación

Los datos con los que trabaja la caja negra son colectados mediante distintos tipos de sensores instalados en diferentes partes de la carrocería del vehículo. La manera en la que estos datos llegan a la caja negra es mediante un conjunto de redes de interconexión de datos.

Esto implica que a medida que se aumente la cantidad de fuentes de datos en el vehículo, el volumen de información transmitida a través de las redes crecerá raudamente, por lo que la capacidad de cada línea de datos es un factor a tener en cuenta en el diseño del sistema.

Además, debido a los diferentes requerimientos para cada tipo de dato, es necesario contar con un sistema de prioridades para los mensajes trasmitidos por las redes.

4.3. Procesamiento

La unidad de procesamiento de la caja negra ejecuta un algoritmo que es relativamente sencillo, pero al tratarse de un sistema de tiempo real debe respetar ciertas condiciones temporales. Por ejemplo debe trabajar a una frecuencia adecuada para tomar los datos de mayor importancia de modo tal que al recuperar los datos almacenados para reconstruir el accidente, carezcan de faltantes o datos vacíos.

Además, el microprocesador debe ser capaz de direccionar la cantidad de memoria necesaria para almacenar el historial de información de los últimos segundos.

5. Funcionamiento

5.1. Descripción general

La caja negra permanece en funcionamiento todo el tiempo en el que el vehículo se encuentra con la llave en posición de contacto.

Está conectada a las redes del vehículo y recibe permanentemente la información provista por los sensores.

La unidad de procesamiento analiza cada paquete de datos recibido y determina si debe descartarlo o no, en base a la frecuencia establecida en el estándar IEEE-1616. Esto significa que si se reciben muestras de un parámetro con mayor frecuencia que la esperada, algunas serán ignoradas.

Los rangos de la mayoría de los datos que el dispositivo debe grabar se encuentran entre -8 hasta +0,5 segundos con respecto a la ocurrencia de un accidente.

Este requerimiento debería asegurar que el EDR capture información de interés para los investigadores de accidentes.

Los datos recibidos en los últimos segundos son almacenados temporalmente en memoria volátil. Dado que la ocurrencia de un accidente es compleja de prever, este proceso asegura la grabación de los datos pre-accidente.

Cuando se detecta la ocurrencia de un accidente toda la información es volcada, si corresponde siguiendo los criterios del estándar, a una memoria no volátil.

Posteriormente, todos los datos recolectados son recuperados a través de una interfaz de extracción de datos.

5.2. Descripción del estándar

El EDR opera en dos modos. En principio se realiza el monitoreo constante de datos pre-accidente. El EDR opera continuamente en este modo siempre que el vehículo esté en operación. Por otra parte, el EDR opera en el segundo modo cuando el vehículo se ve involucrado en un accidente. En este modo, se deben tomar dos decisiones: la primera es la determinación de la ocurrencia de un accidente y es tomada en base a un umbral disparador. La segunda es la decisión de almacenar los datos capturados y se logra mediante un proceso comparativo. Basado en el resultado del proceso, los datos capturados asociados con el accidente se almacenan en memoria no volátil o se descartan.

En las aplicaciones de vehículos de carga liviana actuales, el umbral de disparo está asociado con el mismo mecanismo o uno similar al utilizado para determinar la explosión de los airbags. Las circunstancias que causan que este umbral se alcance se las denomina evento.

El inicio de un evento que causa que el EDR comience a volcar los datos en la memoria no volátil se determina cuando el vehículo excede un umbral de desaceleración específico. Posteriormente los datos pueden ser almacenados en el EDR. El estándar requiere que el EDR comience a grabar los datos cuando el cambio de velocidad del vehículo durante cualquier intervalo de 20ms sea igual o superior a 0,8 km/h. Esto es equivalente a 1,152gn de desaceleración contínua. El cambio de velocidad del vehículo puede determinarse o bien en base a la aceleración

longitudinal, o bien mediante la combinación entre la aceleración longitudinal y la aceleración lateral, es decir se calculando el cambio de velocidad del vehículo en el plano horizontal.

Un disparador es uno o más eventos que causan que el EDR comience a grabar. Los datos disparadores pueden recolectarse de un único sensor o a través de una red, por ejemplo un bus CAN.

6. Implementación

6.1. Estructura general

El sistema está representado por una jerarquía de módulos de SystemC. El módulo más general que contiene a todos los demás es el denominado Car. Está compuesto por una caja negra, un bus CAN y diversos sensores y actuadores.

La caja negra está compuesta por un microprocesador, una memoria volátil RAM, una memoria no volátil y un reloj de tiempo real.

6.2. Microprocesador

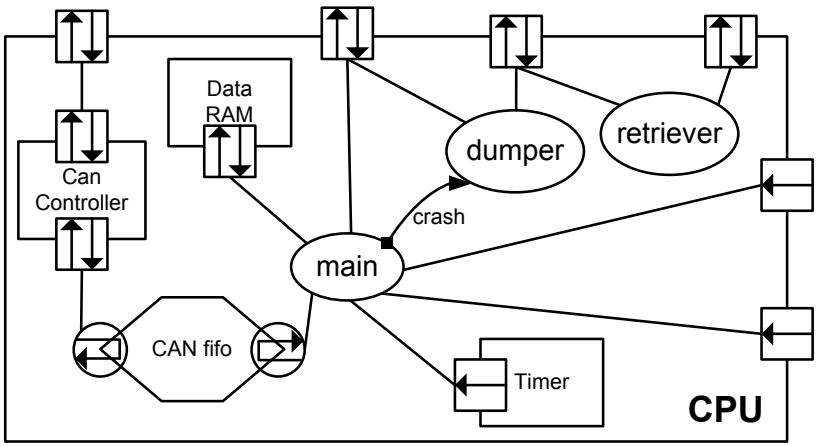

Figura 11: Diseño del Procesador

Para su diseño, se tomó como referencia la arquitectura de un procesador similar al P8xC591 de Phillips [PHI00], si bien no es ni intenta ser una reproducción exacta.

Está compuesto por una unidad aritmético lógica, una memoria RAM de datos interna, un controlador CAN, un timer y un conjunto de interfaces, representadas en SystemC mediante sc_ports, para comunicarse con los demás componentes de la arquitectura, es decir la memoria RAM, la memoria no volátil, el reloj y el o los acelerómetros.

La unidad de procesamiento no está representada mediante un módulo de SystemC, sino que se representa la utilización de la misma mediante un sc_mutex. El código de los hilos de SystemC que implementan el comportamiento de la caja está dividido en bloques que están encerrados entre sentencias de bloqueo y desbloqueo del mutex de la ALU.

El procesador posee registros de 16 y de 32 bits. Para los casos en los que se debe resaltar el uso de un determinado tamaño de registro se utilizan los tipos de SystemC (*sc_uint<n> y sc_int<n>)* mientras que en el resto de los casos se utilizan los tipos nativos de C++.

El CPU está conectado a través de señales con tres memorias diferentes: la memoria de datos interna del CPU, la memoria RAM y la memoria no volátil.

Para cada una de estas memorias, se declaran seis puertos:

- Puerto Read/Write: está conectado a una señal binaria a través de la cual el CPU determina la operación que se va a realizar. Escribe un 1 para un acceso de lectura o un 0 para un acceso de escritura.
- Puerto Address: donde el CPU escribe la dirección de memoria donde se realizará la operación. Esta dirección es de 32 bits de ancho.
- Puerto Data Out: es donde el CPU coloca el dato en el caso de que vaya a realizar una escritura en memoria.
- Puerto Do Operation: Una vez que el CPU definió el tipo y la dirección de memoria sobre la cual se realizará la operación y colocó el dato en el puerto Data Out en el caso de que se trate

de una escritura, debe invertir esta señal para que la memoria ejecute dicha operación.

- Puerto Memory Done: luego de invertir la señal Do Operation, como los accesos son sincrónicos, el CPU espera por un cambio en el estado de esta señal que le indique que la memoria terminó de ejecutar la operación.
- Puerto Data In: en el caso de que se haya realizado una operación de lectura, el CPU puede leer de este puerto el dato proveniente de la memoria.

6.3. Acceso a memoria

Dado que las memorias simulan el paso del tiempo que toma accederlas, los accesos deben realizarse siempre desde *sc_threads* ya que los *sc_methods* no pueden suspender su ejecución como los primeros. Cada memoria conectada al CPU tiene un hilo y un evento asociados para ser accedidas.

Cada vez que un hilo del CPU debe realizar un acceso a una memoria, coloca el tipo de acceso, dirección y valor (en caso de que sea una escritura) en registros y luego dispara el evento asociado a la memoria a la cual se va a acceder. Luego de esto, el hilo se suspende hasta que el evento Memory Access Done ocurra.

Al disparar el evento, un hilo asociado se reanuda, escribe los valores de los registros en los puertos que están conectados a esa memoria, invierte la señal Do Operation correspondiente y se suspende hasta que haya un cambio en la señal Memory Done de la memoria accedida. Al reanudarse, dispara el evento Memory Access Done para que el hilo que hizo el acceso continúe su ejecución.

De esta manera, la memoria simula el paso del tiempo que toma cada acceso.

```
memoryOperation = RWB_SIGNAL_WRITE;
addrRegister = direccionDeMemoria;
unsignedWordRegister = dato;
dataRamAccessEvent.notify();
wait(memAccessEventDone);
```

```
memoryOperation = RWB_SIGNAL_READ;
addrRegister = direccionDeMemoria;
dataRamAccessEvent.notify();
wait(memAccessEventDone);
variable = unsignedWordRegister;
```

6.4. Memoria caché del CPU

El CPU cuenta con una memoria caché de 512 bytes en la que se almacenan los datos básicos que se necesitan para administrar la información que se recibe desde los sensores.

Los primeros bytes están reservados para datos tales como maxCurrentDeltaV, maxRecordedDeltaV, eventCount, etc. Luego de estos datos, se guardan los metadatos sobre los elementos que la caja registra. Se utilizan 14 words (28bytes) de metadatos para cada tipo de elemento en la memoria caché del CPU.

Estos datos incluyen, de cada elemento, el id, la cantidad de muestras por segundo que se espera recibir, el espacio de tiempo con respecto al accidente cuyos datos deben almacenarse en memoria no volátil, el tamaño de la muestra en bytes, el timestamp del último elemento recibido y finalmente los punteros necesarios para el manejo de la memoria circular que cada elemento tiene asignada.

6.5. Timer interno

El CPU cuenta con un timer que genera pulsos cada 1 milisegundo. Por cada pulso, se incrementa una unidad en un contador de 16 bits, generando números entre 0 y 65535.

Estos números son utilizados por los hilos del CPU para obtener timestamps relativos y evitar de esa manera hacer accesos permanentes al reloj de tiempo real. Dado que la caja negra, según los estándares actuales, no necesita representar más de 18,5 segundos de datos en el peor de los casos, los 65 segundos y medio circulares representados por este timer son suficientes.

6.6. El controlador CAN

El controlador CAN está implementado como un módulo integrado dentro del CPU. Por un extremo se conecta al puerto del CPU que se conecta al bus CAN externo.

Por el otro extremo se conecta a una cola (*sc_fifo*) en la que deposita los mensajes recibidos para que sean leídos de allí por los hilos del CPU.

Además, el controlador implementa el método sendMessage para que el módulo que lo contiene envíe mensajes.

Este módulo implementa la arbitración del medio según corresponde al protocolo CAN.

6.7. Memoria RAM

Las memorias están implementadas con un tamaño de palabra de 2 bytes. Tanto el tamaño total de la memoria como la velocidad de escritura y la velocidad de lectura son parametrizables.

```
Mem::Mem(sc_module_name nm, int size, int writeTime,
int readTime){
    SC_THREAD(execute);
    sensitive << doOperation;
    this->readTime = readTime; // En nanosegundos
    this->writeTime = writeTime;
    this->memorySize = size;
    this->memData =
(sc_uint<16>*)malloc(memorySize*sizeof(sc_uint<16>));
}
```

En el constructor de la memoria se recibe como parámetro el tamaño y el tiempo de acceso para lectura y escritura. Se declara un *sc_thread* sensible a la señal doOperation y se aloca la memoria necesaria para simular el espacio de memoria disponible.

Como el tamaño de palabra es de 2 bytes y muchos de los datos con los que trabaja la caja negra son de 4 bytes, generalmente se requieren dos accesos para escribir o leer cada registro.

Este módulo cuenta con los puertos que representan las señales de control, datos y direcciones necesarias para realizar los accesos desde el CPU.

Se utiliza la señal rwb para indicar el tipo de operación que se va a realizar, la señal de control doOperation para indicar cuándo realizar la operación, la señal address para especificar la posición de memoria sobre la cual se va a realizar la operación, las señales dataIn y dataOut como canales de entrada y salida de datos, y la señal memDone para notificar de la finalización de una operación al módulo que accede a la memoria.

El hilo principal de la memoria se inicia suspendido esperando por un cambio en la señal doOperation. Cuando esto ocurre, el hilo se reanuda y lee desde el puerto rwb cuál es la operación que debe realizar y en base a eso invoca a la función read() o write(). Luego de la invocación a la operación correspondiente se ejecuta una sentencia wait() para simular el tiempo que tomaría realizar ese acceso en una memoria real. Finalmente se invierte la señal memDone para indicar que la operación fue ejecutada y vuelve a suspenderse a la espera de otro acceso.

La función *read* lee la señal address para obtener la dirección de memoria y escribe en la señal dataOut el dato que está en esa dirección.

La función *write* lee la señal address para obtener la dirección de memoria y escribe en esa posición el dato leído desde la señal dataIn.

Si bien en algunos lenguajes se utiliza una sola señal bidireccional para transferir datos desde y hacia la memoria, en este caso se utilizaron dos señales (*dataIn* y *dataOut*) porque a partir de la versión 2.2 de SystemC, a menos que se fuerce mediante una directiva al compilador, no se permiten escrituras en una misma señal desde diferentes fuentes.

```
void Mem::execute(){
     while (true) {
          wait(doOperation.value_changed_event());
          switch(rwb.read()) {
          case false : {
               read();
               wait(readTime,SC_NS);
```

```
                    break; }
               case true  : {
                    write();
                    wait(writeTime,SC_NS);
                    break; }
               }
          memDone.write(!memDone.read());
          }
     }
```

6.8. Memoria no volátil

La principal preocupación de direccionar conectores es el interés de recuperar datos luego de un evento severo o accidente y permitir suficiente capacidad en el conector para recolectar los datos. Por ejemplo, si el conector o el MVEDR está tan dañado que la descarga normal de datos no puede ocurrir, entonces la memoria debe ser accesible de alguna forma, y leído por un equipo especial o reconstrucción de laboratorio.

Esto no sólo implica consideraciones al momento de diseñar el hardware, sino que también se debe establecer una convención sobre la disposición lógica de los datos en la memoria.

El estandar IEEE 1616 no establece cómo deben ubicarse los datos en la memoria no volátil, sino que sólo define los tipos de datos que deben usarse para cada elemento.

Por eso, en este trabajo se definió arbitrariamente la siguiente forma de ubicar los datos en memoria.

En el principio de la memoria se definen datos generales sobre el evento:

- Flag de grabación terminada (1 word)
- Fecha del choque (2 words)
- Hora del choque (2 words)

- Datos pre-choque
- Cantidad de eventos (1 word)
- Tiempo entre el evento 1 y el evento 2 (en caso de que haya ocurrido más de un evento)
- Tiempo entre el evento 1 y el evento 3 (en caso de que hayan ocurrido tres eventos)
- Datos post-choque

Los datos pre y post-choque incluyen, de cada elemento:

- id del elemento (1 word)
- Cantidad de registros volcados (1 word)
- Tamaño en words de cada unidad de datos (1 word)
- Unidades de datos antecedidos por su timestamp (2 o 3 words incluyendo uno de timestamp)

6.9. Acelerómetros

Los acelerómetros no están necesariamente ubicados físicamente dentro del EDR, pero generalmente se encontrarían en el mismo módulo junto a él y a la unidad de control de Airbags. En este caso, los tres ejes X,Y y Z son muestreados por el mismo dispositivo, que envía sucesivamente las muestras por una interfaz que está conectada al CPU.

La velocidad de muestreo es parametrizable y generalmente será relativamente superior a la de los demás sensores del sistema.

Cada paquete enviado por este módulo contiene el valor en el formato estipulado por el estándar IEEE 1616 junto con un campo que indica a qué eje pertenece.

6.10. Reloj

Para satisfacer el estándar IEEE 1616, la caja cuenta con un reloj de tiempo real que provee la fecha y hora actual. De esta forma se almacena la fecha y hora en la que se detecto la ocurrencia del accidente.

Según el estándar, la fecha puede ser tomada de un receptor de GPS o de un reloj de tiempo real integrado y puede tener un error de ±15 minutos y la hora un error de ±1 segundo. Por simplicidad se optó por la segunda opción.

El mencionado documento establece que tanto la fecha como la hora deben ser representadas cada una con un long integer de 4 bytes.

Para la fecha, en este modelo se utilizan los 5 bits menos significativos para almacenar el día del mes, los siguientes 4 bits para el número de mes y 13 bits de los restantes para el año.

En cuanto a la hora, se utilizan los 6 bits menos significativos para los segundos, los siguientes 6 bits para los minutos y 5 bits de los restantes para la hora.

El reloj implementado (módulo *RTClock*) implementa una interfaz (interfaz *RTClock_if*) que incluye las funciones *getTime* y *getDate* que son las invocadas por los procesos del CPU para recibir la hora o la fecha.

7. Semántica

El hilo principal del CPU es *main*. Este hilo es el que se encarga de recibir los paquetes de datos de los diferentes sensores, analizarlos y almacenarlos en memoria RAM.

Al iniciarse, construye la tabla de referencia de elementos en la memoria caché a partir de la información leída desde un archivo generado para cada simulación.

De acuerdo a la cantidad de muestras por segundo requerida para cada elemento, el tiempo de muestreo requerido con respecto al momento en que ocurre el accidente y el tamaño de cada muestra, se asigna para cada tipo de dato una porción de la memoria RAM que será utilizada a modo de buffer circular. Por lo tanto, en la tabla de referencia se incluye información sobre cada una de esas áreas de memoria junto con los punteros para administrarla.

La distribución se realiza de acuerdo a los tiempos, frecuencia y tamaño de muestra de cada elemento. En principio, a cada elemento se le asigna el espacio que requeriría para almacenar los datos del choque de mayor duración posible según las restricciones del estándar. Luego, en base a la memoria libre y los words por segundo

que requiere cada elemento, se les asigna un espacio de memoria que les hace ganar una ventana temporal del mismo tamaño que a todos los demás. Por ejemplo, si luego de realizar el cálculo, se determina que la memoria libre es suficiente para almacenar un segundo extra de datos, un elemento de 1000 muestras por segundo con 3 words por muestra recibirá un espacio extra de 3000 words mientras que un elemento de 10 muestras por segundo de 2 words cada una recibirá 20 words.

Dado que el ancho de palabra de las memorias es de 16 bits, y el CPU debe llevar en la memoria interna información de punteros (y los punteros son de 32 bits), cada vez que el CPU debe acceder a datos de un puntero debe realizar dos accesos.

8. Requerimientos mínimos de memoria

Nombre del elemento	Frecuencia	Bytes	Unidad	Intervalo Min	Intervalo Max	Segundos	Bytes/seg	Tamaño total
Accelerometer X	1000	4	g_n = 9.806 65 m/s2	-8	5	13	4000	52000
Accelerometer Y	1000	4	g_n = 9.806 65 m/s2	-8	5	13	4000	52000
Accelerometer Z	1000	4	g_n = 9.806 65 m/s2	-8	5	13	4000	52000
Velocidad x rueda	400	2	km/h	-8	5	13	800	10400
RPM	5	2	r/min	-8	5	13	10	130
Mariposa	4	1	%	-8	5	13	4	52
Service Brake	2	1	on/off	-8	0	8	2	16
Ciclos de encendido (trigger)	1	4	int	0	1	1	4	4
Ciclos de encendido (retrieval)	1	4	int	0	1	1	4	4
Cinturón de seguridad, conductor	10	1	on/off	-8	5	13	10	130
Supresor de airbag, pasajero	1	1	on/off	-8	5	13	1	13
Tiempo de explosión, airbag conductor	1	2	time	0	1	1	2	2
Tiempo de explosión, airbag pasajero	1	2	time	0	1	1	2	2

Total	3426	32				116	12839	166753

Como observamos en la tabla anterior, los acelerómetros son los que demandan mayor ancho de banda, mayor tamaño de memoria y por ende, mayor tiempo de CPU.

En general, se nota que los tiempos de captura de los datos abarcan desde los 8 segundos previos al evento y los 5 segundos posteriores al mismo. Esto se debe a que, dado que en la definición de evento éste puede durar un máximo de 5 segundos, se necesitan los datos previos al mismo para marcar diferentes umbrales, y los segundos posteriores por la misma razón. Con esta ventana de 13 segundos, también se incluye la posibilidad de múltiples eventos dentro de un mismo accidente.

Es interesante notar que los ciclos de encendido y los tiempos de explosión del airbag son datos que se envían una sola vez en el momento del evento o descarga de datos de la caja negra. Son los datos que menos porción de memoria ocupan pero a su vez los más sensibles a pérdidas, dada su baja ocurrencia.

Existen ciertos valores que se toman según tecnologías, como por ejemplo la mariposa del motor, que si posee el automóvil la tecnología Drive-by-wire, podría medirse según un acelerador electrónico. En el caso de no utilizarla, se podría optar por la colocación de un sensor o caudalímetro para medir este dato.

Muchos de estos datos pueden obtenerse por tiempo real con un cable OBD2, común en todos los autos desde fines de 1995. El conector OBD2 emite diferentes datos según la empresa automotriz.

Capítulo IV – Pruebas y Resultados

Pruebas y Resultados Obtenidos

Teniendo el modelo mencionado como referencia, construimos una implementación en el lenguaje de diseño TLM C++/SystemC. Como resultado pudimos probar, por medio de simulaciones, la viabilidad de un dispositivo conforme a las reglas del IEEE.

Las simulaciones involucran un automóvil en funcionamiento equipado con un EDR grabando el tiempo transcurrido hasta que se dispara un accidente.

Luego, los datos volcados por el EDR a la memoria no volátil en el momento del evento se descargan, a través de la interfaz correspondiente, para verificar su conformidad con el protocolo definido por el estándar del IEEE.

Las simulaciones proveen un amplio espectro de flexibilidad. Se probaron varias instancias con diferentes configuraciones que incluyen aspectos parametrizados, tales como la velocidad del CPU, latencias de memoria, etc. Naturalmente, no todas las instancias completaron las pruebas exitosamente, y esto nos ayudó a determinar los requerimientos de hardware mínimos aproximados para cumplir con el estándar previamente mencionado.

El modelo presentado ha ido pasando por varios cambios hasta alcanzar su último estado. Inicialmente, era un SAM (léase untimed). Las escrituras de memoria se realizaban sin ninguna latencia, tanto como las instrucciones de la CPU. De esta forma, podría verificarse, por ejemplo, la correcta administración por parte del hilo *main* del buffer circular asignado para cada tipo de datos. De la misma manera, podemos comprobar que lo volcado de la RAM a la memoria no volátil se realizó correctamente por el hilo *dumper.*

Como detalle adicional, el bus CAN que implementamos era capaz de manejar prioridades, pero no representamos el tiempo requerido para transmitir un mensaje.

En las etapas subsecuentes, los componentes funcionales y de comunicación se refinaron para lograr que el modelo se comporte de manera más realista en relación a una implementación real de hardware. Las declaraciones *wait* se agregaron al código de manera de representar el paso del tiempo. La duración de las declaraciones *wait* se parametrizaron para permitir instanciar hardware con diferentes propiedades, tales como la velocidad de lectura/escritura de la memoria.

Las simulaciones corrieron con los elementos de datos recomendados por el estándar del IEEE.

1. Interfaz

Para realizar las simulaciones de funcionamiento de la caja negra es necesario también emular el funcionamiento de un automóvil real con sensores del cual capturar los datos.

Para esto, se implementó un módulo llamado *SimInput.* Este modulo está conectado a todos los sensores del modelo y les provee valores que son leídos desde un archivo previamente generado.

De este modo, cada sensor cuenta con valores para ser enviados a través del bus CAN según la frecuencia que le corresponde.

La ejecución de cada simulación recibe los parámetros con los que se construirán los módulos que representan el sistema (Fig. 12). Los parámetros que recibe el ejecutable son:

- Velocidad del CPU en MHZ
- Tamaño de la memoria RAM en bytes
- Tiempo de escritura de la memoria RAM en nanosegundos
- Tiempo de lectura de la memoria RAM en nanosegundos
- Tamaño de la memoria no volátil en bytes
- Tiempo de escritura de la memoria no volátil en nanosegundos
- Tiempo de lectura de la memoria no volátil en nanosegundos
- Tamaño del buffer CAN en cantidad de paquetes CAN

- Archivo del cual el módulo SimInput leerá los valores simulados de los sensores

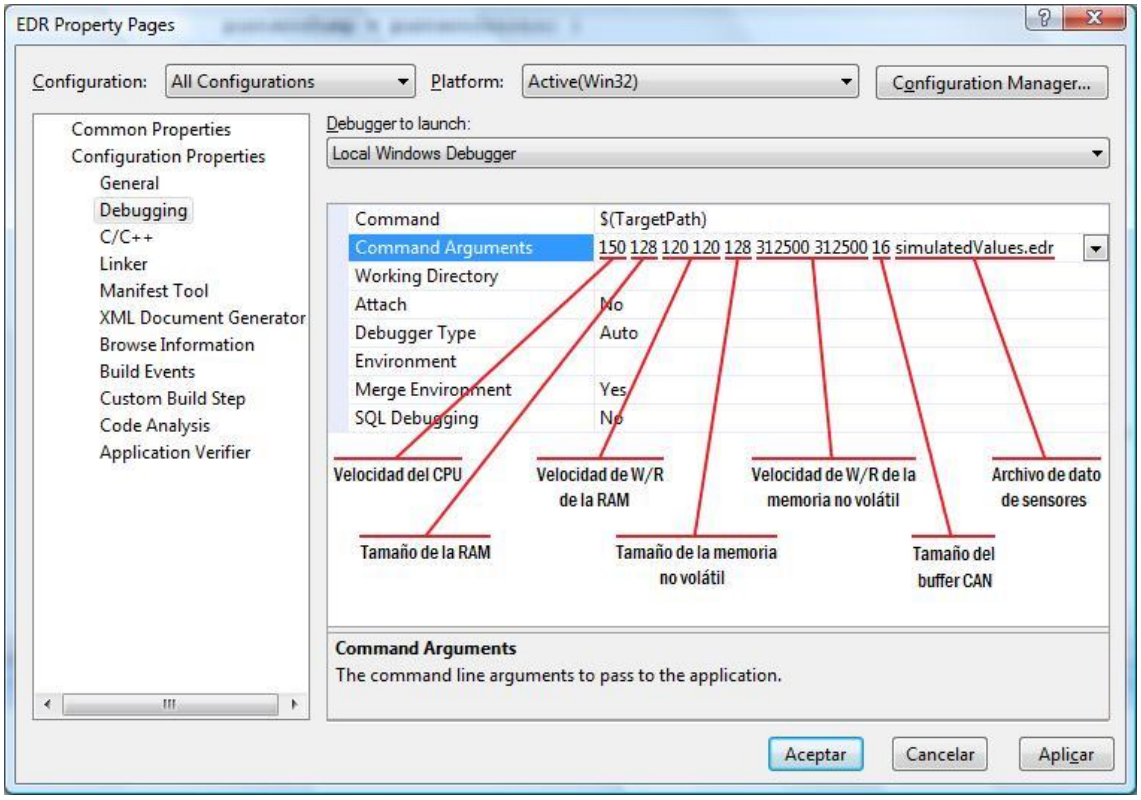

Figura 12: Parámetros de las simulaciones

La ejecución de las simulaciones genera cuatro salidas diferentes.

Durante la ejecución de las simulaciones se puede ver en la consola (Fig. 13) una línea de tiempo básica en la que se muestran determinados eventos que ocurren a medida que avanza el tiempo. Por ejemplo, se imprime el momento en el que se recibe el paquete que indica que un airbag fue detonado, el momento en el que inicia y finaliza el proceso de dump, etc.

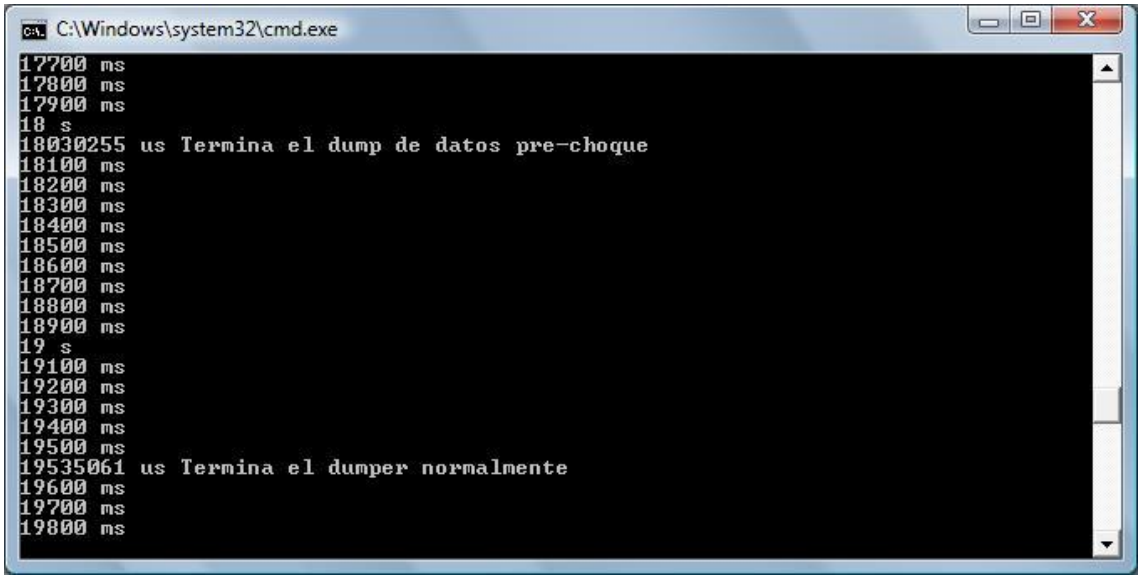

Figura 13: Consola

Además, la ejecución genera tres archivos de texto plano: dump.txt, test.txt y estadisticas.txt.

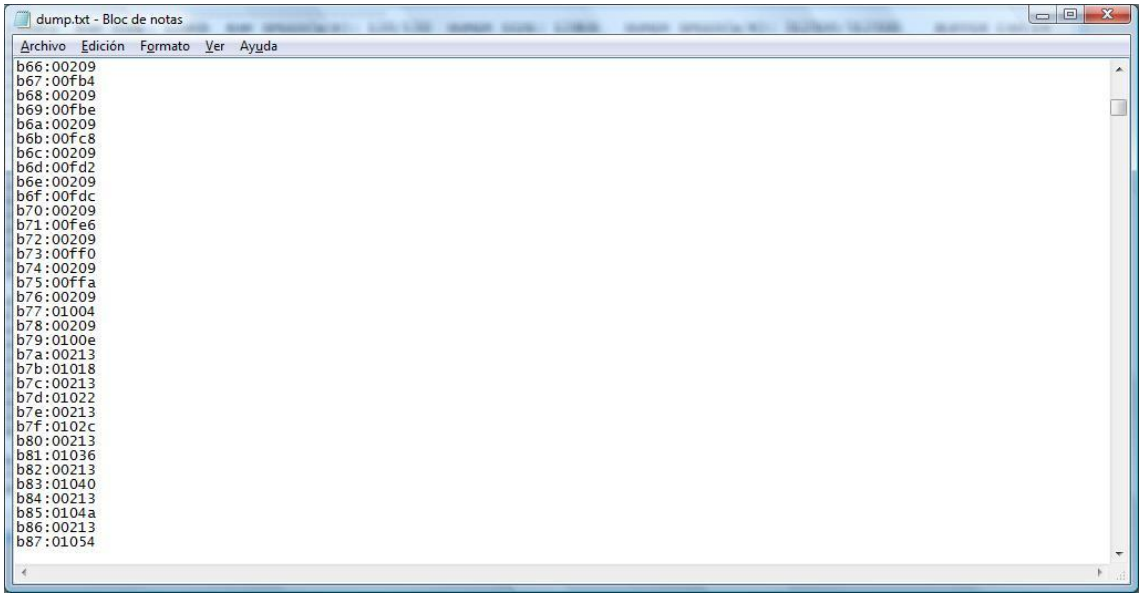

Figura 14: Archivo dump.txt

El archivo dump.txt (Fig. 14) es generado por el proceso retriever y es un volcado de la memoria no volátil del EDR. Tanto las direcciones de memoria como los valores están en hexadecimal.

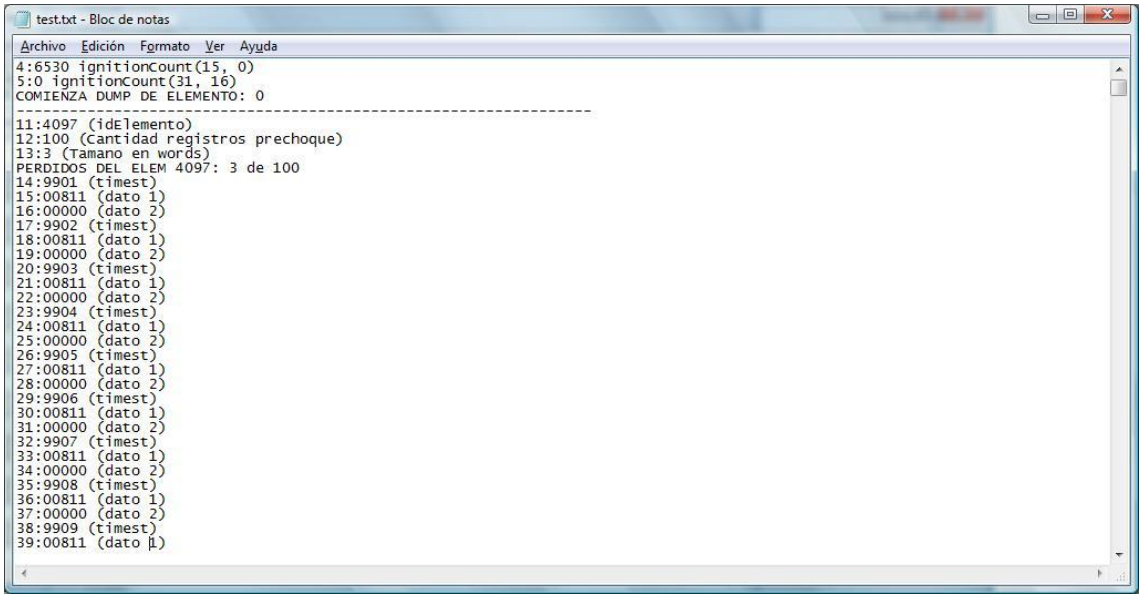

Figura 15: Archivo test.txt

El archivo test.txt (Fig. 15) contiene información sobre las escrituras que se fueron realizando en memoria no volátil con comentarios e información extra para facilitar su lectura. La parte alta y baja de los elementos de dos words se imprime por separado. En este caso las direcciones de memoria están en decimal.

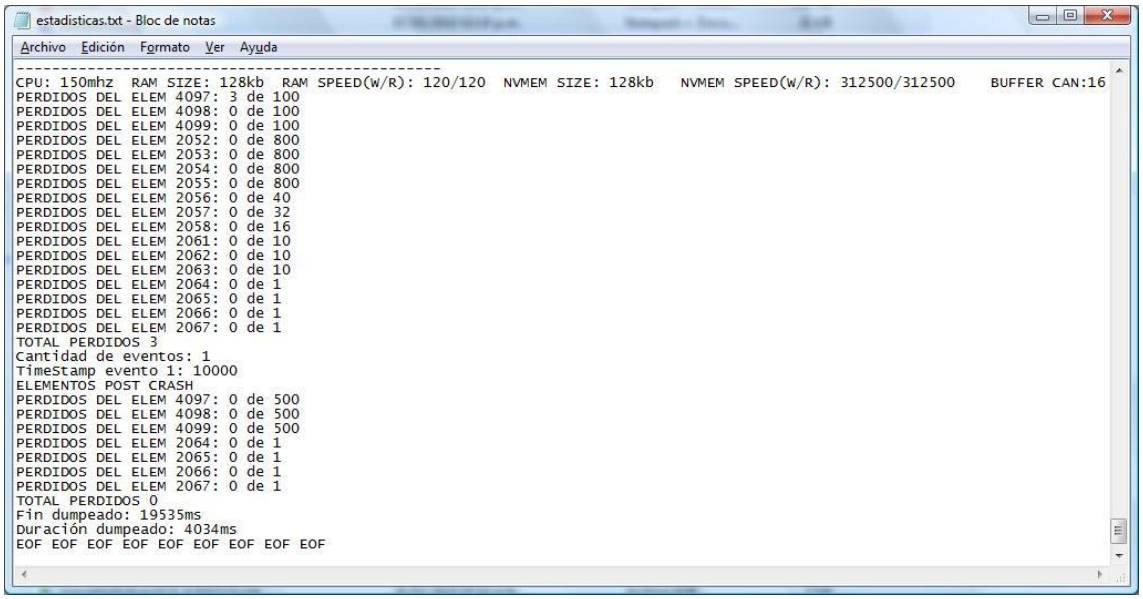

Figura 16: Archivo estadisticas.txt

Finalmente, el archivo estadisticas.txt (Fig. 16) acumula información sobre cada una de las ejecuciones de simulaciones. Incluye los parámetros de la simulación, es decir las características del CPU y memorias con las que fue instanciada, la cantidad de

datos que se guardaron y que se perdieron de cada tipo, la cantidad de eventos que ocurrieron, el timeStamp de cada evento, el timeStamp del momento en el que finalizó el proceso de dump y su duración total.

2. Simulaciones

Escenario 1

Se simula una colisión frontal a una velocidad de 120 km/h.

Tanto el conductor como el pasajero tienen el cinturón de seguridad abrochado.

Aproximadamente dos segundos antes de la colisión, el conductor presiona el freno logrando reducir levemente la velocidad antes del choque.

Al producirse la colisión, ambos airbags son detonados.

A continuación se muestran dos diferentes opciones de volcado de memoria de un evento.

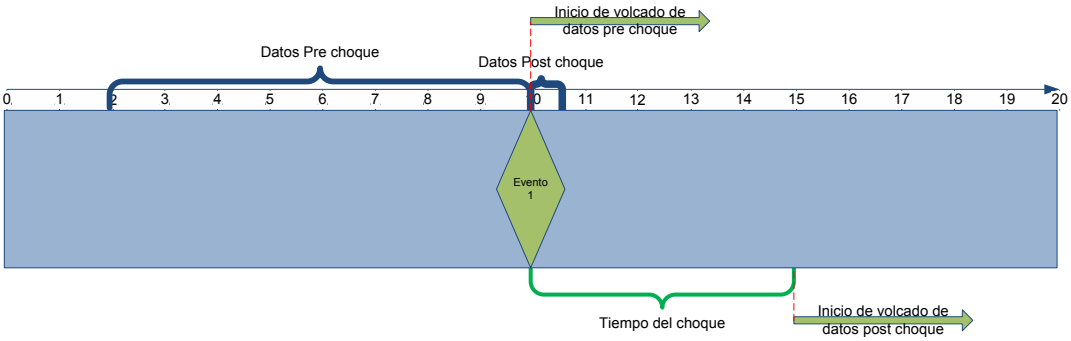

Figura 17: Escenario 1 con volcado de memoria pre choque y post choque separados

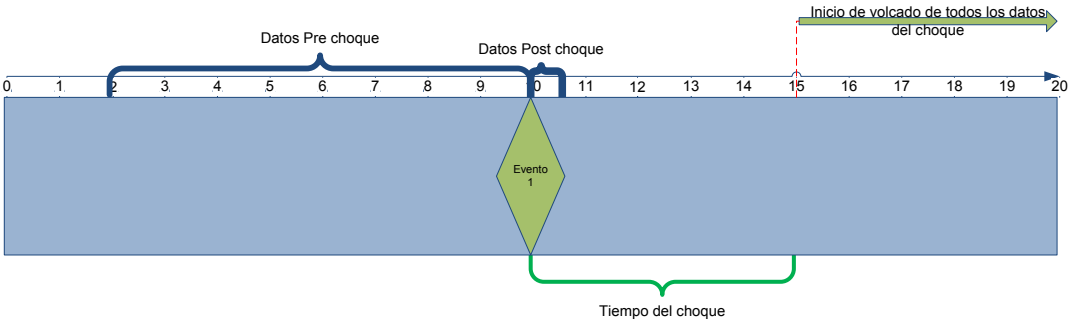

Figura 18: Escenario 1 con volcado de memoria total

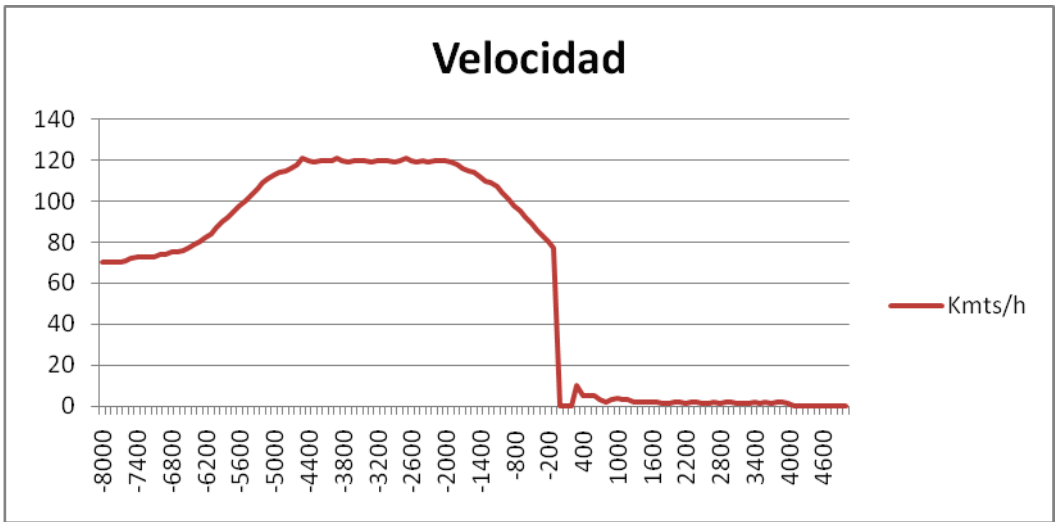

Figura 19: Velocidad

Figura 20: Revoluciones por minuto

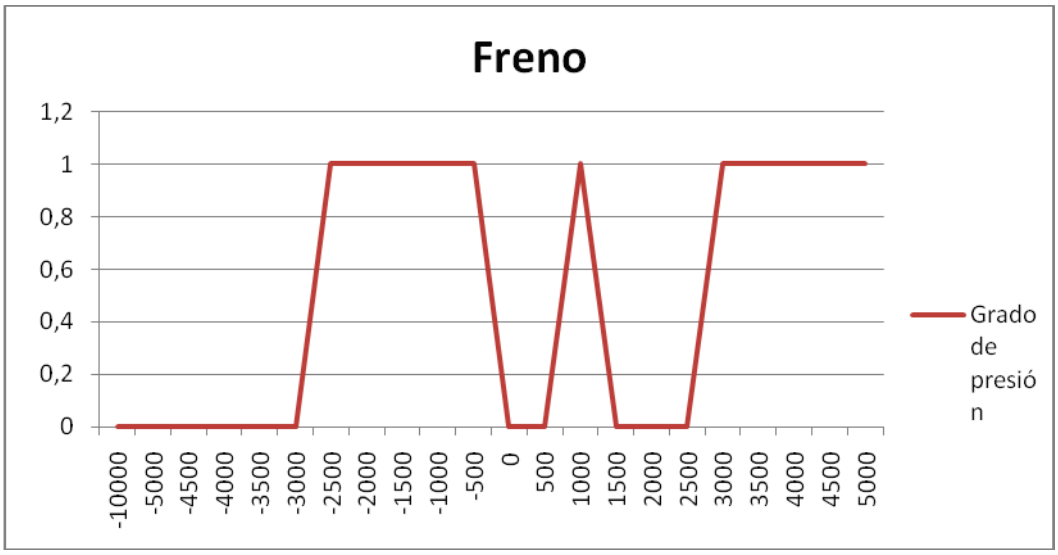

Figura 21: Uso del freno

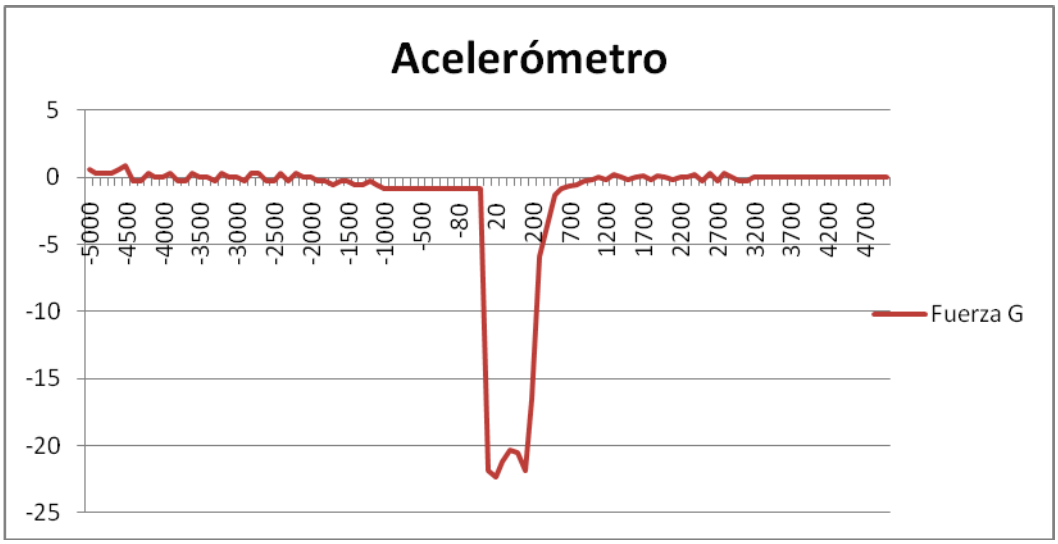

Figura 22: Acelerómetro eje X

	Valor
Cinturón Abrochado Conductor	Abrochado
Cinturón Abrochado Pasajero	Abrochado
Nivel de explosión del airbag del conductor	1
Nivel de explosión del airbag del pasajero	1
Tiempo de explosión del airbag del conductor	37 ms
Tiempo de explosión del airbag del pasajero	46 ms
Grabado completo del archivo	True
Hora del evento	11:47:49
Fecha del evento	20/04/2010
Ciclos de encendido	3718

A modo de ejemplificación de los datos que se pueden obtener a partir de la información provista por las cajas negras, se muestra el log del accidente del escenario 1 (Figs. 18 a 21). Se puede apreciar la velocidad incrementando de 70 a 120, pasando de cuarta a quinta marcha, con la apertura de mariposa que acompaña esta acción. El pie del freno es accionado en los últimos 2 segundos con la consecuente disminución de la velocidad hasta llegar a la colisión. Aquí se puede observar una desaceleración brusca de más de $20g_n$ que supera el umbral, accionando los airbags y posteriormente grabando los datos en la memoria permanente del EDR. Un eventual accidente posterior a este no causaría ningún efecto en la caja negra debido a que al haberse activado el airbag, la memoria no volátil deshabilita escrituras posteriores.

Escenario 2

Se simula un choque en cadena en el que el vehículo impacta 3 veces. La diferencia entre el primer y el tercer impacto es de 4950 milisegundos, 50 milisegundos por debajo de lo que el estándar del IEEE define como duración total de un choque.

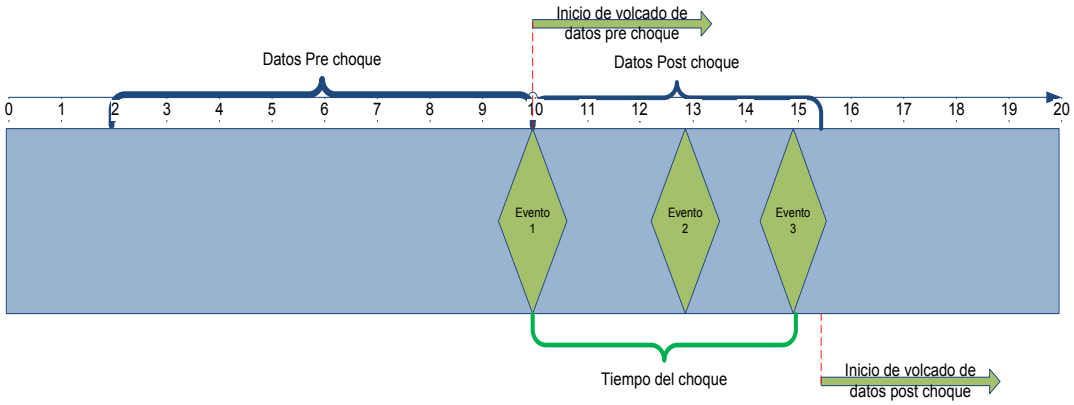

Figura 23: Escenario 2 - Multievento

Escenario 3

Como tercer escenario tenemos a una variación del anterior, agregando un segundo accidente pocos milisegundos luego del primero. El airbag es detonado recién en el evento del segundo choque.

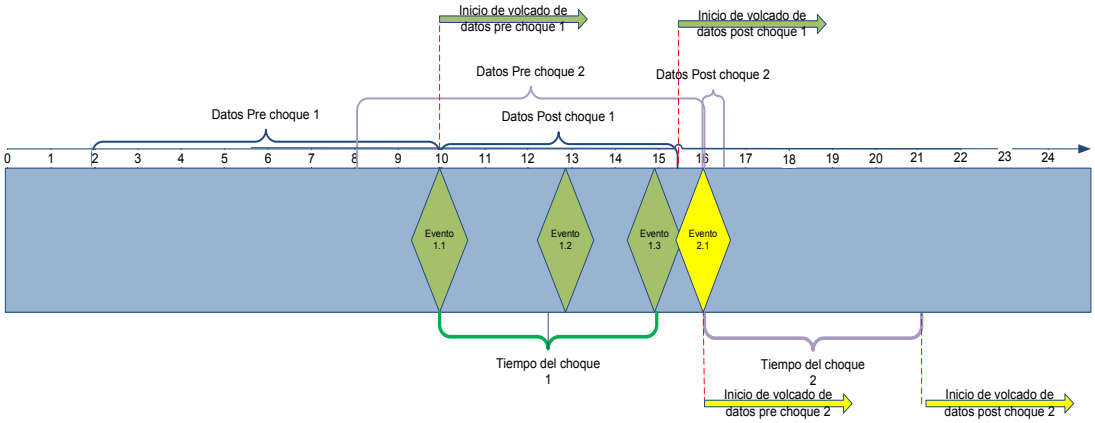

Figura 24: Escenario 3 - 2 choques

3. Interpretación de los escenarios

Dado un escenario, consideramos que una simulación con una determinada configuración de hardware termina satisfactoriamente cuando el proceso de volcado de datos finaliza correctamente con una pérdida de datos menor a 30%.

Ese porcentaje fue fijado arbitrariamente, considerando que un 70% de los datos de un choque son al menos suficientes para lograr realizar una reconstrucción aceptablemente fiel de un accidente.

En caso de que una simulación termine satisfactoriamente, se analiza si hubo datos perdidos provenientes del bus CAN o de los acelerómetros. Esto puede ocurrir por tamaño insuficiente del buffer de recepción de paquetes CAN, por capacidad insuficiente de procesamiento del CPU, por altas latencias de las memorias del sistema o por una combinación de todos estos factores.

4. Tablas de resultados

Tabla 1: Escenario 1

Velocidad del CPU (en MHz)	**Tamaño del buffer CAN**	**Tamaño de Memoria RAM mínimo**	**T. de Lectura/Escritura RAM (en ns)**	**Tiempo de escritura en memoria no volátil (por word)**	**RESULTADO**	**Tiempo de dump**
~150Mhz	16	64 kb	20/20	120 ns	Pasa. Pérdidas insignificantes.	5011 ms
~50Mhz	16	64 kb	20/20	120 ns	Pasa. Pérdidas menores al 2%	5035 ms
~10Mhz	16	128 kb	20/20	120 ns	Dudoso. Pierde aproximadamente 32% de los datos de los acelerómetros post choque	5125 ms
~150Mhz	16	64 kb	20/20	0,3125 ms	No pasa. Pierde más del 74% de los datos de los acelerómetros post choque.	
~150Mhz	16	1024 kb	20/20	0,3125 ms	No pasa. Pierde más del 74% de los datos de los acelerómetros post	

					choque.	
~150Mhz	16	64 kb	100/100	120 ns	Pasa. Pérdidas insignificantes.	5012 ms
~150Mhz	16	64 kb	10000/10000	120 ns	Pasa. Pérdidas insignificantes.	5093 ms
~50Mhz	16	64 kb	10000/10000	120 ns	Pasa. Menos del 5% de pérdidas.	5115ms

Tabla 2: Escenario 2

Velocidad del CPU (en MHz)	**Tamaño del buffer CAN**	**Tamaño de Memoria RAM mínimo**	**T. de Lectura/Escritura RAM (en ns)**	**Tiempo de escritura en memoria no volátil (por word)**	**RESULTADO**	**Tiempo de dump**
~150Mhz	16	64 kb	20/20	120 ns	Pasa. Pérdidas insignificantes.	5553 ms
~50Mhz	16	64 kb	20/20	120 ns	Pasa. Menos del 1% de pérdidas.	5750 ms
~25Mhz	16	64 kb	20/20	120 ns	Pasa. Menos del 2% de pérdidas.	6024 ms
~10Mhz	16	128 kb	20/20	120 ns	Pasa. 22% de valores de acelerómetro pre choque perdidos. 10,6% de valores de acelerómetro post	6944 ms

					choque perdidos.	
~10Mhz	16	64 kb	20/20	120 ns	No pasa.	
~150Mhz	16	64 kb	20/20	0,3125 ms	No pasa. Más del 61% de los valores del acelerómetro post choque perdidos.	
~150Mhz	16	1024 kb	20/20	0,3125 ms	No pasa. Más del 55% de los valores del acelerómetro post choque perdidos.	
~150Mhz	16	64 kb	100/100	120 ns	Pasa. 0,12% de valores de acelerómetro post choque perdidos.	5553 ms
~150Mhz	16	64 kb	10000/10000	120 ns	Pasa. Menos del 2% de pérdidas.	6264 ms
~50Mhz	16	64 kb	10000/10000	120 ns	Pasa. Menos del 2% de pérdidas.	6428 ms

Tabla 3: Escenario 3

Velocidad del CPU (en MHz)	**Tamaño del buffer CAN**	**Tamaño de Memoria RAM mínimo**	**T. de Lectura/Escritura RAM (en ns)**	**Tiempo de escritura en memoria no volátil (por word)**	**RESULTADO**	**Tiempo de dump**
~150Mhz	16	64 kb	20/20	120 ns	Pasa. Menos del 15% de pérdidas.	5012 ms
~50Mhz	16	64 kb	20/20	120 ns	Pasa. Menos del 15% de pérdidas.	5036 ms
~10Mhz	16	64 kb	20/20	120 ns	No Pasa. Más del 65% de los datos pre-crash del acelerómetro perdidos.	5140 ms
~150Mhz	16	64 kb	20/20	0,3125 ms	No pasa.	
~150Mhz	16	1024 kb	20/20	0,3125 ms	No pasa.	
~150Mhz	16	64 kb	100/100	120 ns	Pasa. Menos del 15% de pérdidas.	5013ms
~150Mhz	16	64 kb	10000/10000	120 ns	Pasa. Menos del 18% de pérdidas.	5094ms
~50Mhz	16	64 kb	10000/10000	120 ns	Pasa. Pierde 19% de los datos pre-crash	5115ms

					de los acelerómetros	

5. Análisis de resultados

Escenario 1

En este caso, dado que ocurre un único choque de un sólo evento, la principal característica que se evalúa es la capacidad del sistema de capturar los datos provenientes de los sensores y de volcar a memoria no volátil los datos capturados. Como se vuelcan los datos de un sólo evento, la cantidad de información a volcar es la mínima de todos los casos posibles.

Dados los resultados de las simulaciones, se puede observar que en caso de poseer una memoria no volátil de baja velocidad como por ejemplo una EEPROM, no es posible lograr un bajo porcentaje de datos post choque perdidos cuando los datos pre choque comienzan a volcarse inmediatamente luego de detectar la ocurrencia de un evento que supera el umbral disparador y que se cumple lo establecido en el estándar con respecto a cuándo guardar los datos en memoria permanente.

Esa pérdida de datos se debe a que se realiza el volcado de información pre choque al mismo tiempo que se reciben datos post choque que también deberán ser posteriormente guardados en memoria permanente. Al demorarse la memoria no volátil en la escritura, el microcontrolador queda bloqueado y no puede procesar los paquetes de datos que están llegando hasta la caja por el canal CAN y desde los acelerómetros.

Una solución a este problema, incumpliendo parcialmente el estándar, consiste en demorar el volcado de datos, tanto pre como post choque, hasta que éste se considere finalizado.

Según el estándar un choque puede durar hasta 5 segundos. Si se configura la caja para que el volcado comience luego de que transcurra ese espacio de tiempo, no se pierden datos pero la finalización del proceso se prolongará un poco más en el tiempo que antes. Además, será necesario contar una mayor cantidad de de memoria RAM para evitar que los datos de los buffers circulares sean sobrescritos antes de ser volcados. Los requisitos mínimos y resultados de simulaciones de este caso son los siguientes:

VOLCADO CON RETRASO					
CPU	LATENCIA NV MEM	TAMAÑO RAM	RESULTADO	TIEMPO	% PÉRDIDA
≥10Mhz	0,3125ms	≥128kb	Pasa	4034ms a 5129ms	≤10%

Tabla 4: Volcado con retraso de los datos del escenario 1

Por otro lado si, omitiendo la sugerencia del estándar, se limita la funcionalidad de la caja negra a que almacene la información de sólo un evento por choque, se puede configurar para que el volcado de datos se inicie quinientos milisegundos posteriores al evento, que es el momento en el cual se completa la captura de datos requerida para el acelerómetro. De este modo, si bien el proceso de volcado toma la misma cantidad de milisegundos, es menor el tiempo que transcurre desde la ocurrencia del evento hasta la finalización del volcado de información en memoria permanente.

Escenario 2

Este escenario es el peor caso de un choque multievento. No sólo se simula la ocurrencia de tres eventos, sino que la distancia en tiempo entre el primero y el tercero (4950 ms) es casi la máxima admitida por el estándar (5000 ms). De este modo se maximiza la cantidad de información que debe ser almacenada en memoria permanente. Esto incluye todos los datos recibidos entre el primer y último evento más todos aquellos que se recibieron antes del primero y después del último según los períodos de muestreo establecidos en el estándar.

Análogamente a lo que ocurre en el escenario 1, si se configura la caja negra con una memoria no volátil de baja velocidad, se pierden la mayoría de los datos. Sin embargo, como en este escenario el volumen de datos a guardar en memoria permanente es mucho mayor, si se configura la caja para comenzar el volcado de datos al finalizar el choque, este proceso va a requerir entre 20 y 24 segundos y entre 256kb y 512kb de memoria RAM dependiendo de la velocidad del CPU para copiar todos los datos a memoria permanente.

Por otro lado, si se configura la caja con una memoria no volátil de más velocidad, se requieren apenas 64kb o 128kb de memoria RAM según la velocidad del CPU. Con una frecuencia de 10Mhz se genera una pérdida de datos significativa pero

aceptable (alrededor del 10%) y con CPUs de frecuencias a partir de los 25Mhz se obtiene una pérdida de datos casi nula.

Escenario 3

En este caso, se representa un escenario similar al 2 con la diferencia de que los airbags son detonados recién al producirse un nuevo choque inmediatamente luego de que finaliza el primero.

Por ese motivo, según establece el estándar, los datos del primer choque son descartados y sólo deben guardarse los del segundo.

La dificultad de este caso reside en que los datos pre-choque del segundo choque que la caja debe guardar en memoria permanente son capturados al mismo tiempo que se realiza el volcado de los datos del primer choque.

De este modo, no sólo se comprometen los datos post-choque como en el escenario 2 sino que también puede haber pérdida de datos pre-choque.

En este caso, retrasar el proceso de volcado no es una alternativa dado que siempre existirá la posibilidad de que ocurra posteriormente un nuevo choque cuyos datos pre-choque se solapen con el volcado de datos anterior.

6. Conclusiones del análisis

Dados los análisis de los escenarios planteados, se puede afirmar que para la mayoría de los casos, si se desea concretar el volcado de información a memoria permanente en el menor tiempo posible y sin pérdidas, es necesario contar con una memoria no volátil de una velocidad considerable, mayor a la de las EEPROM tradicionales.

Sin embargo, si por razones de costos se deben utilizar EEPROMs, puede retrasarse el inicio del proceso de volcado, corriendo el riesgo de sufrir la pérdida de un cierto porcentaje de datos en caso de falla en el suministro de energía, dado el largo tiempo que toma este proceso con este tipo de memorias. Además, también puede haber pérdida de datos en el caso de un siniestro que involucre más de tres eventos distribuidos en más de un choque, como ocurre en el escenario 3, aunque este tipo de accidentes es poco probable [NHT00].

Por otro lado, contando con una memoria no volátil de alta velocidad de escritura, como por ejemplo una F-RAM [RAM10], el factor que pasa a ser el más influyente en la cantidad de datos perdidos es la frecuencia del procesador. Según los resultados obtenidos en las simulaciones y plasmados en las tablas, la configuración mínima que devuelve resultados aceptables es la siguiente:

Frecuencia del CPU	50Mhz
Memoria RAM	64kb
Latencia de Memoria no volátil	120ns
Tamaño de Buffer CAN del CPU	8 paquetes

Capítulo V – Conclusiones y Trabajos a Futuro

Conclusiones

La adopción de TLM, utilizando un lenguaje de descripción de hardware como SystemC provee una metodología altamente flexible para el desarrollo de sistemas embebidos de tiempo real. El modelo ejecutable temprano, junto a un adecuado nivel de abstracción, permite a los diseñadores realizar una verificación funcional antes de realizarse una implementación.

La principal ventaja adquirida en este proyecto de las simulaciones es que proveyeron retroalimentación temprana de la performance del sistema y permitió probar la funcionalidad bajo diferentes y diversas condiciones.

Los resultados obtenidos de este proyecto nos permitieron prever una especificación muy aproximada de los requerimientos acerca de los componentes de hardware. Además, la funcionalidad implementada está probada de estar acorde al estándar IEEE.

En una posible próxima etapa, este modelo puede refinarse en un modelo RTL para ser finalmente desplegada en una implementación de hardware real.

Este desarrollo demostró la posibilidad de realizar un EDR que cumpla con el estándar del IEEE-1616. Además, puede ser el punto de partida para el desarrollo de dispositivos costo-efectivos para suministrar a la industria automotriz con EDRs estandarizados.

Posibles desarrollos a futuro

El documento nos presenta un futuro abierto a más de un posible desarrollo a partir de lo realizado. Cabe mencionar alguno de ellos:

Debido a que nuestra implementación se basó en simular una caja negra en un lenguaje de medio nivel como lo es C++, carece de elementos visuales amenos a los potenciales usuarios. Se podría desarrollar una interfaz de interacción en tiempo real

con las simulaciones que sea más amigable al usuario, que muestre los elementos y sus variaciones temporales, así como también la posibilidad de ingresar manualmente modificaciones a estos.

Habiendo mencionado el hecho de que nuestro modelo se basaba en una técnica de modelado e mayor nivel de abstracción que RTL, podría considerarse el hecho de realizar alguna implementación más concreta y de bajo nivel, de modo de facilitar la transición al modelo real. Por otro lado, vale la pena mencionar que se ha presentado un estándar llamado AUTomotive Open System ARchitecture (AUTOSAR), que es una arquitectura de software abierta y estandarizada desarrollada conjuntamente por fabricantes de automóviles, proveedores y desarrolladores de herramientas de software. AUTOSAR provee herramientas de modelado con un mayor nivel de abstracción, lo que trae como beneficios desarrollos más rápidos, reusables y de costos reducidos.

El modelo propuesto puede extenderse a otros ámbitos. La arquitectura desarrollada permite que no sólo las cajas negras sean el dominio sobre las que se aplican. Se consideran demás dispositivos electrónicos que reciben continuamente datos de sensores y envían otros a actuadores, como lo podrían ser hoy en día los celulares, PDAs, etc. De esta manera, se podrían reutilizar varios elementos y acelerar el ciclo de desarrollo de productos.

1. AUTOSAR

El objetivo que se intenta lograr con la aplicación de AUTOSAR es la creación de estándares abiertos para arquitecturas de componentes electrónicos del sector automotriz, proveyendo infraestructura básica en base a módulos, interfaces de usuario y control de dominios. Esto abarca la estandarización de funciones elementales del sistema, portabilidad entre vehículos y plataformas, portabilidad entre redes de interconexión, integración de múltiples proveedores y mantenimiento y evolución del software en el ciclo de vida del vehículo.

1.1. Áreas de trabajo

Se desglosan tres áreas de trabajo principales.

Arquitectura: Una arquitectura de software de tres niveles incluyendo una pila de módulos básicos (BSW) que proveen una plataforma para desarrollo de aplicaciones con independencia del hardware.

Metodología: Intercambio de formatos o templates de descripción para permitir un proceso natural de configuración de la pila de módulos básicos y una fluida integración del software de aplicación en las Electronic Control Unit (ECU). AUTOSAR incluye una metodología para usar este framework.

Interfaces de aplicación: Especificación de interfaces de las aplicaciones automotrices típicas de todos los dominios en términos de sintaxis y semántica que deben actuar como estándar para los softwares de aplicación.

2. Derivación a chip de silicona

Actualmente la implementación en SystemC no permite una derivación 100% fiel a un SoC. Esto eventualmente puede solucionarse mediante productos derivadores tales como el software C-to-Sillicon de la desarrolladora Cadence. Con estos productos se puede generar código RTL a partir de código C/SystemC, realizando una síntesis de alto nivel.

Acrónimos

ABS = Anti-Lock Brake System / Sistema Antibloqueo de Frenos

ACM = Airbag Control Module / Módulo de Control del Airbag

AUTOSAR = AUTomotive Open System ARchitecture / Arquitectura de Sistemas Abierta para el sector del Automóvil

ADC = Analog to Digital Converter / Conversor Analógico-Digital

CAN = Controller Area Network / Red de Área de Controladores

CDR = Crash Data Retrieval / Recuperación de Datos de Accidentes

DAC = Digital to Analog Converter / Conversor Digital-Analógico

DLC = Diagnostic Link Connector / Conector de Diagnóstico

ECU = Electronic Control Unit / Unidad de Control Electrónico

EDR = Event Data Recorder / Grabador de Datos de Eventos

FDR = Flight Data Recorder / Grabador de Datos de Vuelo

IEEE = The Institute of Electrical and Electronics Engineers / Instituto de Ingenieros Eléctricos y Electrónicos

NCHRP = National Cooperative Highway Research Program / Programa Nacional Cooperativo de Investigación Vial

NHTSA = National Highway Traffic Safety Administration / Administración Nacional de Seguridad de Tráfico Vial

OEM = Original Equipment Manufacturer / Fabricante de Equipos Originales

OSCI = Open SystemC Initiative / Iniciativa Abierta de SystemC

RAM = Random Access Memory / Memoria de Acceso Aleatorio

RCM = Restraint Control Module / Módulo de Control de Contención

RTL = Register Transfer Level / Nivel de Transferencia de Registro

SAE = Society of Automotive Engineers / Sociedad de Ingenieros Automotores

SDM = Sensing Diagnostic Module / Módulo de Diagnóstico y Sensado

TLM = Transaction Level Modelling / Modelado a Nivel de Transacción

Lista de Figuras

Lista de Tablas

Referencias

[BLA04] **Black, David C. and Donovan, Jack.** *SystemC: From the Ground Up.* 2004.

[CAM07] **Campbell, Neil.** *The Evolution Of Flight Data Analysis.* s.l. : Australian Society of Air Safety Investigators, 2007.

[COR02] **Corrigan, Steve.** *Introduction to Controller Area Network.* s.l. : Texas Instrument, 2002.

[DAV10] **Davis Instruments.** *Vehicle Tracking and Monitoring Solutions for Small Business, Fleets, Teens, and Hobbyists by Davis*. Professional Quality Weather Stations and Software, Automotive Tracking, and Marine Accessories by Davis. Web. Mayo 2010. http://www.davisnet.com/drive/products/carchip_products.asp

[DOU10] **Doulos.** "Deprecated Features in SystemC 2.2." Web. Mayo 2010. http://www.doulos.com/knowhow/systemc/deprecated/ .

[GHE05] **Ghenassia, Frank.** *Transaction Level Modeling With SystemC - TLM Concepts and Application for Embedded Systems.* s.l. : Springer, 2005. ISBN 13 978-0-387-26233-4.

[GRO05a] **Group, IEEE.** *IEEE Standard for Motor Vehicle Event Data Recorders - 1616-2004.* 2005.

[GRO05b] **Group, IEEE.** *IEEE Standard SystemC Language Reference Manual - 1666-2005.* s.l. : IEEE Computer Society, 2005.

[HAI01] **Haight, W.R. Rusty.** *Automobile Event Data Recorder - Evolution, Data And Reliability.* s.l. : Collision Safety Institute, 2001.

[KOP00] **Kopetz, Hermann.** *Software Engineering for Real-Time: A Roadmap.* s.l. : ACM, 2000.

[LAW06] **Law, Averill M.** *Simulation Modeling and Analysis, Fourth Edition.* s.l. : McGraw-Hill, 2006.

[LEF02] **LeFort, Bob and Ries, Bob.** *Taking the Uncertainty Out of Thermocouple Temperature Measurement.* s.l. : Analog Devices, 2002.

[LEY08] *Ley 26.363 - Créase la Agencia Nacional de Seguridad Vial. Funciones. Modificaciones a la Ley Nº 24.449. Disposiciones Transitorias.* 2008.

[MCH09] **McHaney, Robert.** *Understanding Compuer Simulation.* s.l. : Book Boon, 2009.

[MEL96] **Melvin, John W., et al.** *Investigation of Indy Car Crashes Using Impact Recorders.* s.l. : Society of Automotive Engineers, 1996.

[NHT00] **NHTSA.** *NASS/CDS database guide.* 2000.

[OPE09] **Open SystemC Initiative.** *OSCI TLM-2.0 Language Reference Manual.* 2009.

[OPE10] **Open SystemC Initiative** (OSCI) – Home. "About OSCI - Open SystemC Initiative (OSCI)." Web. 06 Mayo 2010. <http://www.systemc.org/about/>.

[PAR03] **Park, John and MacKay, Steve.** *Practical Data Acquisition For Instrumentation And Control Systems.* s.l. : Newnes Press, 2003.

[PHI00] **Philips Semiconductors.** *P8xC591 Single-chip 8-bit microcontroller with CAN controller.* 2000.

[RAM10] **Ramtron.** Ramtron Products. *Ramtron Products - Nonvolatile Memory.* Web. Mayo 2010. http://www.ramtron.com/products/nonvolatile-memory/.

[VDO10] **VDO.** *Accident Data Recorder - UDS*. Web. Mayo 2010. http://www.vdo.com/generator/www/com/en/vdo/main/products_solutions/commercial_vehicles/tachographs/uds/uds_en.html

Otras fuentes consultadas

1. **Ching-Yao, Chan.** Trends in Crash Detection and Occupant Restraint Technology. Proceedings of the IEEE, vol. 95, no. 2, pp 388-396 (2007)

2. **C. Shelton, C. Martin.** Using Models to Improve the Availability of Automotive Software Architectures. Proceeding of Software Engineering for Automotive Systems. ICSE Workshop, SEAS'07, 20-26 May 2007, pp 9-15, 2007.

3. **Cai, L., Gajski, D.** Transaction-level Modeling: an Overview. Proc. of the Int. Conf. on Hardware/Software Codesign and System Synthesis, pp. 19–24. ACM Press, New York (2003)

4. **Shuqing Zhao, D. D. Gajski.** Defining an enhanced RTL semantics. Proceeding of Design, Automation and Test in Europe, pp 548-552 (2005).

5. **J. Ehrlichl, A. Zerroukill, N. Demassieux.** Distributed architecture for data acquisition: a generic model. Proc. of the IEEE Instrumentation and Measurement Technology Conference, pp 1180-1185 (1997)

6. **Bosch, Robert.** CAN Specification Version 2.0. 1991.

7. **Navet, Nicolas and Simonot-Lion, Francoise.** Automotive Embedded Systems Handbook (2009)

8. **NHTSA, Department of Transportation.** Event Data Recorders; Final Rule (2006)

Anexo A – Elementos del IEEE 1616

Nombre del elemento: Tasa de giro (Yaw rate)	
Descripción: Velocidad de rotación con respecto al eje vertical (eje z) del vehículo en los momentos circundantes al event trigger	
Definición de datos/convención de signos: ±180°/s	**Unidad de medida:** °/s
Resolución: 0.1°/s	**Precisión:** ±0.1°/s
Velocidad de muestreo: 10 muestras por segundo de -8 s a -1 s antes del evento. 1000 muestras por segundo de -1 s antes del evento disparador hasta 5 s posteriores al evento disparador.	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 280 bytes desde -8 hasta -1 24000 bytes desde -1 hasta 5

Nombre del elemento: Velocidad de las ruedas (cada rueda)	
Descripción: Velocidad individual de las ruedas en los momentos circundantes al evento disparador	
Definición de datos/convención de signos: Valores positivos representan giro hacia adelante. Rango: -80km/h hasta 240 km/h Casos especiales: 998 = datos inválidos, 999 = datos no disponibles	**Unidad de medida:** Km/h
Resolución: 0.32 km/h por bit	**Precisión:** ±5.12 km/h
Velocidad de muestreo: 100 muestras por segundo (100 Hz)	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Integer	**Tamaño total de datos:** 6400bytes

Nombre del elemento: Hora del evento disparador
Descripción: Hora, minutos y segundos en las que ocurrió el evento disparador

Definición de datos/convención de signos: No aplicable	**Unidad de medida:** No aplicable
Resolución: 1 s	**Precisión:** ±1 s
Velocidad de muestreo: Una sola vez en el momento del evento disparador	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Long integer (4 bytes) usando “hhmmss” en formato UTC con “hh” = hora, “mm” = minutos y “ss” = segundos	**Tamaño total de datos:** 4 bytes

Nombre del elemento: Fecha en que ocurrió el evento disparador	
Descripción: Día, mes y año en que ocurrió el evento disparador	
Definición de datos/convención de signos:	**Unidad de medida:** No aplicable
Resolución: 1 día	**Precisión:** ±15 minutos
Velocidad de muestreo: Una sola vez en el momento del evento disparador	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Long integer (4 bytes) usando “ddmmyy” donde “dd” = día, “mm” = mes y “yy” = año	**Tamaño total de datos:** 4 bytes

Nombre del elemento: Tiempo transcurrido entre eventos	
Descripción: Tiempo transcurrido entre dos eventos de impacto	
Definición de datos/convención de signos: El valor es siempre positivo	**Unidad de medida:** milisegundo
Resolución: 1 ms	**Precisión:** ±1 ms
Velocidad de muestreo: N/A	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Integer (4 bytes)	**Tamaño total de datos:** 4 bytes

Nombre del elemento: Ángulo de giro del volante	
Descripción: Posición y dirección (sentido horario o antihorario) del volante en referencia a la posición neutra en los momentos circundantes al evento disparador	
Definición de datos/convención de signos: Rango: -720° (antihorario) hasta 720° (horario)	**Unidad de medida:** grado

0 = volante en posición neutra Casos especiales: 998 = datos inválidos, 999 = datos no disponibles	
Resolución: 1°	**Precisión:** ±1°
Velocidad de muestreo: 10 muestras por segundo	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Integer (2 bytes)	**Tamaño total de datos:** 260 bytes

Nombre del elemento: Clasificación de ocupante del asiento del pasajero	
Descripción: Presencia y clasificación del ocupante del asiento del pasajero según tamaño y peso.	
Definición de datos/convención de signos: Determinado por el fabricante, pero de acuerdo al FMVSS 208 y FARS	**Unidad de medida:** No aplicable
Resolución: Determinada por los diseñadores del sistema del vehículo	**Precisión:** Debe ser compatible con la performance del sistema de sujeción
Velocidad de muestreo: Debe ser establecida por el fabricante como parámetro de la performance del sistema de sujeción	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: 4 bytes	**Tamaño total de datos:** Según implementación

Nombre del elemento: Posición del asiento del pasajero	
Descripción: Posición del asiento del pasajero sobre los rieles sobre los que está montado en los momentos circundantes al evento disparador	
Definición de datos/convención de signos: 0 = posición más hacia adelante 1 = posición más hacia atrás 254 = datos inválidos 255 = datos no disponibles	**Unidad de medida:** Milímetro
Resolución: Rango: 25 mm a 300 mm con 1.25 mm de resolución	**Precisión:** ±1.25 mm
Velocidad de muestreo: Una muestra por segundo antes del evento disparador,	**Tiempo de muestreo:** -8 s disparador +5 s

10 muestras por segundo luego del evento	
Formato de datos: Byte	**Tamaño total de datos:** 58 bytes

Nombre del elemento: Posición del asiento del conductor	
Descripción: Posición del asiento del conductor sobre los rieles sobre los que está montado en los momentos circundantes al evento disparador	
Definición de datos/convención de signos: 0 = posición más hacia adelante 1 = posición más hacia atrás 254 = datos inválidos 255 = datos no disponibles	**Unidad de medida:** Milímetro
Resolución: Rango: 25 mm a 300 mm con 1.25 mm de resolución	**Precisión:** ±1.25 mm
Velocidad de muestreo: Una muestra por segundo antes del evento disparador, 10 muestras por segundo luego del evento	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Byte	**Tamaño total de datos:** 58 bytes

Nombre del elemento: Tasa de vuelco	
Descripción: Velocidad de rotación alrededor del eje longitudinal en los momentos circundantes al evento disparador.	
Definición de datos/convención de signos: ±180°/s convención de signo (SAE J211/1-2003 7.2) sistema de coordenadas (SAE 670-1976 [B68], Fig. 2)	**Unidad de medida:** grados por segundo
Resolución: 0.1°/s	**Precisión:** ±0.1°/s
Velocidad de muestreo: 10 muestras por segundo de -8 s a -1 s antes del evento. 1000 muestras por segundo de -1 s antes del evento disparador hasta 5 s posteriores al evento disparador.	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos:	**Tamaño total de datos:**

Punto flotante (4 bytes)	24280 bytes

Nombre del elemento: Aceleración – Pre-choque – Eje x	
Descripción: Describe el movimiento longitudinal del vehículo antes del disparador del evento	
Definición de datos/convención de signos: Rango de datos: -100g_n a +100 g_n Casos especiales: 998 = datos inválidos 999 = datos no disponibles	**Unidad de medida:** g_n = 9.806 65 m/s^2.
Resolución: 0.048 g_n por bit	**Precisión:** ±0.01 g_n
Velocidad de muestreo: 10 muestras por segundo (mínimo), es decir ≥ 10 hz	**Tiempo de muestreo:** –8 s disparador 0 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 320bytes

Nombre del elemento: Aceleración – Pre-choque – eje y	
Descripción: Describe el movimiento lateral del vehículo antes del disparador del evento	
Definición de datos/convención de signos: Rango de datos: -100g_n a +100 g_n Casos especiales: 998 = datos inválidos 999 = datos no disponibles	**Unidad de medida:** g_n = 9.806 65 m/s^2.
Resolución: 0.048 g_n por bit	**Precisión:** ±0.01 g_n
Velocidad de muestreo: 10 muestras por segundo (mínimo), es decir ≥ 10 hz	**Tiempo de muestreo:** –8 s disparador +0 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 320bytes

Nombre del elemento: Aceleración – Choque – Eje x	
Descripción: Describe el movimiento longitudinal del vehículo durante el accidente	
Definición de datos/convención de signos: Rango de datos: -100g_n a +100 g_n Casos especiales: 998 = datos inválidos	**Unidad de medida:** g_n = 9.806 65 m/s^2.

999 = datos no disponibles	
Resolución: 0.048 g_n por bit	**Precisión:** ±0.01 g_n
Velocidad de muestreo: 1000 muestras por segundo (mínimo), es decir ≥ 1000 hz	**Tiempo de muestreo:** disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 20000 bytes

Nombre del elemento: Aceleración – Choque – eje y	
Descripción: Describe el movimiento lateral del vehículo durante el accidente	
Definición de datos/convención de signos: Rango de datos: -100 g_n a +100 g_n Casos especiales: 998 = datos inválidos 999 = datos no disponibles	**Unidad de medida:** g_n = 9.806 65 m/s^2.
Resolución: 0.048 g_n por bit	**Precisión:** ±0.01 g_n
Velocidad de muestreo: 1000 muestras por segundo (mínimo), es decir ≥ 1000 hz	**Tiempo de muestreo:** disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 20000 bytes

Nombre del elemento: Aceleración – Choque – eje z	
Descripción: Ayuda a describir el movimiento del vehículo en el eje vertical durante el accidente	
Definición de datos/convención de signos: Rango de datos: -100 g_n a +100 g_n Casos especiales: 998 = datos inválidos 999 = datos no disponibles	**Unidad de medida:** g_n = 9.806 65 m/s^2.
Resolución: 0.048 g_n por bit	**Precisión:** ±0.01 g_n
Velocidad de muestreo: 1000 muestras por segundo (mínimo), es decir ≥ 1000 hz	**Tiempo de muestreo:** disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 20000 bytes

Nombre del elemento: Supresión del airbag, airbag del pasajero
Descripción:

Este dato registra si el sistema de airbag del pasajero fue colocado manualmente en posición de encendido “on” o apagado “off” en el tiempo colindante al disparo del evento	
Definición de datos/convención de signos: 0 = no activo 1 = activo Casos especiales: 3 = datos inválidos 4 = datos no disponibles	**Unidad de medida:** N/D
Resolución: N/D	**Precisión:** N/D
Velocidad de muestreo: 1 muestra por segundo, es decir ≥ 1 hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: 1 Byte	**Tamaño total de datos:** 13 bytes

Nombre del elemento: Airbag, ejecución del airbag frontal del conductor, tiempo	
Descripción: Identifica cuando se activó el airbag del conductor durante el accidente	
Definición de datos/convención de signos: N/D	**Unidad de medida:** Tiempo desde que se detectó el impacto
Resolución: 1 ms	**Precisión:** ±001 s
Velocidad de muestreo: N/D	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Entero	**Tamaño total de datos:** 2 bytes

Nombre del elemento: Airbag, ejecución del airbag lateral del conductor, tiempo	
Descripción: Identifica cuando se activó el airbag lateral del conductor durante el accidente	
Definición de datos/convención de signos: N/D	**Unidad de medida:** Tiempo desde que se detectó el impacto
Resolución: 1 ms	**Precisión:** ±001 s
Velocidad de muestreo: N/D	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Entero	**Tamaño total de datos:** 2 bytes

Nombre del elemento: Airbag, ejecución del airbag frontal del pasajero, tiempo

Descripción: Identifica cuando se activó el airbag del pasajero durante el accidente	
Definición de datos/convención de signos: N/D	**Unidad de medida:** Tiempo desde que se detectó el impacto
Resolución: 1 ms	**Precisión:** ±001 s
Velocidad de muestreo: N/D	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Entero	**Tamaño total de datos:** 2 bytes

Nombre del elemento: Airbag, ejecución del airbag lateral del pasajero, tiempo	
Descripción: Identifica cuando se activó el airbag lateral del pasajero durante el accidente	
Definición de datos/convención de signos: N/D	**Unidad de medida:** Tiempo desde que se detectó el impacto
Resolución: 1 ms	**Precisión:** ±001 s
Velocidad de muestreo: N/D	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Entero	**Tamaño total de datos:** 2 bytes

Nombre del elemento: Actividad de frenado, ABS	
Descripción: Identifica la actividad del ABS en el tiempo colindante al disparo del evento	
Definición de datos/convención de signos: Rango de datos: 0 = no activo 1 = activo Casos especiales: 2 = datos no válidos 3 = datos no disponibles	**Unidad de medida:** ABS activo (on) o no activo (off)
Resolución: Resolución on/off	**Precisión:** N/D
Velocidad de muestreo: 2 muestras por segundo, es decir ≥ 2 Hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: 1 Byte por muestra	**Tamaño total de datos:** 26 bytes

Nombre del elemento: Actividad de frenado, freno de servicio

Descripción: Identifica la actividad del freno de servicio en el tiempo colindante al disparo del evento	
Definición de datos/convención de signos: Rango de datos: 0 = no activo 1 = activo Casos especiales: 2 = datos no válidos 3 = datos no disponibles	**Unidad de medida:** Switch de la luz de freno activo (on) o no activo (off)
Resolución: Resolución on/off	**Precisión:** N/D
Velocidad de muestreo: 2 muestras por segundo, es decir ≥ 2 Hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: 1 Byte por muestra	**Tamaño total de datos:** 26 bytes

Nombre del elemento: Control de estabilidad de frenado (conectado/desconectado)	
Descripción: Identifica el estado del control de estabilidad en el tiempo colindante al disparo del evento	
Definición de datos/convención de signos: Rango de datos: 0 = no conectado 1 = conectado 2 = activo Casos especiales: 3 = datos no válidos 4 = datos no disponibles	**Unidad de medida:** Conectado (on) o no conectado(off)
Resolución: Resolución on/off	**Precisión:** 10 ms
Velocidad de muestreo: 2 muestras por segundo, es decir ≥ 2 Hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: 1 Byte por muestra	**Tamaño total de datos:** 26 bytes

Nombre del elemento: Cambio de velocidad (eje x)	
Descripción: Identifica el cambio de velocidad en el eje x durante el evento del impacto para entender la magnitud del impacto. En conjunto con el cambio de velocidad del eje y, puede determinar la principal dirección de la fuerza (PDOF)	
Definición de datos/convención de signos: Convención de signo: Ver SAE J211/1 Sistema de coordinadas: Ver SAE J670-	**Unidad de medida:** Kilómetro por hora (km/h) 1 km/h = 1000 m/h = ~16.7 m/min = ~0.28 m/s

1976	1 km/h = ~0.62 mi/h
Resolución: 1 km/h	**Precisión:** ±1 km/h
Velocidad de muestreo: ≥ 500 Hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 26000 bytes

Nombre del elemento: Cambio de velocidad (eje y)	
Descripción: Identifica el cambio de velocidad en el eje y durante el evento del impacto para entender la magnitud del impacto. En conjunto con el cambio de velocidad del eje x, puede determinar PDOF	
Definición de datos/convención de signos: Convención de signo: Ver SAE J211/1 Sistema de coordinadas: Ver SAE J670-1976	**Unidad de medida:** Kilómetro por hora (km/h) 1 km/h = 1000 m/h = ~16.7 m/min = ~0.28 m/s 1 km/h = ~0.62 mi/h
Resolución: 1 km/h	**Precisión:** ±1 km/h
Velocidad de muestreo: ≥ 500 Hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 26000 bytes

Nombre del elemento: Cambio de velocidad (eje z)	
Descripción: Identifica el cambio de velocidad en el eje z durante el evento del impacto para entender la magnitud del impacto	
Definición de datos/convención de signos: Convención de signo: Ver SAE J211/1 Sistema de coordinadas: Ver SAE J670-1976	**Unidad de medida:** Kilómetro por hora (km/h) 1 km/h = 1000 m/h = ~16.7 m/min = ~0.28 m/s 1 km/h = ~0.62 mi/h
Resolución: 1 km/h	**Precisión:** ±1 km/h
Velocidad de muestreo: ≥ 500 Hz	**Tiempo de muestreo:** –8 s disparador +5 s
Formato de datos: Punto flotante	**Tamaño total de datos:** 26000 bytes

Nombre del elemento:

Principal dirección de la fuerza	
Descripción: Reconstrucción post-choque a través de la reducción de datos y el análisis	
Definición de datos/convención de signos: Convención de signo: Ver SAE J211/1 Sistema de coordinadas: Ver SAE J670-1976	**Unidad de medida:** Kilómetro por hora (km/h) 1 km/h = 1000 m/h = ~16.7 m/min = ~0.28 m/s 1 km/h = ~0.62 mi/h
Resolución: Limitada por los datos descargados del EDR	**Precisión:** Limitado por los datos de aceleración disponibles (pueden no ser determinísticos a menos que ambas 3 aceleraciones lineares y 3 aceleraciones rotacionales sean medidas)
Velocidad de muestreo: Calculada por el EDR	**Tiempo de muestreo:** Derivado de las aceleraciones medidas durante los períodos de grabación pre y post choque
Formato de datos: N/D	

Nombre del elemento: Datos del motor, revoluciones por minuto	
Descripción: Velocidad del motor, expresado en revoluciones por minuto (r/min). Medida de velocidad del motor. La ECU del motor determina la velocidad del motor y presenta los datos en la red vehicular.	
Definición de datos/convención de signos: 0 a 12.000 r/min	**Unidad de medida:** r/min
Resolución: 0,25 (r/min)/bit	**Precisión:** ±1%
Velocidad de muestreo: 5 Hz	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Entero 2 bytes	

Nombre del elemento: Datos del motor, mariposa	
Descripción: Posición de la mariposa del motor, expresada en porcentaje de apertura total. La ECU del motor lee el sensor de posición de la mariposa y presenta estos datos en la red de datos del vehículo.	
Definición de datos/convención de	**Unidad de medida:**

signos: 0 = 0%, 250 = 100% Casos especiales: 254 = Datos inválidos, 255 = Datos no disponibles	**%** (Porcentaje)
Resolución: 0,4%/bit	**Precisión:** ±1%
Velocidad de muestreo: 4 muestras por segundo	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: 1 byte	

Nombre del elemento: Ciclos de encendido, evento	
Descripción: Indica el número de veces que se encendió el motor del automóvil comenzando desde el momento que se fabricó y hasta el momento en el que ocurrió el accidente	
Definición de datos/convención de signos: 0 = Sin encendido, 1= Un encendido … Casos especiales: 999,999,998 = Dato inválido 999,999,999 = Dato no disponible	**Unidad de medida:** Cada encendido
Resolución: 1 bit por encendido	**Precisión:** ±
Velocidad de muestreo: Una sola vez, en el evento	**Tiempo de muestreo:** En el momento del trigger
Formato de datos: Entero (4 bytes)	

Nombre del elemento: Ciclos de encendido, descarga	
Descripción: Indica el número de veces que se encendió el motor del automóvil comenzando desde el momento que se fabricó y hasta el momento en el que se descargaron los datos	
Definición de datos/convención de signos: 0 = Sin encendido, 1= Un encendido … Casos especiales: 999,999,998 = Dato inválido 999,999,999 = Dato no disponible	**Unidad de medida:** Cada encendido
Resolución: 1 bit por encendido	**Precisión:** ±
Velocidad de muestreo: Una sola vez, en la descarga	**Tiempo de muestreo:** A petición
Formato de datos: Entero (4 bytes)	

Nombre del elemento: Uso del cinturón de seguridad, Conductor	
Descripción: Conector del cinturón de seguridad del conductor se comunica con la ECM para determinar si el cinturón de seguridad está conectado o no.	
Definición de datos/convención de signos: 0=abrochado 1=desabrochado 254=datos inválidos 255 = datos no disponibles	**Unidad de medida:**
Resolución:	**Precisión:** ±
Velocidad de muestreo: 10 muestras por segundo	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Entero de 2 bytes	

Nombre del elemento: Uso del cinturón de seguridad, Conductor	
Descripción: Conector del cinturón de seguridad del conductor se comunica con la ECM para determinar si el cinturón de seguridad está conectado o no.	
Definición de datos/convención de signos: 0=abrochado 1=desabrochado 254=datos inválidos 255 = datos no disponibles	**Unidad de medida:**
Resolución:	**Precisión:** ±
Velocidad de muestreo: 10 muestras por segundo	**Tiempo de muestreo:** -8 s disparador +5 s
Formato de datos: Entero de 2 bytes	

Anexo B – Paper presentado en el CACIC 2009

Ver el archivo "*CACIC 2009 - Description of an Architecture for Critical Distributed and Real-Time Data Collection Applied to MVEDR.pdf"* adjunto.