Análisis y simulación de un protocolo de comunicaciones para una red de sensores: Aplicación en dispositivos de vuelo

Autor: Diego Omar Encinas

Director: R. Marcelo Naiouf

Codirector: Hugo D. Ramon

Tesis presentada para obtener el grado de Magister en Redes de Datos

Facultad de Informática – Universidad Nacional de La Plata

Febrero 2013

Agradecimientos

Principalmente a mis padres, gracias por todo lo que hicieron ya que esto es el resultado de su esfuerzo, trabajo y perseverancia.

A mis hermanos por su paciencia y apoyo, gracias Natalia por ser la correctora de todos los capítulos mientras cuidabas a Simón.

A la Abuelita porque siempre supo que terminaría satisfecho con este trabajo y que con su reciente partida se fue para quedarse eternamente.

A Emmanuel por su constante ánimo y ayuda en los momentos claves del trabajo de investigación, pero sobre todo por brindarme su amistad.

A Diego, mi compañero de box, que siempre estuvo dispuesto a escuchar mis problemas.

Al Colo por ayudarme desinteresadamente a aprender sobre el mundo de la programación orientada a objetos.

A Diego Alustiza, mi ex compañero en la Facultad de Ingeniería que inicialmente me motivó a iniciar esta tesis y por extensión a mis ex compañeros de trabajo del Centro de Investigaciones Ópticas.

A Hugo que con su experiencia me ayudó a sobrellevar, superar y aprender de los momentos más difíciles del trabajo de tesis.

A mi Director de tesis, Dr. Naiouf, quien rápidamente respondió a mis inquietudes y solucionó mis inconvenientes.

Al Director del III-LIDI, Ing. De Giusti, por darme la oportunidad de formarme como investigador y docente.

A todos mis compañeros del III-LIDI de la Facultad de Informática, que siempre me alentaron y prestaron su ayuda cuando la requerí.

A Jesús, mi Señor y Salvador, gracias por todo lo que me diste y por tu Paz que sobrepasa todo entendimiento humano.

"Si logras atravesar la oscuridad de la noche, habrá un día muy soleado"
José Gabriel Condorcanqui Noguera (Túpac Amaru II)

"La mejor forma de aburrir es contarlo todo"
François Marie Arouet (Voltaire)

Índice

Agradecimientos 1

Tabla de Figuras 7

Lista de acrónimos 9

1 Capítulo 1: Introducción 11

1.1 Introducción y motivación 11

1.2 Objetivos y alcance 13

1.3 Estructura del trabajo 14

2 Capítulo 2: Estado del Arte 16

2.1 Protocolos de comunicaciones en sistemas de tiempo real 16

2.1.1 Sistemas de tiempo real 16

2.1.2 Comunicaciones en sistemas de tiempo real distribuido 17

2.1.2.1 Clasificación 18

2.2 Sensores 19

2.2.1 Desarrollo de girómetros y su utilidad 19

2.3 Redes de Sensores 20

2.4 Sistemas de Comunicaciones Aeroespaciales 20

2.5 Simulación 22

2.5.1 Sistemas, Modelos y Simulación 23

2.5.2 Simulación de eventos discretos 24

2.5.2.1 Mecanismos de avance de tiempo 25

2.5.2.2 Componentes y organización de un modelo de simulación de eventos discretos 25

2.5.3 Construcción del modelo 26

2.5.4 Simulación en Sistemas Aeroespaciales 27

2.6 TLM y SystemC 28

2.6.1 Conceptos de TLM 28

2.6.2 Conceptos de SystemC 29

3 Capítulo 3: Descripción del problema 32

3.1 Experiencias previas 32

3.1.1 Ámbito de desarrollo de la implementación 32
3.1.2 Soluciones implementadas 33
3.1.2.1 Protocolo CAN 33
3.1.2.1.1 Conceptos básicos 33
3.1.2.1.2 Características 35
3.1.2.1.3 Estructura de mensajes 38
3.1.2.2 Soluciones de hardware en estos casos 39
3.1.3 Clasificación y tolerancia a fallas 40
3.2 Análisis de las posibles soluciones a los problemas abiertos 42
3.2.1 Tolerancia a fallas en la implementación 42
3.2.2 Simulación para acortar tiempos de desarrollo 44
3.2.2.1 Flujo de diseño tradicional 45
3.2.2.2 Flujo de diseño con herramientas de codiseño 45
3.2.2.3 Simulación funcional 47
4 Capítulo 4: Herramientas utilizadas para la propuesta de solución 48
4.1 Propuesta de solución 48
4.2 Implementación del sistema 48
4.2.1 Diseño hardware inicial 48
4.2.2 Lenguajes de diseño 50
4.3 Estructura de SystemC 51
4.3.1 Módulos 52
4.3.2 Procesos 52
4.3.3 Canales 53
4.3.4 Interfaces 53
4.3.5 Puertos 54
4.3.6 Scheduler (Planificador) 54
4.3.7 Eventos 54
4.3.8 Relojes 54
4.3.9 Dinámica de la simulación 54
4.3.10 Utilización de wait 55
4.3.11 Trabajos relacionados al tema 56
4.4 Requerimientos del sistema a modelar 57

4.5 Herramientas de desarrollo 57
4.5.1 Herramientas de hardware 57
4.5.2 Herramientas de Software 60
5 Capítulo 5: Diseño y desarrollo del modelo de simulación 61
5.1 Descripción del sistema a modelar 61
5.1.1 Nodos 62
5.1.2 Bus 64
5.1.3 Configuración del nodo 64
5.2 Conceptos de diseño 66
5.2.1 Restricciones de modelado 67
5.2.1.1 Análisis de planificación del protocolo CAN 67
5.2.1.2 Implicancias del *Jitter* en los tiempos de transmisión 69
5.2.1.3 Implicancias de la temperatura en el jitter. 69
5.2.1.4 Implicancias de la temperatura en el bus 70
5.3 Modelo de simulación 71
5.3.1 Modelo final 71
5.3.1.1 Composición 73
5.3.1.1.1 Microcontrolador 73
5.3.1.1.2 Controlador 73
5.3.1.1.3 Bus 74
5.3.1.2 Semántica 75
6 Capítulo 6: Experimentación y análisis de resultados 77
6.1 Experimentación con el prototipo 77
6.1.1 Métricas obtenidas 79
6.2 Experimentación con el modelo de simulación 80
6.2.1 Métricas obtenidas 83
6.3 Validación del modelo de simulación 84
6.3.1 Ejemplo de experimentación 85
7 Capítulo 7: Conclusiones 87
7.1 Alcance de la simulación 87
7.2 Líneas futuras de investigación 88
Anexo A: Trabajos Publicados 89

Anexo-B: Evolución del modelo de simulación 90
Bibliografía 94

Tabla de Figuras

Figura 1.1: Satélite SAC-D/Aquarius 11
Figura 1.2: Instrumentos del SAC-D/Aquarius 12
Figura 2.1: Comunicación disparada por eventos 18
Figura 2.2: Circuito óptico del IFOG 19
Figura 2.3: Conexionado de un terminal MIL-STD-1553 al bus 21
Figura 2.4: Red SpiceWire 21
Figura 2.5: Red CAN 22
Figura 2.6: Formas de estudiar un sistema 23
Figura 2.7: Modelos de abstracción TLM 28
Figura 2.8: Arquitectura del framework SystemC 30
Figura 3.1: Unidad de Medición Inercial 33
Figura 3.2: Modelo de Referencia OSI del protocolo CAN 35
Figura 3.3: Arbitraje en bus CAN 37
Figura 3.4: Trama CAN 38
Figura 3.5: Tramas estándar y extendida 39
Figura 3.6: Clasificación de fallas 40
Figura 3.7: Curva de la bañera 41
Figura 3.8: Flujo de diseño tradicional 45
Figura 3.9: Flujo de diseño con herramientas de codiseño 46
Figura 4.1: Prototipo de prueba 49
Figura 4.2: Tipos de redundancias de hardware 50
Figura 4.3: Componentes de SystemC 51
Figura 4.4: Kernel de simulación de SystemC 55
Figura 4.5: Entorno de desarrollo KEIL 58
Figura 4.6: Interfaz del analizador de tráfico CANBuilder 59
Figura 4.7: Visualizador de señales GTKWave 60
Figura 5.1: Fotografía del prototipo desarrollado e interfaz CAN-USB 61
Figura 5.2: Componentes del prototipo 62
Figura 5.3: Controlador CAN 62
Figura 5.4: Diagrama en bloques del SJA1000 63
Figura 5.5: Arquitectura de capas del protocolo CAN 64
Figura 5.6: Diagrama de flujo general 65
Figura 5.7: Diagrama de flujo de transmisión 65
Figura 5.8: Diagrama de flujo de recepción 66
Figura 5.9: Modelo de línea de transmisión del bus 70
Figura 5.10: Modelo de simulación final 72
Figura 5.11: Diagrama de clases del microcontrolador 73
Figura 5.12: Diagrama de clases del Controlador CAN 74
Figura 5.13: Diagrama de clases del Bus CAN 75

Figura 5.14: Diagrama de actividades del modelo de simulación final 76
Figura 6.1: Fotografía del horno y el nodo 78
Figura 6.2: Fotografía del nodo en la mufla 78
Figura 6.3: Metodología para la experimentación del hardware 79
Figura 6.4: IATs a diferentes temperaturas en el prototipo 80
Figura 6.5: Modelo de simulación 82
Figura 6.6: IATs a diferentes temperaturas en modelo de simulación 84
Figura 6.7: Valores medios de IATs del prototipo y la simulación 85
Figura 6.8: Comparación de IATs 86
Figura 7.1: Diagrama en bloques y modelado del controlador CAN 88
Figura B-1: Primer modelo 90
Figura B-2: Tracing del primer modelo 90
Figura B-3: Segundo modelo 91
Figura B-4: Cuarto modelo 91
Figura B-5: Tracing del cuarto modelo 92
Figura B-6: Sexto modelo 93
Figura B-7: Tracing del séptimo modelo 93

Lista de acrónimos

CoNAE: Comisión Nacional de Actividades Espaciales

NASA: National Aeronautics and Space Administration

IRU: Inertial Reference Unit

COTS: Commercial off the Shelf

CAN: Controller Area Network

STR: Sistemas de Tiempo Real

ISO: International Organization for Standardization

IMU: Inertial Measurement Unit

IFOG: Interferometric Fiber Optic Gyro

ESA: European Space Agency

ECSS: European Cooperation on Space Standardization

TLM: Transaction Level Modeling

SAM: System Architectural Model

RTL: Register Transfer Level

LLC: Logical Link Control

MAC: Medium Access Control

SOF: Start of Frame

CMOS: Complementary Metal-Oxide-Semiconductor

SoC: System on Chip

NoC: Network on Chip

PCB: Printed Circuit Board

FPGA: Field Programmable Gate Array

UML: Unified Modeling Language

SysML: System Modeling Language

VCD: Value Change Dump

IML: Interface Management Logic

BSP: Bit Stream Processor

BTL: Bit Timing Logic

EML: Error Management Logic

STP: Shielded Twisted Pair

PTFE: Politetrafluoroetileno (Teflón)

PVDF: Polyvinylidene fluoride

IAT: Inter Arrivals Time

1 Capítulo 1: Introducción

En este capítulo se efectúa una explicación de la motivación del tema de tesis. También se comentan algunas características del dispositivo que inspiró la realización del presente trabajo y el problema a resolver.

Además, se menciona el objetivo general de este trabajo y el alcance del mismo. Igualmente se detallan los sub-objetivos para conseguir el propósito principal.

Finalmente, se presenta una visión general de la tesis mediante un esquema por capítulos.

1.1 Introducción y motivación

El ansia de conocimiento ha llevado al hombre a experimentar y explorar desde el principio de los tiempos empujándolo a internarse a través del globo de un continente a otro, enfrentando cada vez mayores desafíos y superándolos de una manera u otra. Actualmente, el conocimiento y aprovechamiento eficiente de los recursos naturales es un tema primordial de cada una de las naciones. Por esta razón los países desarrollan y construyen tecnología para estudiar uno de sus bienes más preciados: las tierras que habitan. Una manera de realizar esto es a través de satélites artificiales y así realizar una observación del planeta constantemente.

El interés por la investigación y desarrollo aeroespacial en el país se ve reflejado en el diseño de un plan nacional desde el año 1961 con la creación de la Comisión Nacional de Investigaciones Espaciales (CNIE). Luego de la disolución de la CNIE se creó la Comisión Nacional de Actividades Espaciales (CoNAE) en 1991 colocando satélites en órbita a través de lanzadores extranjeros. En la actualidad se continúa con el diseño de otros satélites y un lanzador (Tronador II). Puede notarse un esfuerzo económico y científico para el desarrollo de tecnología espacial propia que posea características iguales o mayores a los estándares internacionales.

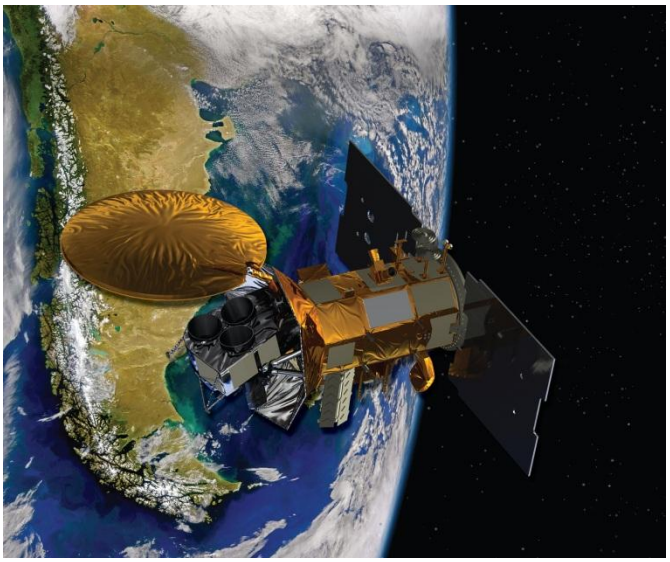

Figura 1.1: Satélite SAC-D/Aquarius

La motivación que predominó en la elección del tema de tesis surge de una experiencia transcurrida en el área aeroespacial, puntualmente en el desarrollo de instrumentación para la misión conjunta entre la NASA (National Aeronautics and Space Administration) y la CoNAE para el desarrollo del satélite SAC-D/Aquarius [1] [2]. En la Figura 1.1 puede verse un gráfico del satélite.

El principal objetivo del satélite es estimar la salinidad de mares y océanos, esta información es de vital importancia para entender las interacciones entre el ciclo del agua, la circulación oceánica y el clima. También se miden la temperatura y humedad del suelo permitiendo elaborar alertas tempranas de inundaciones, aparición y/o dispersión de enfermedades y modelos climáticos a largo plazo. Es decir, el sistema es un observatorio que permite efectuar estudios oceanográficos, atmosféricos, ictícolas, hidrológicos, ambientales y epidemiológicos.

El diseño de sistemas que forman parte de un satélite constituye un serio reto que hace que el desarrollo de estas aplicaciones sea especialmente atractivo, ya que el entorno espacial presenta infinidad de problemas. Generalmente los requerimientos científicos imponen condiciones que demandan la implementación de diseños novedosos, pero con la utilización de tecnologías probadas y preparadas para vuelo.

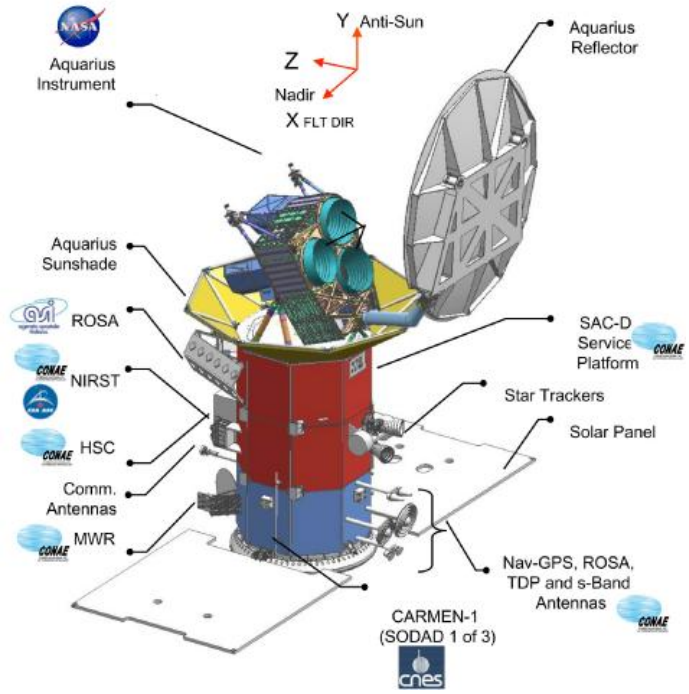

Figura 1.2: Instrumentos del SAC-D/Aquarius

El satélite está constituido por ocho instrumentos de los cuales cinco son de fabricación nacional [3]. El tesista integró el equipo que diseñó, desarrolló y construyó una Inertial Reference Unit (IRU) cuya función es brindar toda información, relativa a los cambios de estado de movimiento del objeto al cual es solidario, a un sistema capaz de procesarla y tomar decisiones (computadora de navegación). En este caso la IRU fue denominada Technological Demostration Package (TDP) [4] ya

que la misma, una vez en vuelo, ofrece la posibilidad de evaluar las tecnologías involucradas en su desarrollo. En la Figura 1.2 puede verse la ubicación de los distintos instrumentos en el SAC-D, y en uno de los cinco instrumentos de CoNAE se indica al TDP (abajo a la derecha).

Al ser un paquete de demostración tecnológico, se apeló a la utilización de componentes electrónicos comerciales (COTS) y protocolos de comunicaciones creados para otras áreas como el Controller Area Network (CAN) [5] reduciendo los costos de construcción al máximo pero obligando a tomar mayores precauciones para que el sistema supere todas las pruebas previas al lanzamiento. Es de destacar que el satélite SAC-D/Aquarius fue puesto en órbita el 10 de junio de 2011 y actualmente está operativo.

En el transcurso de la participación de este tesista en el proyecto, se observó las restricciones que presentan las técnicas tradicionales utilizadas durante la implementación de los sistemas híbridos (hardware/software). Puntualmente en el desarrollo del sistema de comunicaciones de la red que forman los sensores inerciales.

La poca flexibilidad de las metodologías tradicionales provoca un gran retraso en este tipo de proyectos, ya que generalmente los sistemas aeroespaciales son sometidos a un constante rediseño y verificación hasta poco antes de terminar con la integración de todos sus subsistemas. La complejidad en atender todos los requerimientos (algorítmicos, de software y hardware), teniendo en cuenta las restricciones a causa de posibles fallas, promovió a que en la actualidad se realice el diseño y verificación en esta área con tecnologías de simulación [6].

1.2 Objetivos y alcance

Esta tesis tiene como objetivo general, desarrollar una herramienta para analizar/predecir condiciones de trabajo de una red de sensores. En particular, se enfoca a un conjunto de dispositivos de vuelo que son parte de sistemas aeroespaciales.

El simulador es obtenido por medio de un modelado específico del sistema y su contrastación es realizada a través de un prototipo hardware. La herramienta propuesta permite conseguir medidas previas a la implementación de una etapa experimental definitiva de un dispositivo de vuelo.

En particular, en los sistemas aeroespaciales, es necesario cambiar las condiciones de trabajo de los prototipos para predecir el comportamiento y posibles fallos durante la etapa de servicio del hardware definitivo. Conseguir estas condiciones de trabajo puede resultar muy costoso debido al valor de los dispositivos para generarlas. El modelo de simulación provee el medio para extrapolar posibles escenarios operativos sin contar con instrumentos físicos.

Los temas a desarrollar en este trabajo son:

- Análisis de protocolos de comunicaciones en sistemas de tiempo real
- Análisis de sistemas de comunicaciones aeroespaciales
- Construcción y validación de una red de sensores utilizando el protocolo CAN

- Clasificación de fallas en comunicaciones de tiempo real
- Principios de modelado y simulación
- Construcción de un modelo de simulación utilizando SystemC [7]
- Validación del modelo de simulación propuesto

Los resultados de esta tesis se utilizarán potencialmente tanto en sistemas satelitales como en vehículos lanzadores, pero al existir diferencias entre los dos sistemas (tiempo de servicio de misión, radiación, sensores, entre otros) deberán realizarse pruebas específicas para cada uno.

1.3 Estructura del trabajo

El resto de la tesis está organizada de la siguiente forma:

En el capítulo 2 se realiza un estudio y clasificación de los protocolos de comunicaciones en sistemas de tiempo real. Se otorgan algunas descripciones de sensores girométricos y redes de sensores. También se realiza un análisis de protocolos de comunicaciones en sistemas aeroespaciales. Además, se definen principios de modelado y simulación enfocando a los sistemas aeroespaciales. Finalmente se detallan algunos conceptos de las herramientas para construir el simulador.

En el capítulo 3 se describe el problema a través de una explicación de experiencias previas. Se realiza una descripción del protocolo CAN y una clasificación de los parámetros que provocan fallas en estos tipos de sistemas. Se propone utilizar tecnologías de simulación como una solución para acortar tiempos de diseño y desarrollo.

En el capítulo 4 se presentan las herramientas utilizadas para realizar la propuesta de solución. Se describe a SystemC como el lenguaje de diseño elegido y los requerimientos del sistema a modelar. También se explican las herramientas de desarrollo empleadas para la construcción del prototipo hardware y el modelo de simulación.

En el capítulo 5 se realiza con mayor detalle la descripción del prototipo hardware, los componentes elegidos y la manera de configurar los nodos. Se efectúa un análisis de planificación del protocolo para definir las restricciones de diseño del modelo de simulación y se describen los distintos componentes del modelo de simulación propuesto.

En el capítulo 6 se efectúa una descripción de los experimentos generados con el prototipo hardware y con el modelo de simulación. Se detallan los valores empleados en los dos entornos (hardware y de simulación) y los escenarios planteados. Se muestran los resultados obtenidos y la validación del modelo de simulación utilizando los experimentos del prototipo hardware. Finalmente se realiza un ejemplo de experimentación con el modelo construido.

En el capítulo 7 se presentan las conclusiones del desarrollo de la tesis y se plantean algunas líneas de investigación futuras.

En el anexo A se presentan los trabajos publicados en encuentros y congresos nacionales e internacionales.

En el anexo B se explican algunos modelos previos al modelo de simulación final presentado en el capítulo 5.

2 Capítulo 2: Estado del Arte

En este capítulo se presentan definiciones y se describen los requerimientos que demandan los sistemas de tiempo real, sensores y simulación, específicamente enfocando a las aplicaciones de comunicaciones en sistemas aeroespaciales.

Dado que el campo de estudio es muy amplio, se ha acotado para resolver un escenario y un subsistema particular de un sistema más grande. Consecuentemente, sólo se describen las herramientas elegidas para resolver las distintas etapas del presente trabajo.

2.1 Protocolos de comunicaciones en sistemas de tiempo real

2.1.1 Sistemas de tiempo real

Puede definirse a un sistema de tiempo real (STR) como aquel que no solo depende de sus resultados lógicos sino también del momento físico en el que ocurren los mismos para asegurar su correcto funcionamiento. Este tipo de sistema no sólo debe proveer la salida esperada de acuerdo a la entrada sino que también debe respetar ciertas restricciones temporales para cumplir los requerimientos del sistema [8].

Los STR reaccionan a estímulos de su entorno, que ocurren en intervalos de tiempo dados por éste. Además las salidas del sistema deben producirse en instantes denominados plazos de tiempo (*deadline*). Si las salidas ocurren luego de vencido el plazo de tiempo pero aún son útiles, al plazo de tiempo se lo clasifica como blando (soft). Caso contrario el plazo de tiempo es llamado duro (hard) ya que no cumplirlo estrictamente podría acarrear graves consecuencias. De esta forma, pueden clasificarse a los sistemas de tiempo real en duros o críticos, si al menos deben respetar un plazo de tiempo duro, o blandos en caso que no exista ningún plazo de tiempo de este tipo.

Los requerimientos que deben cumplir los STR pueden agruparse en funcionales, temporales y de fiabilidad. En los funcionales se analiza cuándo y cómo tomar datos del entorno para luego ejecutar el procesamiento que generará la salida adecuada. En esta etapa puede notarse la característica concurrente de los STR, causada por la multiplicidad de eventos que pueden ocurrir en el momento de observar la situación de todo el sistema.

Los requerimientos temporales están fuertemente relacionados con el retardo de respuesta del sistema; el cálculo de este parámetro incumbe directamente a otros como latencia, *jitter*, entre otros. Cuantificar el retardo es la forma de encontrar las limitaciones del sistema diseñado y asegurar que cumple con la funcionalidad buscada.

Los requerimientos de fiabilidad cubren los datos que relacionan la calidad de servicio (QoS) del sistema en un intervalo de tiempo definido. Estos datos abarcan tasa de fallas, tiempo medio entre fallas, tiempo medio de reparación, entre otros. Es decir, deben encontrarse todos los datos necesarios para medir la seguridad, el mantenimiento y disponibilidad del sistema.

La clasificación de los STR en duros y blandos está dada por las características de entorno, es decir, parámetros externos al sistema. También se pueden clasificar en disparados por eventos o

disparados por tiempo, y esta manera de agruparlos está relacionada a los factores internos del sistema.

Un disparo es un evento que provoca el comienzo de una acción en el procesador, por ejemplo la ejecución de una tarea o la transmisión de un mensaje. Dependiendo del mecanismo de disparo para realizar acciones en cada nodo del sistema pueden distinguirse dos tipos de control: disparado por eventos o disparado por tiempo.

Cuando se utiliza el control por eventos las acciones del sistema se llevan a cabo cuando un evento ocurre. Generalmente la aparición de un evento se indica por medio del mecanismo de interrupción, por esta razón el sistema requerirá de una planificación dinámica para ejecutar la tarea que atienda al mismo.

Un sistema que implementa el control disparado por tiempo ejecuta todas las acciones a medida que transcurre el tiempo real. Luego, cada comunicación o procesamiento se inicia con un pulso de reloj que generalmente es asociado a una interrupción periódica que se ejecuta en cada nodo de un sistema distribuido. Se supone que los relojes de todos los nodos están sincronizados para formar un tiempo global, y de esta manera las acciones podrán ser marcadas con tiempos dados por este tiempo global.

2.1.2 Comunicaciones en sistemas de tiempo real distribuido

Un sistema de tiempo real distribuido está formado por un conjunto de nodos (procesadores) interconectados a través de una red de comunicaciones de tiempo real. La interfaz entre el objeto controlado y el sistema de tiempo real se denomina interfaz de instrumentación y consiste en un conjunto de sensores y actuadores que transforman las señales físicas medidas (velocidad, luz, tensión, entre otros) en señales digitales.

La infraestructura de comunicación en un sistema de tiempo real distribuido debe cumplir con requerimientos muy diferentes a los demandados por otros sistemas. Los requerimientos pueden clasificarse en: exactitud, fiabilidad, flexibilidad y estructural.

En el requerimiento de exactitud se puede notar la necesidad de conseguir una latencia corta y un *jitter* mínimo, siendo esta la principal diferencia entre los sistemas de comunicaciones de tiempo real y los otros. La latencia tiene una duración que inicia con la lectura de los sensores en cada uno de los nodos y finaliza con la salida del actuador correspondiente, teniendo en cuenta el procesamiento requerido; en muchos casos se toma como latencia del sistema a la que corresponde al peor caso. El *jitter* es la diferencia entre las latencias del peor y mejor caso, y un valor alto de este parámetro puede producir efectos negativos para calcular el instante de aceptación de un mensaje. En un sistema distribuido de tiempo real la exactitud temporal sólo se asegura si se dispone de una base de tiempo global que garantiza la precisión de los instantes de tiempo en todo el sistema. Corresponde al sistema de comunicaciones establecer este tiempo global y sincronizar los nodos.

La fiabilidad engloba características que necesitan poseer los sistemas de comunicaciones para aportar seguridad a la comunicación (esto se consigue por medio de redundancia); estas son:

contención de fallas temporales para permitir el funcionamiento de la red a pesar de ciertas averías y detección de errores para identificar las fallas.

Un protocolo de comunicaciones de tiempo real debe ser lo suficientemente flexible para soportar cambios sin la necesidad de requerir modificaciones en el software. Si bien se puede implementar un escalamiento de la red, siempre estará limitada por el ancho de banda y las limitaciones temporales a medida que aumenta el tráfico en la misma.

Los requerimientos estructurales de la red están determinados no solo por consideraciones técnicas sino también económicas. Los entornos de aplicación son variados pero en casos de exigencia extrema, como los campos automotrices o de aviónica, debe seleccionarse un medio de transmisión robusto y que no insuma un costo alto.

2.1.2.1 Clasificación

Desde un punto de vista temporal los protocolos de comunicaciones de bajo nivel para tiempo real pueden clasificarse en tres grandes grupos: disparados por eventos, de tasa restringida y disparados por tiempo [9].

En las comunicaciones disparadas por eventos un emisor envía un mensaje en el momento que ocurre un evento. El mensaje es ubicado en una cola del emisor hasta que puede ser enviado por el canal de comunicaciones al receptor. Luego de que el mensaje arriba al receptor es colocado en una cola hasta que el mismo puede recibir el mensaje. En la Figura 2.1 puede verse una representación de lo explicado.

Figura 2.1: Comunicación disparada por eventos

Entre los protocolos de comunicaciones disparados por eventos pueden mencionarse a Ethernet, Controller Area Network (CAN) normalizado por la International Organization for Standardization (ISO) y User Datagram Protocol (UDP).

Las comunicaciones de tasa restringida establecen un ancho de banda mínimo para cada canal y para conseguir esto se aseguran latencias y *jitters* máximos que no superen un cierto límite. Con el propósito de conseguir este ancho de banda mínimo el sistema de comunicaciones debe poseer información sobre el ancho de banda garantizado para cada emisor. Algunos de los protocolos de tasa restringida son: Token, Aeronautical Radio Incorporated (ARINC) 629 que es utilizado en el Boeing 777 y Avionics Full Duplex Switched Ethernet (AFDX) usado en Airbus A 380.

En los sistemas de comunicaciones disparados por tiempo, el emisor y el receptor acuerdan una planificación comunicacional controlada por un tiempo cíclico para enviar mensajes disparados por tiempo. De cierto modo una comunicación disparada por tiempo se asemeja a un time-

controlled circuit switching (TCCS), en donde se establece un canal dedicado controlado por tiempo entre el emisor y el receptor de una duración suficiente para enviar un mensaje. Un control por tiempo requiere señales temporales que deben derivar de una única fuente de tiempo. Generalmente esta fuente es el tiempo global sincronizado del sistema o puede usarse un periodo que establece un proceso autónomo. Algunos protocolos que necesitan del establecimiento de un tiempo global son: Time Triggered que fue estandarizado por la Society of Automotive Engineers (SAE) en 2011, Time Triggered Ethernet utilizado por la NASA y en proceso de estandarización por la ARINC y FlexRay.

2.2 Sensores

2.2.1 Desarrollo de girómetros y su utilidad

La navegación de alta mar o de vuelo es más segura y económica si se utilizan sistemas automáticos se consigue esta seguridad porque los sistemas son inagotables y siempre están en funcionamiento, son más económicos porque se evita la utilización de recursos humanos calificados. Obviamente la navegación espacial no-tripulada requiere de esta característica que se consigue muchas veces a través de sistemas autónomos denominados inerciales [10].

En navegación inercial se mide el comportamiento dinámico del sistema por medio de girómetros y acelerómetros. Inicialmente los sensores estaban basados en complejos sistemas mecánicos con ruedas y ejes giratorios. Actualmente el desarrollo tecnológico de fibra óptica y óptica integrada junto a la utilización de procesadores ha permitido implementar dispositivos más confiables y pequeños. De esta manera los sistemas de navegación inerciales están conformados por una computadora de navegación y un conjunto de girómetros y acelerómetros generalmente llamados sensores inerciales. El grupo de sensores inerciales unido a un procesador que modula las señales ópticas es denominado Inertial Measurement Unit (IMU) o Inertial Reference Unit (IRU).

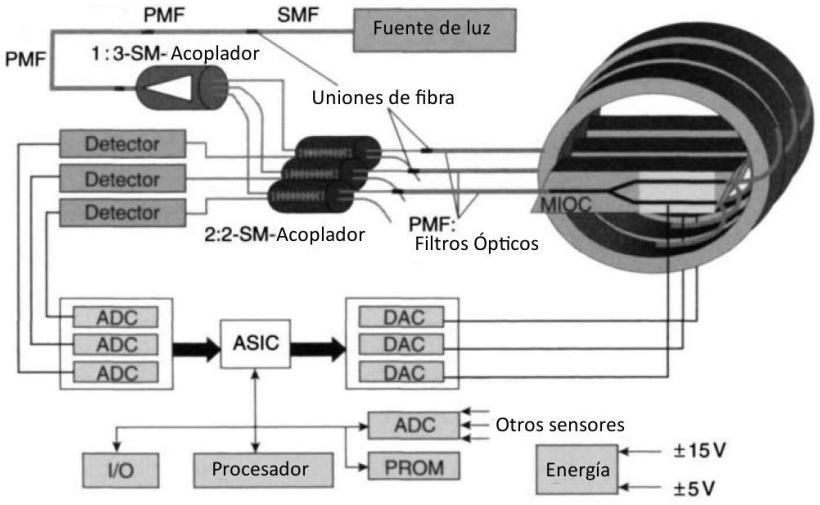

Figura 2.2: Circuito óptico del IFOG

Por otra parte los girómetros ópticos, aprovechando las características inerciales de la luz, han demostrado ofrecer grandes ventajas por sobre los mecánicos ya que no poseen piezas con un

mecanizado complejo que aumentan el costo y acortan el tiempo de vida del dispositivo. Los sensores ópticos tipo Interferometric Fiber Optic Gyro (IFOG) se basan en el efecto Sagnac y se han posicionado en el mercado como sensores confiables y precisos para desarrollar unidades de referencia inerciales particularmente para sistemas aeroespaciales. En la Figura 2.2 puede observarse el grafico de un paquete de sensores IFOG que proveen información de los tres ejes del sistema en cuestión.

Los sistemas de vuelo cuentan con computadoras de navegación que continuamente reciben información de los tres ejes de referencia del vehículo (yaw, pitch y roll) para ser comparados con la ubicación esperada y cumplir con los planes de guiado. En caso que la comparación arroje resultados no aceptables, se generan acciones de corrección manteniendo el control, en cada momento, de la trayectoria seguida por el sistema [11].

2.3 Redes de Sensores

En este tipo de redes el componente principal es el sensor que esencialmente debe medir parámetros físicos del mundo real; en el caso de sistemas vuelo estos datos pueden ser velocidad angular, aceleración, temperatura, entre otros. Generalmente los sensores cuentan con los circuitos electrónicos necesarios para transformar los parámetros medidos en señales eléctricas. Además deben poseer cierta capacidad de procesamiento para acondicionar estas señales en mensajes que se transmiten a una base central o sumidero en donde se generan las acciones de control [12]. Es decir, una red de sensores ejecuta tres funciones básicas: sensar, comunicar y calcular por medio de tres componentes fundamentales: hardware, software y algoritmos, respectivamente.

Las redes de sensores deben utilizar protocolos de comunicaciones para implementar arquitecturas de comunicaciones que proporcionen soluciones escalables, robustez y eficiencia energética. En la mayoría de las aplicaciones de redes de sensores distribuidos para control se utilizan redes alámbricas tales como CAN, PROFIBUS o Ethernet [13]. Las características de la red elegida afectan directamente la performance del sistema de control; una manera de estudiar simultáneamente el comportamiento de los sistemas de comunicaciones y de control es por medio de las herramientas de simulación.

2.4 Sistemas de Comunicaciones Aeroespaciales

Actualmente las agencias espaciales más reconocidas del mundo, como la European Space Agency (ESA) o la NASA, no han definido un protocolo de comunicaciones estándar único, aunque mencionan estándares usados o propuestos para las distintas misiones espaciales. A continuación se mencionan algunos protocolos que ha usado y usa la ESA en sus proyectos [14].

El protocolo MIL-STD-1553B o Milbus es un estándar militar desarrollado para el área de aviónica y define un bus de datos serie multiplexado. El medio de transmisión es un cable trenzado y apantallado que soporta una tasa fija de datos de 1 Mbps, y típicamente es implementado en forma redundante ofreciendo dos canales de transmisión. El protocolo define tres componentes: el Bus Controller (BC), el Remote Terminal (RT) y Monitor Bus (BM). El BC se desempeña como productor e inicia la transacción, los RTs proveen la interfaz entre los nodos y el bus. El BM es

pasivo y registra el tráfico en el bus. En aplicaciones espaciales el protocolo ha sido utilizado en varios proyectos de agencias espaciales como la International Space Station (ISS). En la Figura 2.3 puede verse el conexionado de un RT al bus.

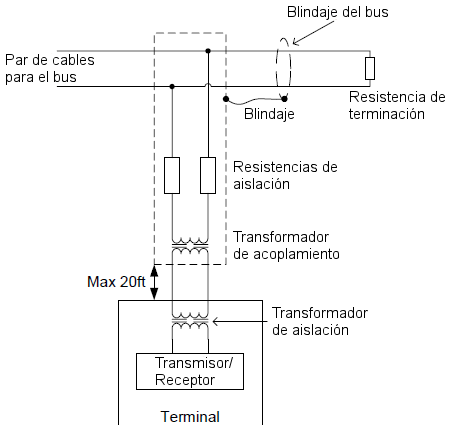

Figura 2.3: Conexionado de un terminal MIL-STD-1553 al bus

El protocolo SpaceWire fue desarrollado por la European Cooperation on Space Standardization (ECSS) y ha sido diseñado especialmente para aplicaciones espaciales. El estándar está basado parcialmente en el IEEE 1355 pero con mejoras en cuanto a consumo de energía, compatibilidad electromagnética y robustez. El protocolo define un enlace de datos serie punto a punto a través de un medio de transmisión de ocho cables en total. Dos pares están dedicados para los datos de entrada/salida y los otros dos para señales de control, siendo destacable que cada par es trenzado y posee apantallamiento individual. La tasa de datos es alta ya que soporta velocidades de 2 Mbps hasta 200 Mbps. La red consiste en dos o más nodos y posiblemente en uno o más routers SpiceWire. En Figura 2.4 puede verse un ejemplo de una red SpiceWire.

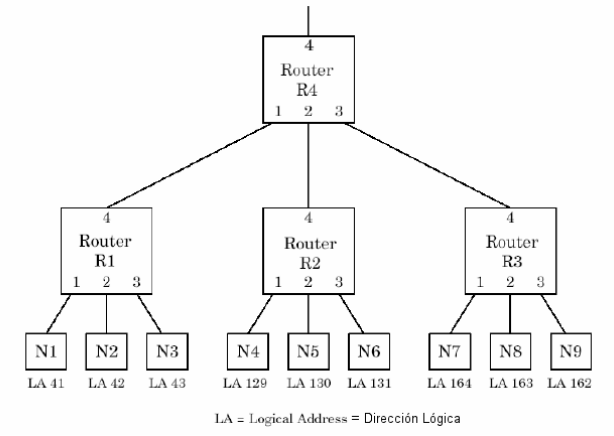

Figura 2.4: Red SpiceWire

El CAN fue diseñado por la compañía Robert Bosch GmbH en Alemania y el propósito inicial era crear una red más rápida y resistente a interferencias dentro del ámbito automotriz. El estándar ofrece una comunicación determinística, con prioridades y detección de errores. La topología del

bus consiste generalmente en un medio de transmisión, formado por un par de cables, y estaciones o nodos. Aunque la especificación del medio de transmisión del cable no está indicada en la norma original puede ser adaptada dependiendo el área de aplicación. La codificación utilizada es de Non Return to Zero (NRZ) con stuffing sobre un canal de señales diferenciales (balanceado) lo que asegura mensajes compactos con un mínimo número de transiciones y una alta resistencia a las perturbaciones externas. La tasa de transferencia de datos es media ya que ofrece velocidades que van de 100 kbps a 1 Mbps. En la Figura 2.5 puede observarse la disposición de una red CAN [5].

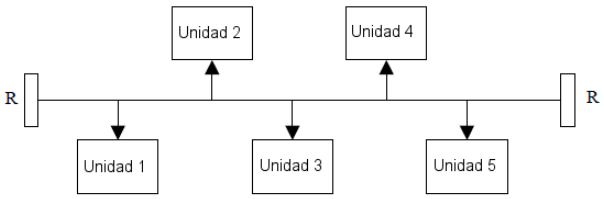

Figura 2.5: Red CAN

Los parámetros a tener en cuenta para realizar una comparación entre protocolos de comunicaciones en aplicaciones aeroespaciales son: compatibilidad electromagnética (EMC), área física de ocupación en el sistema, potencia consumida, tasa de transmisión y complejidad de implementación. Los tres protocolos antes mencionados probablemente sean los que mejor cumplen con estos requisitos en la actualidad, pero queda un último factor a tener en cuenta que es el peso del sistema y en este sentido el CAN-Bus ampliamente supera a los otros dos protocolos debido al medio de transmisión de éstos últimos. SpiceWire posee ocho conductores y MIL-STD-1553 suma el peso de sus transformadores, mientras que CAN utiliza sólo dos conductores.

2.5 Simulación

Una simulación se puede definir como la imitación de las operaciones de un sistema del mundo real en el tiempo, y de esta manera generar una historia artificial para inferir posibles características de esas operaciones. El comportamiento de un sistema en el tiempo generalmente es estudiado desarrollando un modelo de simulación que se realiza a partir de un conjunto de suposiciones. Estas suposiciones, que normalmente toman forma de relaciones lógicas o matemáticas entre entidades u objetos de interés, una vez validadas permiten realizar un análisis del sistema, aún antes de contar con el modelo físico [15].

Si las relaciones que componen el modelo son lo suficientemente simples, entonces es 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 modelos realistas sean evaluados analíticamente, y estos modelos deben ser estudiados mediante simulaciones. En una simulación se utilizan computadoras para evaluar un modelo numéricamente, y los datos son recolectados para estimar las características deseadas del modelo [16]. Se debe analizar con detalle el sistema,

su grado de complejidad y el modelo a adoptar para generar la programación que permita obtener la simulación necesaria.

2.5.1 Sistemas, Modelos y Simulación

Un sistema se define como una colección de entidades, como personas o máquinas, que actúan e interactúan en conjunto para lograr un fin lógico. En la práctica, la definición de "el sistema" depende de los objetivos de estudio en particular. El estado de un sistema se define como una colección de variables necesarias para describir al sistema en un momento particular, relacionado a los objetivos de estudio.

Los sistemas se pueden categorizar en dos tipos: discretos y continuos. Un sistema discreto es aquel en el cual las variables de estado cambian instantáneamente en puntos separados del tiempo. Un sistema continuo es aquel en el que las variables de estado cambian continuamente con respecto al tiempo. En la práctica, pocos sistemas son completamente discretos o continuos; pero dado que un tipo de cambio predomina para la mayoría de los sistemas, generalmente es posible clasificar a un sistema como discreto o continuo. En la Figura 2.6 puede observarse un gráfico que representa las distintas maneras de estudiar un sistema.

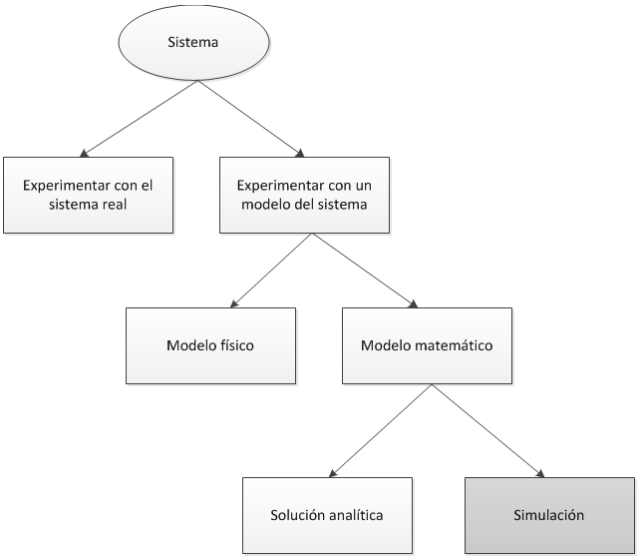

Figura 2.6: Formas de estudiar un sistema

A continuación se explican las características de las distintas formas de estudiar un sistema:

- *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 sería demasiado costoso o perjudicaría interrumpir el sistema. También se debe considerar el caso en el que el sistema no exista y se quieran plantear alternativas de configuración para ver cómo 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 validación del modelo.

- *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 las preguntas 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, es decir utilizar una solución analítica. Algunas soluciones analíticas pueden volverse muy complejas, tornando casi como única opción la simulación.

Generalmente los modelos de simulación pueden clasificarse 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 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 han 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 características reales del modelo; esta es una de las desventajas principales de la simulación.

- Modelos de Simulación Continuo vs Discreto. Estos tipos de simulaciones se definen de forma análoga a las definiciones de sistemas discretos y continuos. 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.

2.5.2 Simulación de eventos discretos

Los modelos de simulación discretos, dinámicos y estocásticos se los denominan modelos de simulación de eventos discretos. Este tipo de simulaciones 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 instantes 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.

2.5.2.1 Mecanismos de avance de tiempo

Debido a la naturaleza dinámica de los modelos de simulación de eventos discretos, se debe estar al tanto de los valores actuales del tiempo simulado a medida que la simulación transcurre, y también es necesario un mecanismo para el avance del tiempo simulado de un valor a otro. La variable en el modelo de simulación que entrega el valor actual del tiempo simulado se 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 han sido sugeridos para el avance del reloj de simulación: avance de tiempo del próximo evento o dirigido por eventos y avance del tiempo de incremento fijo o periódico. El primer enfoque es utilizado por la gran mayoría de los lenguajes para simulación y por las personas que codifican modelos en un lenguaje de propósito general, aunque el segundo enfoque en realidad es un caso especial del primero.

2.5.2.2 Componentes y organización de un modelo de simulación de eventos discretos

Aunque las técnicas de simulación han sido aplicadas a una gran diversidad de sistemas del mundo real, todos los modelos de simulación de eventos discretos comparten un número común de componentes y existe una organización lógica para estos componentes que promueve la codificación, depuración y futuros cambios en la programación del modelo de simulación. Los siguientes componentes se encuentran en la mayoría de los modelos de simulación de eventos discretos, utilizando el enfoque de avance de tiempo dirigido por eventos 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.

- Lista de eventos: es una lista que contiene el próximo instante en el que un evento ocurrirá.

- 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 cero.

- Rutina de tiempo: subprograma que actualiza el estado del sistema desde la lista de eventos y de esta forma avanza el reloj de simulación al instante en el que ocurre el evento.

- 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 la 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.

2.5.3 Construcción del modelo

Teniendo en cuenta que la construcción de un modelo de simulación no es un proceso secuencial, se pueden distinguir los siguientes pasos:

1. Análisis del problema y planificación. El primer paso en la construcción de un modelo de simulación es el de analizar el problema y decidir qué objetivos serán abordados. Para facilitar una solución, en el análisis primero se 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. Luego se realiza una planificación teniendo en cuenta el tiempo requerido para el desarrollo del modelo, costo, entre otros.
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 son recolectados, se procede a la construcción de un modelo e implementación del mismo en un programa de computadora. Se debe tener en cuenta que un excesivo nivel de detalle en el modelo puede acarrear una exagerada complejidad en la codificación.
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 efectuar. 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 validar. La validación del modelo examina si éste 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 se ha verificado, 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. Documentación. 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.

2.5.4 Simulación en Sistemas Aeroespaciales

Los sistemas complejos requieren de una ingeniería detallada para su diseño, construcción, verificación y validación. El desarrollo de sistemas espaciales se puede dividir en cuatro fases que incluyen la concepción del sistema y su análisis, los requerimientos de los componentes, el diseño y definición de los dispositivos y finalmente la construcción y verificación del proyecto.

Uno de los primeros pasos en la fase diseño de estos dispositivos es efectuar un análisis funcional de los mismos. En la actualidad esto se consigue por medio de técnicas de simulación, de esta manera se permite estudiar la interacción entre los distintos componentes, por ejemplo, protocolos de comunicaciones, modos de operación o potencia consumida de los distintos subsistemas. Debe quedar claro que el propósito de la simulación funcional no es representar el funcionamiento exacto de los componentes ya que se perdería la ventaja de abstracción de las características físicas [6].

Los sistemas aeroespaciales se construyen por medio de una combinación de hardware y software; esto implica encontrar los algoritmos adecuados, transfórmalos en software, integrar el software con el hardware y finalmente constituir el sistema completo. Entonces, a causa de la complejidad en la construcción de los modelos de simulación debido a los requerimientos de tiempo real, la tecnología de simulación debe ofrecer los siguientes recursos que se adaptarán a la aplicación: *kernel* de simulación, módulos configurables y líneas de conexión para representar buses.

Estas características permiten afirmar que la simulación debe ser preferiblemente implementada aplicando tecnologías de software orientada a objetos. Los lenguajes de programación orientados a objetos permiten implementar una clase para cada tipo de elemento funcional del sistema, declarando las variables y funciones (métodos) en el mismo, incluyendo interfaces a otras clases. De esta forma la topología del sistema real puede ser reproducida en el programa fuente y el mantenimiento del código se simplifica. Los lenguajes de programación orientados a objetos como C++, C# o Java son la mejor elección para desarrollar modelos de simulación de sistemas aeroespaciales.

2.6 TLM y SystemC

2.6.1 Conceptos de TLM

Transaction Level Modeling (TLM) es una metodología de descripción de sistemas que permite simulaciones rápidas y eficientes de gran número de ciclos en sistemas complejos, modelando módulos sin la necesidad de especificar si serán sintetizados en Hardware o serán partes Software del sistema. Esta descripción de muy alto nivel se basa en abstraer la comunicación entre módulos, de manera que pueda ser descripta independientemente de todo lo demás [17].

La utilización del TLM junto con el lenguaje de descripción de sistemas como SystemC, permite el uso de un modelo temprano de ejecución del sistema, mejorando tareas de diseño y facilitando trabajos de pruebas mediante 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 las comunicaciones e interfaces de los componentes.

Se pueden distinguir cuatro niveles de refinamiento o abstracción:

- Sin tiempo (Un-Timed): ya que sólo se representa la funcionalidad, un modelo con comunicación sin tiempo y funcionalidad generalmente se lo refiere como un modelo arquitectónico del sistema System Architectural Model (SAM).

- De tiempo débil (Loosely Timed): es el caso de una comunicación con mayor refinamiento que sus nodos.

- De tiempo aproximado (Approximately Timed): se incluye cierta información básica sobre el tiempo, la duración real es aproximada.

- A nivel de ciclos (Pin and Cycle Accurate): los detalles de implementación permiten una simulación con precisión a nivel de ciclos de reloj [7].

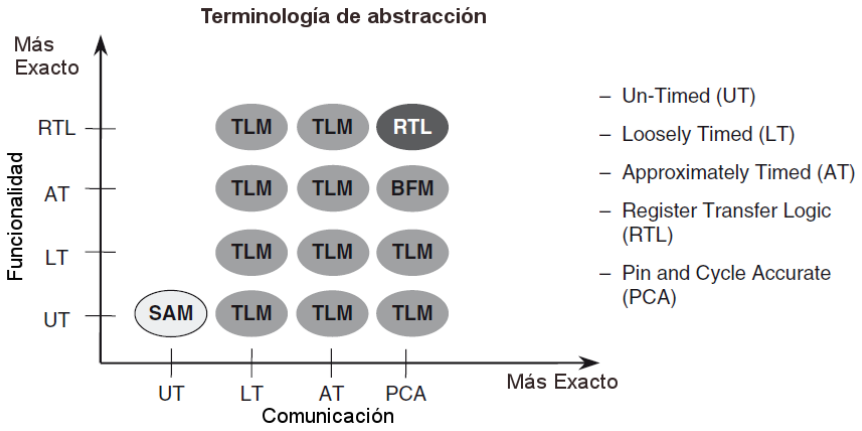

Figura 2.7: Modelos de abstracción TLM

En la Figura 2.7 puede verse la representación del nivel de abstracción de la funcionalidad del modelo a construir frente al nivel de abstracción de las comunicaciones del mismo.

Generalmente se comienza con un SAM, considerando que este es sin tiempo tanto en funcionalidad como en comunicación, para realizar el refinamiento algorítmico y luego 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 a nivel de ciclos presenta precisión tanto a nivel de comunicación como de funcionalidad. Generalmente se lo denomina modelo a nivel de registro de transferencia Register Transfer Level (RTL). Este tipo de abstracción hace referencia a una descripción en lenguaje que utiliza una combinación de construcciones descriptivas realizadas a nivel de flujo de datos o comportamiento que puede ser aceptada por una herramienta de síntesis lógica. 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 un mayor reúso de componentes de diseño, no sólo dentro de un proyecto específico, sino también en proyectos futuros, ocultando detalles de implementación más finos.

2.6.2 Conceptos de SystemC

Aunque se describe a SystemC como un lenguaje de descripción de sistemas, en realidad es una librería de clases dentro de un lenguaje bien establecido como es C++. Entonces se puede describir como un *framework* que provee los constructores necesarios para modelado de hardware y que permite representar conceptos de tiempo, tipos de datos de hardware, comunicación y concurrencia.

Como los sistemas pueden estar compuestos por subsistemas de comunicación que son modelados de manera heterogénea, éstos necesitan ser simulados eficientemente. SystemC provee constructores de alto nivel que comparten un poderoso motor de simulación, un *kernel* de simulación dirigido por eventos. Este *kernel* trabaja con eventos y procesos de manera abstracta. Otros elementos incluyen módulos y puertos para representar la información estructural, e interfaces y canales para las comunicaciones. El *kernel* y estos elementos forman el corazón del *framework* [18].

Este *framework* 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 del tiempo simulado y que se sincronizan y comunican emulando sistemas electrónicos que contienen hardware y software embebido. En la Figura 2.8 se ilustra la mayoría de los componentes de SystemC.

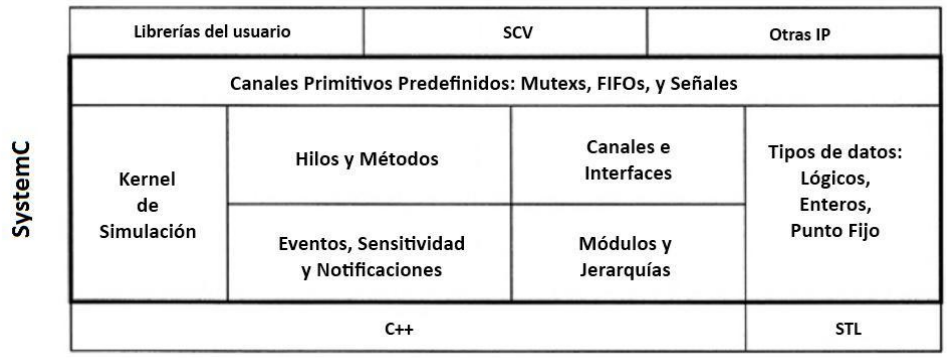

Figura 2.8: Arquitectura del framework SystemC

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

SystemC provee mecanismos para modelar hardware pero utilizando un lenguaje compatible con el desarrollo de software. El *framework* provee varios constructores orientados a hardware pero que generalmente no están disponibles en lenguajes de software; sin embargo, estos constructores son requeridos para modelar hardware y están implementados dentro del contexto del lenguaje C++. La mayoría de las características orientadas a hardware implementadas en SystemC son:

- Modelo de tiempo
- Tipos de dato hardware
- Jerarquía de módulos para el manejo estructural y conectivo
- Manejo de comunicaciones entre unidades concurrentes de ejecución
- Modelo de concurrencia

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

Se explicará luego con más detalle, en el capítulo de herramientas utilizadas para la propuesta de solución, los distintos elementos del *framework*.

Una de las grandes ventajas del *framework* es que fue estandarizado bajo la norma IEEE 1666 [19] aunque distintas agrupaciones aportaron pre-estandarizaciones para refinar a SystemC. 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:

- 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

En 2011 las asociaciones Accellera y la OSCI se unieron para formar la organización Accellera Systems Initiative con el propósito de cubrir las necesidades de los diseñadores de sistemas y semiconductores; y buscar nuevas formas de crear y producir nuevas soluciones para sistemas embebidos. Así mismo para colaborar en la creación de nuevos estándares de diseño y verificación requeridos por estos sistemas [20].

3 Capítulo 3: Descripción del problema

En este capítulo se presenta una descripción general del protocolo de comunicaciones, utilizado para generar una red de sensores inerciales.

También se considera oportuno definir conceptos de clasificación y tolerancia a fallas enfocados al área de incumbencia e implementación.

En función de las definiciones presentadas y las características de la implementación se plantean las herramientas de codiseño como una solución para generar la simulación de todo el sistema.

3.1 Experiencias previas

3.1.1 Ámbito de desarrollo de la implementación

El área de aplicación de esta tesis es la de sistemas de vuelo y específicamente el subsistema control y guiado de un satélite. Como ya se ha mencionado en el capítulo anterior, los sistemas de vuelo cuentan con una serie de dispositivos sensores que miden los valores de las diversas magnitudes involucradas en los algoritmos de navegación a ser resueltos. En este trabajo, se entiende por navegación a la actividad por la cual se determina la ubicación del sistema aeroespacial en un momento dado respecto de un punto de referencia específico, para ser comparada con la ubicación esperada y cumplir con los planes de guiado. En caso que la comparación arroje resultados no aceptables, se generan acciones de corrección manteniendo el control, en cada momento, de la trayectoria seguida por el vehículo.

También se ha indicado en el capítulo previo (sección 2.2.1) que las mediciones inerciales se obtienen por sensores de tales como acelerómetros y girómetros. En este caso se desarrolló el circuito óptico (bobina sensora de fibra, modulador o MIOC, acoplador, fotodetector, fuente de luz) y el sistema electrónico que produce la modulación de la señal de luz que circula por el circuito, para formar un girómetro del tipo IFOG. Además, como la salida del fotodetector es no lineal se convierte a lineal mediante un procesamiento de señal adecuado. El mismo sistema acondiciona y muestrea la señal sensada para suministrar un dato de 16 bits y formar la IMU. En la Figura 3.1 [21] se puede observar una IMU con cuatro ejes (uno redundante) junto al módulo de electrónica que envía las señales a la computadora de navegación. El autor de este trabajo participó en el diseño e implementación de los circuitos ópticos y electrónicos de una IMU que fue utilizada en un satélite.

Luego de desarrollar las unidades optoelectrónicas fue indudable la necesidad de seleccionar un protocolo de comunicaciones, para formar con cada uno de los ejes de la IMU una red que los contenga. Debido a las razones mencionadas en la sección 2.4 y a otras que se desarrollarán más adelante se optó por el protocolo CAN o CAN-Bus.

Figura 3.1: Unidad de Medición Inercial

3.1.2 Soluciones implementadas

3.1.2.1 Protocolo CAN

Esta red es definida como un protocolo de comunicación serie 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 [5]. El protocolo CAN fue definido por la ISO bajo el estándar ISO 11898 [22].

Inicialmente fue diseñado para proveer comunicaciones simples, eficientes y robustas en la industria automotriz alemana. Cómo en los automóviles el número de Electronic Control Unit (ECU) pasó de 5 a principios de los 90 a más de 100 en la actualidad, rápidamente el protocolo fue adoptado por la mayoría de las empresas más reconocidas a nivel mundial [23].

3.1.2.1.1 Conceptos básicos

Para conseguir transparencia en el diseño y flexibilidad en la implementación, CAN ha sido subdividida en diferentes capas de acuerdo al modelo de referencia ISO/OSI. Las principales capas son la de enlace de datos y la física.

La capa de enlace de datos se subdivide en:

- Subcapa de control de enlace lógico (LLC, Logical Link Control)
- Subcapa de control de acceso al medio (MAC, Medium Access Control)

En versiones previas de la especificación los servicios y las funciones de las subcapas LLC y MAC se denominaban capa objeto y capa de transferencia. El alcance de la subcapa LLC es:

- Proveer servicios para transferencia de datos y requerimientos remotos de datos.
- Decidir qué mensajes recibidos por la subcapa LLC serán aceptados
- Proveer un manejo de recuperación y notificación de sobrecarga

El alcance de la subcapa MAC principalmente es el protocolo de transferencia, lo que significa controlar el entramado, realizar el arbitraje, señalización y búsqueda de errores y captura de

fallas. Dentro de la subcapa MAC 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 subcapa MAC. Es importante mencionar que la subcapa MAC 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.

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 en el sistema
- Multimaster
- Detección y señalización de errores
- Retransmisión automática de mensajes corruptos tan pronto como el bus esté disponible nuevamente
- Distinción entre errores temporales y fallas permanentes de nodos y desconexiones autómaticas de nodos defectuosos

En la Figura 3.2 puede verse la arquitectura en capas de CAN de acuerdo al modelo de referencia OSI.

La capa física define cómo son realmente transmitidas las señales. Dentro de esta especificación las características de la Interfaz/Receptor no son definidas, por lo tanto el medio de transmisión y la implementación a nivel señal son optimizadas de acuerdo a la aplicación.

La subcapa MAC representa el *kernel* del protocolo CAN. Presenta mensajes recibidos de la subcapa LLC y acepta mensajes para ser transmitidos desde la subcapa LLC. La subcapa MAC es responsable del *timing* y la sincronización de bits, entramado de mensajes, arbitraje, ack, detección y señalización de errores y manejo de fallas. La subcapa MAC es supervisada por una entidad denominada confinamiento de fallas, que se autochequea para detectar disturbios cortos a causa de averías permanentes.

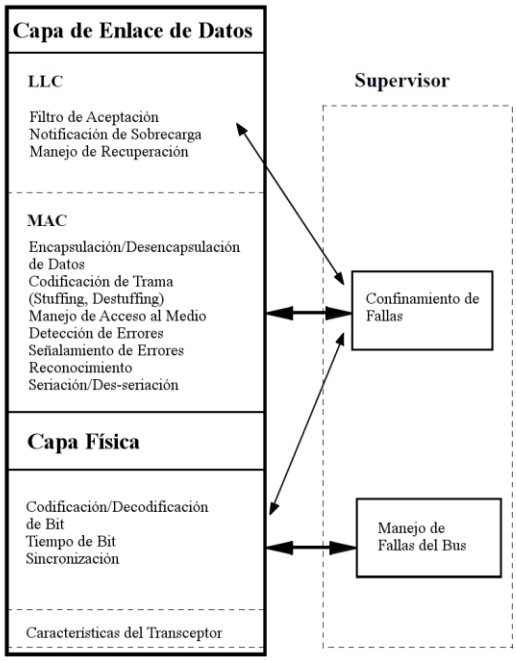

Figura 3.2: Modelo de Referencia OSI del protocolo CAN

La subcapa LLC se encarga del filtrado de mensaje así como también del manejo de la recuperación y notificación de sobrecarga.

3.1.2.1.2 Características

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.

Información de ruteado: 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 y/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 por ninguno. Entonces, la consistencia de datos de un sistema se logra bajo los conceptos de multicast y manejo de errores.

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

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

Petición de datos remota: al enviar una trama remota un nodo que requiere el dato puede pedir a otro nodo que le envíe la trama de datos correspondiente. La trama de datos y su trama remota correspondiente son llamadas por el mismo identificador.

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.

Arbitraje: si dos o más unidades comienzan a transmitir un mensaje en el mismo momento, el conflicto de acceso al bus se resuelve por arbitraje de diferencia de bits utilizando el identificador. El mecanismo de arbitraje garantiza que no se pierde información ni tiempo. Si una trama de datos y una trama remota con el mismo identificador comienzan a transmitir al mismo tiempo, la trama de datos prevalece por sobre la trama remota. Durante el arbitraje 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 en el arbitraje y debe dejar de transmitir.

Seguridad: para lograr seguridad en la transferencia de datos, se implementan en cada nodo CAN medidas para la detección, señalización y auto-comprobación de errores. Entonces para detectar errores el protocolo toma 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)

Los mecanismos de detección de errores tienen las siguientes propiedades:

- Se detectan todos los errores globales
- Se detectan todos los errores locales en los transmisores
- Hasta 5 errores aleatoreamente 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 total de error residual para mensajes corruptos sin detectar es menor que:

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

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. Si no hay otro error, 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 31 bits por la red.

Confinamiento de fallas: los nodos CAN pueden distinguir disturbios intermitentes de fallas permanentes, en caso que ocurra esto último los nodos defectuosos se desconectan.

Conexiones: el enlace de comunicación serie CAN es un bus el cual no posee un límite teórico de unidades que pueden conectarse. Aunque en la práctica todas las unidades pueden ser limitadas por tiempos de retraso y/o cargas eléctricas en la línea del bus.

Canal único: el bus consiste en un canal único que transporta bits. La información de resincronización puede derivarse de estos datos. La forma de implementación del canal no está detallada por la especificación.

Comprobación de consistencia (ACK): todos los receptores comprueban la consistencia de un mensaje recibido y reconocerán un mensaje consistente y marcarán uno que no lo sea.

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. Por ejemplo en caso de una implementación wired-AND del bus, los bits dominantes serían representados por un nivel lógico cero mientras que un nivel recesivo por uno lógico uno. Los estados físicos que representan los niveles lógicos no son detallados por la especificación. En la Figura 3.3 se muestran tres nodos transmitiendo conectados al bus y aplicando el mecanismo recién mencionado.

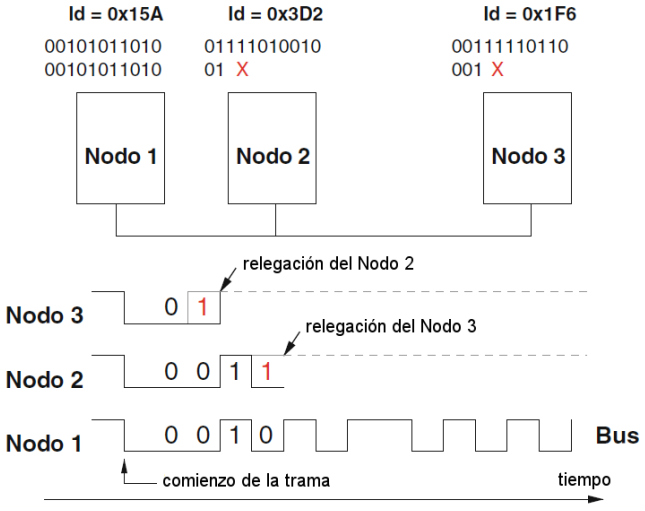

Figura 3.3: Arbitraje en bus CAN

3.1.2.1.3 Estructura de mensajes

Existen dos tramas diferentes que se distinguen por la longitud del campo de identificación. Las tramas con 11 bits para el identificador son denominadas tramas estándar, las tramas con 29 bits en el mismo campo se llaman tramas extendidas.

Tipos de tramas: la transferencia de mensajes se manifiesta y controla por cuatro tipos de trama diferentes. La 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. Las tramas de error se transmiten 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.

Trama de datos: se compone de siete campos de bit diferentes: comienzo de trama, campo de arbitraje, campo de control, campo de datos, campo CRC, campo ACK y fin de trama. El campo de datos puede ser de longitud cero. En la Figura 3.4 se puede observar los campos de la trama de datos.

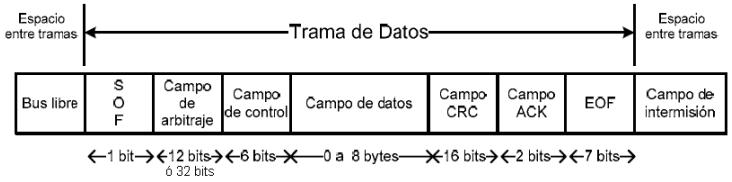

Figura 3.4: Trama CAN

Comienzo de trama (SOF, Start of Frame): 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 desocupado. Todas las estaciones se tienen que sincronizar con el flanco de comienzo de la trama de la estación que empezó la transmisión.

Campo de arbitraje: el formato de este campo es distinto para las tramas estándar y las extendidas. En las tramas estándar el campo de arbitraje está formado por los 11 bits de identificación más el bit de petición de transmisión remota (Remote Transmission Request). En la trama extendida el campo consiste en los 29 bits de identificación, el bit Substitute Remote Request (SRR), el bit Identifier Extension (IDE) y el RTR. En la Figura 3.5 se grafican las diferencias de las tramas estándar y extendida.

Campo de control: consiste en seis bits que son diferentes para las tramas estándar y extendida. Las tramas estándar incluyen al código de longitud de datos (DLC, Data Length Code), el bit IDE que es dominante y el bit reservado r0. La trama extendida está compuesta por el DLC y dos bits reservados r1 y r0 que son enviados en estado recesivos. El código de longitud de datos es de cuatro bits y representa el número de bytes en el campo de datos.

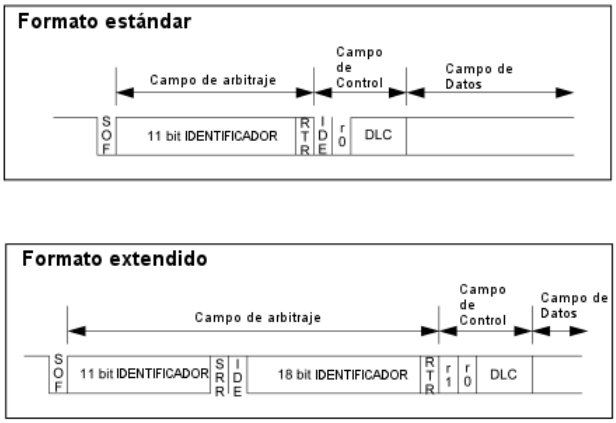

Figura 3.5: Tramas estándar y extendida

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 de 15 bits 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.

3.1.2.2 Soluciones de hardware en estos casos

Para la elección del protocolo CAN se ha tenido en cuenta no sólo sus características sino también sus antecedentes en el área de incumbencia. Entre los trabajos referidos se menciona su utilización para realizar la comunicación central entre subsistemas en satélites de órbita baja (LEO, Low Earth Orbit) [24]. También en el diseño de sistemas modulares distribuidos de vuelo (MDAS, Modular Distributed Avionics Systems) en vehículos lanzadores [25]. Y como se ha explicado en la sección 2.4 la ESA propone al protocolo CAN como uno de los buses más usados en los sistemas de a bordo de sus misiones espaciales.

Como se detallará en la siguiente sección, los circuitos integrados de los sistemas aeroespaciales deben soportar ciertas condiciones de operación. Entre ellas se encuentra la exposición a la radiación cósmica que obliga a proteger al encapsulado de los componentes, encareciendo la fabricación de los mismos.

Se puede notar la utilización de componentes comerciales o Commercial off the Shelf (COTS) para realizar la comunicación entre los distintos instrumentos de vehículos de vuelo comerciales [26].

Aunque se encuentran pocos trabajos referidos a este tema en el campo aeroespacial existe una gran tendencia al uso de componentes COTS. Específicamente en el desarrollo de satélites se han realizado experiencias que abarcan tanto a sistemas de navegación [27] como a misiones completas [28] con este tipo de componentes.

Las etapas de diseño físico y constructivo del hardware deben ser precedidas por un análisis de fallas exigente y riguroso, ya que la utilización de componentes COTS exige determinar el rango de operación más allá del estipulado por el fabricante.

3.1.3 Clasificación y tolerancia a fallas

Los sistemas están formados por componentes que a su vez pueden considerarse subsistemas en donde existe la posibilidad de errores como consecuencia de fallas. En el caso que estos errores provoquen una anomalía notable en el funcionamiento del sistema, las fallas habrán generado una avería. Se utilizan técnicas de tolerancia a fallas para evitar averías y para implementarlas es necesario realizar una correcta evaluación del sistema que consiste en identificar posibles fallas, evaluarlas, seleccionarlas y gestionarlas. Además se deben determinar qué riesgos necesitan ser manejados a través del plan de contingencias [29]. En la Figura 3.6 se muestra en un gráfico la clasificación de fallas.

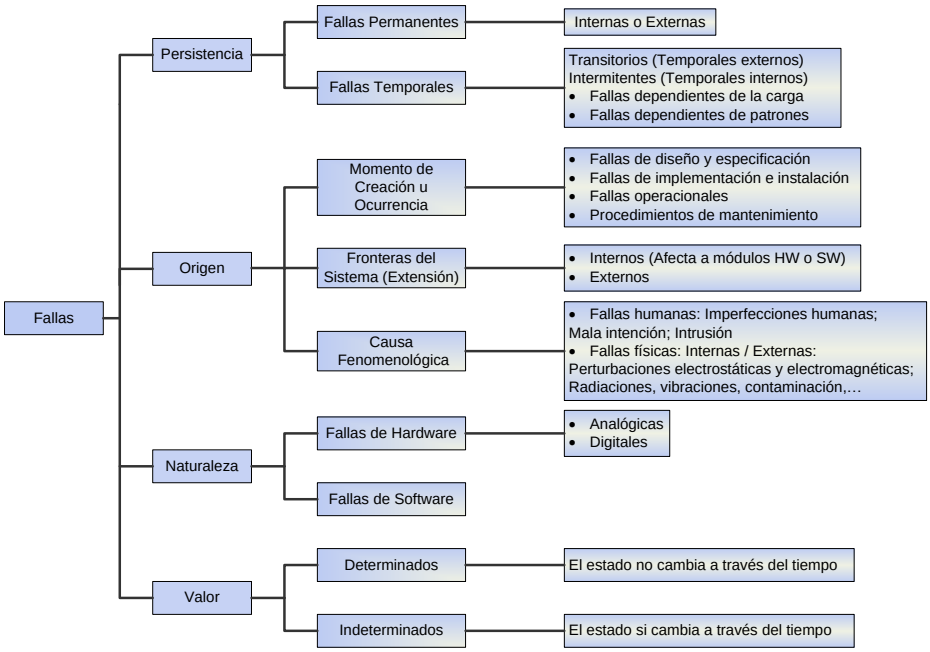

Figura 3.6: Clasificación de fallas

Una manera de cuantificar la robustez de una red es estimar la conectividad de la misma. Es decir, calcular la probabilidad de fallas de un enlace o nodo. Para conseguir esto generalmente se realiza un análisis de confiabilidad del hardware por medio de la tasa de fallas de los componentes.

La tasa de fallas se define como el número de fallas que puede ocurrir en un componente por unidad de tiempo durante un determinado intervalo de tiempo futuro. La tasa de fallas depende

del tiempo de vida del componente, la tensión de alimentación, estrés del entorno, temperatura ambiente y la tecnología utilizada para su fabricación.

En cuanto al tiempo de vida se puede decir que los componentes tienen una tasa de fallas alta en su etapa de funcionamiento inicial, luego a medida que pasa el tiempo prevalecen los que poseen pocas fallas y finalmente la tasa de fallas aumenta por el envejecimiento de los materiales. Esto puede verse reflejado en la conocida curva de la bañera mostrada en la Figura 3.7.

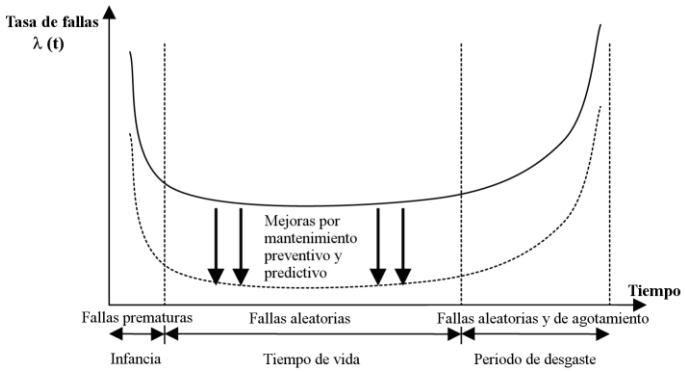

Figura 3.7: Curva de la bañera

El impacto de los otros factores puede reflejarse en la siguiente formula empírica [30]:

$$\lambda = \pi_L \pi_Q (C_1 \pi_T \pi_V + C_2 \pi_E) \tag{3.1}$$

En donde

λ es la tasa de fallas.

π_L es el factor de aprendizaje y está vinculado a la madurez de la tecnología utilizada.

π_Q es el factor de calidad y representa la calidad del proceso de fabricación. El rango del factor es de 0,25 a 20.

π_T es el factor de temperatura, su rango es de 0,1 a 1000. Es proporcional a $e^{-Ea/kT}$, en donde E_a es la energía de activación asociada a la tecnología en electrón-volts, k es la constante de Boltzmann ($0{,}8625 \times 10^{-4}$ eV/K) y T es la temperatura en Kelvin.

π_V es el factor de estrés a causa de la tensión eléctrica en dispositivos CMOS; su rango es de 1 a 10 y depende de la tensión de alimentación con la temperatura.

π_E es el factor de shock ambiental; su rango varia de 0,4, en entornos muy benignos como pueden ser los refrigerados, hasta 13 en entornos hostiles a causa de por ejemplo las vibraciones.

C_1 y C_2 los factores de complejidad están en función del número de compuertas en el circuito integrado y el número de pines en el encapsulado.

Los dispositivos que operan en el espacio, como están en contacto con partículas cargadas y sufren cambios bruscos de temperatura, pueden fallar más a menudo que otros en entornos más benignos. A causa de esto se han desarrollado técnicas de tolerancia a fallas para este tipo de aplicaciones críticas.

El objetivo de la tolerancia a fallas es permitir la ejecución correcta de las tareas a pesar de la ocurrencia de fallas, detectándolas, enmascarándolas y recuperando al sistema utilizando redundancia. Pero aplicarla incrementa la complejidad de todo el sistema ya que agregar componentes puede aumentar la probabilidad de fallas, el costo, latencia y *overhead*. Por otra parte introducir elementos para detectar fallas y recuperar el sistema provocará una disminución del rendimiento, por lo que surge la necesidad de reducir al mínimo la redundancia.

El grado de tolerancia a fallas necesario depende de la aplicación y de las consecuencias de las fallas. En sistemas de tiempo real críticos la probabilidad de fallas es muy pequeña, se encuentra en el rango de 10^{-4} a 10^{-10} [31]. Dependiendo del subsistema, en los vehículos lanzadores se requiere de una probabilidad de fallas mucho menor que en los satélites, ya que éstas provocan perdidas más importantes que abortar o no completar una misión. Generalmente en estos sistemas se implementa una tolerancia completa aunque en ciertos casos puede contemplarse una degradación aceptable.

Para incorporar tolerancia a fallas debe tenerse en cuenta que el trabajo de recuperación no puede ocupar un tiempo mayor al que llevaría realizar nuevamente el trabajo.

Existen dos estrategias básicas para llevar a cabo la recuperación de errores, directa o hacia delante FER (forward error recovery) e inversa o hacia atrás BER (backward error recovery). En la recuperación directa se avanza desde un estado erróneo haciendo correcciones sobre partes del estado mientras que en la recuperación inversa se retrocede a un estado anterior correcto que se ha guardado previamente [8].

En los sistemas de tiempo real generalmente se implementa la recuperación directa ya que el sistema no recupera el último estado consistente. Además las restricciones temporales no permiten retraerse a estados anteriores, sino que utiliza la información de diagnóstico de falla e intenta reconstruir un estado de ejecución válido para continuar con los servicios [29].

3.2 Análisis de las posibles soluciones a los problemas abiertos

3.2.1 Tolerancia a fallas en la implementación

Las condiciones de funcionamiento que imperan en el entorno de servicio de misiones aeroespaciales son de máxima exigencia en varios aspectos. Se enumeran a continuación algunas características básicas que deben poseer los instrumentos.

- Que soporten grandes variaciones térmicas: las variaciones provocadas en las características de medición del instrumento no deben ser mayores a aquellas a partir de las cuales se compromete la veracidad de los datos de navegación. Básicamente se

requiere una sensibilidad a las variaciones de temperatura acotada en todo el rango operativo térmico que imponga la misión [32].

- Que soporten altas exigencias mecánicas (vibraciones y shock): este requerimiento está condicionado por el perfil de despegue, momento en el cual se establecen las excitaciones mecánicas más violentas de cualquier misión aeroespacial.

- Que soporten exposición a ambientes de baja presión (vacío): dependiendo del tipo de misión, este requerimiento condicionará la elección de materiales. Existen ciertos materiales que son estrictamente rechazados debido a que se deterioran estructuralmente cuando son sometidos a vacío, por fenómenos de "desgase" (*outgassing*) [33]. Un ejemplo típico es el uso de polímeros, que manifiestan un rápido deterioro inclusive en órbitas bajas (a partir de los 100 km de altura).

- Que soporten exposición a radiación cósmica y bombardeo de partículas provenientes del espacio profundo: en componentes electrónicos el bombardeo de partículas generan comportamientos anómalos progresivos, si es que no se toman los recaudos convenientes. Muchos fabricantes de dispositivos electrónicos brindan versiones "endurecidas" (*hardrad*) de sus productos, que cuentan con blindajes que protegen las pastillas de silicio. Dichos dispositivos son conocidos como componentes de calidad espacial [34].

- Que verifiquen todo requerimiento funcional con un empleo mínimo de recursos energéticos, el diseño empleado en vehículos aeroespaciales está fuertemente condicionado por las limitaciones impuestas en los sistemas de potencia eléctrica abordo. El suministro energético a los subsistemas es estrictamente calculado con el fin de dimensionar correctamente el reservorio de energía eléctrica. Para minimizar tanto el tamaño como el peso de las baterías empleadas, se imponen límites de consumo eléctrico a todo dispositivo.

- Que sus diseños mecánicos sean capaces de evacuar la energía térmica generada por el funcionamiento de las unidades electrónicas que contienen, mediante mecanismos únicamente conductivos, debido a la carga ambiental representada por el vacío [35]. La ausencia de atmósfera no permite el empleo de vías de evacuación de calor mediante la transmisión basada en fenómenos convectivos.

- Que sus diseños mecánicos electrónicos verifiquen los requerimientos de compatibilidad electromagnética ya que todo sistema de abordo no debe generar perturbaciones electromagnéticas que condicionen el funcionamiento de otro sistema vecino. Asimismo, los diseños deben garantizar el grado de inmunidad electromagnética mínimo especificado por los requerimientos de la misión [36].

Todos estos requisitos se cumplen mediante el empleo de diversas consideraciones en las estrategias de diseño mecánico, térmico, electromagnético, electrónico, de programación, entre otros. La confiabilidad del producto debe ser la principal meta a alcanzar, por lo cual se

implementan políticas de diseño con factores de seguridad altamente conservadores. Las fallas que potencialmente pueden ser provocadas por las condiciones de trabajo citadas más arriba son muy variadas pero puede mencionarse algunas de las más importantes para el sistema de comunicaciones.

- Fallas mecánicas.
- Fallas de hardware (electrónica).
- Fallas de software.

Las fallas mecánicas provocadas por vibraciones o presión generalmente se solucionan seleccionando componentes de montaje apropiado y resinas que embeben a estos completamente [37].

Como se ha mencionado en la sección anterior, luego del análisis de fallas se implementa la estrategia de tolerancia a fallas que se adecue al área de aplicación. Generalmente se fabrica un prototipo con el que se demostrará de forma práctica el funcionamiento del sistema. En caso de no resultar satisfactorio en alguno de los requerimientos, se deben realizar reformas hasta cumplir con las metas fijadas. El tiempo de desarrollo es fundamental en este tipo de proyectos y es por esto que se propone como solución las herramientas de codiseño.

3.2.2 Simulación para acortar tiempos de desarrollo

Una de las características de estos sistemas es el alto grado de heterogeneidad por la combinación de hardware a medida y software empotrado. Esta heterogeneidad impone, en una de las principales etapas del diseño, decidir qué funcionalidad se implementará vía software y cual vía hardware. De hecho esta decisión define la arquitectura del sistema ya que indicará los diferentes componentes que son necesarios diseñar.

A causa del aumento de la complejidad del hardware se han propuesto soluciones tecnológicas como los System on Chip (SoC), que generalmente están formados por bloques prediseñados llamados Intellectual Property (IP). Muchas veces los IPs se interconectan dentro de los SoCs por medio de estructuras de comunicación como los buses compartidos o Network on Chip (NoC). El paradigma para realizar el diseño con esta tecnología es el denominado Codiseño Hardware/Software.

Al enfrentar un problema de codiseño se deben explotar las ventajas de la heterogeneidad del sistema objetivo. Es decir, aprovechar las fortalezas del software (mayor flexibilidad) y las del hardware (mayor velocidad) pero también es imperioso acotar los tiempos de desarrollo y depuración. Si bien utilizar codiseño reduce la complejidad individual de los componentes del sistema, por otro lado puede complicar el proceso de diseño.

Desarrollar una tarea de diseño implica la ejecución de un conjunto de subtareas conocidas comúnmente como flujo de diseño. En un sistema hibrido o heterogéneo el flujo de diseño debe contemplar el desarrollo en forma conjunta de hardware y software incluyendo descripción a nivel

de sistemas, simulación funcional, partición hardware-software, generación de interfaces, co-simulación, síntesis e implementación de hardware /software e interfaces. Esta actividad interdisciplinar es lo que se denomina codiseño.

3.2.2.1 Flujo de diseño tradicional

Si bien existen desde hace años herramientas para realizar la compilación del software y la síntesis del hardware, las decisiones más importantes para tener en cuenta la partición hardware-software se han ido tomando basándose en experiencias previas de diseño. En la Figura 3.8 puede verse el flujo de diseño tradicional.

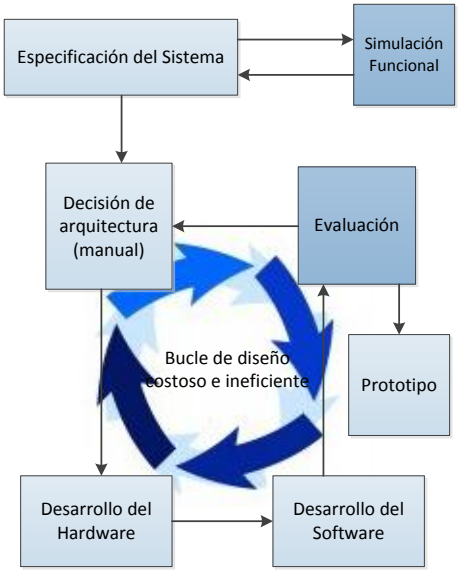

Figura 3.8: Flujo de diseño tradicional

3.2.2.2 Flujo de diseño con herramientas de codiseño

Posiblemente uno de los problemas más significativos del codiseño sea la partición hardware-software. Esta partición puede depender de distintos factores como el tamaño, de área y de memoria, velocidad de procesamiento o el consumo de potencia.

Utilizando herramientas específicas de software para realizar codiseño hardware-software se puede esperar un diagrama de flujo de diseño como el de la Figura 3.9.

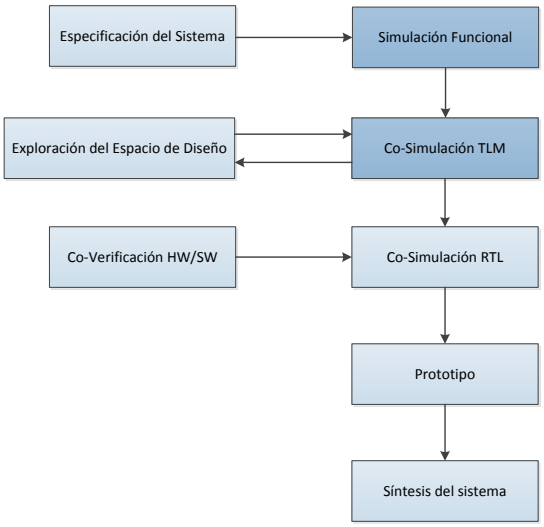

Figura 3.9: Flujo de diseño con herramientas de codiseño

Al comparar este flujo de diseño con el tradicional se puede notar, como principal diferencia, que la etapa de simulación funcional es obligatoria para llegar al prototipo y síntesis del sistema.

Generalmente cada uno de los pasos se realiza con una herramienta diferente, por lo que la integración de las distintas herramientas en un único flujo de diseño no es una tarea sencilla.

La descripción del sistema se realiza en la etapa de especificación del sistema y por medio de un lenguaje de abstracción que esté por encima de los lenguajes con los que se describirá el software (C, Pascal) o el hardware (VHDL, Verilog). Esto es fundamental para poder estudiar individualmente los bloques funcionales, abstrayéndose de la implementación, hardware o software, de los mismos.

En la etapa de simulación funcional del sistema se comprueba si realmente la aplicación descripta coincide con las especificaciones. Los lenguajes con los que se realiza la descripción a alto nivel normalmente cuentan con simuladores que permiten realizar simulaciones funcionales. Sólo cuando la simulación funcional cumple con las especificaciones del sistema tiene sentido continuar con el flujo de diseño para encontrar la arquitectura adecuada.

La etapa de exploración del espacio de diseño es el momento de evaluar las distintas alternativas de particionado y en la co-simulación TLM se realiza una simulación a nivel transaccional. En las dos etapas se utiliza la herramienta TLM.

En la coverificación de hardware-software se comprueba si el sistema cumple con las funcionalidades conseguidas durante la simulación funcional y con requisitos adicionales de *timing* impuestos luego del particionado. Esta etapa se realiza normalmente a través de una cosimulación a nivel RTL.

En las siguientes etapas se implementa el prototipo con el que se demuestra de forma práctica el sistema hibrido funcionando. Tras comprobar que el prototipo funciona correctamente y es apropiado para la aplicación se procede a realizar la síntesis del sistema final.

3.2.2.3 Simulación funcional

El desarrollo de un sistema aeroespacial está divido en cuatro etapas [6] antes del lanzamiento del mismo. Prácticamente el sistema es sometido a un constante rediseño y verificación poco antes de terminar con la integración de todos los subsistemas. La complejidad en atender todos los requerimientos (algorítmicos, de software y hardware), teniendo en cuenta las restricciones a causa de posibles fallas, promovió a que en la actualidad se realice el diseño y verificación en esta área con tecnologías de simulación. En estos casos los modelos de simulación sólo reflejan la funcionalidad del sistema, es decir, modos de operación, protocolos de comunicación del sistema y consumo de potencia.

En este trabajo se abordaron las etapas de especificación del sistema y simulación funcional, se utilizó como lenguaje de descripción de sistemas al *framework* SystemC y por medio de su *kernel* de simulación se implementó la simulación funcional. Esto se detallará en los siguientes dos capítulos.

4 Capítulo 4: Herramientas utilizadas para la propuesta de solución

En este capítulo se realiza una descripción general del prototipo desarrollado para llevar a cabo las pruebas del sistema de comunicaciones y posteriormente la verificación del modelo de simulación.

Luego se detallan las características de SystemC con las cuales se realiza el diseño y desarrollo del modelo de simulación de la red CAN.

Finalmente se presentan los requerimientos del sistema y las herramientas de hardware y software utilizadas en este trabajo.

4.1 Propuesta de solución

En el final del capítulo anterior (sección 3.2.2) se propone el uso del codiseño hardware-software como solución para acortar tiempos de diseño y desarrollo de un sistema de comunicaciones. Como se ha explicado, esta propuesta confluirá en una simulación funcional que implica la planificación y construcción de un modelo de simulación. La validación del modelo de simulación es llevada a cabo por medio de un prototipo hardware que inicialmente fue construido para comprobar la viabilidad del protocolo de comunicaciones CAN en un sistema aeroespacial.

4.2 Implementación del sistema

4.2.1 Diseño hardware inicial

Como se ha indicado en el capítulo anterior (sección 3.1.1) se seleccionó al protocolo de comunicaciones CAN para formar una red de sensores con las unidades optoelectrónicas y otros dispositivos. Las unidades optoelectrónicas fueron dispuestas para proveer la velocidad angular inercial tri-axial a una computadora de a bordo por medio del bus CAN.

Inicialmente se diseñó y fabricó un prototipo para verificar la funcionalidad y características deseadas que debe poseer la red en este tipo de aplicación. A causa de los antecedentes mencionados en la sección 3.1.2.2 se decidió utilizar componentes del tipo COTS para la selección de los circuitos integrados. Esto último implicó tomar mayores recaudos en el momento de diseño del Printed Circuit Board (PCB), que integra a todos los componentes, para cumplir con las condiciones operativas señaladas en la sección 3.2.1.

El prototipo desarrollado está conformado por seis nodos con las siguientes características: tres unidades que se han construido y armado completamente, dos unidades integradas en una interfaz CAN-PCI [38] y otra en una interfaz CAN-USB [39]. En la Figura 4.1 puede verse el conexionado de los nodos de la red y los elementos que posee cada estación.

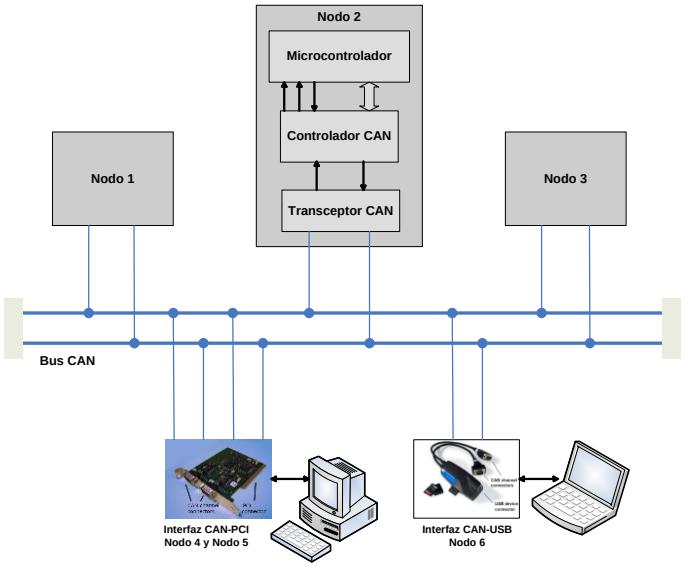

Figura 4.1: Prototipo de prueba

Cada nodo está integrado por tres componentes: microcontrolador, controlador CAN y tranceptor CAN. El microcontrolador funciona como *host* del controlador CAN configurándolo y administrando el tráfico de datos en los buffers de recepción y transmisión del mismo. También realiza el direccionamiento del IFOG y la lectura del bus de datos de este último.

La función del controlador es implementar en hardware toda la especificación CAN relativa a la descripción de la capa de enlace y parte de la capa física del modelo OSI. Con este componente se realiza la comunicación entre el microcontrolador y el transceptor.

La función del transceptor o transceiver es acondicionar las señales entregadas por el controlador confiriéndoles las características eléctricas estipuladas en la especificación CAN. Este elemento une al controlador CAN con el medio de transmisión.

Los detalles de los circuitos integrados se describirán en el siguiente capítulo para puntualizar técnicamente el entorno experimental del hardware.

Con este prototipo se llevaron a cabo tres tipos de pruebas para garantizar el funcionamiento del sistema en condiciones operativas:

- Pruebas mecánicas: se sometió al sistema ensamblado a un ensayo mecánico que consiste en excitarlo mediante vibraciones de diferente tipo. Esto pone a prueba la fortaleza de las estructuras mecánicas así como también la calidad de los vínculos eléctricos representados por soldaduras. Luego se ejecutó el plan de ensayos a partir de los cuales se determina el estado de salud (aliveness) del sistema.
- Pruebas térmicas: se sometió al prototipo a un ciclado térmico en un ambiente de baja presión. Se generan solicitaciones térmicas que redundan en estrés mecánico debido a dilataciones y contracciones. También se prueban los materiales empleados así como

también el funcionamiento de los dispositivos electrónicos luego del ciclado, verificando los rangos operativos de los mismos.

- Pruebas del sistema de comunicaciones: se realizaron pruebas con el fin de evaluar la funcionalidad del bus CAN. Para ello se generó un tráfico en la red que consistió en la emisión, por parte del nodo representado por el IFOG, de un mismo dato a una tasa constante. Un nodo receptor se encargó de monitorizar el tráfico y hacer un registro del mismo.

Luego de la evaluación del prototipo por medio de estas pruebas se implementó la red CAN en las placas de vuelo del sistema aeroespacial, en este caso la IMU. Aunque el protocolo actualmente está operativo en el sistema aeroespacial, éste no contempla el caso de un fallo transitorio o permanente en alguno de los nodos que conforman la red.

Como se ha mencionado en el capítulo 3, una de las técnicas de tolerancia a fallas más usada en sistemas de tiempo real distribuido es la redundancia. De esta manera, en los sistemas aeroespaciales se incluyen dos o más elementos capaces de realizar la misma función. La penalización en masa, volumen y consumo es importante pero el riesgo asumido en caso de falla sin la existencia de redundancia es mayor en estos tipos de sistemas. Las redundancias hardware más importantes en aplicaciones de vuelo [40] pueden verse en la Figura 4.2.

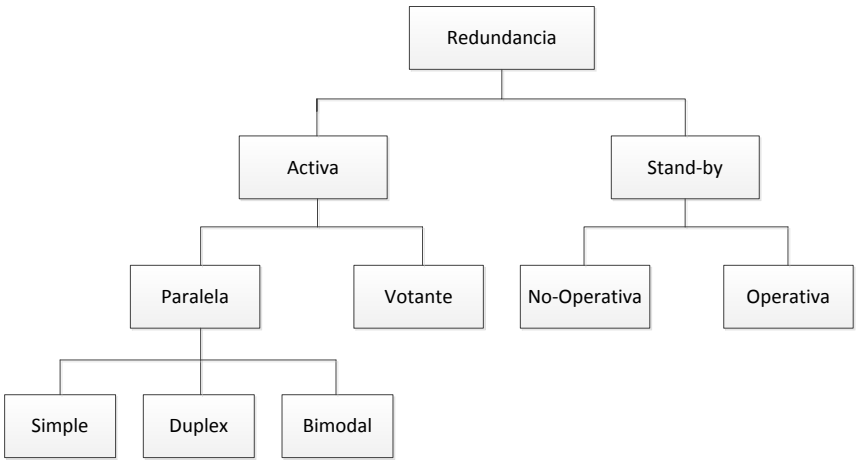

Figura 4.2: Tipos de redundancias de hardware

Como se puede notar de la explicación del desarrollo del prototipo, el flujo de diseño utilizado fue el tradicional pero sin ningún tipo de simulación funcional ya que se requería acortar al máximo los tiempos de fabricación. Es decir, la metodología de diseño fue Bottom-Up o ascendente, seleccionando los circuitos integrados para conseguir que la suma de sus efectos realice la aplicación deseada. Si la intención es implementar redundancia de alguno de los componentes del prototipo, las falencias de este flujo de diseño se reflejan en la poca flexibilidad para realizar cambios, el tiempo de rediseño y el costo para generar el nuevo hardware.

4.2.2 Lenguajes de diseño

En base a lo explicado anteriormente y en la sección 3.2.2.2, la utilización del flujo de diseño con herramientas de co-diseño o metodologías Top-Down permiten la descripción del sistema al más

alto nivel. En este tipo de diseño, la dependencia de la implementación final es prácticamente inexistente en las etapas de definición funcional hasta llegar a la síntesis del sistema sobre cualquiera de las tecnologías existentes, como ser full-custom, ASICs, FPGAs, CPLDs, entre otros. Conseguir a estos dispositivos con encapsulado endurecido no es problema ya que actualmente existen FPGAs para aplicaciones aeroespaciales. Entonces se puede concentrar el esfuerzo en la descripción del sistema a nivel funcional y de comportamiento antes de abordar el diseño detallado a nivel físico.

Al utilizar metodologías a nivel de sistemas la etapa inicial de especificación se convierte en una fase crucial del diseño. Es necesario que la metodología esté basada en un lenguaje unificado para la especificación de sistemas heterogéneos o hardware-software. Y los lenguajes de descripción de hardware como VHDL o Verilog, o los lenguajes de programación como C, ADA o Java no presentan por sí solos características suficientes para describir el sistema completo.

Por lo tanto, se proponen modelos de la plataforma hardware más abstractos, como los modelos de nivel de transacciones (TLM) [17] y la utilización del *framework* SystemC como lenguaje de especificación unificado para el diseño a nivel de sistemas.

4.3 Estructura de SystemC

Con el fin de reducir la complejidad de diseño, un sistema de SystemC consiste en una jerarquía de módulos [7]. El diseño de los componentes es encapsulado en los módulos, y éstos 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. Además, la comunicación entre módulos se realiza mediante canales. En la Figura 4.3 pueden verse distintos componentes de SystemC.

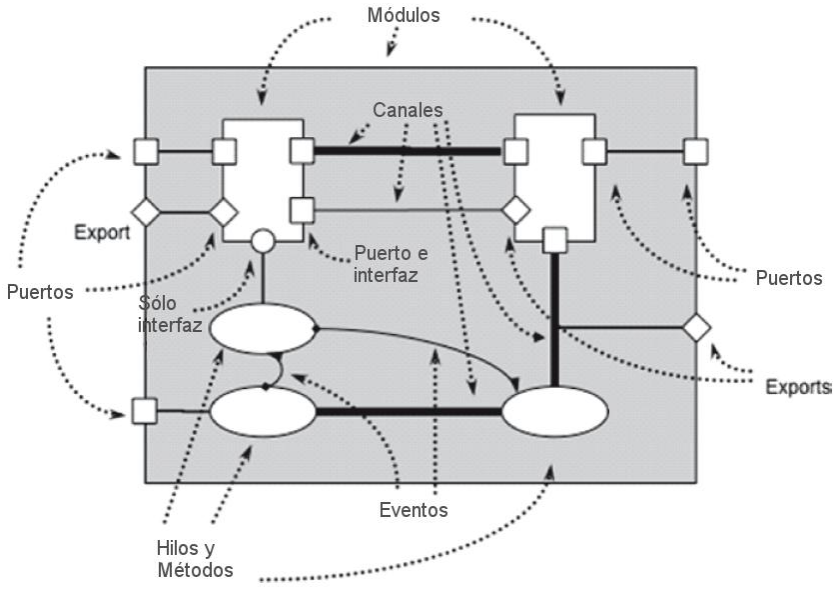

Figura 4.3: Componentes de SystemC

4.3.1 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; y pueden ser grandes o pequeños aunque generalmente contienen una jerarquía de componentes pequeños.

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 para conectividad jerárquica. Puede representar hardware, software o cualquier entidad física.

De esta manera, los diseñadores pueden dividir sistemas complejos en partes más pequeñas. Un módulo puede contener:

- Puertos
- Canales
- Procesos
- Eventos
- Otros módulos (sub-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++.

4.3.2 Procesos

El proceso de simulación es la unidad básica de ejecución en SystemC. Todos los procesos son registrados con el *kernel* de simulación de SystemC y son llamados sólo por éste. Durante la simulación, todo el código que se ejecuta es iniciado a partir de uno o más procesos, que son ejecutados en forma concurrente.

Entonces se puede definir a un proceso SystemC como una función miembro o método de clase de un módulo que es invocado por el *scheduler* del *kernel* de simulación.

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

Un hilo es igual a un 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 terminar con su ejecución, 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 un canal de comunicación.

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

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)*.

4.3.3 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. Entre los canales primitivos pueden encontrarse a los *sc_mutex* y *sc_semaphore* que son iguales a un mutex y semáforo de software tradicional, generalmente utilizadas como primitivas de sincronización.

Por otra parte, 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, entre otros.

4.3.4 Interfaces

Una interfaz es un conjunto de métodos de acceso. No proporciona la implementación de los métodos sino que es puramente declarativa, o sea, es una clase abstracta que provee métodos declarados pero implementados en canales o puertos.

Las interfaces se vinculan 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.

4.3.5 Puertos

Un puerto es el mecanismo principal para permitir la comunicación a través de las fronteras de un módulo. Básicamente es un puntero a un canal que no se encuentra en el módulo donde está definido el puerto.

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.

4.3.6 Scheduler (Planificador)

El *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*.

4.3.7 Eventos

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.

4.3.8 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.

4.3.9 Dinámica de la simulación

El simulador de SystemC se divide en dos etapas principales de operación: elaboración y ejecución. Una tercera etapa ocurre al final de la ejecución, está etapa puede definirse como de post-procesamiento o limpieza. En la Figura 4.4 se puede observar el *kernel* de simulación de SystemC.

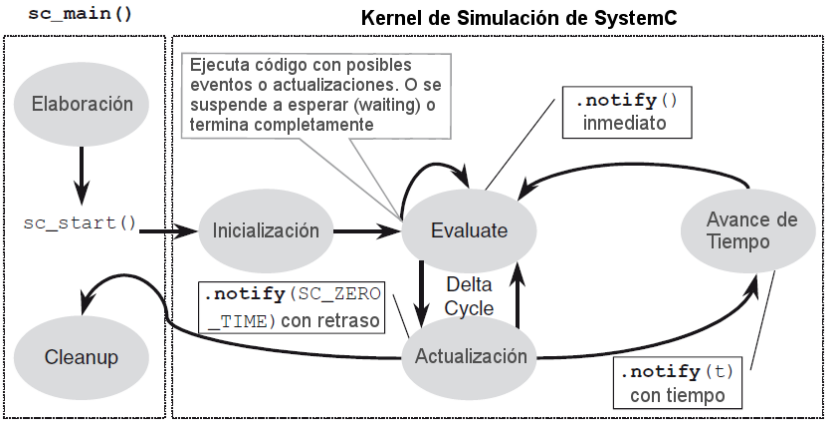

Figura 4.4: Kernel de simulación de SystemC

Inicialmente, durante la etapa de *elaboración,* se inicializan las estructuras de datos, se establece la conectividad y se prepara la segunda etapa, de *ejecución*. 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)*.

Varios procesos pueden empezar a ejecutarse en un mismo instante de simulación. En este caso, todos los procesos son evaluados y sus salidas son actualizadas (*update*). Una evaluación seguida de una actualización es denominada un *delta cycle*.

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.

4.3.10 Utilización 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. 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.

4.3.11 Trabajos relacionados al tema

Existen pocos antecedentes sobre la utilización del framework SystemC para el desarrollo de simuladores de subsistemas aeroespaciales. Se puede mencionar al uso de SystemC para la evaluación de buses de datos como el protocolo FlexRay [41]. En la comparación de la velocidad de diseño y simulación utilizando SystemC frente a VHDL, para el desarrollo del bus MIL-STD-1553 en el área de aviónica [42]. También en el análisis de performance y diseño de componentes, en la arquitectura de cajas negras o Motor Vehicle Event Data Recorders (MVEDR) en automóviles [43].

En cuanto al análisis del protocolo CAN por medio de SystemC, se pueden encontrar algunos trabajos en el área automotriz. Por ejemplo como lenguaje de base para el diseño de un framework de redes de sensores [44]. Y para realizar verificación de modelos de controladores CAN, en implementaciones de propósito general, a través de SystemC [45].

En el área espacial se recomienda la utilización del lenguaje unificado de modelado o UML [46] para el diseño de simuladores en sistemas espaciales [6]. Esto conlleva al uso de un lenguaje orientado a objetos que permita realizar una verificación del prototipo y soporte aplicaciones de tiempo real. Esta funcionalidad es provista por Java o SystemC (en realidad una variante de C++) que cuenta con clases para el modelado de redes y procesos de comunicaciones los cuales son esenciales en la representación de interconexiones de los equipos aeroespaciales.

4.4 Requerimientos del sistema a modelar

Las especificaciones del sistema están dadas por el tipo de aplicación, en este caso una IMU que envía las mediciones inerciales realizadas periódicamente a una computadora de a bordo. Al elegir al bus CAN como protocolo de comunicaciones de la IMU, se tienen en cuenta las siguientes características de la especificación CAN 2.0:

- Tasa de transferencia para cumplir con el tiempo de actualización o *refresco* de la computadora de a bordo.
- Incidencia del método de sincronización de señales de reloj para tener una referencia central, a pesar del *drift* del oscilador de cada nodo.
- El tipo de arbitraje que implementa el protocolo.
- Utilización de colas o buffers para llevar a cabo la transacción, al tratarse de una comunicación disparada por eventos.

Además en el diseño del modelo de simulación se debe contemplar el efecto del entorno de servicio sobre los distintos componentes desarrollados, con el propósito de recrear condiciones reales de funcionamiento. Este es el caso de la temperatura, como puede verse en la ecuación 3.1 es uno de los parámetros que más influye en la generación de fallas en este tipo de sistemas.

4.5 Herramientas de desarrollo

Se ha mencionado en la sección 4.1.1 que se desarrolló un prototipo previo al modelo de simulación. Con el fin de ordenar la descripción de las herramientas utilizadas en este trabajo, se las ha divido en dos secciones para discriminar las usadas tanto en el prototipo hardware como en el modelo de simulación.

4.5.1 Herramientas de hardware

Por medio de los microcontroladores se realizó la configuración de los nodos. Debido a que el microcontrolador elegido es de la marca ATMEL [47] se utilizó el entorno de programación KEIL IDE µVision [48] y el lenguaje C [49]. En la Figura 4.5 se puede ver una captura de pantalla del entorno.

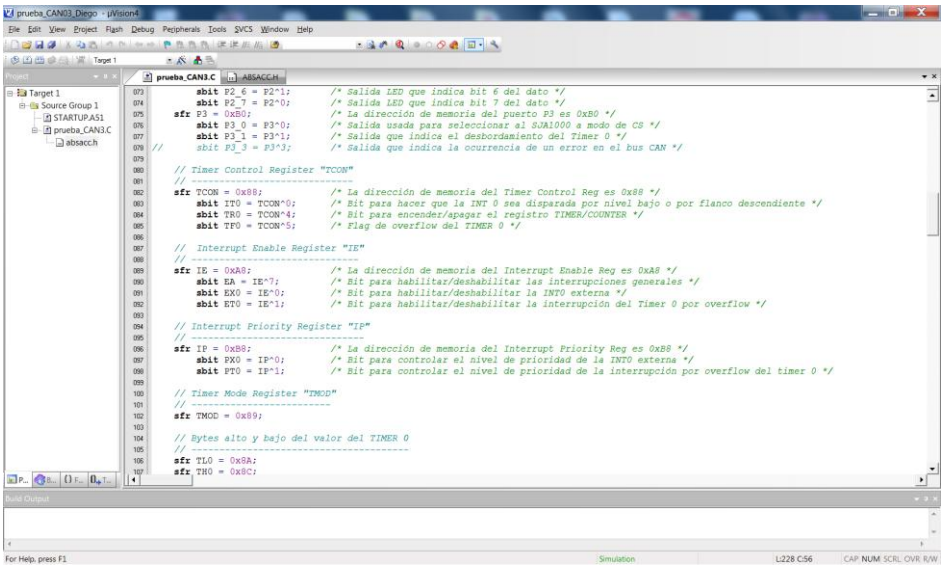

Figura 4.5: Entorno de desarrollo KEIL

Por medio de este entorno se genera el firmware que es embebido en el microcontrolador; esto se realiza a través de una conexión In-System Programmable (ISP) que posee cada placa y permite la conexión de una PC para efectuar la reprogramación del microcontrolador sin necesidad de removerlo del hardware de aplicación. El Software de programación ISP utilizado es el AT89ISP [50].

La validación del prototipo se realizó utilizando una herramienta de tráfico de redes propietaria llamada CanKing (basada en el protocolo de capa de aplicación Can Kingdom [51]) que actúa como monitor del sistema. Aunque con esta herramienta se puede realizar un relevamiento inicial de la red, no es posible mostrar información sobre las características principales de los mensajes que envían los IFOG en la etapa de pruebas. A causa de la necesidad de conseguir una traza de comunicaciones que incluya esta información, para este trabajo el autor participó en el desarrollo de una nueva herramienta denominada CanBuilder [52] que, además de validar el correcto funcionamiento de cada nodo de la red, es capaz de mostrar los principales parámetros medidos por el sensor IFOG en tiempo real. Esto permite una correcta depuración del sistema bajo una gran variedad de escenarios. En la Figura 4.6 se puede observar la interfaz de usuario de la aplicación CanBuilder.

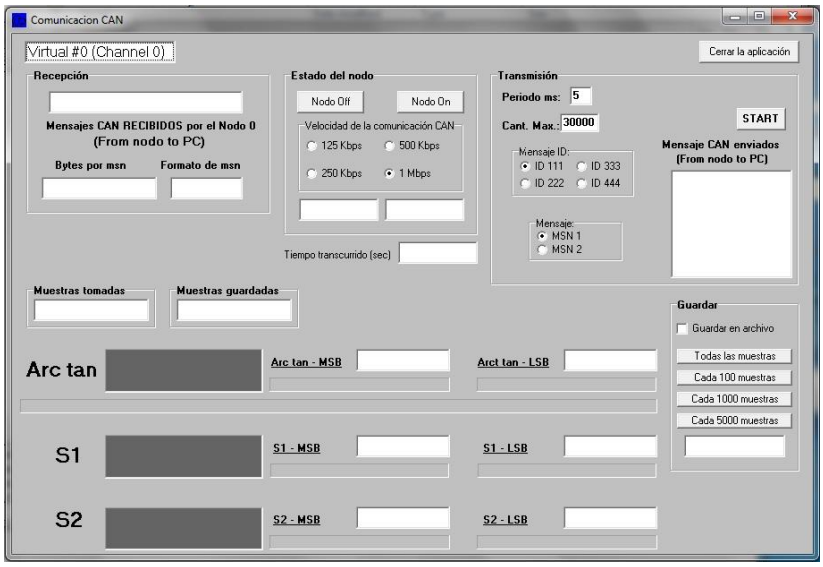

Figura 4.6: Interfaz del analizador de tráfico CANBuilder

CanBuilder es una herramienta desarrollada en C++ que, utilizando la librería freeware canlib.h [53], proporciona el acceso a la red CAN a través de la interfaz CAN-USB. De esta manera permite consultar dinámicamente los mensajes emitidos por los sensores IFOG, es decir los parámetros S1, S2 y Arc tan. La razón de medir estos parámetros surge de lo detallado en la sección 2.2.1, los sensores IFOG se basan en el efecto Sagnac, en este caso la determinación de la velocidad angular de rotación depende de la capacidad de medir el corrimiento de fase Sagnac. Debido a la modulación implementada en este sensor, el corrimiento de fase puede escribirse como:

$$\Delta\emptyset = \arctan\frac{S_1}{S_2}$$

En donde $\Delta\emptyset$ es el corrimiento de fase, S_1 y S_2 son los dos primeros armónicos de la serie de Fourier de la señal entregada por el sensor.

Además, CanBuilder provee mecanismos para el encendido/apagado del nodo CAN, configuración de la velocidad de comunicación (bit rate), personalización de los mensajes que circularán por la red (ID, datos del mensaje) y permite guardar un registro del experimento para un posterior análisis de los datos. También facilita el filtrado de mensajes y brinda información sobre la cantidad de mensajes entrantes al nodo de la interfaz y la cantidad de mensajes guardados, en tiempo real.

Si bien la empresa Kvaser proporciona una herramienta propietaria (CanKing) que permite realizar consultas interactivas sobre una red CAN, no soporta la visualización de los mensajes que circulan por la red en tiempo real. Aunque durante las pruebas iniciales para validar el correcto funcionamiento del prototipo puede prescindirse de esta característica, resultará indispensable para obtener información en tiempo real durante pruebas térmicas y mecánicas del prototipo. No obstante, CanKing resultó de utilidad para validar la fidelidad de los resultados obtenidos con CanBuilder.

4.5.2 *Herramientas de Software*

Como ya fue mencionado, el framework elegido es SystemC (en este trabajo se utiliza la versión 2.3.0 lanzada en Julio de 2012) que en realidad es una librería del lenguaje C++. El entorno de desarrollo para utilizar SystemC es Eclipse CDT que es de dominio público.

UML ya ha sido citado como una herramienta gráfica apropiada para el diseño de software. Pero el diseño de simuladores de sistemas aeroespaciales debe contemplar tanto el comportamiento de los componentes modelados como la topología del sistema, es decir, la forma de conexión entre los componentes. Una notación estandarizada para atender este problema es el Lenguaje de Modelado de Sistemas o System Modeling Language (SysML).

La herramienta para visualizar las señales producidas por el modelado de simulación es GTKWave [54]. Es un visualizador de formas de onda basado en GTK+, con el cual se pueden leer archivos tipo Value Change Dump (VCD) y generalmente es usado para depurar modelos de simulación Verilog o VHDL. En la Figura 4.7 se puede ver la visualización de señales por medio de esta herramienta.

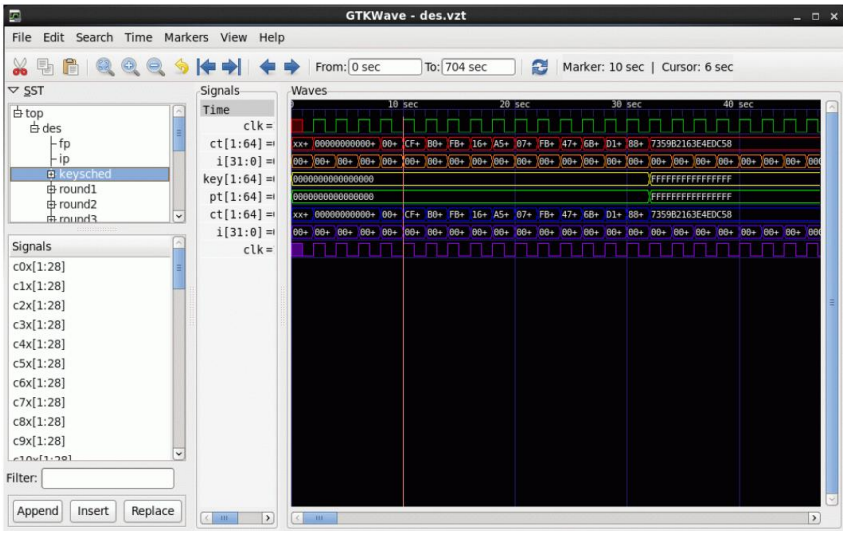

Figura 4.7: Visualizador de señales GTKWave

En el estado del arte actual se ha establecido y son conocidas variantes de estas herramientas a nivel de descripción de hardware. Aunque, como se ha indicado en los trabajos relacionados, la utilización de metodologías a nivel de sistemas en este tipo de aplicaciones ya es un hecho.

5 Capítulo 5: Diseño y desarrollo del modelo de simulación

En este capítulo se presenta una descripción de cada uno de los componentes del hardware a modelar y la manera de configurar al prototipo para cumplir con la funcionalidad del sistema.

También se describen los conceptos de diseño utilizado y un análisis de planificación del protocolo para realizar el diseño del modelo de simulación.

Por último se detalla el modelo final del sistema, su composición y semántica.

5.1 Descripción del sistema a modelar

Como se mencionó en el capítulo 2, antes de iniciar la construcción del modelo de simulación se debe recolectar información estructural del sistema a simular. En este caso la arquitectura de referencia es la del prototipo hardware implementado para comprobar la viabilidad del protocolo de comunicaciones CAN en un sistema de vuelo y explicado en el capítulo anterior (sección 4.2.1). En la Figura 5.1 pueden verse los nodos y la interfaz CAN-USB conectados al bus.

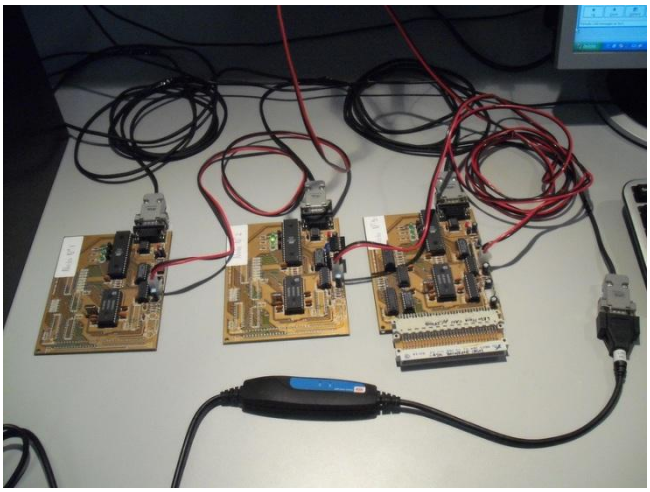

Figura 5.1: Fotografía del prototipo desarrollado e interfaz CAN-USB

Con el propósito de encontrar los datos que permitan la construcción del modelo de simulación, a continuación se realiza una descripción en mayor detalle de la composición y funcionamiento de los principales elementos del sistema de comunicaciones: los nodos y el bus. En la Figura 5.2 puede observarse un gráfico con detalles de uno de los nodos.

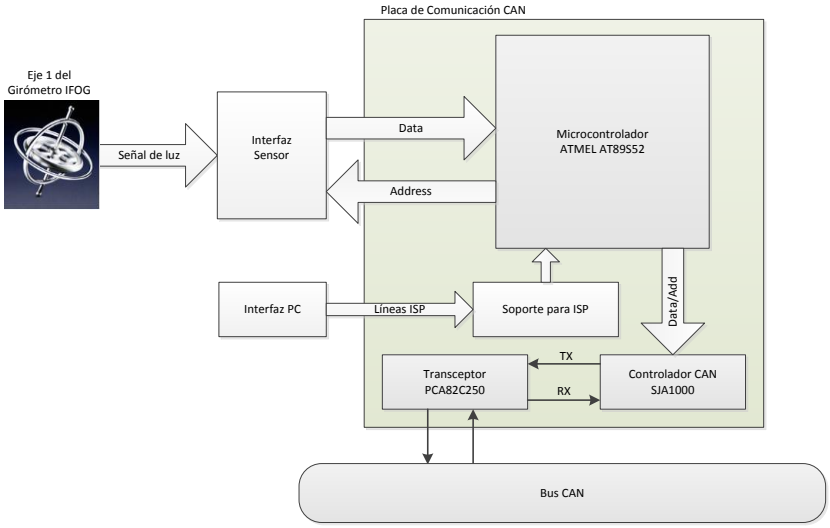

Figura 5.2: Componentes del prototipo

5.1.1 Nodos

Ya se ha explicado que cada placa posee tres componentes principales: microcontrolador, controlador CAN y transceptor CAN. El microcontrolador seleccionado fue el AT89S52 [55], que una vez configurado adecuadamente recibe el paquete de datos de la Interfaz Sensor (conectada a los girómetros IFOG) y transmite al controlador CAN realizando el *handshake* correspondiente para la correcta recepción del dato. En la sección 4.5.1 se han mencionado las distintas herramientas utilizadas para generar el firmware y la manera de embeberlo en el microcontrolador.

El controlador CAN elegido es el SJ1000 [56]: en la Figura 5.3 puede verse el diagrama en bloques provisto por su hoja de datos.

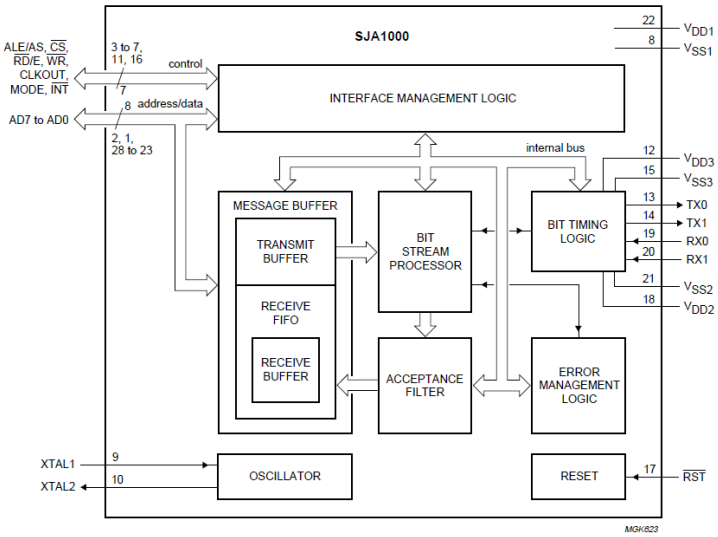

Figura 5.3: Controlador CAN

En el bloque indicado como Message Buffer se deposita la trama que es recibida o transmitida por el controlador, Transmit Buffer es la interfaz entre el *host* y el Bit Stream Processor (BSP) y Receive Buffer es la interfaz entre el Acceptance Filter y el *host*. La unidad Interface Management Logic (IML) recibe datos e identificadores a través de las líneas AD0/7 y completa los campos correspondientes para generar la trama CAN. El elemento BSP realiza la recepción (des-seriación) y la transmisión (seriación) de la trama. Además ejecuta el arbitraje, stuffing, dis-stuffing y detección de errores. Acceptance Filter compara el identificador recibido con el registrado en el nodo, y si son iguales el mensaje es depositado en el Receive Buffer. El componente Bit Timing Logic (BTL) monitorea el bus y controla el tiempo de bit, también se encarga de la sincronización, detección de comienzo de trama, desfasaje, retardo de bits y punto de muestreo. Finalmente el Error Management Logic (EML) lleva a cabo el confinamiento de errores, recibe el anuncio de errores del BSP e informa las estadísticas al BSP e IML.

Un análisis simplificado del funcionamiento del controlador CAN puede verse en la Figura 5.4.

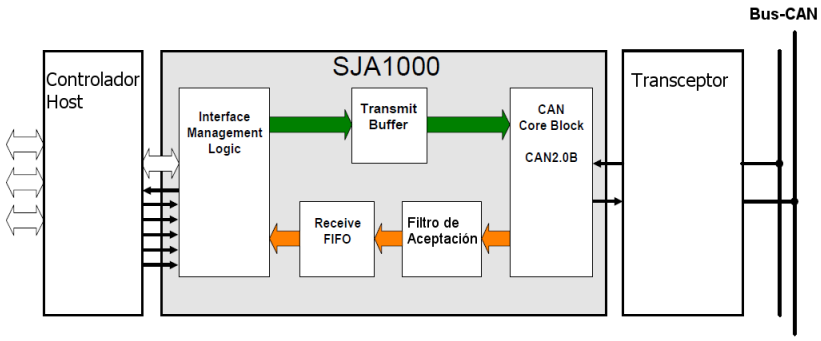

Figura 5.4: Diagrama en bloques del SJA1000

En donde el CAN Core Block agruparía funciones de las unidades BSP, BTL y EML; es decir, realizaría el control transaccional de las tramas. Puede notarse que la unidad IML efectúa el enlace con el controlador *host* externo que puede ser un microcontrolador siendo generalmente parte de un sensor. El acceso a cada registro del SJA1000 se ejecuta en esta unidad por medio del bus address/data y las líneas de control (strobe) read/write. Cuando se inicia una transmisión por el microcontrolador, el mensaje es depositado en el Transmit Buffer y la unidad IML fuerza al CAN Core Block a leer el mensaje del buffer. Todos los mensajes aceptados por el filtro de aceptación son depositados en el Receive Fifo.

El transceptor CAN es el PCA82C250 [57] e implementa el acceso al medio físico del controlador CAN en la red de acuerdo al estándar ISO 11898 [58]. En la Figura 5.5 puede verse una comparación de las especificaciones, modelo OSI e implementación en circuitos integrados del protocolo CAN.

<table>
<tr><th colspan="2">Especificación</th><th colspan="2">Capa OSI</th><th>Implementación</th></tr>
<tr><td colspan="2">Especificado por el diseñador del sistema</td><td colspan="2">Capa de Aplicación</td><td></td></tr>
<tr><td colspan="5"></td></tr>
<tr><td rowspan="5" colspan="2">Especificado por ISO 11898</td><td rowspan="2">Especificación Protocolo CAN</td><td rowspan="2">Capa de Enlace de Datos</td><td>Enlace de control Lógico</td><td rowspan="2">Controlador CAN</td></tr>
<tr><td>Acceso Control del medio</td></tr>
<tr><td rowspan="3">Capa Física</td><td>Señalización Física</td><td rowspan="2">Transceptor CAN</td></tr>
<tr><td>Acceso al medio físico</td></tr>
<tr><td>Interfaz dependiente del medio</td><td></td></tr>
<tr><td colspan="3">Medio de transmisión</td><td></td></tr>
</table>

Figura 5.5: Arquitectura de capas del protocolo CAN

5.1.2 Bus

Cada nodo se unió, por medio de conectores del tipo D-SUB 9, a un único bus de dos cables que poseen resistencias de terminación y adaptadoras de impedancia para impedir fenómenos de reflexión. Como la norma no especifica el cable, se eligió uno estándar que responde a la denominación AWG 24 y es muy utilizado para las redes RS-485. Se trata de un cable apantallado con un par trenzado en su interior del tipo Shielded Twisted Pair (STP) y de una impedancia característica de 120 Ohms.

5.1.3 Configuración del nodo

Con el propósito de conseguir la vinculación de los nodos a la red CAN se deben programar éstos para que realicen los siguientes pasos:

- Luego del encendido del sistema
  - Configurar al microcontrolador considerando los enlaces hardware y software con el SJA1000
  - Configurar las características de la comunicación en el controlador CAN tales como: selección del modo, filtro de aceptación, tiempo de bit, entre otros – también hacer esto luego de cada reset por hardware del SJA1000
- Durante el proceso principal de la aplicación
  - Preparar los mensajes a ser transmitidos y activar al SJA1000 para que los transmita
  - Reaccionar con los mensajes recibidos por el controlador CAN
  - Reaccionar con los errores ocurridos durante la comunicación

En la Figura 5.6 se puede ver un diagrama de flujo general de un programa de configuración para embeber en el microcontrolador.

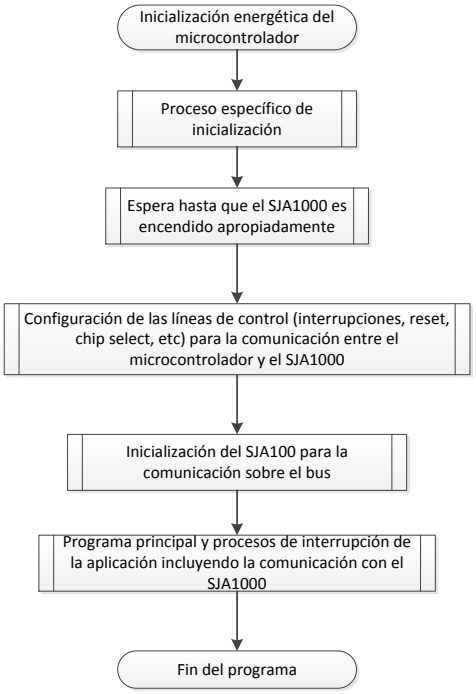

Figura 5.6: Diagrama de flujo general

La transmisión de un mensaje es realizada de manera autónoma por el controlador CAN SJA1000 de acuerdo a las especificaciones del protocolo. El proceso de transmisión puede ser controlado por interrupción o polling de un registro del SJA1000.

El diagrama de flujo para una transmisión controlada por polling puede verse en la Figura 5.7.

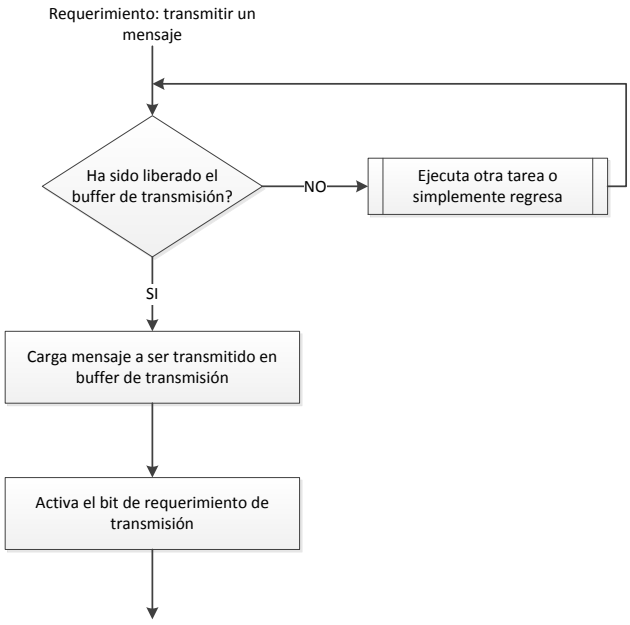

Figura 5.7: Diagrama de flujo de transmisión

El proceso de recepción se efectúa de manera similar al de transmisión ya que se ejecuta de manera autónoma y la transferencia del mensaje desde el SJA1000 al microcontrolador se puede controlar por una interrupción o por polling. El diagrama de flujo de una recepción controlada por polling se puede ver en la Figura 5.8.

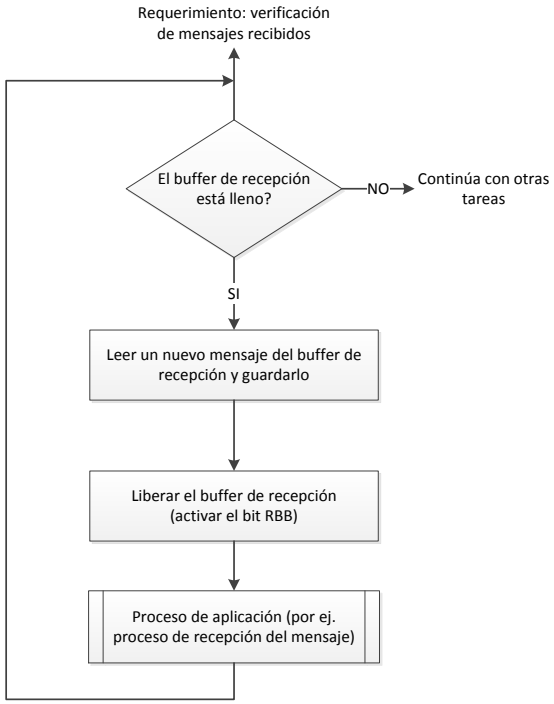

Figura 5.8: Diagrama de flujo de recepción

De esta manera los nodos fueron configurados y programados para enviar mensajes a una tasa periódica y a la máxima velocidad de transferencia que permite el protocolo. La monitorización de la red fue llevada a cabo por medio de la interfaz USB-CAN que también posee un controlador CAN del tipo SJA1000 y el analizador de tráfico CanBuilder ya explicado en la sección 4.5.1. En el próximo capítulo se mencionan los datos exactos de configuración de los nodos para realizar las pruebas de validación del modelo de simulación, las cuales se describirán.

5.2 Conceptos de diseño

En el transcurso de diseño del sistema, básicamente fueron aplicados dos conceptos: uno es el modelado a nivel de sistemas, y el otro es implementar las especificaciones y mecanismos de comunicación del protocolo de comunicaciones, pero siempre teniendo en cuenta los requerimientos del sistema en cuestión.

El modelado a nivel de sistemas tiene la ventaja de reducir la complejidad del sistema e incrementar la comprensión del mismo a un alto nivel de abstracción. La ejecución de simulaciones para detectar cuellos de botella en el diseño, más allá de los beneficios de implementación de hardware, permite explorar y validar alternativas de diseño en una fase inicial del proyecto lo cual aumenta significativamente la productividad [7].

Utilizando las clases provistas por SystemC, los nodos y los componentes que lo integran fueron modelados empleando la clase módulo. Los mecanismos de comunicación entre módulos se modelan mediante canales y se conectan a los módulos por medio de puertos. Mientras que las solicitudes de transacciones, se llevan a cabo a través de llamadas a métodos de las interfaces de los modelos de canal, que encapsulan los detalles de intercambio de información a bajo nivel.

Los parámetros configurables del modelo de simulación fueron planteados para estimar el retardo de comunicación del sistema frente a cambios de la temperatura de operación. Teniendo en cuenta este objetivo y contemplando la complejidad que conlleva generalmente la construcción de un modelo de simulación es que se implementan diferentes restricciones.

5.2.1 Restricciones de modelado

Considerando que el sistema modelado es de tiempo real y lo mencionado en la sección 2.1.1, cuantificar el retardo de respuesta es la forma de encontrar las limitaciones del sistema diseñado y asegurar que cumple con la funcionalidad buscada.

Existe una dependencia directa entre el tiempo de bit y el retardo de propagación de la señal pero éste está sujeto a varios retardos que derivan de la calidad de los componentes electrónicos elegidos en cada nodo: el controlador CAN aporta entre 50 y 62 nseg, el transceptor entre 120 y 250 nseg y el medio de transmisión alrededor de 5 nseg/m [59]. Para el modelo de simulación desarrollado en este trabajo no se tomó en cuenta estos tiempos de retardo ya que el resultado de sumar a todos ellos se encuentra en el orden de los nanosegundos, lo cual es muy bajo comparado con el tiempo de transmisión de un bit (1 µseg) a la tasa seleccionada.

En los sistemas de tiempo real es necesario restringir el no-determinismo encontrado dentro de los distintos subsistemas concurrentes que lo componen. Esta actividad es conocida como planificación y provee dos características: un algoritmo para ordenar el uso de los recursos y una manera de predecir el peor caso de las posibles reacciones del sistema cuando el algoritmo de planificación es aplicado. De esta manera se puede confirmar si los requerimientos temporales del sistema se cumplen [8].

5.2.1.1 Análisis de planificación del protocolo CAN

El modelo de planificación CAN utilizado es el propuesto en [23] y en sus trabajos previos. Los conceptos de planificación más importantes son:

- Se supone que cada nodo se ocupa de introducir, en el proceso de arbitraje, al mensaje de mayor prioridad encolado en su buffer de transmisión.
- Se supone que el sistema contiene un conjunto estático de mensajes de tiempo real duro. Cada uno (m) asignado con un ID fijo a un nodo de la red y un tiempo máximo de transmisión (C_m).
- Cada mensaje es encolado por una tarea, proceso o interrupción software que se ejecuta en el procesador *host*. Esta tarea es invocada por un evento o consulta de estado con una cantidad de tiempo limitada para encolar el mensaje que esté listo para transmitir. Este tiempo varía entre 0 y J_m, donde J_m es denominado el *jitter* de encolamiento del mensaje

- Se supone que el evento iniciador del encolamiento ocurre con un tiempo entre arribos mínimo de T_m, denominado periodo del mensaje. El modelo soporta eventos que son estrictamente periódicos, con periodo T_m. Eventos que ocurren esporádicamente con una separación mínima de T_m. Eventos que ocurren solamente una vez antes que el sistema sea reinicializado, en cuyo caso T_m es infinito.
- Cada mensaje posee un plazo estricto D_m, correspondiente al tiempo máximo permitido que transcurre desde el evento iniciador del mensaje hasta que el mismo se encuentra disponible en el nodo receptor. Las tareas en el nodo receptor podrían agregar otros requerimientos temporales pero en esos casos se asume que D_m es la restricción de tiempo más ajustada.
- El peor tiempo de respuesta (R_m) de un mensaje se define como el mayor tiempo que puede insumir la transmisión de un mensaje, desde que ocurre un evento de transmisión hasta que es recibido por el nodo receptor.

Se dice que un mensaje se puede planificar si y solo si su peor tiempo de respuesta es menor o igual a su plazo estricto ($R_m \leq D_m$). Por lo tanto, el sistema se puede planificar si y solo si todos los mensajes del sistema pueden planificarse.

Aunque el controlador CAN accede efectivamente a un mensaje luego de leer el último bit del campo EOF (Figura 3.4) de la trama, en este modelo se incluye al campo de intermisión (3 bits más) y así se considera el caso más pesimista para el peor tiempo de respuesta. También se asume un reloj global suponiendo que los métodos Hard Synchronizatión y Resynchronization [60] son suficientes para sincronizar las señales de reloj de cada nodo.

De acuerdo al modelo presentado y desde el punto de vista de planificación de tráfico, CAN implementa una planificación de prioridad fija no *apropiativa*. Además se considera al bus libre de errores. El peor tiempo de respuesta (R_m) posee los siguientes componentes:

$$R_m = J_m + w_m + C_m \tag{5.1}$$

En donde se contempla el máximo número de bits debido al *stuffing* que implementa el protocolo CAN, para calcular el peor tiempo de transmisión (C_m). Entonces, la fórmula para calcular C_m será:

$$C_m = \left(g + 8s_m + 13 + \left\lfloor \frac{g+8s_m-1}{4} \right\rfloor\right)\tau_{bit} \tag{5.2}$$

En (5.2) s_m es el número de bytes en el campo de datos del mensaje, g es el número de bits adicionales por el *stuffing* que corresponde a 34 para el formato estándar y a 54 para el formato extendido recordando que los últimos tres campos de la trama y el campo de intermisión no son afectados por el *stuffing*; y finalmente τ_{bit} es el tiempo de transmisión para un solo bit.

El mayor tiempo de espera de un mensaje en el controlador CAN, antes de comenzar una transmisión exitosa, se denomina retardo de encolamiento (w_m). Este tiempo está compuesto por dos elementos:

$$w_m = B_m + I_m \tag{5.3}$$

En (5.3) B_m, denominado tiempo de bloqueo, es el mayor tiempo de espera de un mensaje a causa de la transmisión de otros mensajes de menor prioridad que estaban en el proceso de transmisión cuando el mensaje de mayor prioridad estaba siendo encolado. El máximo tiempo de bloqueo está dado por:

$$B_m = \max_{k \in lp(m)}(C_k) \tag{5.4}$$

donde $lp(m)$ es el conjunto de mensajes con menor prioridad que m.

El elemento denominado interferencia (I_m) es el mayor tiempo de espera a causa de la transmisión de otros mensajes de mayor prioridad que ganarán en el arbitraje y serán transmitidos antes que el mensaje en cuestión.

Finalmente w_m se puede expresar de la siguiente manera:

$$w_m = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{5.5}$$

donde $hp(m)$ es el conjunto de mensajes con mayor prioridad que m.

5.2.1.2 Implicancias del *Jitter* en los tiempos de transmisión

Generalmente los modelos de planificación de prioridad fija como es el caso del protocolo CAN, presentan *jitters* a pesar de poseer sólo tareas periódicas [8]. Existen varios trabajos que realizan un análisis de *planificabilidad* del protocolo [61] [62] [63] e introducen al *jitter* como un parámetro necesario para efectuar la planificación de mensajes que se envían periódicamente.

La presencia de un *jitter* aleatorio provoca un retardo del tiempo de transmisión y afecta directamente al peor tiempo de respuesta del mensaje como ya se ha mostrado por medio de la ecuación (5.1). En [59] se realiza un análisis estocástico de los tiempos de respuesta y se propone una función de distribución uniforme para el *jitter*. Pero si se considera que el *jitter* de un componente está formado por la contribución de distintos subcomponentes, que poseen a su vez otros *jitters*, puede apelarse al teorema del límite central. El teorema del límite central establece que la función distribución de probabilidad de la suma de un número suficientemente grande de variables aleatorias, con distintas distribuciones, puede aproximarse a una normal o gaussiana [64]. Esta es una de las razones, junto con la descripta en la siguiente sección, para proponer como función de distribución de probabilidad del *jitter* a la normal o gaussiana.

5.2.1.3 Implicancias de la temperatura en el jitter.

Como se ha explicado en la sección 3.1.3 la tasa de tolerancia a fallas de los componentes electrónicos que integran un sistema, puede calcularse en base a la ecuación 3.1 y en este caso para el controlador CAN será de 1.0 x 10^{-6} fallas/hora [65]. Especialmente uno de los parámetros que tiene mayor peso en la ecuación 3.1 es el factor de temperatura (π_T).

Al utilizarse componentes electrónicos de la familia lógica Complementary Metal-Oxide-Semiconductor (CMOS), éstos se verán afectados directamente por la temperatura del entorno [66]. Existe un ruido temporal aleatorio en los circuitos electrónicos y en la práctica estos

dispositivos generan una o más fuentes internas de ruido. Sin embargo, la mayor parte de éste es causada por el ruido térmico originada por la agitación térmica de los electrones libres [67]. Este tipo de ruido sigue una función de distribución normal o gaussiana.

5.2.1.4 Implicancias de la temperatura en el bus

Al principio de la sección se ha mencionado que el bus aporta un retardo fijo pero realizando un análisis electromagnético simple se puede modelar a éste como una línea de transmisión en donde el retardo no será constante. Físicamente el medio de transmisión es un cable tipo STP con una aislación que introduce capacidades e inductancias parásitas distribuidas en la línea de transmisión. En la Figura 5.9 se muestra un esquema del modelo.

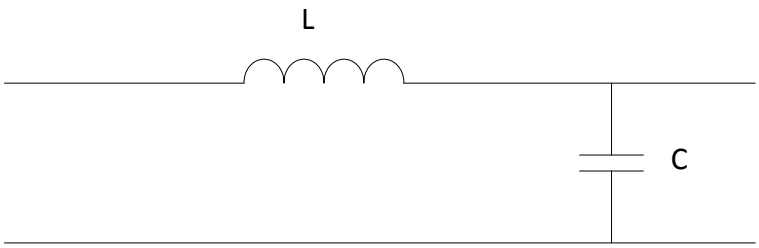

Figura 5.9: Modelo de línea de transmisión del bus

De acuerdo a las ecuaciones de Maxwell, la velocidad de propagación de la señal en la línea está dada por [68]:

$$v = \frac{1}{\sqrt{lc}} = \frac{1}{\sqrt{\mu\varepsilon}} \tag{5.6}$$

En donde, l y c representan la inductancia y capacidad distribuidas en la línea de transmisión. Además:

$$l = \frac{\pi\varepsilon}{\ln(d/a)} \qquad c = \mu\frac{\ln(d/a)}{\pi}$$

Donde a es el radio de los conductores y d la separación entre ellos, ε y μ representan la permitividad y permeabilidad del dieléctrico respectivamente.

Ahora el retardo de la línea de transmisión puede describirse como:

$$t_D = \frac{L}{v} \tag{5.7}$$

Ya que L es la longitud de la línea de transmisión y debido a que en las pruebas se utilizó una medida de 1 metro, reemplazando en la ecuación (5.7) con la ecuación (5.6) queda:

$$t_D = \sqrt{\mu\varepsilon} \tag{5.8}$$

Considerando que $\mu = \mu_0 \mu_r$ y $\varepsilon = \varepsilon_0 \varepsilon_r$, teniendo en cuenta que μ_0 y μ_r representan las permeabilidades en el vacío y relativa respectivamente, y que lo mismo ocurre con ε_0 y ε_r pero para la permitividad. El retardo de línea de transmisión está dado por:

$$t_D = \sqrt{\mu_0 \mu_r \varepsilon_0 \varepsilon_r} = \sqrt{\mu_0 \varepsilon_0}\ \sqrt{\mu_r \varepsilon_r} = 3{,}33\ \sqrt{\mu_r \varepsilon_r}\ nseg \tag{5.9}$$

En las áreas automotrices, de aviónica y espacial generalmente se utilizan materiales poliméricos como el Politetrafluoroetileno (PTFE, comercialmente Teflón) o el Polyvinylidene fluoride (PVDF) en los cables que interconectan a los distintos dispositivos. Esto se debe a que presentan características que se adecuan a las necesidades de seguridad de los sistemas de tiempo real, como es el caso del comportamiento de estos materiales frente a variaciones de temperatura y frecuencia [69] [70].

5.3 Modelo de simulación

El desarrollo del diseño del modelo de simulación se realizó considerando el concepto de refinamiento explicado en el capítulo 2. De esta forma se comenzó efectuando un modelo de simulación con un nivel de abstracción sin tiempo hasta llegar a un modelo con un nivel de refinamiento a nivel de ciclos. En el transcurso de conseguir el modelo de simulación final se implementaron distintos modelos de experimentación, los cuales permitieron comprobar la verificación del modelo. Éstos se describen y muestran en el anexo A.

5.3.1 Modelo final

En el sistema básicamente existen dos módulos principales: el Nodo y el Bus. Como sucede físicamente, el Nodo modelado está compuesto por un microcontrolador y un controlador CAN. En este modelo de simulación no se ha tenido en cuenta al transceptor CAN ya que como se ha expuesto en la sección anterior este componente agrega un retardo fijo del orden de los nanosegundos, el cual es un valor pequeño en comparación al tiempo necesario para la transmisión de un bit (1 µseg). En la Figura 5.10 puede verse un gráfico del modelo final utilizando los componentes de SystemC mencionados en la sección 4.3.

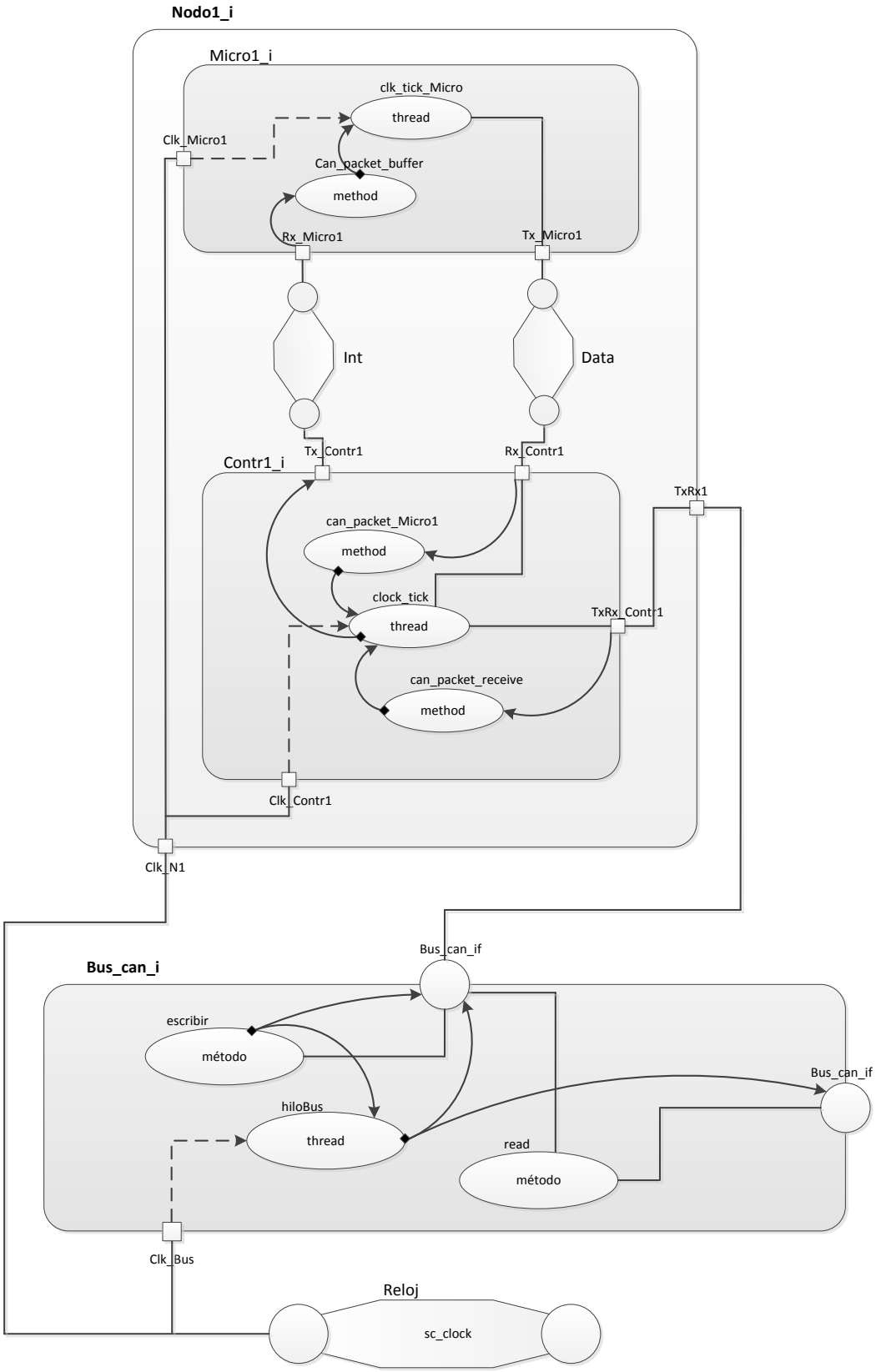

Figura 5.10: Modelo de simulación final

5.3.1.1 Composición

El sistema está representado por una jerarquía de módulos de SystemC, en donde, el módulo más general que contiene a todos los demás es denominado CAN y se encarga de inicializar al sistema simulado. Está compuesto por un Nodo CAN, un Bus CAN y un reloj global cuya función es la de marcar un único tiempo de referencia para el sistema completo.

A su vez, el Bus está integrado por un microcontrolador, un controlador CAN y canales tipo *sc_signal* para realizar la interconexión entre los dos módulos.

5.3.1.1.1 Microcontrolador

Este componente no sólo cumple con las funciones del *host* del controlador sino también con las del sensor IFOG y la Interfaz Sensor, ya que genera los datos acondicionados y listos para ser transmitidos a través del bus. La cantidad de datos a enviar y la tasa de transmisión son recibidas como parámetros antes de ejecutar cada simulación. En la Figura 5.11 se observa un diagrama de clases del módulo microcontrolador.

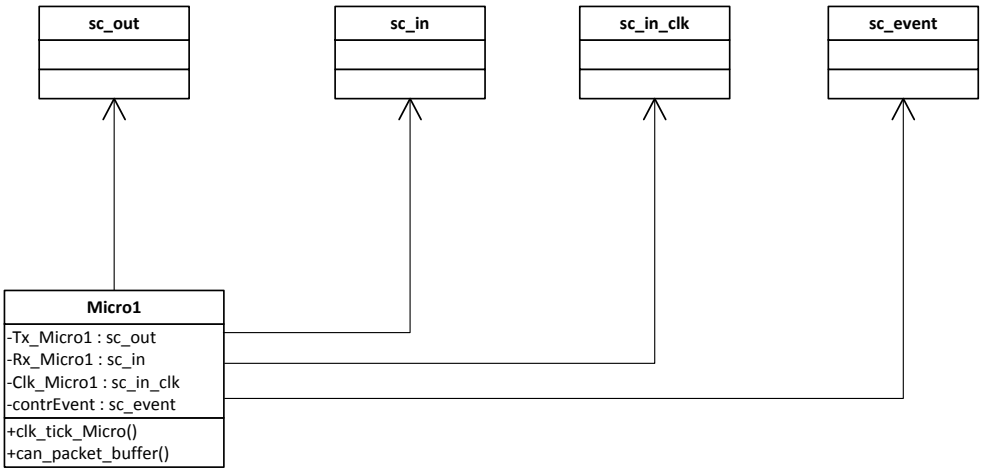

Figura 5.11: Diagrama de clases del microcontrolador

Está compuesto por un proceso SC_THREAD (clk_tick_Micro()) generador de una cierta cantidad de datos aleatorios a una tasa de transmisión fija, otro proceso SC_METHOD (can_packet_buffer ()) que marca al proceso anterior la disponibilidad de envío de mensajes en el buffer del controlador y un conjunto de interfaces, representadas en SystemC mediante puertos sc_in y sc_out para comunicarse con el controlador. Además, posee un puerto sc_in_clk para la conexión con el reloj global.

5.3.1.1.2 Controlador

El componente Controlador junto con el Bus, es uno de los principales módulos del sistema de simulación. En él se completan los campos de la trama con datos aleatorios pero respetando las especificaciones del estándar, se introduce el mensaje a transmitir y se monitoriza el bus para realizar el arbitraje correctamente. Finalmente también se agrega el *jitter* correspondiente al buffer de trasmisión, que es recibido por parámetro junto con la temperatura de operación. En la Figura 5.12 se observa un diagrama de clases del módulo controlador.

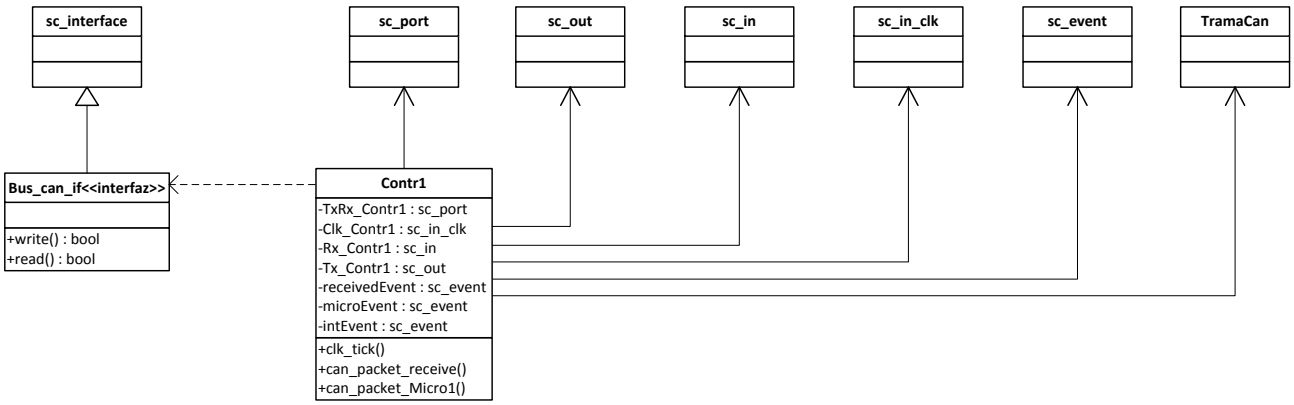

Figura 5.12: Diagrama de clases del Controlador CAN

El proceso SC_THREAD (clk_tick()) es el encargado de realizar la construcción de la trama CAN, introducir el *jitter* del buffer de transmisión, notificar al microcontrolador la recepción del dato y efectuar la monitorización del bus para ejecutar el arbitraje en el bus. Es decir, cumple con las tareas de los bloques IML, Message Buffer y en parte del BSP del SJA1000. Los procesos SC_METHOD (can_packet_Micro1() y can_packet_receive()) comunican al controlador el requerimiento de transmisión de un mensaje por parte del microcontrolador y la trama de datos que está ocupando el bus, desempeñando las funciones de los buses de direcciones y control junto con las del BSP. Esto último se notifica por medio de eventos de SystemC. Nuevamente se utilizan interfaces tipo sc_port, sc_in y sc_out pero en este caso para conseguir la conexión con el microcontrolador y el Bus, o sea las líneas de control y direcciones/datos. Además, se recurre a un tipo TramaCan construido especialmente para respetar los campos del protocolo CAN.

5.3.1.1.3 Bus

El Bus CAN es modelado por medio de un canal jerárquico de SystemC y permite la conexión de un número ilimitado de nodos a través de la interfaz Bus_can_if. En este componente se implementa el proceso de arbitraje, que se realiza junto con el controlador, se fija el tiempo de transmisión del canal (C_m) y se agrega el retardo debido a los elementos parásitos de la línea de transmisión que dependerá de la temperatura de operación del sistema. La sincronización de los procesos que se ejecutan concurrentemente se realiza utilizando canales provistos por SystemC para acceder a recursos compartidos. Además, aquí se genera el seguimiento o traza de los mensajes para realizar un posterior análisis de los tiempos de transmisión. En la Figura 5.13 se observa un diagrama de clases del módulo Bus.

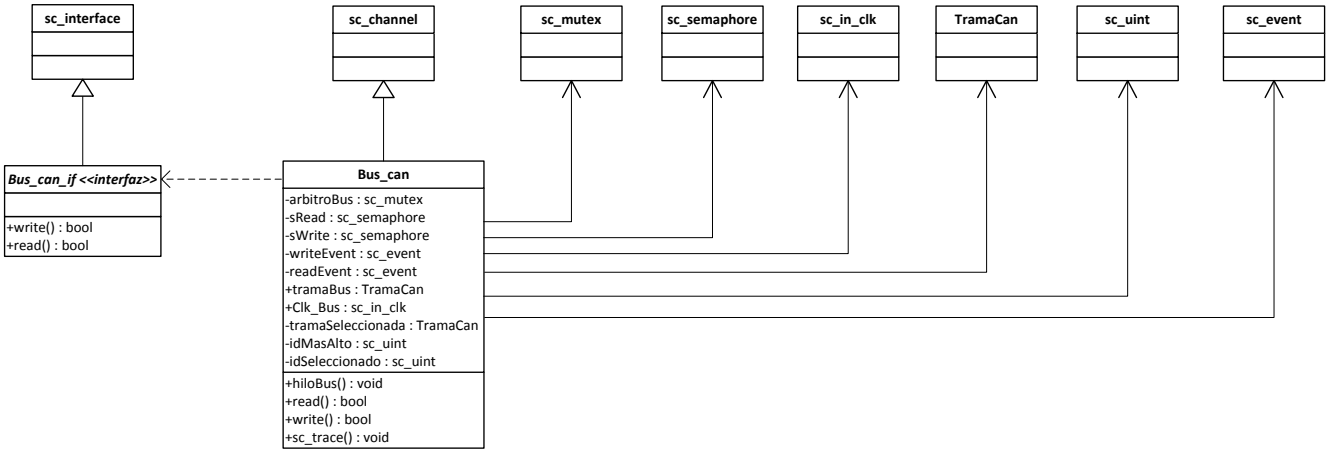

Figura 5.13: Diagrama de clases del Bus CAN

El proceso SC_THREAD (hiloBus) efectúa el arbitraje utilizando los identificadores de origen y prioridad de los mensajes. A su vez, por medio de los métodos de escritura y lectura se definen las funciones para acceder al canal. El proceso hiloBus interactúa sincrónicamente con los dos métodos de acceso para introducir los tiempos que impone la norma y establecer el mensaje ganador del arbitraje teniendo en cuenta que el protocolo no es *apropiativo*.

5.3.1.2 Semántica

De la Figura 5.10 puede observarse que el microcontrolador envía mensajes por medio del canal Data a una tasa periódica y luego espera hasta recibir la confirmación del controlador CAN, a través del canal Int. Una vez en el controlador, el mensaje es encolado para ser transmitido e insertado el *jitter* (J_m) correspondiente, si el Bus lo permite el mensaje se introduce en el canal para participar del arbitraje si otro mensaje quiere acceder al Bus.

En el Bus se implementa el concepto de ventanas de escritura y lectura que se corresponden con las etapas de escritura y lectura, definidas por el proceso del bus. Las ventanas fueron diseñadas para que tengan un intervalo de tiempo fijo y representan las restricciones que poseen los mensajes cuando llegan en un momento o periodo en el que no pueden transmitir a pesar de tener una alta prioridad, este periodo es también conocido como *busy period*. Es decir, cuando un mensaje llega durante el intervalo de tiempo ventana de lectura debe esperar hasta el inicio del intervalo ventana de escritura para competir con otros posibles mensajes en el momento de arbitraje. La suma de los tiempos de las ventanas de escritura y lectura es igual al tiempo de transmisión del canal.

En la Figura 5.14 se observa un diagrama de actividades que muestra la interacción entre los distintos componentes del modelo de simulación.

Hasta aquí se han descripto los criterios y justificaciones para realizar la construcción del modelo de simulación (punto 3 de la sección 2.5.3) y parte de la verificación por medio de las pruebas realizadas durante su diseño (Anexo B). Pero aún falta definir los valores seleccionados para el

tiempo de transmisión y los *jitters* del controlador y el bus, éstos se proponen y explican en el próximo capítulo.

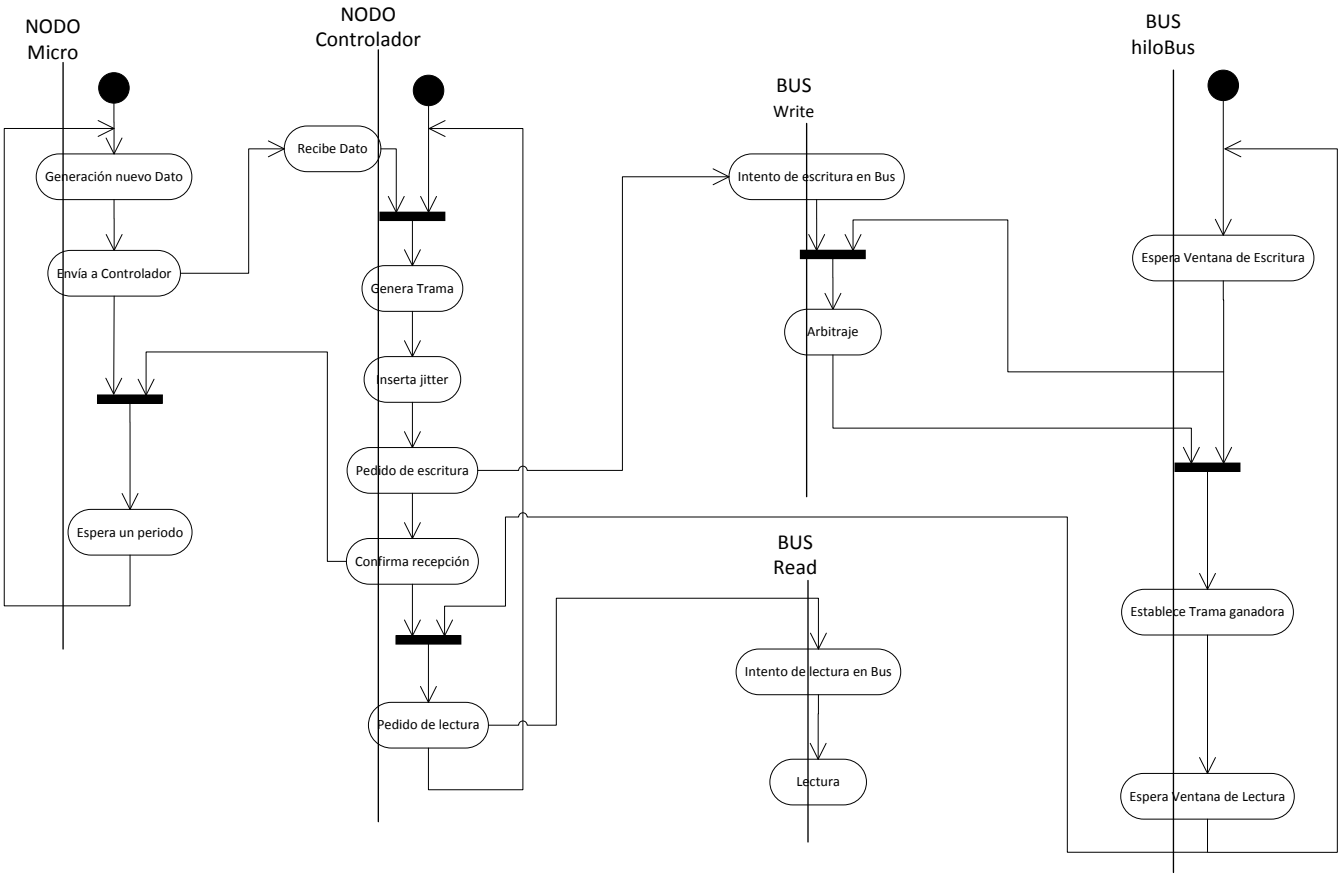

Figura 5.14: Diagrama de actividades del modelo de simulación final

6 Capítulo 6: Experimentación y análisis de resultados

En este capítulo se describen los experimentos realizados con el prototipo hardware y el modelo de simulación. Por este motivo, se definen los valores utilizados para configurar a los dos modelos junto con las justificaciones de elección de los mismos.

También se realiza un análisis de los datos conseguidos con el prototipo y el modelo de simulación para formalizar la validación de este último.

6.1 Experimentación con el prototipo

El prototipo de prueba se configuró para cumplir con el tiempo de actualización de la computadora de a bordo del sistema de vuelo. En este caso la información de los tres ejes inerciales debía ser actualizada cada 20 mseg, por lo tanto la periodicidad de envío elegida de cada sensor fue de 4 mseg. La programación de los microcontroladores se realizó siguiendo los pasos señalados en la sección 5.1.3 y utilizando las herramientas descriptas en la sección 4.5.1.

Como consecuencia de la arquitectura del microcontrolador (tamaño de los registros), el proceso de transmisión del controlador CAN y la frecuencia de los cristales de cada uno de los nodos, la tasa de transmisión programada fue de 3,96 mseg.

En general, el tráfico en redes de datos tradicionales se analiza a partir del throughput, latencia y jitter [71] de la red. Aunque en una red para sistemas de tiempo real estricto estos parámetros son necesarios para cuantificar la calidad de servicio (QoS), por las características específicas de las redes en el protocolo CAN medir el throughput es poco significativo. De hecho el throughput es fijo y de 1 Mbps, ya que es la máxima velocidad de transferencia que admite el protocolo. Teniendo en cuenta que el proceso de medición inercial implica que los algoritmos que procesan la información de los sensores cumplan con los requerimientos temporales y que los nodos deben transmitir en forma periódica los datos sensados a la Computadora de Navegación para que esta efectúe las correcciones de guiado, se mide el tiempo de llegada de cada mensaje a esta unidad central de procesamiento. Es por esto que el parámetro que se evalúa es el Inter Arrivals Time (IAT) de mensajes.

En varias secciones de capítulos anteriores (3.1.3, 4.4, 5.2.1.2, 5.2.1.3 y 5.2.1.4) se ha mencionado que la temperatura es uno de los factores que más influye en la generación de fallas e incremento de tiempos de respuestas o retardos en el sistema de comunicaciones desarrollado para este trabajo. Además, la utilización de componentes COTS en la construcción del prototipo hace que sea de particular importancia conocer el desempeño del sistema bajo condiciones de estrés térmico para determinar si el mismo puede constituirse como una alternativa viable.

Todo el sistema de comunicaciones está diseñado para ser instalado en un ambiente controlado térmicamente y apantallado al chasis del satélite asegurándose una temperatura de trabajo de 40 °C con una tolerancia de ± 5 °C. Debido a las condiciones ambientales a las que se puede encontrar expuesto un satélite, se sabe que el rango de temperatura exterior oscila entre -25 °C y 85 °C [25]. No obstante, la técnica de apantallamiento utilizada permite acotar este rango desde 30 °C hasta

65 °C. Cabe destacar que una falla en el control térmico, que provocaría temperaturas fuera de rango, tendría serias consecuencias en todo el sistema (particularmente con temperaturas elevadas).

Por las razones mencionadas el rango de temperaturas, para realizar las pruebas, fue de 25 °C a 70 °C ya que es de principal interés el comportamiento de la red a temperaturas altas. Estas condiciones se recrearon utilizando un horno eléctrico de laboratorio [72] que cuenta con un módulo de control por realimentación Proportional-Integral-Derivative (PID). Este módulo permite asegurar una temperatura de trabajo con una resolución de 1 °C. El instrumento fue diseñado para realizar distintas pruebas de componentes electrónicos, por lo tanto cuenta con un orificio que se utiliza para introducir en la mufla (cámara interna del horno) el medio de transmisión de red y de potencia de los dispositivos. En la Figura 6.1 se puede ver el horno descripto y en la Figura 6.2 el nodo en la mufla del horno.

Figura 6.1: Fotografía del horno y el nodo

Figura 6.2: Fotografía del nodo en la mufla

En las pruebas se configuró un nodo para que envíe mensajes en forma periódica a la tasa de transmisión antes señalada, con un paquete sobredimensionado de datos de 6 bytes (la información de cada girómetro no supera los 2 bytes) y de esta manera representar el sensado de la velocidad angular en uno de los ejes del IFOG. Un nodo receptor, en este caso el que se encuentra en la interfaz USB-CAN, realiza la monitorización del tráfico a través de la herramienta CANBuilder.

En este contexto los tiempos de prueba están sujetos a la disponibilidad de las herramientas para llevar a cabo las mismas. Generalmente se dispone de unos pocos instrumentos para realizar los tests de varios dispositivos. Además, se debe considerar que el tiempo de preparación de la prueba insume un tiempo relevante. Teniendo en cuenta que las pruebas sólo serían funcionales, se optó por seleccionar un tiempo sujeto a la cantidad de mensajes a enviar en modo free running. La cantidad elegida fue de 37339 mensajes, aproximadamente 150 seg (tasa de 3,96 mseg). En la Figura 6.3 se puede observar la metodología utilizada para realizar las pruebas y las herramientas usadas en cada uno de los pasos.

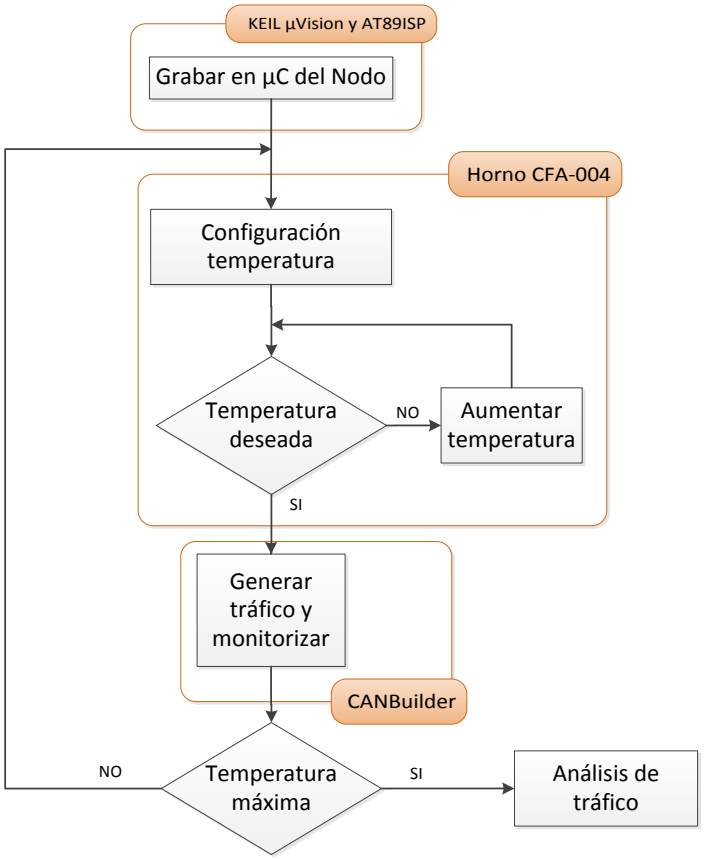

Figura 6.3: Metodología para la experimentación del hardware

6.1.1 *Métricas obtenidas*

Las pruebas consistieron en enviar 37339 mensajes con ocho temperaturas de operación diferentes por la red de sensores. La Tabla 6-1 muestra los resultados obtenidos.

Tabla 6-1: Resultados obtenidos con el prototipo

IAT [mseg]	Temperaturas [ºC]							
	25	30	35	40	45	55	65	70
3,96	108	93	94	86	85	71	67	56
3,97	411	414	397	394	377	345	297	188
4,03	29448	29472	29480	29496	29506	29548	29579	29657
4,04	7368	7357	7365	7360	7367	7373	7384	7416
4,09	1		1		1	1	1	3
4,10	2	2	1	2	2		10	18
Totales	**37338**	**37338**	**37338**	**37338**	**37338**	**37338**	**37338**	**37338**

Como puede observarse en la tabla, en promedio, el mayor porcentaje de mensajes se recibe a un IAT promedio de 4,03 mseg y el resto de los mensajes no se desvían más de 70 µseg de ese valor. Al aumentar la temperatura de operación los mensajes con IAT menores al promedio (3,96 mseg y 3,97 mseg) disminuyen prácticamente a la mitad. Cabe destacar el aumento de mensajes con IAT mayores (4,09 mseg y 4,1 mseg) a partir de 65 °C de temperatura de operación. Debe notarse que aún con este aumento bajo temperaturas elevadas (la temperatura promedio es de 40 °C) el IAT cumple con las restricciones impuestas por la tasa de refresco. En la Figura 6.4 puede verse los mensajes recibidos a diferentes temperaturas e IATs obtenidos.

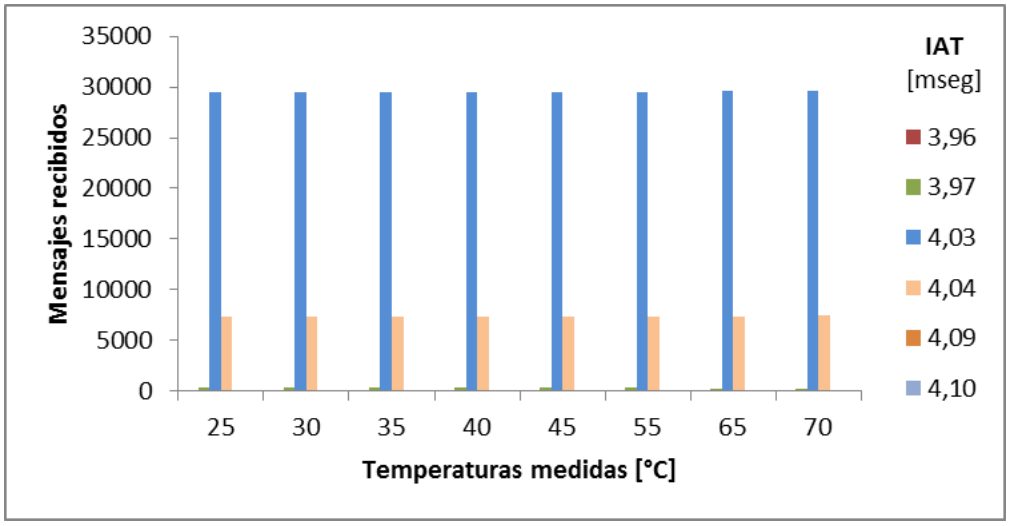

Figura 6.4: IATs a diferentes temperaturas en el prototipo

6.2 Experimentación con el modelo de simulación

Como se explicó en la sección 5.2.1.1, para calcular el peor tiempo de respuesta en una transmisión es necesario encontrar C_m, J_m y w_m. Dado que no existen mensajes con mayor o menor prioridad a los que se transmiten por la red en esta aplicación, se puede suponer a w_m igual a cero.

El tiempo máximo de transmisión utilizado en el modelo de simulación se calculó por medio de la ecuación (5.2). Considerando que se envían 6 bytes por mensaje, se selecciona el formato

extendido y la velocidad de transferencia es la máxima (tiempo de transmisión de un solo bit es igual a 1 µseg); entonces C_m es igual 140 µseg.

En cuanto al *jitter*, ya se ha indicado en la sección 5.2.1.2 que se propone representar al mismo con una función de distribución normal. Se hace uso de la librería tr1/random (C++ Standards Committee Technical Report 1) del estándar de C++ [73] que define funciones y clases para la generación de números aleatorios distribuidos con diferentes funciones de densidad de probabilidad, entre ellas la normal o gaussiana. En este caso, se plantea un J_m que como máximo es igual a C_m y una media (µ) igual a la mitad de C_m con una desviación estándar (σ) que sea un cuarto de µ.

Al utilizar componentes CMOS con tecnología de 180 nm, se puede representar la variación de J_m en función de la temperatura con la siguiente ecuación [66]:

$$y = 0{,}349\, x + 10.875 \tag{6.1}$$

donde y representa el porcentaje de tiempo de retardo respecto a la temperatura de 25 °C que ya posee un retardo del 19,6% y el parámetro x representa la temperatura. El *jitter* es implementado en la clase Controlador del modelo de simulación.

A pesar de que en la sección 5.2.1.4 se ha explicado y encontrado la ecuación (5.9) para representar el retardo de línea de transmisión (t_D), al utilizarse en esta aplicación Teflón como material aislante para el medio de transmisión la incidencia de t_D no es notoria (sólo del orden de los 10 nseg en el peor de los casos) en función del rango de temperatura de operación. De todas maneras la ecuación (5.9) podría ser implementada en la clase Bus del modelo de simulación, ya que en el área aeroespacial se utilizan polímeros como el PVDF cuyo t_D sí es evidente con el aumento de la temperatura.

El escenario que se modeló fue realizado en base a la experimentación con el prototipo. Es decir, un nodo (Nodo1_i) que transmite a una tasa periódica de 3,96 mseg un mensaje con 6 bytes de datos a un nodo receptor (Nodo2_i). Los objetos son instanciados de las clases explicadas en la sección 5.3. En la Figura 6.5 puede verse un gráfico simplificado del escenario planteado.

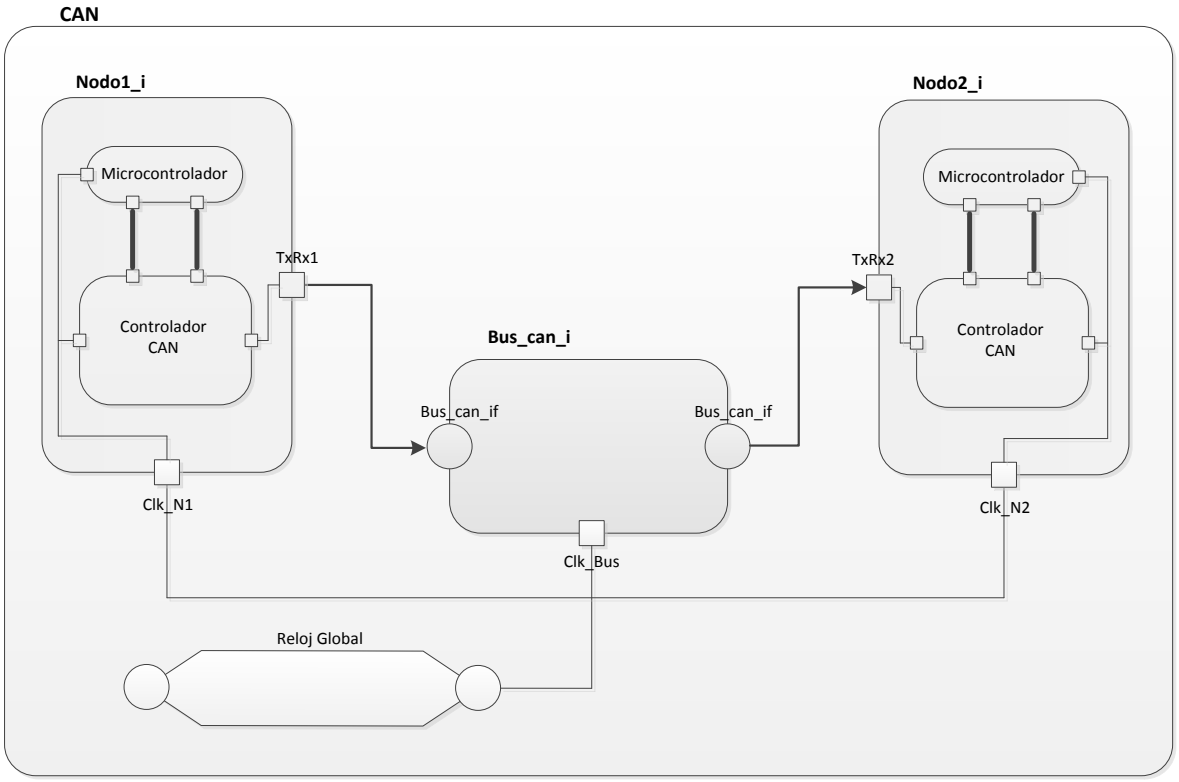

Figura 6.5: Modelo de simulación

Para simplificar el dibujo se ha representado los canales tipo *sc_signal* con líneas gruesas y no se han incorporado las figuras de los métodos, procesos y eventos mostrados en el capítulo anterior para explicar el modelo de simulación final.

Debido a la naturaleza probabilística del sistema físico fue imprescindible crear un modelo de simulación cuyos resultados sean estadísticamente iguales a los del sistema real. Uno de los factores que afectan en forma directa esos resultados es el tamaño de la corrida de simulación o bien el número de corridas de simulación realizadas para encontrar resultados confiables. Una forma de lograr la estabilización del modelo de simulación consiste en seleccionar condiciones iniciales de arranque que sean más representativas de la condición de estado estable y por lo tanto reduzcan el periodo transitorio [74]. Es por esto que se recurre a la configuración utilizada con el prototipo para realizar las pruebas con el modelo de simulación.

Cada simulación recibe los datos de entrada desde consola mediante argumentos por línea de comando para configurar el entorno dentro del cual se realiza la ejecución de la simulación, en este caso temperatura y cantidad de mensajes. Luego, la ejecución de la simulación genera dos archivos diferentes: uno del tipo VCD para visualizar las señales producidas en el bus y un archivo de bitácora (log) que detalla los distintos hitos temporales en el nodo receptor, con este último se efectúa el análisis de tráfico de la red.

6.2.1 *Métricas obtenidas*

Nuevamente las pruebas consistieron en enviar 37339 mensajes con ocho temperaturas de operación diferentes por la red de sensores simulada. La Tabla 6-2 muestra los resultados obtenidos.

Tabla 6-2: Resultados obtenidos con el modelo de simulación

IAT [mseg]	Temperaturas [°C]							
	25	**30**	**35**	**40**	**45**	**55**	**65**	**70**
3,96	3	3	3	3	3	3	2	2
3,97	28	25	25	25	23	18	16	16
3,98	134	128	116	106	100	91	88	82
3,99	683	638	597	542	496	417	341	326
4	1969	1778	1653	1521	1401	1174	1020	953
4,01	4460	4175	3843	3574	3309	2889	2474	2281
4,02	7178	6787	6477	6132	5857	5185	4628	4365
4,03	8373	8236	8021	7801	7492	7069	6547	6285
4,04	7169	7331	7467	7569	7654	7555	7372	7261
4,05	4469	4819	5158	5452	5676	6158	6474	6615
4,06	2020	2353	2618	2909	3305	3919	4529	4727
4,07	669	816	999	1227	1404	1913	2389	2632
4,08	152	202	298	386	486	703	1012	1210
4,09	24	39	52	75	109	190	348	449
4,1	6	7	9	12	16	44	79	105
4,11	1	1	2	3	6	8	13	22
4,12				1	1	2	5	6
4,13							1	1
Totales	**37338**	**37338**	**37338**	**37338**	**37338**	**37338**	**37338**	**37338**

Se observa que la Tabla 6.2 posee mayor número de IATs que la Tabla 6.1, esto se debe a que la simulación permite conseguir resoluciones mucho mayores que el modelo físico. Al existir más IATs y habiendo elegido una distribución normalizada para el *jitter*, los mensajes se distribuyen de esa manera entre los tiempos de arribo. Aun así puede observarse un comportamiento similar en los dos modelos.

La principal diferencia se advierte en el valor del IAT promedio, varía con la temperatura de 4,03 mseg a 4,04 mseg. Además, el resto de los IATs se alejan de la media en 70/80 µseg con una temperatura de 25 °C finalizando con una desviación de 80/90 µseg a la mayor temperatura.

De todas formas, al aumentar la temperatura los mensajes con IATs más bajos (3,96 mseg y 3,97 mseg) disminuyen prácticamente a la mitad como en el modelo físico. Mientras que el aumento de

los mensajes con IATs mayores a 4,09 mseg, es más notorio que en el sistema físico. De hecho aparecen valores que antes no existían (4,11 mseg, 4,12 mseg y 4,13 mseg), pero esto lejos de invalidar el modelo de simulación lo hace, en el peor de los casos, más pesimista y con un error que no llega al 1% en los IATs más altos en comparación con el sistema real. En la Figura 6.6 puede verse los mensajes recibidos a diferentes temperaturas e IATs obtenidos.

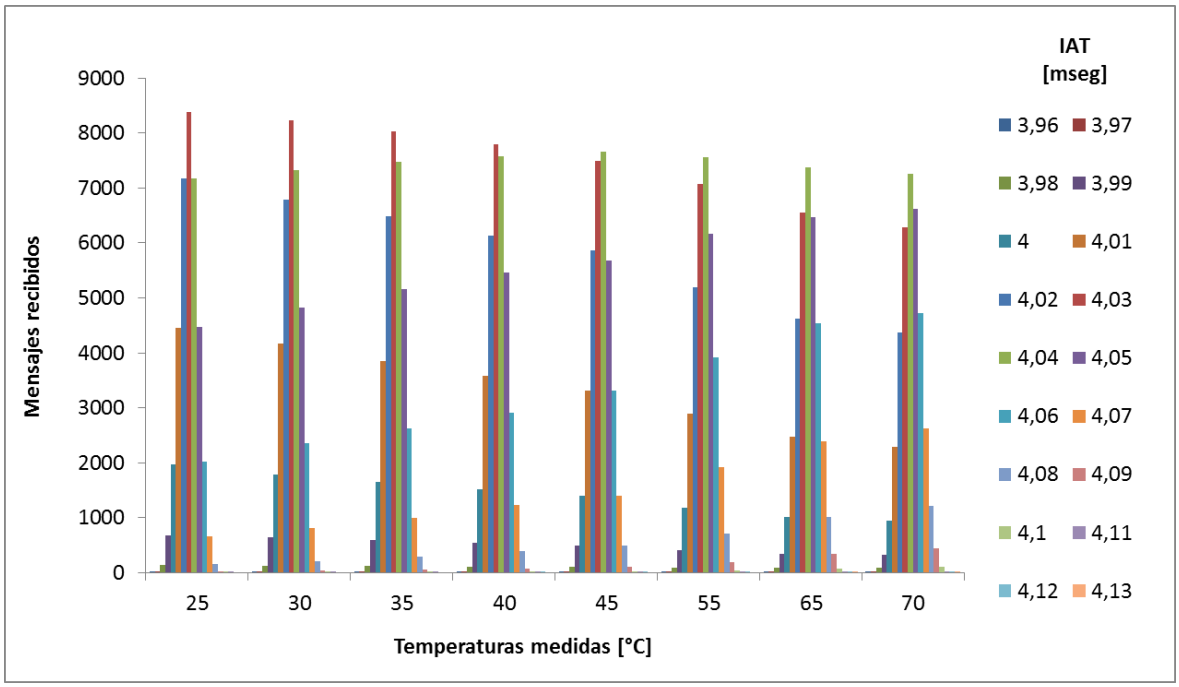

Figura 6.6: IATs a diferentes temperaturas en modelo de simulación

6.3 Validación del modelo de simulación

La prueba para validar un modelo de simulación es establecer que los resultados obtenidos con éste se asemejan a los que podrían encontrarse con un sistema similar al propuesto con la simulación. Si los dos conjuntos de datos encontrados con el modelo y el sistema son similares, el modelo de simulación es considerado válido [16]. Esto se comprobó en las secciones anteriores, permitiendo finalizar con los últimos pasos en la construcción del modelo, mencionados en la sección 2.5.3. Es decir, diseñar y ejecutar experimentos de simulación para estimar la performance del sistema ante posibles escenarios.

Un procedimiento estadístico simple para comparar observaciones del sistema del mundo real con datos de salida de un modelo de simulación es contrastar los valores medios (μ) o esperanzas (E) de los dos conjuntos de muestras [16]. En la Figura 6.7 se muestran los valores medios de los IATs obtenidos con los experimentos llevados a cabo en las secciones 6.1 y 6.2.

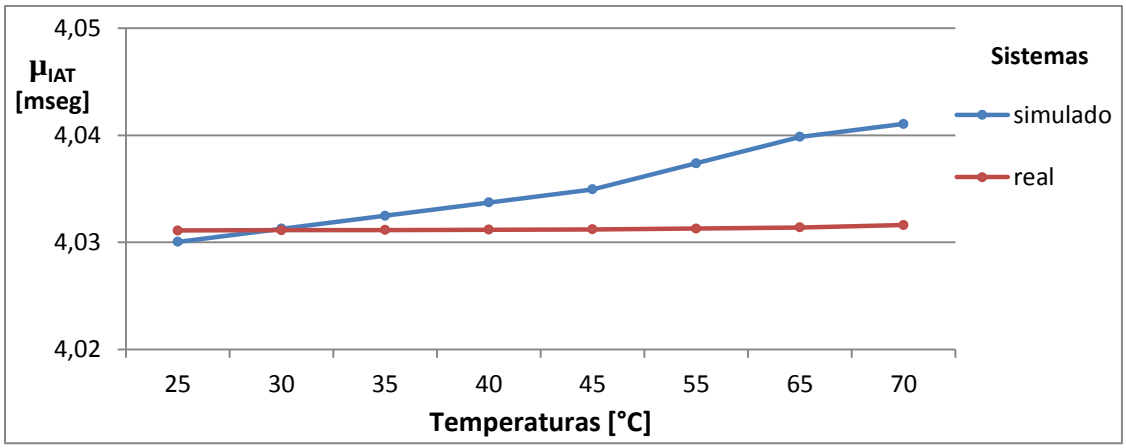

Figura 6.7: Valores medios de IATs del prototipo y la simulación

Recordando que la temperatura de operación del sistema en el satélite es de 40 °C, puede verse que la diferencia de las medias es del 0,05% aproximadamente para este entorno. Aunque la pendiente de la curva del modelo de simulación es más acentuada que la del sistema real, se observa que para la máxima temperatura (70 °C) la diferencia entre medias es del 0,25% aproximadamente. Como ya se mencionó la simulación posee una resolución mucho mayor que el modelo real, esto permite afirmar que el comportamiento del sistema simulado mostrado en la figura es más que conservador para altas temperaturas.

Si bien sólo se tuvo en cuenta al factor térmico para contemplar el efecto del entorno de servicio sobre los distintos componentes del sistema, no se debe olvidar que en este trabajo se realiza un análisis del comportamiento del bus en condiciones operativas (satélite en órbita). Además, se ha demostrado en la sección 3.1 que entre todos los parámetros causantes de fallas en este tipo de sistema, la temperatura es el preponderante.

Finalmente cabe señalar que tomando en cuenta las especificaciones y restricciones de las secciones 4.4 y 5.2.1 respectivamente, puede decirse que el bus ha sido desarrollado y cumple con las especificaciones.

6.3.1 Ejemplo de experimentación

Suponiendo que existe la posibilidad que el entorno de operación del sistema de comunicaciones llegue a temperaturas extremas, a causa de una avería en el sistema de control de temperatura del equipo, podría simularse el comportamiento de la red en estas condiciones. Se envían la misma cantidad de mensajes que en los experimentos anteriores pero a una temperatura de 120 °C. En la Figura 6.8 se ven los tiempos de arribo para la temperatura supuesta y la máxima comparada con el prototipo hardware.

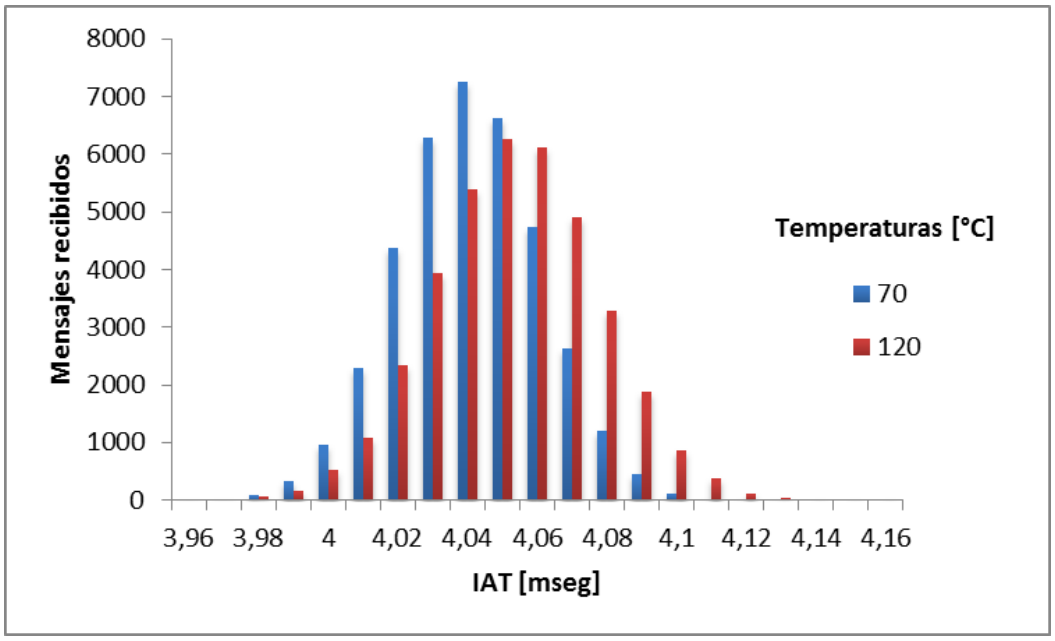

Figura 6.8: Comparación de IATs

Con estos valores se puede predecir un aumento aproximado del 25% de los mensajes con IATs mayores (IATs mayores o iguales a 4,06 mseg a 70 °C y 4,07 mseg a 120 °C) en comparación a los medidos a 70 °C y considerando un error del 1% se puede afirmar que el IAT mayor será de 4,12 mseg. Esta información es de gran importancia para diseñar e implementar una planificación en la computadora de navegación ante este posible escenario.

7 Capítulo 7: Conclusiones

El diseño y verificación de dispositivos aeroespaciales, como se ha reflejada en este trabajo, contempla una gran inversión de tiempo de análisis en distintas áreas de la ciencia y la utilización de instrumentos generalmente costosos. En nuestro país, en la mayoría de los casos existe una baja disponibilidad de herramientas de prueba. Recurrir a técnicas de simulación puede reducir el tiempo de ocupación de ciertas herramientas físicas. Aunque, debe tenerse en cuenta el tiempo de desarrollo, ya que generalmente llevan más tiempo de implementación que la construcción del modelo físico.

La elección de TLM, empleando un lenguaje de descripción de sistemas como SystemC proporcionó una metodología altamente flexible para el desarrollo de sistemas empotrados de tiempo real en dispositivos de vuelo, como es el caso de este trabajo. Disponer de modelos ejecutables tempranamente, junto con un adecuado nivel de abstracción, proveyó una performance del sistema a medida que se desarrollaba el refinamiento.

Si bien el diseño del modelo de simulación fue llevado a cabo teniendo en cuenta conceptos e hipótesis puramente teóricas, la experiencia de construcción y configuración del prototipo hardware facilitó la comprensión del análisis de planificación del sistema de comunicaciones simulado. De hecho el estudio inicial de los distintos protocolos previo a la implementación del prototipo hardware junto con el análisis y clasificación de fallas, permitieron arribar a hipótesis y propuestas de funcionamiento del modelo de simulación final.

7.1 Alcance de la simulación

Observando los resultados del modelo de simulación propuesto, presentados en el capítulo 6, se puede decir que cumple con la especificación del protocolo CAN y de esta forma es validado. También, de la comparación de los resultados del simulador con el prototipo hardware se puede afirmar que con los mismos datos de entrada los dos sistemas producen salidas similares. Es decir, el modelo de simulación está realmente verificado.

Generalmente se realiza un gran número de pruebas estadísticas con los datos de salida de un modelo de simulación pero en este caso se cuenta con un sistema físico que permite realizar una comparación directa. Contar con un sistema hardware similar al modelo de simulación no sólo ofrece la gran ventaja recién mencionada sino que otorga una mayor credibilidad al estudio y diseño del simulador. Aunque no es un objetivo de esta tesis, realizar un análisis estocástico más profundo de los datos de salida sería el próximo paso para proseguir con el presente trabajo.

De esta manera, el principal objetivo de la tesis se concretó a través de un simulador que permite evaluar el comportamiento de una red CAN configurando distintos escenarios. Para conseguirlo, en este trabajo, se propuso configurar la cantidad de mensajes a transmitir y el entorno térmico. Aunque los resultados pueden parecer más pesimistas que los encontrados con el prototipo hardware, ésta es una característica buscada en el área de tiempo real (encontrar el mayor tiempo de respuesta del sistema).

En la Figura 7.1 se muestra una comparación grafica entre el diagrama en bloques del controlador CAN y el modelado del mismo en este trabajo.

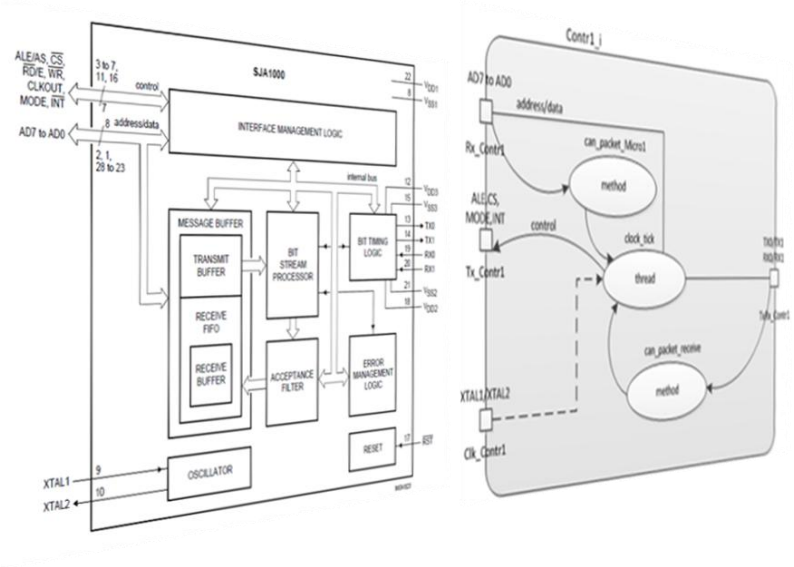

Figura 7.1: Diagrama en bloques y modelado del controlador CAN

7.2 Líneas futuras de investigación

Vale remarcar que el trabajo realizado en esta tesis no está cerrado ya que puede ser ampliado en varios aspectos. Si bien la elección de la función de distribución gaussiana ha demostrado ser acertada para encontrar un máximo de tiempo de respuesta muy similar al real, la distribución observada en los resultados del prototipo no es exactamente normal. Esto se debe aparentemente a que en la distribución real prevalecen distribuciones de un par de componentes del controlador CAN. Una solución posible sería un análisis más profundo de la incidencia del buffer de transmisión/recepción y su posterior inclusión en el modelo de simulación.

Aun habiendo conseguido la funcionalidad necesaria para realizar los análisis requeridos del sistema, en la Figura 7.1 puede verse que no se han modelado varios componentes del controlador CAN. Esto sería fundamental para efectuar una derivación 100% fiel a un SoC. En la actualidad la organización Accellera Systems Initiative cuenta con un grupo que se avoca a desarrollar un producto derivador [75]. También es destacable el esfuerzo de distintas empresas [76] [77] de crear herramientas que permitan generar código RTL a partir de código C/SystemC. Una posible línea de investigación sería realizar la síntesis de alto nivel del modelado en un field programmable gate array (FPGA) y utilizar técnicas de reúso para su refinamiento.

En cuanto a las herramientas gráficas para el diseño del software, en este trabajo se utilizó UML pero se encontró cierta dificultad para representar el comportamiento de los componentes modelados, por ejemplo en la confección de diagramas de actividades e interacción. Un camino para mejorar esto sería utilizar System Modeling Language (SysML) [78] que podría ser la herramienta más adecuada para este tipo de diseños.

Anexo A: Trabajos Publicados

Como resultado del presente trabajo de tesis, durante los años 2011 y 2012 se generaron y publicaron los siguientes artículos en encuentros y congresos nacionales e internacionales:

- "Análisis de tráfico en una red de sensores girométricos del tipo IFOG aplicado a sistemas aeroespaciales". VI Congreso Argentino de Tecnología Espacial (CATE 2011). Ciudad de La Punta, San Luis, Argentina. http://www.aate.org/LISTA_CONGRESO_2011.pdf
- "Integración de Sensores y Sistemas Autónomos Inteligentes en un Sistema Distribuido". IV Encuentro de Becarios de la UNLP (EBec 2011). La Plata, Buenos Aires, Argentina. http://secyt.presi.unlp.edu.ar/Wordpress/wp-content/uploads/2011/08/ebec2011_trabajos1.pdf
- "Caracterización del comportamiento de una red de sensores basado en COTS bajo condiciones de temperatura hostil para sistemas aeroespaciales". XVII Congreso Argentino de Ciencias de la Computación (CACIC 2011). La Plata, Buenos Aires, Argentina. http://sedici.unlp.edu.ar/handle/10915/18792
- "Utilización de un reloj global para el modelado de un ambiente simulado distribuido". XVIII Congreso Argentino de Ciencias de la Computación (CACIC 2012). Bahía Blanca, Buenos Aires, Argentina. http://sedici.unlp.edu.ar/handle/10915/23818
- "Simulación de una red CAN para dimensionar las comunicaciones de una IMU". VII Congreso Argentino de Tecnología Espacial (CATE 2013). Ciudad de Mendoza, Mendoza, Argentina. *Resumen de artículo aprobado*. http://www.aate.org/congreso_mdza.htm

Anexo-B: Evolución del modelo de simulación

El modelo final presentado ha ido pasando por numerosos refinamientos hasta alcanzar su último estado. En este anexo se presentan algunos modelos previos para conseguir ese fin. Inicialmente el primer modelo era un SAM (sección 2.6.1), por medio del cual se modeló al bus CAN con un canal *sc_signal* y a los nodos emisor y receptor con dos procesos del tipo *sc_thread* y *sc_method* respectivamente. En la Figura B-1 se presenta una vista de este modelo.

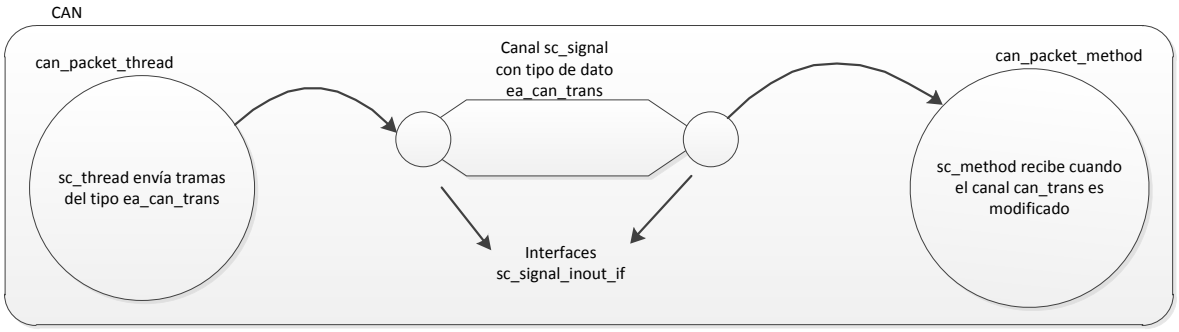

Figura B-1: Primer modelo

Tanto las escrituras como las lecturas en el canal *sc_signal* se realizan a través de la interfaz *sc_signal_inout_if<T>* que hereda el canal. Si bien *ea_can_trans* es una clase que contiene la definición de la trama con cada uno de los campos del *frame* CAN, también contiene la sobrecarga de los operadores =, == y la declaración de los operadores << y *sc_trace* como *friend* de la clase. También en esta clase se define qué parámetros se seguirán (*trace*) para luego generar el archivo VCD. En la Figura B-2 se pueden ver algunos campos graficados en el GTKWave.

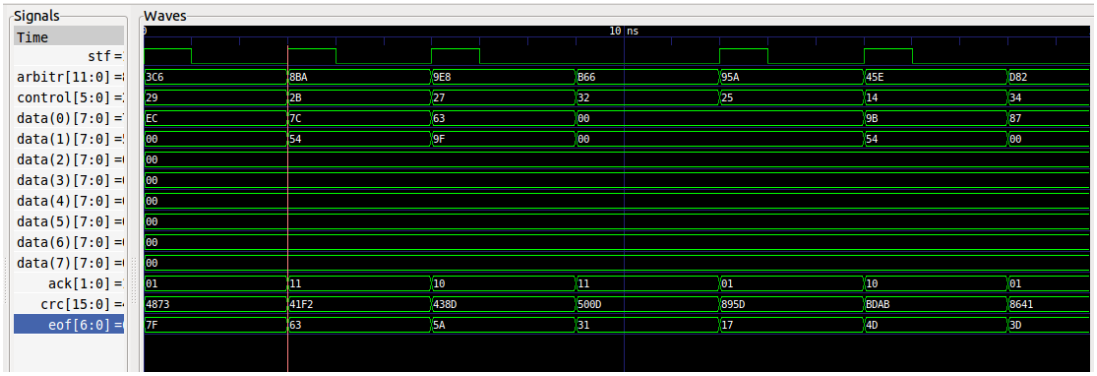

Figura B-2: Tracing del primer modelo

Aunque el canal *sc_signal* está diseñado para usarse en casos similares a éste (modelar un canal que transporte señales eléctricas que son cadenas de bits), de todas maneras no podría utilizarse para el bus CAN ya que en el canal se implementará el método que simule el comportamiento del mecanismo *wired-and*, cuando esto se efectúa en un canal, la metodología recomienda usar canales jerárquicos [7].

En el segundo refinamiento del modelo se repite el funcionamiento del primero pero remplazando el canal *sc_signal* por uno personalizado Bus_can que hereda de *sc_prim_channel* y Bus_can_if (la interface del canal). En la Figura B-3 se presenta una vista del modelo mencionado.

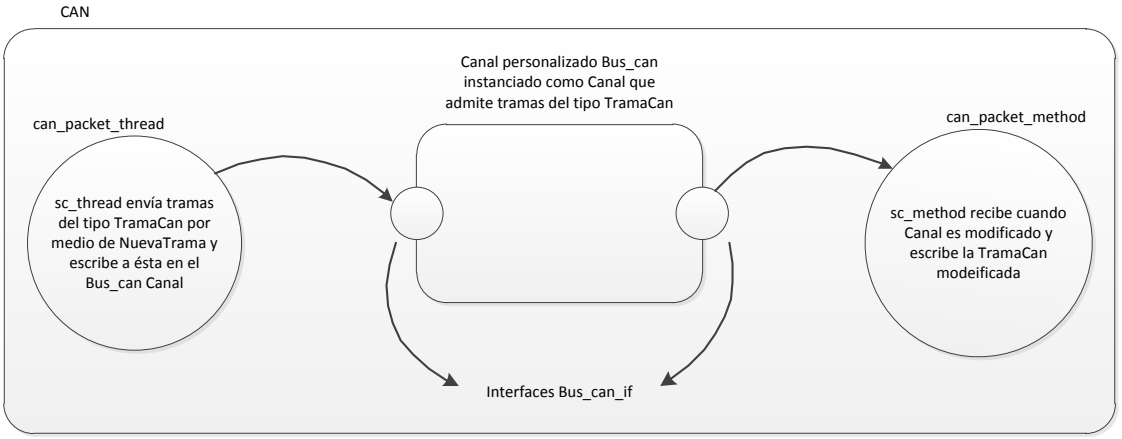

Figura B-3: Segundo modelo

Además, en Bus_can debe definirse un método write para modificar a tramaBus (instancia de TramaCan); esta escritura dispara el evento writeEvent para que can_packet_method (sensible a él) imprima el valor de la trama que fue modificada en el canal.

En los siguientes modelos, se realizaron tareas de refinamiento como encapsular los dos procesos de generación de trama y de recepción en dos módulos (Nodo1 y Nodo2). Pero también se hicieron reformas tales como implementar un método read que devuelve un frame CAN y agregar un canal *sc_clock* para utilizar procesos del tipo *sc_cthread* (que se disparan por flancos de reloj). En la Figura B-4 se puede observar el cuarto modelo.

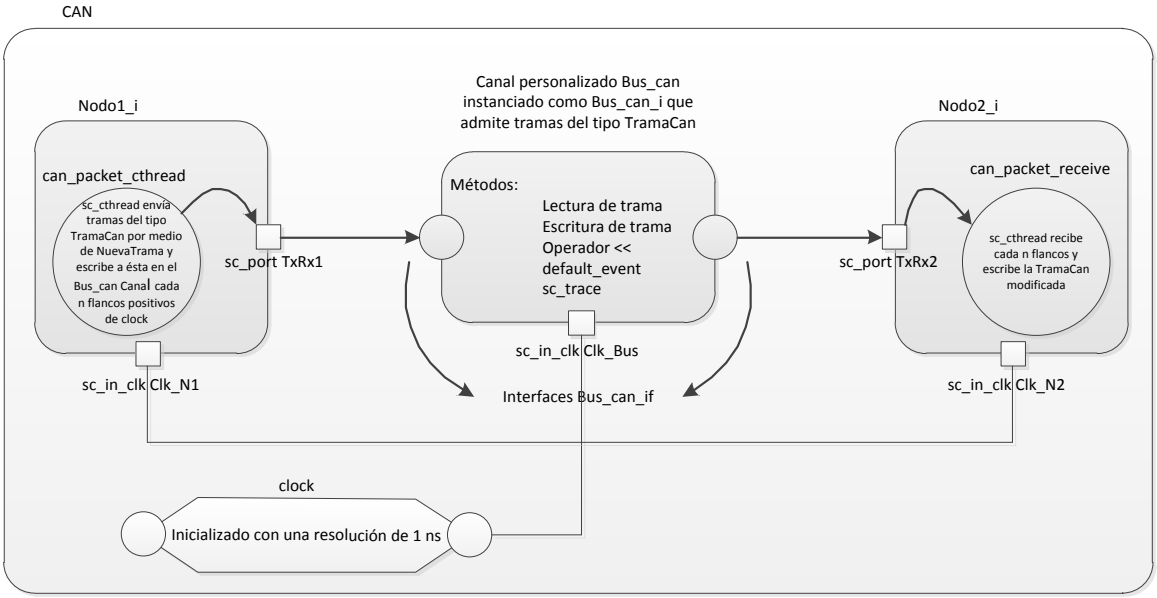

Figura B-4: Cuarto modelo

En la Figura B-5 puede verse algunos campos del frame junto con el nuevo clock. Se puede observar que el can_packet_cthread respeta el flanco ascendente para iniciar el armado de la trama y luego espera 3 flancos ascendentes para armar la próxima trama.

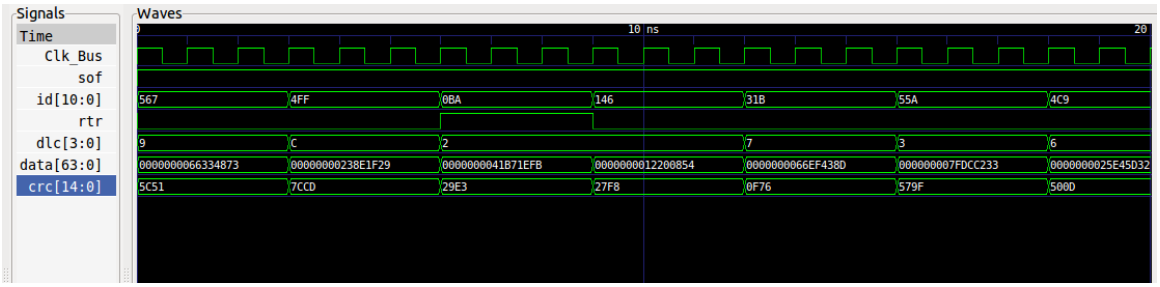

Figura B-5: Tracing del cuarto modelo

A continuación se introdujo el mecanismo de arbitraje, para conseguirlo se recurrió al canal *sc_mutex* (semáforo binario). Contemplar que dos nodos envíen al mismo tiempo sus tramas implicó redefinir el método write de las clases Nodo1, Nodo3, Bus_can_if y Bus_can para que se arbitre qué nodo depositará su trama en el bus de acuerdo a la prioridad de su ID (el protocolo CAN toma al menor ID como el más prioritario). Además, se creó un método, llamado hiloBus, que permite implementar las ventanas de escritura y lectura cada 3 ciclos de reloj para facilitar resolver el problema de concurrencia. Cuando un nodo envía un mensaje llama al método escribir definido en el bus, en éste se utiliza un objeto tipo *sc_mutex* para proteger la región crítica que define el arbitraje (si dos nodos están compitiendo). El método hiloBus define sincrónicamente las ventanas de escritura y lectura. En la Figura B-6 se puede observar el sexto modelo.

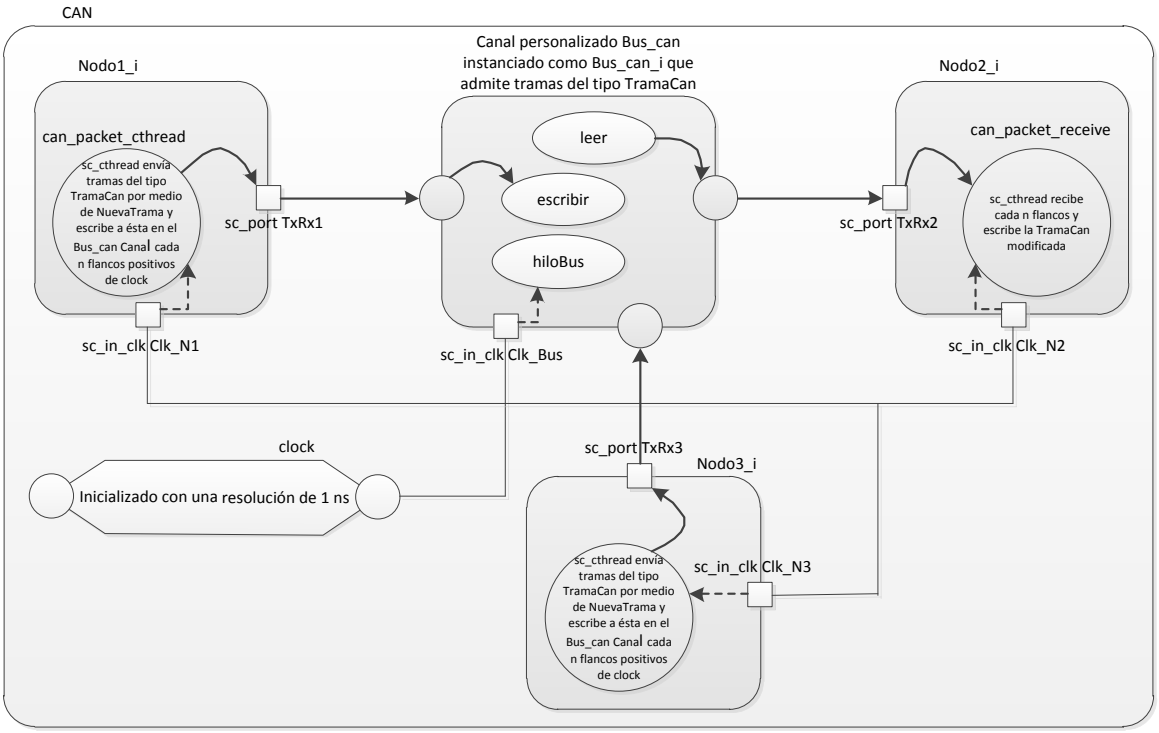

Figura B-6: Sexto modelo

En el siguiente modelo se utilizaron semáforos para sincronizar a los procesos de otra forma, en el caso que más de un nodo envié datos al mismo tiempo. Además se modificó la manera de definir la clase de Bus_can, ya no como sc_module y heredando de sc_prim_channel sino como una clase típica heredando de sc_channel y Bus_can_if. En la Figura B-7 pueden verse los campos ID y Data del frame en el séptimo modelo.

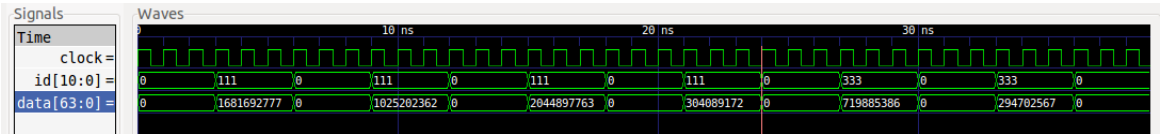

Figura B-7: Tracing del séptimo modelo

Estos modelos cubren las etapas de aprendizaje de la metodología TLM y uso del *framework* SystemC. Para llegar al modelo de simulación final se desarrollaron seis modelos adicionales que básicamente resolvieron dos problemas: la correcta sincronización de todos los procesos para que cumplan con los verdaderos tiempos del protocolo y el *refactoring* [79] del modelo de simulación. De esta forma se implementó un refinamiento de la arquitectura para conseguir una simulación a nivel de ciclos de reloj.

Bibliografía

[1] CoNAE, «Comisión Nacional de Actividades Espaciales,» 2013. [En línea]. Available: http://www.conae.gov.ar.

[2] NASA, «Sea Surface Salinity from Space,» 2013. [En línea]. Available: http://aquarius.gsfc.nasa.gov.

[3] CoNAE, «Detalles técnicos de los intrumentos a bordo de la Misión SAC-D Aquarius,» 2013. [En línea]. Available: http://www.conae.gov.ar/satelites/sac-d_instrum.html.

[4] Centro de Investigaciones Ópticas, «Satélite SAC-D,» 2013. [En línea]. Available: http://www.ciop.unlp.edu.ar/Espanhol/Actividades/SAC-D.html.

[5] R. Bosch, «CAN Specification. Version 2.0,» Robert Bosch GmbH, 1991.

[6] J. Eickhoff, Simulating Spacecraft Systems, Springer, 2009.

[7] D. Black, SystemC: From the Ground Up. Second Edition, Springer, 2010.

[8] A. Burns, Real-time systems and programming languages, Addison Wesley, 2009.

[9] H. Kopetz, Real-Time Systems. Design Principles for Distributed Embedded Applications. Second Edition, Springer, 2011.

[10] A. Lawrence, Modern Inertial Technology. Navigation, Guidance, and Control. Second Edition, Springer, 1998.

[11] D. W. J. Titterton, Strapdown Inertial Navigation Technology. Second Edition, The Institution of Electrical Engineers and The American Institute of Aeronautics and Astronautics, 2004.

[12] I. Stojmenovic, Handbook of Sensor Network. Algorithms and Architectures., John Wiley & Sons, 2005.

[13] N. Mahalik, Sensor Networks and Configuration. Fundamentals, Standard, Platforms, and Applications., Springer, 2007.

[14] «URL European Space Agencies,» 2013. [En línea]. Available: http://www.esa.int.

[15] J. Banks, Discrete-Event System Simulation, Prentice-Hall, 2010.

[16] A. Law, Simulation Modeling and Analysis, McGraw-Hill, Inc., 2006.

[17] F. Ghenassia, Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems., Springer, 2005.

[18] T. Grötker, System Design with SystemC, Kluwer Academic Publisher, 2004.

[19] IEEE Standards Association, *IEEE Standard for standard SystemC language reference manual,* IEEE Computer Society, 2012.

[20] «URL Accellera Systems Initiative,» 2013. [En línea]. Available: http://www.accellera.org.

[21] «URL European Aeronautic Defence and Space Company N.V.,» [En línea]. Available: http://www.eads.com.

[22] International Organization for Standardization, *ISO 11898 - Road vehicles - Controller Area Network (CAN) - Part 1 - Part 2 - Part 3 - Part 4,* 2005.

[23] R. Davis, «Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised.,» *Real-Time Systems. Springer,* vol. 35, nº 3, pp. 239-272, 2007.

[24] M. Khurram, «CAN as a spacecraft communication bus in LEO satellite mission,» de *Proceedings of 2nd International Conference on Recent Advances in Space Technologies*, 2005.

[25] S. Prevost, «A modular distributed avionics architecture for launch vehicles,» de *16th Digital Avionics Systems Conference. AIAA/IEEE*, 1997.

[26] M. Sveda, «Experience with integration and certification of COTS based embedded system into advanced avionics system,» de *International Symposium on Industrial Embedded Systems*, 2007.

[27] K. Ure, «The development of a software and hardware in the loop test system for ITU-PSAT II nano satellite ADCS,» de *IEEE Aerospace Conference*, 2011.

[28] S. Franklin, «The space technology 8 mission,» de *IEEE Aerospace Conference*, 2006.

[29] I. Koren, Fault Tolerant Systems, Morgan Kaufmann Publishers. Elsevier, 2007.

[30] MIL-HDBK-217E. Military handbook. Reliability prediction of electronic equipment, U.S. Department of Defence Interface Standard, 1990.

[31] J. Lala, «Architectural principles for safety-critical real-time applications,» *Proceedings of IEEE:9214152,* vol. 82, nº 1, pp. 25-40, 1994.

[32] European Cooperation for Space Standardization (ECSS), *Space Engineerig. Space Enviroment,* ECSS-E10-04B, 2008.

[33] European Cooperation for Space Standardization, *Space product assurance. Thermal vacuum outgassing test for the screening of space materials.,* ECSS-Q-ST-70-02C, 2008.

[34] NASA Preferred reability practices, *Space radiation effects on electronic components in low-earth orbit,* PD-ED-1258, 1996.

[35] Therm-a-gap 500 Series, *Thermally conductive gap fillers,* Chomerics. Parker Hannifin Corporation, 2006.

[36] European Cooperation for Space Standardization, *Space engineering. Electromagnetic compatibility,* ECSS-E-ST-20-07C, 2008.

[37] EPO-TEK 330, *Material safety data sheet,* Epoxy Technology, 2010.

[38] Kvaser, *PCIcan Hardware reference manual,* Kvaser Inc., 2002.

[39] Kvaser, *Leaf user's guide,* Kvaser Inc., 2006.

[40] MIL-HDBK-338B. Military handbook. Electronic reliability design handbook., U.S.A. Department of defense, 1998.

[41] Woo Sim Kim, «System-level development and verification of the FlexRay communication controller model based on SystemC,» *Proceedings of the 2008 second international conference on future generation communication and networking,* vol. 2, pp. 124 - 127, 2008.

[42] R. Pichappan, «A bus level SystemC model for evaluation of avionics mission system data bus,» *TENCON IEEE Region 10,* pp. 1 - 6, 2005.

[43] F. Consigli, «RAM Characteristics exploration with an MVEDR model,» *XVI Congreso Argentino de Ciencias de la Computación,* pp. 794 - 801, 2010.

[44] H. Shao, «Design of a simulation framework for automotive networks using SystemC,» Technische Universität Darmstadt, 2008.

[45] G. Defo, «Verification of a CAN bus model in SystemC with functional coverage,» *International Symposium on Industrial Embedded Systems,* pp. 28 - 35, 2010.

[46] «Object Management Group,» [En línea]. Available: http://www.omg.org/uml.

[47] ATMEL. [En línea]. Available: www.atmel.com.

[48] KEIL, *Getting started. Creating applications with µVision 4,* 2009.

[49] B. Kernighan, The C programming language. Second edition, Prentice Hall Software Series, 1988.

[50] ATMEL, «Atmel microcontroller ISP Software User´s Manual,» [En línea]. Available: http://www.atmel.com.

[51] L. Fredriksson, *A CanKingdom rev 3.01,* Kvaser Inc., 1995.

[52] D. Encinas, «Análisis de tráfico en una red de sensores girométricos del tipo IFOG aplicado a sistemas aeroespaciales,» *VI Congreso Argentino de Tecnología Espacial,* 2011.

[53] Kvaser, *Kvaser CANLib SDK v4.9. Development Kits and Tools,* Kvaser Inc., 2013.

[54] GNU, *GTKWave 3.3 Wave Analyzer User's Guide,* GNU, 2013.

[55] Atmel corporation, *AT89S52. 8-bits Microcontroller with 8KBytes In-System Programmable Flash,* 2008.

[56] Philips semiconductors, *SJA1000. Stand-alone CAN controller,* 2000.

[57] Philips semiconductors, *PCA82C250. CAN controller interface,* 2000.

[58] Philips semiconductors, *Application note. PCA82C250/251 CAN Transceiver. AN96116,* 1996.

[59] M. Di Natale, Undertanding and Using the Controller Area Network Communication Protocol. Theory and Practice, Springer, 2012.

[60] Philips semiconductors, *Application note. Determination of Bit Timing Parameters for the CAN Controller SJA 1000. AN97046,* 1997.

[61] R. Davis, «Controller Area Network (CAN) Schedulability analysis with FIFO queues,» *23rd Euromicro conference on Real-Time Systems,* pp. 45-56, 2011.

[62] P. Yomsi, «Controller Area Network (CAN): Response time analysis with offsets,» *9th IEEE International workshop on Factory Communication Systems,* pp. 43-52, 2012.

[63] N. Navet, «Controller Area Network (CAN) Schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues,» *9th IEEE International workshop on Factory Communication Systems,* pp. 33-42, 2012.

[64] J. Devore, Probabilidad y estadística para ingeniería y ciencias. Sexta Edición, Thomson Learning, 2005.

[65] M. Short, «Fault-Tolerant Time-Triggered communication using CAN,» *IEEE Transactions on Industrial Informatics,* vol. 3, nº 2, pp. 131-142, 2007.

[66] R. Kumar, «Impact of temperature fluctuations on circuit characteristics in 180nm and 65nm CMOS technologies,» *IEEE International Symposium on Circuits and Systems,* 2006.

[67] P. Peebles, Probability, Random Variables, and Random Signal Principles. Fourth Edition, McGraw-Hill, Inc., 2000.

[68] J. Kraus, Electromagnetismo con aplicaciones, McGraw-Hill, 2000.

[69] A. de Paula, «Experimental measurements and numerical simulation of permittivity and permeability of Teflon in X band,» *Journal of Aerospace Technology and Management,* vol. 3, nº 1, pp. 59-64, 2011.

[70] V. Yadav, «The effect of frequency and temperature on dielectric properties of Pure Poly Vinylidene Fluoride (PVDF) thin films,» *International MultiConference of Engineers and Computer Scientists,* vol. 3, 2010.

[71] J. Kurose, Redes de Computadores. 2° Edición, Pearson-Addison Wesley, 2004.

[72] INDEF S.A., «Horno eléctrico de circulación forzada. Modelo CFA-004,» 2013. [En línea]. Available: www.indef.com.ar.

[73] International Organization for Standardization. International Electrotechnical Commission, *ISO/IEC 14882. Information technology -- Programming languages -- C++,* 2011.

[74] M. Azarang, Simulación y análisis de modelos estocásticos, McGraw-Hill, 1997.

[75] Accellera Systems Initiative, «SystemC Synthesis Working Group (SWG),» [En línea]. Available: http://www.accellera.org/activities/committees/systemc-synthesis.

[76] Cadence Inc, «C-to-Silicon Compiler,» [En línea]. Available: http://www.cadence.com/products/sd/silicon_compiler/pages/default.aspx.

[77] Synopsys Inc, «System -Level Catalyst Member,» [En línea]. Available: http://www.synopsys.com/Community/Interoperability/SystemLevelCatalyst/Pages/MForte.aspx.

[78] OMG System Modeling Language, «The official OMG SysML,» [En línea]. Available: http://www.omgsysml.org/.

[79] E. Gamma, Patrones de diseño: Elementos de software orientado a objetos reutilizables, Addison-Wesley, 2002.