UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"RED BASADA EN EL PROTOCOLO FLEXRAY UTILIZANDO LA TARJETA LAUNCHPAD HERCULES TMS570LC43X"

TESIS

PARA OBTENER EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

PRESENTA

JOSÉ LUIS REY CABRERA BELLO

DIRECTOR DE TESIS

MEDI. HERIBERTO ILDEFONSO HERNÁNDEZ MARTÍNEZ

H. CD. DE HUAJUAPAN DE LEÓN, OAXACA; DICIEMBRE DE 2021

Índice General

Índice General iii

Índice de Figuras ix

Índice de Tablas xv

Capítulo 1. Introducción 1

1.1. Antecedentes 1

1.2. Protocolo de Comunicaciones FlexRay 2

1.3. Planteamiento del Problema 3

1.4. Justificación 4

1.5. Hipótesis 5

1.6. Objetivos 5

1.6.1. Objetivo general 5

1.6.2. Objetivos específicos 5

1.7. Metas 6

1.8. Metodología de Desarrollo 7

1.8.1. Fase 1: Especificación del producto 8

1.8.2. Fase 2: Partición hardware y software 9

1.8.3. Fase 3: Iteración e Implementación 9

1.8.4. Fase 4: Diseño detallado hardware y software 10

1.8.5. Fase 5: Integración hardware y software 10

1.8.6. Fase 6: Pruebas de aceptación 10

1.8.7. Fase 7: Mantenimiento y actualización 11

Capítulo 2. Redes de Comunicaciones en Automoción 13

2.1. Bus de Campo 13
2.2. Buses de Campo en Automoción 14
2.3. Protocolo de Comunicaciones CAN 15
2.3.1. Características principales del protocolo CAN 16
2.3.2. Control de acceso al medio 17
2.3.3. Formato de tramas CAN 18
2.3.4. Estados de los nodos 18
2.3.5. Limitaciones de CAN 19
2.4. Aspectos de Activación por Eventos y Activación por Tiempo 20
2.4.1. El lado probabilístico de CAN 20
2.4.2. El lado determinista de las aplicaciones 21
2.5. Protocolo TTCAN 22
2.6. Sistemas Embebidos 23
2.6.1. Interrupciones en los sistemas embebidos 24
2.6.2. Sistema operativo de tiempo real 25
Capítulo 3. Sistema de Protección Contra Impactos Airbag 27
3.1.1. Unidad de control electrónico del airbag 27
3.1.1.1. Sensores de accidente 28
3.1.1.2. Sensores de seguridad 29
3.1.2. Estructura del *airbag* 29
3.1.2.1. Sistema de detección de ocupación de los asientos 30
3.1.2.2. Cableado del airbag 30
3.1.2.3. Pretensor del cinturón 30
3.1.2.3.1. Delimitador de potencia del cinturón 31

3.1.2.4. Desconexión de la batería 31
3.1.2.5. Apagado de bomba de combustible 31
3.1.3. Sistema de restricción múltiple III 31
3.1.3.1. Umbrales de activación 32
Capítulo 4. Arquitectura de Protocolos Flexray 35
4.1. Revisión Histórica del Protocolo FlexRay 35
4.2. Conceptos Básicos de FlexRay 38
4.3. Capa Física FlexRay 39
4.3.1. Representación y sincronización de los bits 39
4.3.2. Especificaciones eléctricas de la señal 41
4.3.2.1. Transceptor FlexRay 41
4.3.2.1.1. Terminador de interfaz de conexión 42
4.3.2.1.2. Bobina de choque 43
4.3.2.1.3. Protección por descarga electroestática 43
4.3.2.2. Topologías de red en FlexRay 44
4.3.3. Implementación del nodo 45
4.3.3.1. Tipos de memoria en la tarjeta Hercules 47
4.3.3.2. Módulo FlexRay 48
4.3.3.2.1. Implementación del módulo FlexRay 49
4.3.3.2.1. Controlador de comunicación del módulo FlexRay 50
4.3.3.2.2. Unidad de transferencia FlexRay 53
4.3.3.2.3. Ciclo de comunicación 54
4.4. Capa de Enlace de Datos 55
4.4.1. Subcapa de control de enlace lógico 55

4.4.1.1. Controlador de interfaz con el host 55
4.4.1.2. Control de operaciones del protocolo 56
4.4.2. Control de acceso al medio 58
4.4.2.1. Representación del tiempo en FlexRay 58
4.4.2.1.1. Ciclo de comunicación 59
4.4.2.2. Formato de trama 63
4.4.2.2.1. Secuencia de inicio de transmisión 66
4.4.2.2.2. Secuencia de inicio de trama 67
4.4.2.2.3. Secuencia de inicio de bytes 67
4.4.2.2.4. Secuencia de finalización de trama 67
4.4.2.2.1. Secuencia de segmento estático y de segmento dinámico 67
4.4.2.3. Formato de símbolo 68
4.4.2.3.1. Patrón 1: Símbolo CAS y símbolo MTS 68
4.4.2.3.2. Patrón 2: Símbolo WUS 69
4.4.2.3.3. Patrón 3: Símbolo WUDOP 70
Capítulo 5. Desarrollo del Sistema FlexREY 73
5.1. Especificaciones del Sistema FlexREY 73
5.1.1. Nodos FlexRay 73
5.1.1.1. Comunicación FlexRay 75
5.1.1.1.1. Software embebido del nodo 77
5.1.2. Paneles de simulación del sistema airbag 78
5.1.3. Interfaz gráfica de usuario 79
5.2. Partición Hardware y Software 79
5.3. Iteración e Implementación 82

5.4. Diseño Detallado Hardware y Software 82
5.4.1. Programa principal del MCU 83
5.4.1.1. Módulos integrados en tarjeta Hercules 83
5.4.1.2. Proyecto nodo A 84
5.4.1.3. Proyecto nodo B 85
5.4.2. Módulo FlexRay 88
5.4.2.1. Configuración por defecto del módulo FlexRay 89
5.4.2.2. Configuración de los búferes 89
5.4.2.3. Inicialización del controlador de comunicaciones 91
5.4.2.4. Inicialización de interrupciones y habilitación coldstart 92
5.4.2.5. Construcción de la trama 92
5.4.2.6. Transmisión de la trama 93
5.4.2.7. Recepción de la trama 93
5.4.3. Módulo Bus Driver 93
5.4.3.1. Conexión y alimentación de la placa 94
5.4.4. Módulos del panel airbag sensores 94
5.4.5. Módulo del panel airbag indicadores 98
5.4.6. Módulo de simulación MRS III 103
5.4.6.1. Determinación de la respuesta del sistema 104
5.4.7. Diseño de la GUI 104
5.4.7.1. Operación 105
5.4.7.2. Interfaz gráfica de usuario 106
5.4.7.3. Módulo de comunicación serial 109
5.5. Integración HW y SW del Sistema FlexREY 109

Capítulo 6. Resultados 115

6.1. Sistema FlexREY 115

6.2. Encendido y Conexión del Sistema 118

6.3. Pruebas de Aceptación 118

6.3.1. Pruebas de caja negra 119

6.3.2. Pruebas de caja gris 125

Conclusiones 135

Bibliografía 141

Anexo A. Circuitos Impresos A-1

Anexo B. Código biblioteca FlexRay(fray.h) B-1

Anexo C. Código biblioteca FlexRay(fray.c) C-1

Anexo D. Código biblioteca MRSIII (MRSIII.h) D-1

Anexo E. Código biblioteca MRSIII (MRSIII.c) E-1

Anexo F. Código módulo FlexRay A F-1

Anexo G. Código módulo FlexRay B G-1

Anexo H. Código proyecto nodo A (HL_sys_main.c) H-1

Anexo I. Código proyecto nodo B (HL_sys_main.c) I-1

Anexo J. Script Matlab del procesamiento digital de la trama FlexRay muestreada. J-1

Índice de Figuras

Figura 1.1. Diagrama del ciclo de vida del diseño de sistemas embebidos (Berger, 2002). 8

Figura 1.2. División HW y SW del sistema embebido. 9

Figura 2.1. Topología del bus CAN. 16

Figura 2.2. Formato base de la trama CAN (Basada en https://es.wikipedia.org/wiki/Bus_ CAN). 18

Figura 2.3. Estados de error de nodo CAN (Martínez Requena, 2017). 19

Figura 2.4. Matriz del principio de funcionamiento del protocolo TTCAN (Paret, 2012). 23

Figura 2.5. Cambio de contexto (Galeano, 2009). 25

Figura 3.1. Esquema del sistema de restricción múltiple III (BMW of North America, Inc., 2001). 32

Figura 4.1. Arquitectura de nodo FlexRay (FlexRay Consortium, 2010). 39

Figura 4.2. Codificación NRZ (González Salinas, 2008). 40

Figura 4.3. Sincronización, muestreo y BitStrobing (FlexRay Consortium, 2010). 40

Figura 4.4. Estados en el bus FlexRay utilizando TJA1080A (FlexRay Consortium, 2010). 42

Figura 4.5. Esquema terminador de conexión (Xin He, 2018). 43

Figura 4.6. Ejemplo de topología híbrida con canal dual (Texas Instruments Incorporated, 2015). 44

Figura 4.7. Estructura MCU+BD (Xin He, 2018). 45

Figura 4.8. Tarjeta LAUNCHXL2-570LC43 (Texas Instruments Incorporated, 2019). 46

Figura 4.9. Diagrama de bloques del módulo FlexRay (Texas Instruments Incorporated, 2018). 49

Figura 4.10. Diagrama de segmentos de memoria del módulo FlexRay (Texas Instruments Incorporated, 2018). 49

Figura 4.11. Diagrama de estados general del controlador de comunicaciones (Texas Instruments Incorporated, 2018)..... 50

Figura 4.12. Diagrama de acceso entre CPU Host y RAM de mensajes (Texas Instruments Incorporated, 2018). 53

Figura 4.13. Estructura del búfer de mensajes (Texas Instruments Incorporated, 2018). 53

Figura 4.14. Capas del protocolo FlexRay (González Salinas, 2008). 55

Figura 4.15. Representación de tiempos FlexRay (Paret, 2012). 59

Figura 4.16. Estructura en ciclo de comunicación FlexRay (Paret, 2012). 62

Figura 4.17. Trama del protocolo FlexRay (FlexRay Consortium, 2010). 63

Figura 4.18. Encapsulamiento de código trama y código de símbolo (Paret, 2012). 64

Figura 4.19. Secuencia de empaquetado de bytes de la trama FlexRay (Iversen Huse, 2017) y (González Salinas, 2008). 66

Figura 4.20. Fin de ranura estática (Paret, 2012). 68

Figura 4.21. Fin de ranura dinámica (Paret, 2012)..... 68

Figura 4.22. Codificación de símbolo CAS/MTS (FlexRay Consortium, 2010).. 69

Figura 4.23. Patrón de activación que consta de dos o mas WUS (FlexRay Consortium, 2010). 70

Figura 4.24. Patrón de activación WUDOP (FlexRay Consortium, 2010)..... 71

Figura 5.1. Diagrama a bloques del sistema FlexREY..... 74

Figura 5.2. Diagrama de funcionamiento de los nodos A y B. 75

Figura 5.3. Diagrama de funcionamiento de sistema MRS III (BMW of North America, Inc., 2001). 78
Figura 5.4. Principio de operación del sistema por estados. 78
Figura 5.5. Diagrama de funcionamiento del nodo A. 86
Figura 5.6. Diagrama de funcionamiento del nodo B. 87
Figura 5.7. Diagrama de flujo de la configuración del módulo FlexRay. 88
Figura 5.8. Integración de las funciones del sistema. 92
Figura 5.9. Diagrama eléctrico del Bus Driver. 95
Figura 5.10. Módulo de panel sensores. 96
Figura 5.11. Diagrama eléctrico del panel de sensores. 99
Figura 5.12. Diseño de panel indicadores (BMW of North America, Inc., 2001). 101
Figura 5.13. Diagrama electrónico del panel de indicadores. 102
Figura 5.14. Diagrama de operación de interfaz. 105
Figura 5.15. Diagrama de funcionamiento a) Proceso Interfaz, b) Proceso Comunicación. 107
Figura 5.16. Diagrama de funcionamiento a) Proceso Monitor, b) Proceso Salir. 108
Figura 5.17. Diagrama a bloques del sistema integrado del nodo A, interfaz de usuario y panel de sensores. 112
Figura 5.18. Diagrama a bloques del sistema integrado del nodo B y panel de indicadores. 113
Figura 6.1. Sistema FlexREY. 115
Figura 6.2.Red FlexRay. 116
Figura 6.3. Placa de sensores. 116
Figura 6.4. Placa de indicadores. 117
Figura 6.5. Interfaz gráfica de usuario GUI. 117

Figura 6.6. Conexión de la interfaz con el nodo A. 118
Figura 6.7. Prueba 1: Error de autocomprobación, sistema deshabilitado..... 119
Figura 6.8. Prueba 2: Autocomprobación pasada, sistema habilitado..... 120
Figura 6.9. Prueba 3: Respuesta a desaceleraciones drásticas. 120
Figura 6.10. Prueba 4: Respuesta a impactos frontales ligeros. 121
Figura 6.11. Prueba 5: Respuesta a impactos frontales medianos. 121
Figura 6.12. Prueba 6: Respuesta a impactos frontales fuertes. 122
Figura 6.13. Prueba 7: Respuesta a impactos laterales fuertes..... 122
Figura 6.14. Cambio en la interfaz al modo de simulador de sensores. 123
Figura 6.15. Construcción de trama personalizada..... 123
Figura 6.16. Respuesta a tramas personalizadas. 124
Figura 6.17. Captura en osciloscopio del sistema completo. 125
Figura 6.18. Diagrama de seguimiento para prueba caja gris. 125
Figura 6.19. Registros de la interfaz gráfica..... 126
Figura 6.20. Trama enviada por puerto serial y decodificación con base en el reloj (Señal muestreada con osciloscopio y procesada digitalmente). 127
Figura 6.21. Registro de la estructura de datos de la trama serial recibida. .. 127
Figura 6.22. Carga de datos en búfer FlexRay. 128
Figura 6.23. Lectura de osciloscopio de la transmisión sobre el bus FlexRay. 128
Figura 6.24. Captura de osciloscopio primer ranura de la transmisión. 129
Figura 6.25. Trama diferencial FlexRay y uBus codificado (Señal muestreada con osciloscopio y procesada digitalmente). 129
Figura 6.26. Inicio de trama FlexRay y decodificación por segmentos del uBus. 130
Figura 6.27. Carga útil en la trama FlexRay y decodificación por segmentos del uBus. 130
Figura 6.28. Tráiler CRC y secuencia de fin de trama. 131

Figura 6.29. Datos del búfer de recepción en la ranura 1 del nodo B. 132
Figura 6.30. Registro de panel de indicadores. 132
Figura 6.31.Registros de escritura del búfer para ranura 4 en el nodo B..... 133
Figura 6.32. Registros de lectura del búfer de la ranura 4 en el nodo A..... 134
Figura 6.33. Lectura de trama de serial en la interfaz. 134

Índice de Tablas

Tabla 2.1. Velocidad de transmisión del bus CAN con respecto a la distancia (Martínez Requena, 2017). 16

Tabla 3.1. Umbrales de activación de airbag de dos etapas. 33

Tabla 4.1. Revisión histórica FlexRay (FlexRay Consortium, 2010). 36

Tabla 4.2. Características del cableado y conectores para el bus. 41

Tabla 4.3. Nivel de señal en el bus en términos de los estados del BD (González Salinas, 2008)..... 42

Tabla 4. Resumen de las restricciones de las topologías..... 44

Tabla 4.5. Rango de direcciones del módulo FlexRay (Texas Instruments Incorporated, 2018)..... 50

Tabla 4.6. División a bloques de los registros del controlador de comunicaciones del módulo FlexRay (Texas Instruments Incorporated, 2018). 51

Tabla 5.1. Asignación de datos en las ranura y mini ranuras para construcción de la trama FlexRay. 77

Tabla 5.2. Valores de configuración del CC. 89

Tabla 5.3. Configuración global para búferes..... 90

Tabla 5.4. Configuración individual del búfer. 90

Tabla 5.5. Asignación de periféricos de los elementos del panel de sensores. .. 97

Tabla 5.6. Asignación de periféricos de los elementos del panel de indicadores. 100

Tabla 5.7. Funciones del simulador del sistema MRS III. 103

Tabla 5.8. Tramas del nodo A. 110

Tabla 5.9. Tramas del nodo B. 110

Tabla 5.10. Tramas para la GUI. 111

Dedicatoria

Con todo mi amor a mis padres Joaquín y María del Rayo.

A mis hermanas, Yesica y Rayo.

A mi sobrina, Alondra.

Luis Rey.

Agradecimientos

Agradezco ante todo a mis padres, Joaquín y María del Rayo, por el apoyo y motivación que siempre me han dado, la confianza depositada y la paciencia brindada, que siempre han sido mis guías en mis peores momentos, pero finalmente puedo decirles lo logramos y los amo.

Yesica y Rayo, ustedes son mi inspiración, les agradezco por todo su apoyo ahora y siempre, espero estén tan orgullosas de mí como yo lo estoy de ser su hermano.

Al profesor Heriberto I. Hernández Martínez, por aceptar trabajar una loca idea mía que resultó en el desarrollo de este trabajo, por brindarme su conocimiento y experiencia, así como una excelente amistad.

A mi amigo Salvador (chaba), por siempre estar dispuesto a ayudarme y a todos los amigos que me dieron todo su apoyo moral.

A los profesores José Eduardo Cruz Mayo, José Antonio Moreno Espinosa, Felipe Santiago Espinosa y Hugo Fermín Ramírez Leyva, por el apoyo e interés mostrado para el desarrollo de este trabajo.

A la Universidad Tecnológica de la Mixteca, por permitirme realizar este trabajo en sus instalaciones.

Luis Rey.

Capítulo 1. Introducción

La aplicación de la electrónica en el automóvil ha tenido un crecimiento acelerado en los últimos años, pasando de tener una única unidad de control electrónico (ECU, *Electronic Control Unit*) a incorporar varias ECU en un vehículo. Cada ECU cumple sus propias funciones, por ejemplo, existen ECU dedicadas a los sistemas de frenado, de suspensión, de seguridad y de confort, entre otros, lo que genera la necesidad de comunicarse entre sí para dar a conocer los resultados de sus tareas a las demás ECU y a la computadora central del automóvil.

Lo anterior dio origen a las redes de comunicaciones automotrices que con el paso de los años han logrado su estandarizaron; dicha estandarización se basó en el establecimiento de reglas normalizadas con el fin de presentar, señalizar, autenticar y detectar los errores en la transmisión de la información, obteniendo arquitecturas de protocolos de comunicaciones automotrices (Carpio Guartambel, 2013).

Por otro lado, el sistema de protección contra impactos '*airbag*', complementa el uso del cinturón de seguridad para ofrecer una mejor protección en caso de accidentes. Dicho sistema se basa en una bolsa de aire que al detectar una colisión se infla para amortiguar el impacto del ocupante (Murcia Educarm, 2019).

1.1. Antecedentes

Como antecedentes a este proyecto de tesis se consideran los siguientes trabajos:

- Para titularse como Ingeniero en Electrónica en la Universidad Tecnológica de la Mixteca, René González Salinas defendió la tesis intitulada "Especificación del protocolo FlexRay utilizando un lenguaje de descripción formal"; en dicho trabajo desarrolló la descripción

formal de las especificaciones del protocolo FlexRay versión 2.1 Rev A utilizando técnicas FDT, particularmente Cinderella SDL, permitiendo múltiples niveles de abstracción para la descripción y especificación del protocolo (González Salinas, 2008).

- El proyecto fin de carrera intitulado "Estudio del nuevo bus de automoción FlexRay y diseño de un prototipo ilustrativo de la tecnología", presentado por el C. Víctor Jiménez García en la Universidad Politécnica de Cataluña, describe un prototipo ilustrativo de la tecnología FlexRay para un sistema de *collision avoidance* para la empresa FICOSA. En dicho trabajo, desarrolló un sistema que controlaba una rueda con un motor eléctrico (sistema *Brake-by-Wire*) de acuerdo con las señales proporcionadas por un sistema de visión ADAS. La comunicación de dichas señales se realiza mediante nodos FlexRay; por lo que diseñó sus PCB basándose en el controlador de comunicaciones MB88121B de Fujitsu y el TJA1080 de NXP como administrador de bus (Jiménez García, 2008).
- En el proyecto de tesis de maestría del C. Markus Iversen Huse intitulado "*FlexRay analysis, configuration parameter estimation, and adversaries*" presentado en la *Norwegian University of Science and Technology* se describe la estimación de los parámetros de configuración para una red FlexRay basada en el microcontrolador TMS570LS1227, así como los resultados obtenidos al conectar dicha red a un analizador de comunicaciones y a un vehículo de pruebas (Iversen Huse, 2017).

1.2. Protocolo de Comunicaciones FlexRay

En el año 2000 se fundó el consorcio FlexRay con el objetivo de desarrollar un estándar de comunicaciones determinístico y tolerante a fallos. Dicho consorcio está conformado por Daimler Chrysler, BMW, Motorola y Philips, entre otras empresas; la versión 3.0.1 es la más reciente de las especificaciones del

protocolo FlexRay y se publicó en 2010 (Universitatea Politehnica Timisoara, 2019).

Uno de los objetivos del protocolo FlexRay es lograr sustituir las partes mecánicas del automóvil por ECU y buses de comunicación, permitiendo la aplicación de los conceptos de frenado por cable (*brake-by-wire*) y conducción por cable (*steer-by-wire*) que son sistemas computarizados de menor costo que los sistemas hidráulicos y mecánicos tradicionales, además de nuevas aplicaciones para los sistemas como asistencia al conductor, gestión de propulsión y sistemas anti colisión, entre otros.

1.3. Planteamiento del Problema

El presente trabajo contempla el desarrollo de un sistema electrónico para la generación de una red basada en el protocolo de comunicaciones FlexRay, considerando como referencia el trabajo realizado por el Ing. René González Salinas y añadiendo una plataforma de pruebas e interfaz de usuario, con lo que se propone facilitar el estudio del protocolo. Las fases de desarrollo del sistema implicaron el manejo y la aplicación de elementos de programación para sistemas embebidos, procesamiento digital de señales, sistemas de comunicaciones e instrumentación electrónica, así como la familiarización con la arquitectura de protocolos FlexRay.

Los recursos software y hardware disponibles en los dispositivos utilizados ayudan a un desarrollo práctico, eficiente y económico para el proyecto, ya que las herramientas utilizadas son de software y de hardware libre.

Los resultados de este proyecto pretenden describir la base para la implementación de una red FlexRay de bajo costo con fines académicos y de investigación, brindando una opción para pruebas a los estudiantes y profesores interesados en el área, así como un prototipo operativo para la evaluación del funcionamiento del protocolo con capacidades reconfigurables.

Se consideró el sistema de protección contra impactos *airbag* de BMW (MRS III), por lo que se desarrollaron dos paneles de prueba comunicados

mediante FlexRay para simular su funcionamiento. El primero integra los módulos de sensores de impacto, correspondientes a la distribución satelital en el automóvil y los sensores de seguridad, los sensores de detección de ocupación de los asientos y los interruptores (*switch*) de los cinturones se implementan con interruptores de palanca. La velocidad del automóvil se simuló utilizando un potenciómetro como pedal de acelerador. El segundo panel cuenta con testigos luminosos de la respuesta del sistema, utilizando luces LED para indicar la activación de cada componente.

El desarrollo de la interfaz gráfica de usuario (GUI, *Graphic User Interface*) se realizó en Java, ya que este lenguaje cuenta con una amplia gama de bibliotecas que ayudan a mejorar la velocidad de desarrollo y permite optimizar las funciones de la interfaz; dicha GUI cuenta con un estructurador de tramas que permite tanto el envío de mensajes personalizados como la decodificación de los mensajes recibidos sobre el nodo FlexRay, así mismo, la misma interfaz se utiliza durante las pruebas generales de la red para probar el funcionamiento correcto de la implementación del protocolo FlexRay.

1.4. Justificación

La industria automotriz se ha constituido como una de las más grandes a nivel global debido a la imperiosa necesidad de transporte para el desarrollo de las sociedades, ocasionando el aumento del parque vehicular. De acuerdo con la Organización Mundial de la Salud (2017) la cantidad de decesos en carreteras a nivel global asciende aproximadamente a 1.3 millones de personas, colocando los accidentes de tránsito como una de las principales causas de muerte. Para tratar de solventar esta problemática se han implementado más y mejores sistemas de seguridad en los automóviles, con capacidades de operar a mayores velocidades, funcionamiento en tiempo real y tolerancia a fallos, además de comenzar a sustituir partes mecánicas del auto por ECU y buses de comunicaciones; estas nuevas tendencias tecnológicas harán que los actuales buses de comunicaciones

automotrices, como LIN y CAN, sean incapaces de cumplir con las exigencias en los próximos años.

El creciente aumento en los requerimientos de los buses de comunicaciones automotrices ha generado el desarrollo de nuevas soluciones que sean capaces de cumplirlos de forma eficiente y natural, tal es el caso del protocolo FlexRay; sin embargo, al ser un "nuevo" protocolo, en comparación a sus antecesores LIN y CAN, su utilización aún se encuentra reducida a productos de gama alta de fabricantes como BMW e Ideco, entre otros. Lo anterior hace necesario que su estudio e implementación a nivel académico sea reducido debido a la complejidad y costo de los equipos que lo soportan. Cabe resaltar que el estudio de estas tecnologías es indispensable para poder mantenerse a la vanguardia en este sector y da pauta a generar nuevas líneas de investigación (Jiménez García, 2008).

1.5. Hipótesis

El desarrollo de una red basada en el protocolo de comunicaciones FlexRay empleando la tarjeta LaunchPad Hercules TMS570LC43x permitirá simular el funcionamiento del sistema MRS III de la firma BMW.

1.6. Objetivos

1.6.1. Objetivo general

Desarrollar de una red FlexRay y simular el comportamiento del sistema *airbag* MRS III de BMW empleando la tarjeta LaunchPad Hercules TMS570LC43x.

1.6.2. Objetivos específicos

Se establecieron los siguientes objetivos específicos:

- Configurar la tarjeta LaunchPad Hercules para su funcionamiento como parte de un nodo FlexRay.

- Diseñar el controlador de bus (BD) FlexRay con base en el transceptor TJA1080A.
- Implementar el nodo FlexRay con base en la tarjeta LaunchPad Hercules y el BD.
- Desarrollar la interfaz gráfica de usuario (GUI) para el monitoreo de la red.
- Desarrollar los paneles de pruebas para simular el sistema *airbag* MRS III de BMW.
- Integrar la red de comunicaciones basada en el protocolo FlexRay.
- Verificar el correcto funcionamiento de la red basada en el protocolo de comunicaciones FlexRay respecto a las especificaciones iniciales.

1.7. Metas

Con base en los objetivos planteados se logró cumplir con las siguientes metas:

- Se analizó la versión 2.1 Rev. A de la especificación del protocolo FlexRay y las especificaciones del sistema de desarrollo de TI en el microcontrolador Hercules TMS570LC4357 para vincular la teoría con la implementación.
- Se configuraron en HALCoGen las bibliotecas requeridas por la tarjeta de desarrollo LaunchPad Hercules TSM570LC43x para activar las funciones del MCU y los periféricos requeridos por el sistema.
- Se programó el software embebido en CCS para habilitar la unidad FlexRay del MCU.
- Se diseñó el BD FlexRay con base en el transceptor TJA1080A.
- Se integró la tarjeta LaunchPad Hercules programada y el módulo BD diseñado para implementar el nodo FlexRay.
- Se investigaron las distintas herramientas de Java para desarrollar la GUI.

- Se diseñó el panel de pruebas para la simulación del sistema de airbag MRS III de BMW.
- Se programó el software embebido para realizar la simulación del comportamiento del sistema MRS III y la comunicación con la interfaz.
- Se integraron la GUI y el panel de pruebas, al nodo FlexRay para acoplar sus respectivas funciones a la red FlexRay.
- Se conectaron los nodos (con su respectiva interfaz y panel) al bus para la formación de la red basada en el protocolo de comunicaciones FlexRay.
- Se probó la correcta transmisión de datos a través de la red para determinar que los nodos se comunicaban correctamente.
- Se probó el funcionamiento de los paneles para verificar que respondían conforme al sistema MRS III.
- Se verificó el funcionamiento de la red FlexRay mediante pruebas de caja negra y de caja gris para comprobar que cumplía con las especificaciones iniciales.

1.8. Metodología de Desarrollo

Para realizar el presente trabajo de tesis, se consideró la metodología basada en el modelo de sistemas embebidos propuesta por Arnold Berger (Berger, 2002), esto implicó que el *hardware* (HW) y el *software* (SW) se diseñaron en paralelo. Berger (2002) propone dividir este proceso en siete fases (véase la Figura 1.1); es importante considerar que el diseño no es tan simple, ya que, dentro de cada fase y entre ellas, existen una gran cantidad de iteraciones y optimizaciones, y en caso de encontrar defectos o deficiencias es común regresar a la fase correspondiente.

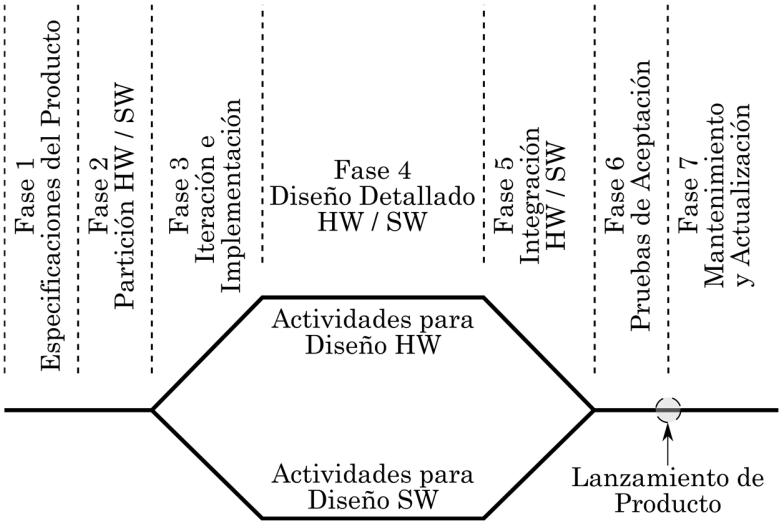


Figura 1.1. Diagrama del ciclo de vida del diseño de sistemas embebidos (Berger, 2002).

1.8.1. Fase 1: Especificación del producto

El proceso de desarrollo del sistema embebido inicia con la especificación del producto, que debe describir *lo que será* y *lo que hará* el producto al finalizarse; define el modo en que este interactuará (entradas y salidas) con el usuario y otros sistemas, sus funcionalidades y condiciones de error. Con esto se logra obtener una descripción del producto de forma abstracta con base en las necesidades, además, en esta fase se eligen las herramientas que se utilizarán en el desarrollo del HW y del SW con el objetivo de establecer la viabilidad en el proceso de desarrollo y minimización de riesgos.

Dentro de la fase de especificación del producto se identifican las siguientes etapas:

- Investigación documental: en esta etapa se lleva a cabo la identificación de la teoría fundamental relacionada con las especificaciones y los componentes de HW y de SW del sistema.
- Definición de requerimientos: en esta etapa se analizan los conceptos teóricos acerca del funcionamiento y el manejo del sistema de acuerdo con el estándar deseado, con la finalidad de establecer claramente los requerimientos de HW y de SW del sistema.

- Caracterización de los componentes del sistema: una vez establecidos lo requerimientos HW y SW, se lleva a cabo una familiarización con los dispositivos propuestos para el desarrollo del sistema, mediante una comprobación de sus recursos.

1.8.2. Fase 2: Partición hardware y software

Generalmente, el desarrollo de un sistema embebido contempla el desarrollo de HW y de SW (véase la Figura 1.2), por lo que se debe realizar la clasificación entre los distintos componentes del sistema, dividiéndolos en componentes HW y componentes SW para un manejo más específico de los requerimientos y partes a desarrollar. Para el HW los requerimientos son más rigurosos que para el SW, debido a que es más complicado y costoso corregir defectos de HW.

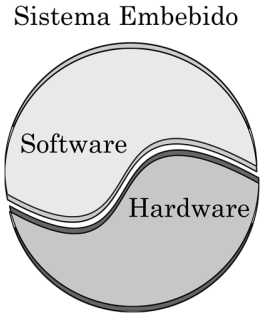


Figura 1.2. División HW y SW del sistema embebido.

1.8.3. Fase 3: Iteración e Implementación

Se repite el proceso de segmentación de compontes con el fin de establecer la división más eficiente de tareas de hardware y de software, el diseño es fluido y en el caso del HW pueden utilizarse simuladores para componentes y memorias del sistema; en el diseño del SW se desarrolla el código para el software embebido. El HW y el SW se trabaja conjuntamente para mantener activo el proceso de iteración.

1.8.4. Fase 4: Diseño detallado hardware y software

El objetivo de esta fase es obtener un diseño a detalle del sistema con base en los requerimientos iniciales, considerando entornos de desarrollo junto a las técnicas especiales de SW, técnicas especiales de programación y el diseño digital junto a la arquitectura de MCU.

Se especifican las funciones para HW y para SW de forma independiente, pero con un desarrollo en paralelo. También, se configuran los módulos y componentes de HW para las diferentes funciones que el sistema requiera, así como la programación de las secuencias para la transmisión de datos y el procesamiento de las señales junto al diseño de la interfaz; al mismo tiempo que se describen las implementaciones realizadas mediante diagramas de flujo, pseudocódigos u otros métodos.

1.8.5. Fase 5: Integración hardware y software

La clave del diseño de sistemas embebidos es la integración del prototipo HW con el software embebido y la aplicación SW sobre el sistema operativo. Una vez definidas y configuradas las funciones específicas de cada uno de los componentes HW y SW, se realiza la integración de los componentes del sistema y se vincula el HW con las rutinas desarrolladas en SW. Los métodos de depuración usualmente utilizados en computadoras son similares a los utilizados en sistemas embebidos, sin embargo, es importante considerar tres requisitos para depurar un sistema embebido: a) Control de ejecución, b) Sustitución de memoria y c) Análisis en tiempo real.

1.8.6. Fase 6: Pruebas de aceptación

En esta fase se cuantifica tanto el funcionamiento como el desempeño del producto mediante pruebas experimentales. Las pruebas para sistemas embebidos son más estrictas que la mayoría de las aplicaciones de escritorio, ya que incluyen aspectos de seguridad y se enfocan en determinar si el sistema opera correctamente. Inicialmente, se evalúa el funcionamiento y confiabilidad

del sistema mediante pruebas simples, fácilmente verificables; además del desempeño del sistema. Posteriormente se eleva el nivel de complejidad de las pruebas para medir la robustez del sistema, así como su respuesta a la simulación de posibles casos de error y su acoplamiento con otros sistemas.

1.8.7. Fase 7: Mantenimiento y actualización

Normalmente, más que diseñarse nuevos sistemas embebidos, se les brinda mantenimiento y actualización a los ya existentes. Esta fase requiere de herramientas adecuadas para reingeniería, que permitan el monitoreo y observación del código de ejecución en tiempo real.

Capítulo 2. Redes de Comunicaciones en Automoción

2.1. Bus de Campo

En los años 90, en el área industrial se comenzó a popularizar la opción de comunicaciones denominada bus de campo (FB, *Field Bus*). Un bus de campo es un sistema de transmisión de información mediante un sólo cable de comunicación, con lo que se lograron simplificar en gran medida la instalación y la operación de sistemas industriales. Debido a la falta de estándares, se diseñaron y desarrollaron diversas soluciones, cada una con diferentes prestaciones y campos de aplicación, algunas de estas son (Salazar Serna y Correa Ortiz, 2011):

- Buses de alta velocidad y baja funcionalidad: están diseñados para integrar dispositivos simples como sensores de fin de carrera, relevadores y actuadores simples, que son utilizados comúnmente en aplicaciones de tiempo real (*real-time*) y agrupados en pequeñas zonas como puede ser una máquina industrial. La arquitectura de protocolos de comunicaciones en la que se basan comprende las capas física y de enlace de datos del modelo de referencia OSI (*Open System Interconnection*).
- Buses de alta velocidad y funcionalidad media: estos integran una capa de enlace de datos para la transmisión eficiente de bloques de datos en forma de mensajes; estos mensajes otorgan mayor funcionalidad a los dispositivos conectados al bus, como la capacidad de configuración, calibración o programación del mismo. Estos buses son capaces de controlar dispositivos de mayor complejidad con gran eficiencia y, además, permiten reducir costos en su implementación. Generalmente, añaden la especificación de la capa de aplicación del modelo de referencia OSI, que proporciona las bases para desarrollar software

que permite acceder, cambiar o controlar los dispositivos mediante funciones estandarizadas que faciliten la interoperabilidad entre dispositivos de diferentes fabricantes.

- Buses de altas prestaciones: son capaces de soportar comunicaciones en todos los niveles de producción industrial; se basan en buses de alta velocidad e implementan mejoras para alcanzar los requerimientos necesarios de seguridad y funcionalidad. Cuentan con una capa de aplicación con una gran cantidad de servicios para la capa de usuario. Entre sus principales características se encuentran:
  - Redes multimaestro con redundancia.
  - Comunicación maestro-esclavo bajo el esquema pregunta-respuesta.
  - Recuperación de datos desde el nodo esclavo con un límite máximo de tiempo (*Maximun latency*).
  - Capacidad de direccionamiento unidireccional (*unicast*), multidireccional (*multicast*) y de difusión (*broadcast*).
  - Petición de servicios a los nodos esclavo basada en eventos.
  - Comunicación de variables y bloques de datos orientada a objetos.
  - Descarga y ejecución remota de programas.
  - Altos niveles de seguridad de la red, opcionalmente con procedimientos de autentificación.
  - Conjunto completo de funciones de administración de la red.

2.2. Buses de Campo en Automoción

Con el paso de los años se estandarizó una gran cantidad de buses de campo, además de enfocarse en sectores específicos dentro de la industria, y la industria automotriz no fue la excepción. Esta estandarización se basó en el establecimiento de reglas normalizadas con el fin de presentar, señalizar, autenticar y detectar los errores en la transmisión de la información sobre el bus

de campo. Como resultado se obtuvieron los protocolos de comunicación automotriz, conocidos inicialmente como *autobus* (Carpio Guartambel, 2013).

La Sociedad de Ingenieros Automotrices (SAE, *Society of Automotive Engineers*) introdujo tres categorías básicas para los buses dependiendo de sus funcionalidades y velocidades de transmisión de datos, estas son las clases A, B y C (Jiménez García, 2008).

Las Clases A y B consideran aplicaciones que no son críticas para la seguridad (*Non Safety Critical*), por lo cual no demandan respuestas en tiempo real, ni máximas latencias en dispositivos que no requieren un gran ancho de banda, ya que comúnmente sólo transmiten pequeños mensajes de información. La Clase A considera velocidades de transferencia de datos por debajo de 10 kbps, mientras que la Clase B considera velocidades entre 10 kbps y 125 kbps.

La Clase C engloba todas aquellas aplicaciones de seguridad crítica (*Safety Critical*) que demandan respuestas en tiempo real, tolerancia a fallos, así como bajos tiempos de latencia, alta predictibilidad, velocidades de transferencia mayores a 125 kbps y detección de errores.

2.3. Protocolo de Comunicaciones CAN

En febrero de 1986, la firma alemana Robert Bosch introdujo el sistema de bus serial llamado CAN (*Controller Area Network*) en la SAE, como una solución para la comunicación en aplicaciones automotrices y debido a sus beneficios ganó una gran aceptación en aplicaciones industriales (CiA, 2018).

CAN es un protocolo serial abierto que implementa tres capas del modelo de referencia OSI y utiliza topología de bus para conectar los distintos elementos o nodos (véase la Figura 2.1); CAN define las capas de nivel físico y de nivel de enlace de datos, mientras que deja abierta la capa de nivel de aplicación para el desarrollo del programador según sus requerimientos y necesidades (Vidal I., Zúñiga G. y Rojas A., 2019).

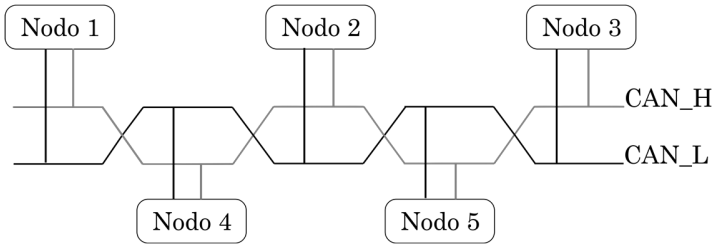


Figura 2.1. Topología del bus CAN.

2.3.1. Características principales del protocolo CAN

El nivel físico está basado en el estándar para aplicaciones ISO11898, que contiene normas específicas para diversos tipos de funcionamientos. El medio de transmisión consta de una línea de dos cables (par trenzado) y un terminador con impedancia de 120 Ω para interconectar todos los elementos de la red, los cuales se componen por (Martínez Requena, 2017):

- Controlador: gestiona la estructuración de la trama (*frame*), realiza la comprobación de errores y la detección de colisiones.
- Transceptor: integra al transmisor y al receptor, se encarga de la codificación y decodificación de los mensajes en el bus, la sincronización, control del nivel en la señal y el acceso al medio.

Tanto el controlador como el transceptor son módulos independientes de los nodos, lo que permite evitar que estos tengan que destinar recursos para llevar a cabo las tareas de los módulos. La velocidad máxima en el bus depende de su longitud como muestra la Tabla 2.1.

Tabla 2.1. Velocidad de transmisión del bus CAN con respecto a la distancia (Martínez Requena, 2017).

Longitud del Bus	**Velocidad máxima**	**Tiempo máximo de transmisión***
Hasta 25 m.	1 Mbps	129 µs
Hasta 100 m.	500 kbps	258 µs
Hasta 500 m.	125 kbps	1 032 ms
Hasta 1 000 m.	50 kbps	2 580 ms

*mensajes de 129 bits de longitud.

Las señales de los cables se denominan como CAN_H (*CAN high*) y CAN_L (*CAN low*). Dependiendo del voltaje diferencial entre CAN_H y CAN_L, el bus puede operar en:

- Modo recesivo: si la diferencia de voltaje entre CAN_L y CAN_H es menor a 1.5 V o igual a cero, toma un valor lógico "1".
- Modo dominante: si la diferencia de voltaje entre CAN_L y CAN_H es por lo menos de 1.5 V, toma un valor lógico "0".

Este modo de comunicación brinda una mayor protección a interferencias electromagnéticas, ya que la lectura de los bits depende de la diferencia entre las señales, por lo que, si se sometieran a interferencias electromagnéticas, ambas señales tendrán la misma variación, manteniendo constante la diferencia entre ellas (Chamú Morales, 2005).

2.3.2. Control de acceso al medio

Otra característica de CAN es el arbitraje, el cual controla el acceso al medio por parte de los nodos para evitar posibles colisiones en la comunicación. Antes de transmitir, los nodos vigilan que en el bus no exista actividad durante un período, tras esto, se realiza la transmisión. Al inicio de la trama se encuentra el campo de arbitraje, el cual coincide con el identificador propio del nodo, que se transmite con bits dominantes y recesivos. Cuando dos nodos tratan de transmitir simultáneamente, los bits dominantes prevalecen sobre los recesivos, lo que permite a los nodos detectar la colisión basándose en el número del identificador que prevalezca (número de identificación menor). Al detectar la colisión, el nodo con menor prioridad (número de identificador mayor) deja de transmitir inmediatamente y espera hasta que se finalice la actividad en el bus para reintentarlo; cuando el bus se encuentra sin actividad se mantiene constante un nivel recesivo en el bus (Martínez Requena, 2017).

2.3.3. Formato de tramas CAN

Cuando el bus se encuentra en modo reposo, no existe intercambio de tramas en el medio físico. El protocolo CAN contempla formatos diferentes para las tramas con base en un formato (véase la Figura 2.2) y un propósito específico para cada uno; los más importantes son (Martínez Requena, 2017) y (Paret, 2007):

- Trama de datos: se utiliza para la transmisión de la información entre los nodos de la red.
- Trama de error: un nodo hace uso de ésta en caso de la detección de algún error en los mensajes transmitidos por otros nodos al corromper el formato base de la trama CAN.
- Trama de petición remota: para realizar la solicitud de envío de información a otro nodo, se envía el identificador del nodo que recibe la petición.
- Trama de sobrecarga: un nodo la envía en caso de encontrarse sobrecargado para que el bus incluya un retraso entre tramas, y con ello se logre un adecuado control de flujo.

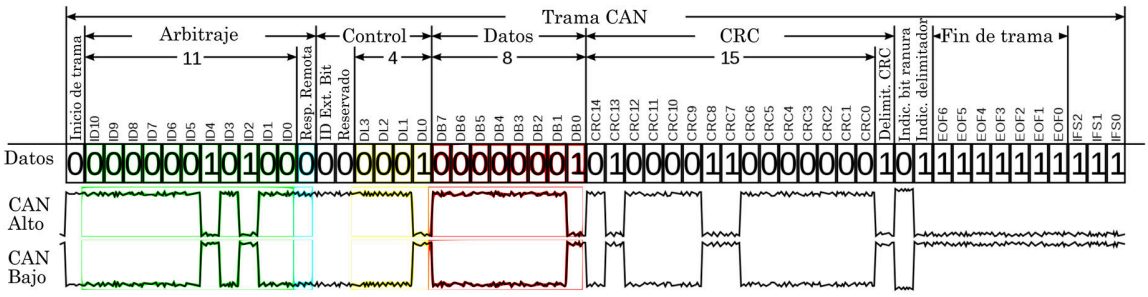


Figura 2.2. Formato base de la trama CAN (Basada en https://es.wikipedia.org/wiki/Bus_CAN).

2.3.4. Estados de los nodos

En caso de encontrarse un nodo con errores dentro de la red, el protocolo cuenta con medidas para desconectarlo y evitar que condicione el funcionamiento pudiendo ocasionar fallos o saturación en dicha red. De acuerdo al estado en el

que se encuentre el respectivo nodo, puede estar en los siguientes estados (véase la Figura 2.3):

- Error activo: estado del nodo en que puede enviar mensajes y tramas de error normalmente.
- Error pasivo: el nodo envía tramas modificadas de error enviando 12 bits recesivos, de modo que los demás nodos sean incapaces de detectarlas evitando las comunicaciones.
- Bus *off* o anulado: el nodo se auto desconecta del bus al deshabilitar su transceptor. Cuando un nodo se desconecta de la red, se reinicia, reconfigura y reconecta de nuevo, pero no será capaz de transmitir hasta recibir 128 secuencias de 11 bits recesivos sin errores en el bus, regresando al estado de error activo.

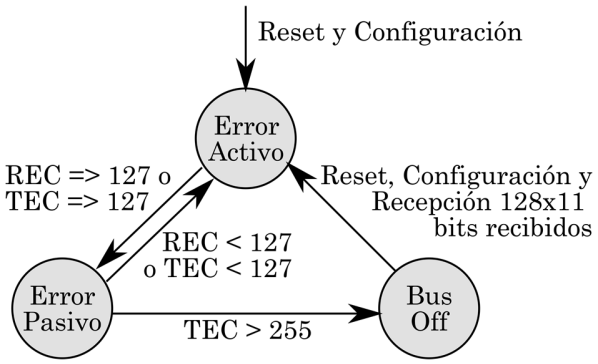


Figura 2.3. Estados de error de nodo CAN (Martínez Requena, 2017).

2.3.5. Limitaciones de CAN

El concepto de CAN fue utilizado desde los años 80, resultando perfectamente útil para muchas aplicaciones hasta la actualidad, sin embargo, con el paso del tiempo se observan las siguientes limitaciones del protocolo CAN (Paret, 2012):

- Limitación en la velocidad (*bit rate*): la máxima velocidad de transferencia está limitada a 1 Mbps, las nuevas y futuras aplicaciones para redes multiplexadas requieren velocidades mayores, en el orden de 5 a 10 Mbps, ya sea por la saturación de la capacidad de

comunicación o por requerimientos en tiempo, estos son los límites de la filosofía en la cual todavía es posible no considerar fenómenos como el efecto del retraso de propagación y coeficiente de reflexión, entre otros. Mejorar la velocidad es imposible si en el diseño del protocolo no se toman en cuenta los parámetros físicos y sus efectos.

- Limitaciones en distancia y flexibilidad de topologías: la máxima velocidad de 1 Mbps de CAN está relacionada con la estructura del bit de reconocimiento del protocolo, de hecho, para que el protocolo funcione correctamente, es necesario asegurar que la suma de los tiempos de entrada y de salida de la señal permitan que la señal de reconocimiento se encuentre dentro de la duración del bit. Esta característica especial del protocolo impone un límite en el tiempo de propagación y, por lo tanto, limita la máxima distancia entre dos nodos presentes en la red si se excluyen topologías que posibiliten propagaciones asimétricas.
- Limitación de la posibilidad de redundancia topológica: este punto está ligado con los dos anteriores, ya que es difícil la creación de una red que posibilite la redundancia en la comunicación al nivel de la capa física de acuerdo con una arquitectura topológica CAN.
- Limitación de acceso al medio en tiempo real: CAN está orientado a eventos, por lo que la comunicación en la red es iniciada de forma esporádica o aleatoria. Además, CAN carece de una orientación al tiempo real, ya que esta filosofía se basa en iniciar la comunicación en función a un reloj, fecha o instante fijo.

2.4. Aspectos de Activación por Eventos y Activación por Tiempo

2.4.1. El lado probabilístico de CAN

El diseño y la estructura de CAN fomentan el inicio de la transmisión de datos cuando ocurre un evento en algún nodo de la red, a esto se le conoce como

activación por evento (*Event-Triggered*); a menudo el participante envía el mensaje después de la acción o respuesta a la solicitud de información de acuerdo a los requisitos de la aplicación o tarea asignada (Paret, 2012).

En CAN los mensajes son priorizados en el sistema por el diseñador del programa, utilizando valores que él mismo asigna a cada identificador; debido a este principio, en un instante dado, ningún nodo puede estar seguro que su mensaje será transmitido de inmediato debido al proceso de arbitraje de conflictos entre los nodos competidores en ese instante. Este concepto y su administración en la red CAN brinda un fuerte énfasis "probabilístico", ya que está sujeto al proceso de arbitraje, que hace que los tiempos de transmisión y de latencia asociados dependan en gran medida del valor de los identificadores asignados. El único mensaje en el que el tiempo de latencia es estrictamente conocido y por lo que se puede considerar "determinista" es *hex 0000*, que es el de máxima prioridad; para otros mensajes esto depende en gran medida de la programación, lo que dificulta el cálculo de la probabilidad en el modelo de actividad de la red, además de que la probabilidad de arbitraje es excesivamente alta. Lo anterior es un problema cuando se busca una comunicación de transmisión o recepción, definida, precisa y en un instante determinado, en donde el tiempo sea determinista. Otro problema consiste en la probabilidad de que un nodo siempre pierda en el arbitraje haciendo que su mensaje nunca sea transmitido (Paret, 2012).

2.4.2. El lado determinista de las aplicaciones

En muchas aplicaciones es o puede ser necesario activar ciertas acciones en instantes precisos, estos sistemas son denominados de activación por tiempo (*time-triggered*) o sistemas que funcionan en modo tiempo real. Cuando un sistema funciona en tiempo real, es decir, cuando se desea asegurar que la transmisión o recepción debe realizarse en un instante o franja de tiempo específicos, agregando así un aspecto determinista a la comunicación (Paret, 2012).

2.5. Protocolo TTCAN

A principio de los años 90s, el protocolo CAN dominaba el mercado, pero el incremento de la complejidad en los sistemas embebidos empezó a demandar protocolos que garantizaran respuestas en tiempo real, fueran determinísticos y de alta seguridad, en consecuencia, estos sistemas demandaron la utilización de un reloj global. El primer protocolo en atender dichos requerimientos fue TTCAN (*time-triggered communication on CAN*), el cual consiste en una capa de aplicación implementada por la compañía Robert Bosch, la cual se encarga de iniciar la comunicación en CAN utilizando eventos por tiempo (Paret, 2012).

El objetivo de TTCAN es definir y garantizar los tiempos de latencia de cada mensaje con un valor que es independiente de la carga en la red CAN, y se puede implementar en dos niveles:

- Nivel 1: Está destinado únicamente para transferencias cíclicas de mensajes
- Nivel 2: Soporta un sistema denominado "Tiempo Global".

TTCAN está basado en intercambios con tiempos determinados, utilizando ventanas de tiempo preestablecidas en el ciclo de operación (Paret, 2012). TTCAN utiliza mensajes de referencia para indicar el inicio del ciclo de operación, este ciclo se divide en diferentes tipos de ventanas de tiempo (Coronel, y otros, 2006):

- Ventanas exclusivas: utilizadas para trasmitir un mensaje específico.
- Ventanas arbitradas: en donde los nodos compiten por el acceso al bus como en una comunicación CAN normal.
- Ventanas libres: reservan espacio libre para cualquier movimiento.

En TTCAN las ventanas de tiempo se organizan en ciclos básicos con tiempos iguales (filas X) y en numerosas zonas de tiempo durante las cuales se permite la transmisión (columnas Y), por lo que el patrón completo de los

mensajes transmitidos en la red CAN se encuentran organizados en una matriz de X por Y como se muestra en la Figura 2.4.

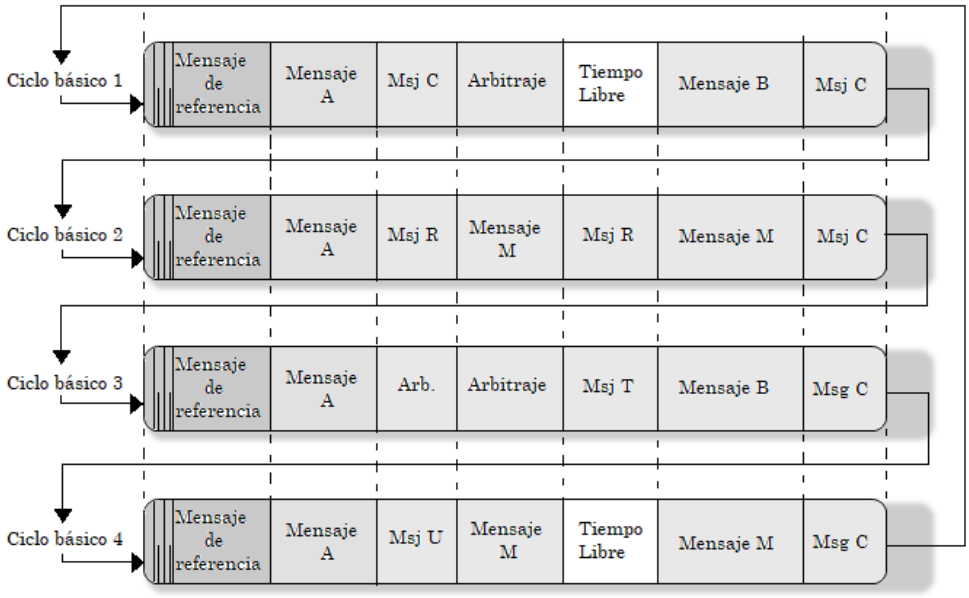


Figura 2.4. Matriz del principio de funcionamiento del protocolo TTCAN (Paret, 2012).

En principio, TTCAN asume que existe un nodo en la red que se encarga de dividir y asignar los tiempos para cada nodo cuando considere necesario, por lo que se obtiene un sistema determinístico ya que cada nodo cuenta con un tiempo determinado y conocido, pero esto sólo puede ser un sistema "casi" en tiempo real, en realidad las restricciones de los sistemas y la frecuente necesidad de reconfigurar la secuencia de tiempos debido a eventos externos previstos o no, hacen que sean bastante diferentes la teoría y la aplicación (Paret, 2012).

2.6. Sistemas Embebidos

Los sistemas embebidos son circuitos electrónicos computarizados mediante microcontroladores que almacenan y ejecutan secuencias y algoritmos diseñados para solucionar un problema específico. Actualmente se encuentran dispersos en todos los aspectos posibles de la vida cotidiana, por ejemplo, en los vehículos es común encontrarlos en los sistemas de frenado ABS, inyección de combustible y protección contra impactos (*Airbag*), entre otros; y el número de aplicaciones soportadas por los sistemas embebidos crece cada vez más.

Dependiendo de sus derivados y su aplicación, los microcontroladores incluyen varios periféricos internos para facilitar la integración de un sistema permitiendo que la aplicación final sea compacta, pequeña y optimizada para el ahorro de energía y reducción de costos. Dentro de los más comunes se encuentran las terminales de entrada/salida, convertidor analógico a digital (ADC, *Analogic Digital Converter*), perro guardián (*Watchdog*), detector de bajo nivel de voltaje, temporizador, modulación de ancho de pulso (PWM, *Pulse Width Modulation*) y comunicación serial síncrona y asíncrona (Galeano, 2009).

2.6.1. Interrupciones en los sistemas embebidos

En los sistemas embebidos, las interrupciones consisten en un evento que notifica al CPU sobre la aparición de una situación específica en alguno de sus periféricos, permitiéndole pausar la ejecución normal de sus funciones para responder de acuerdo a una rutina establecida al evento en cuestión (a esto se le conoce como cambio de contexto) y posteriormente reanudar sus funciones normales.

Utilizar interrupciones es una forma de mejorar la rapidez de la respuesta del sistema; al tiempo que tarda en responder se le denomina latencia de interrupción y este depende de los siguientes factores: a) máximo tiempo de deshabilitación de las interrupciones, b) tiempo de ejecución de la rutina de interrupción y c) tiempo que el procesador tarda en realizar el cambio de contexto, de estos sólo es posible controlar los dos primeros factores, ya que el tercero depende del fabricante del microprocesador.

Durante la ejecución del programa principal, existen zonas en que es necesario deshabilitar las interrupciones, ya que se ejecutará un proceso crítico que no puede ser interrumpido, ya sea de datos o de hardware; a esta sección de código se le denomina “zona crítica de software”, una vez terminada la ejecución del código crítico se deben habilitar nuevamente las interrupciones (Galeano, 2009).

2.6.2. Sistema operativo de tiempo real

Un sistema operativo de tiempo real (RTOS, *Real Time Operating System*) es un programa que coordina el funcionamiento de otros programas como tareas, permitiendo el manejo de datos y recursos de manera óptima. Los RTOS son capaces de adaptarse a los requerimientos de la aplicación y es posible reconfigurarlos o cambiarlos por completo para llevar a cabo las tareas necesarias. El cambio de contexto es la base de los RTOS, que consiste en realizar diferentes tareas de forma periódica y en un tiempo determinado como muestra la Figura 2.5.

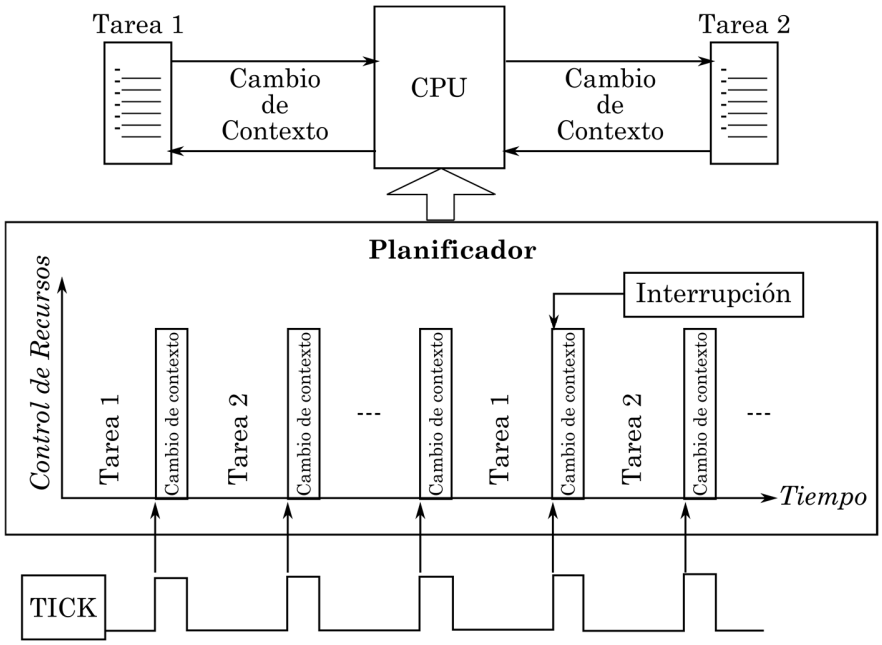


Figura 2.5. Cambio de contexto (Galeano, 2009).

El método más común para coordinar la ejecución de las tareas es incluir en el sistema un componente llamado planificador (*scheduler)*, que se encarga de asignar pequeños tiempos de ejecución a cada tarea y suspender su ejecución para pasar el control de los recursos a la siguiente tarea en la lista, logrando así que todas las tareas se ejecuten. El tiempo y el cambio de tarea (cambio de contexto) son gobernados por un reloj externo al RTOS llamado TICK, que es una base de tiempo generada a partir de una interrupción periódica en el microcontrolador (Galeano, 2009).

Capítulo 3. Sistema de Protección Contra Impactos Airbag

El sistema de seguridad *airbag,* es un que complementa el uso del cinturón de seguridad ofreciendo una mejor protección en caso de accidente, este se basa en una bolsa de aire que al detectar una colisión se infla para amortiguar el impacto del ocupante. Existen límites de funcionamiento para evitar disparos accidentales del *airbag*, como circular a velocidades superiores a 28 km/h y la dirección de impacto debe encontrarse dentro de un rango específico, además de integrar sensores de seguridad en serie con los sensores de accidente (Murcia Educarm, 2019).

El sistema *airbag* se compone principalmente por la ECU y sensores, múltiples módulos que contienen su bolsa de aire con su respectivo generador de gas y una espira de metal en el volante que conecta ambos módulos.

3.1.1. Unidad de control electrónico del airbag

La ECU es el núcleo del sistema *airbag* y se encarga de controlar su funcionamiento considerando las señales recibidas de los respectivos sensores; normalmente se ubica en el centro del vehículo y cumple con las siguientes funciones (HELLA S.A., 2019):

- Detección de accidentes.
- Detección a tiempo de las señales emitidas por los sensores.
- Activación a tiempo de los circuitos de encendido necesarios.
- Autodiagnóstico de todo el sistema.
- Registro de los fallos surgidos en la memoria de averías.
- Encendido del testigo luminoso de control del *airbag* si falla el sistema.
- Comunicación con las otras ECU.

En las nuevas unidades de control se almacena información que se ha obtenido gracias a diversas simulaciones de accidentes, lo que permite clasificar el accidente por grado de gravedad en:

- Gravedad 0: accidente leve, no se ha accionado ningún *airbag*.
- Gravedad 1: accidente de gravedad media, es posible que se hayan activado los *airbags* en una primera fase.
- Gravedad 2: accidente grave, se han accionado los *airbags* en la primera fase.
- Gravedad 3: accidente muy grave, se han accionado los *airbags* en la primera y en la segunda fase

Además del grado de gravedad, para la estrategia de activación del sistema se considera otro tipo de información como el sentido de la marcha, potencia de impacto y el tipo de accidente; también tiene en cuenta si los ocupantes llevaban puesto el cinturón de seguridad o no.

3.1.1.1. Sensores de accidente

Dependiendo del sistema *airbag* y del número de bolsas disponibles, los sensores de accidente y de aceleración se encuentran conectados directamente en la ECU, o también pueden ubicarse de forma satelital en el frente y laterales del vehículo (Murcia Educarm, 2019).

Los sensores frontales siempre se colocan en pares, generalmente son sensores que trabajan conforme el sistema masa-resorte, en los cuales, si se aplica la fuerza suficiente en una dirección determinada, el circuito se cierra permitiendo el flujo de corriente hacia la unidad de control. Para los laterales se emplean sensores de presión, estos se montan en las puertas y reaccionan ante un cambio de presión dentro de ellas en caso de accidente.

Los sensores de desaceleración emplean una masa de silicio que se moverá al recibir la fuerza y dependiendo de la posición de la masa modificará su la

lectura eléctrica, proporcionando así la información sobre la magnitud de la desaceleración a la ECU (HELLA S.A., 2019).

3.1.1.2. Sensores de seguridad

Por motivos de seguridad y para evitar una activación involuntaria del sistema, siempre deberán enviarse las señales de dos sensores que trabajen de manera independiente.

El sensor de seguridad tiene la función de evitar que el *airbag* se active involuntariamente siendo este un interruptor electromecánico. El sensor *Safing* está conectado en serie con los sensores frontales, este se compone por un contacto Reed abierto dentro de un tubo lleno de resina sobre el que se encuentra un imán con forma de anillo que va sujeto al extremo de la carcasa por medio de un muelle; si se aplica fuerza, el imán se desliza en contra de la fuerza del muelle a través del tubo lleno de resina y cierra el contacto Reed del circuito que activa el *airbag* (Murcia Educarm, 2019).

3.1.2. Estructura del *airbag*

El *airbag* del volante está compuesto por una bolsa de aire, de un soporte para la bolsa, de un generador situado sobre el soporte del generador y de una cubierta del *airbag* (cubierta del volante). En caso de accidente, la ECU encenderá el generador que calienta un fino alambre mediante una corriente de encendido, este alambre enciende el cebador que produce la combustión de la carga propulsora, el gas se expande y llena la bolsa de aire aproximadamente en 30 ms (Murcia Educarm, 2019).

En los *airbags* laterales, llamados *Thorax bags*, el proceso es muy parecido, aunque debido a la deformación que sufren las bolsas, se necesita un encendido mucho más rápido de los generadores del gas para llenar las bolsas de aire tras 22 ms.

3.1.2.1. Sistema de detección de ocupación de los asientos

Para controlar mejor el uso de los *airbags* y para evitar que se activen innecesariamente, existen sensores de detección de ocupación de los asientos (SBE, *Seat Occupancy Sensor*). Usualmente se utilizan unas alfombrillas que actúan como sensores de peso y que están formadas por sensores de presión. También existe la opción de utilizar sensores infrarrojos y de ultrasonido que se montan en la zona de las luces interiores o del espejo retrovisor, y se encargan de supervisar tanto la ocupación de los asientos como la posición del acompañante. La información sobre la ocupación de los asientos tiene una influencia directa sobre la activación de los *airbags*, la activación de los pretensores del cinturón y también sobre los reposacabezas activos. Si algún asiento no está ocupado, el sistema *airbag* lo detecta y, en caso de accidente, no activa los correspondientes sistemas de protección (Murcia Educarm, 2019).

3.1.2.2. Cableado del airbag

Para una mejor identificación de los cables y de los enchufes del *airbag*, estos son de un color amarillo muy llamativo. Dentro de los enchufes se encuentra un puente para cortocircuitos que evita que el *airbag* se active de manera involuntaria cuando haya que realizar alguna tarea de mantenimiento. Esto podría ocurrir, por ejemplo, si existiera carga estática. El puente para cortocircuitos es un contacto que, al separar la conexión del enchufe, une los dos contactos dentro del enchufe (HELLA S.A., 2019).

3.1.2.3. Pretensor del cinturón

El pretensor del cinturón tiene la función de evitar que el cinturón esté "muy flojo" en caso de accidente. Esta situación puede producirse por llevar ropa muy amplia o por adoptar una posición incorrecta al estar sentado. Este puede ir integrado en el cierre o en la bobina del cinturón. Si el pretensor del cinturón va integrado en el cierre, en caso de accidente un generador de gas, como el del *airbag*, desliza un pistón sobre un tubo rígido y un cordón, que une el pistón con

el cierre del cinturón, se desplaza hacia abajo tensando el cinturón; si el pretensor del cinturón va integrado en la bobina del cinturón, se tensa el cinturón por medio de un sistema mecánico especial para enrollarlo, por ejemplo, utilizando un motor que acciona un pistón circular que gracias a su movimiento giratorio tensa el cinturón (HELLA S.A., 2019).

3.1.2.3.1. Delimitador de potencia del cinturón

En caso de accidente, para minimizar la carga en el pecho se integra un delimitador de potencia en el tensor del cinturón, este es un sistema automático y adaptativo que mediante un generador de gas, produce una conmutación entre un nivel de fuerza muy elevado y otro muy bajo que junto al óptimo ajuste entre el pretensor del cinturón y el *airbag*, hace que la energía cinética de los ocupantes del vehículo se reparta lentamente durante la duración del accidente, esto con la finalidad de reducir la carga sobre los cuerpos.

3.1.2.4. Desconexión de la batería

El peligro que supone la generación de fuego causado por un cortocircuito requiere la desconexión de la batería en red de a bordo; para ello, la ECU del *airbag* envía una señal a un generador de gas para la desconexión, este sistema de denomina BST (*Battery Safety Terminal*) (BMW of North America, Inc., 2001).

3.1.2.5. Apagado de bomba de combustible

El sistema *airbag* se encuentra conectado al módulo de control del motor que le indicará apagar la bomba de combustible en caso de que se produzca un choque.

3.1.3. Sistema de restricción múltiple III

El sistema de restricción múltiple (MRS, *Multiple Restriction System*) III (véase la Figura 3.1) es un sistema de seguridad de equipamiento estándar en los modelos Sedan E46 de BMW, este utiliza tecnología inteligente, la cual se refiere a la programación del módulo de la ECU que permite el despliegue del

sistema para generar un efecto de amortiguación por etapas con base en la gravedad del impacto, además de la desactivación de la bomba de combustible y de la desconexión eléctrica de la red de abordo, entre otros (BMW of North America, Inc., 2001).

El MRS integra dos sensores de desaceleración que en conjunto con los sensores satelitales montados debajo de los asientos delanteros y en el marco del asiento, tienen la función de detectar la gravedad de los impactos laterales y enviar una señal al módulo MRS III, utilizando una señal modulada por pulsos al ocurrir un choque. El módulo MRS III realiza una autocomprobación del sistema cada vez que se enciende (esto incluye los sensores satelitales, de aceleración y SBE). Cualquier falla detectada en el sistema hará que la luz de advertencia del testigo luminoso permanezca iluminada después de arrancar el motor en el tablero de instrumentos.

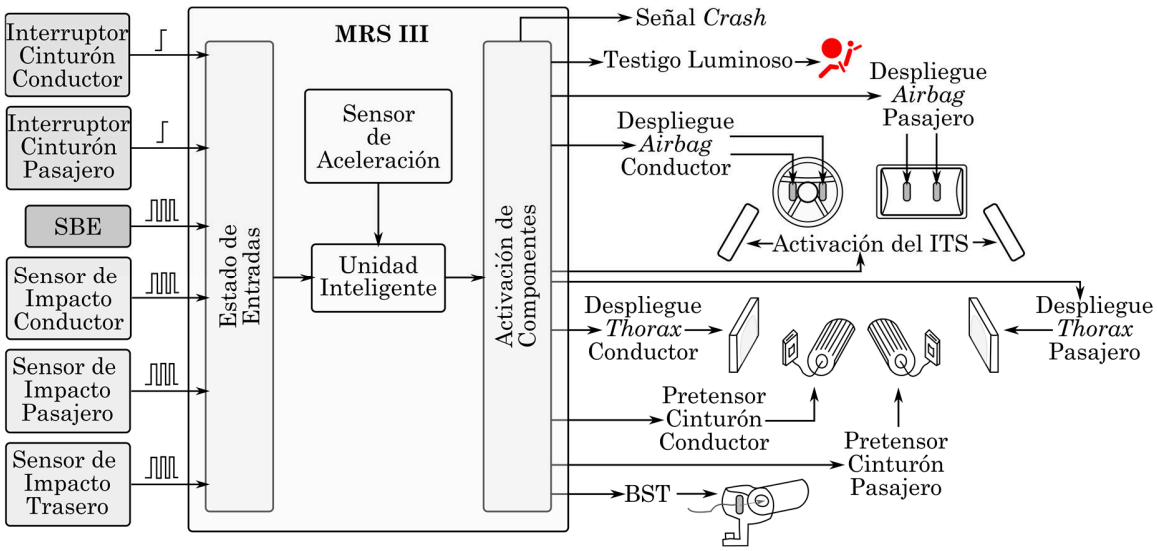


Figura 3.1. Esquema del sistema de restricción múltiple III (BMW of North America, Inc., 2001).

3.1.3.1. Umbrales de activación

El módulo de control utiliza las señales de los sensores satelitales junto con su señal del sensor de impacto interno para determinar el despliegue de los *airbags*. Existen diferentes umbrales y restricciones de seguridad para el despliegue de los *airbags* (BMW of North America, Inc., 2001):

- Umbral de pretensor del cinturón: para la activación de los tensores del cinturón de seguridad.
- Umbral de *airbag* 1: primera etapa de activación para los *airbags* frontales de dos etapas, desplegando la primera etapa al alcanzar el umbral de activación frontal.
- Umbral de airbag 2: segunda etapa de activación para los *airbags* frontales de dos etapas; se puede desplegar simultáneamente o después de un retraso, dependiendo de la gravedad del impacto.
- Umbral de choque trasero: para la activación de los pretensores del cinturón de seguridad con un impacto trasero.
- Umbral del BST: para la activación del BST con despliegue del *airbag*.
- *Airbag* lateral / umbral ITS: para el despliegue de los *airbags* laterales y de tórax.

Los umbrales de activación son independientes de los tensores de cinturón. El MRS III incluye cuatro umbrales de activación para los *airbags* frontales de dos etapas, ya que la activación de los *airbags* delanteros también depende de si los cinturones de seguridad están conectados y si el asiento del pasajero delantero está ocupado, como se muestra en la Tabla 3.1.

Tabla 3.1. Umbrales de activación de airbag de dos etapas.

Umbral	**Sin cinturón de seguridad**	**Con cinturón de seguridad**
1	Ignición de carga 1	Sin activación
2	Ignición de cargas 1 y 2 con retraso	Ignición de carga 1
3	Ignición de cargas 1 y 2 con retraso	
4	Ignición de cargas 1 y 2, simultáneamente	

Si la señal del SBE es defectuosa al disparar, el MRS III se desplegará como si el asiento estuviera ocupado; si la señal de los contactos del cinturón de seguridad es defectuosa, el MRS III se desplegará como si los cinturones no estuvieran abrochados.

El BST se desplegará con un impacto frontal que alcance el umbral 2 o superior. El umbral para la activación de BST con un impacto lateral se programa por separado en los criterios de despliegue lateral, también se desplegará cuando se exceda el umbral de impacto trasero (BMW of North America, Inc., 2001).

Capítulo 4. Arquitectura de Protocolos Flexray

FlexRay es un sistema de comunicaciones diseñado para soportar las necesidades que han surgido para las nuevas aplicaciones en el ámbito de la automoción, ya que provee flexibilidad y determinismo combinando un segmento estático de ranuras basado en el esquema de acceso múltiple por división en el tiempo (TDMA, *Time Division Multiple Access*) de capacidad escalable y un segmento dinámico orientado a eventos basado en acceso múltiple por división flexible en el tiempo (FTDMA, *Flexible Time Division Multiple Access*). Dentro de sus características más importantes se encuentran (FlexRay Consortium, 2010) y (Jiménez García, 2008):

- Sistema escalable y redundante al soportar dos canales de comunicación.
- Velocidad de transmisión de datos de hasta de 10 Mbps (20 Mbps en caso de utilizar ambos canales sin redundancia).
- Garantiza la latencia de los mensajes.
- Sistema tolerante a fallos.
- Sincronización de nodos mediante reloj global.
- Acceso al bus libre de colisiones.
- Orientado a mensajes, direccionando mediante identificadores.
- Soporte para diferentes topologías de red.

4.1. Revisión Histórica del Protocolo FlexRay

Desde la primera publicación de las especificaciones del protocolo FlexRay en el año 2004 con la versión 2.0, se han realizado cambios llegando actualmente a la versión 3.0.1 como se muestra en la Tabla 4.1 (FlexRay Consortium, 2010).

Tabla 4.1. Revisión histórica FlexRay (FlexRay Consortium, 2010).

Versión y Fecha	**Modificaciones**
2.0 06/2004	Lanzamiento de la primera publicación.
2.1 05/2005	Restructuración de los procesos de codificación SDL. Reescritura del apéndice B. Eliminación del antiguo capítulo 10 y referencias BG. Especificación de cambios significativos al CHI: • Dependencias de canal *pMicroInitialOffset[Ch]* y *pMacroInitialOffset[Ch]* ha sido introducidas en 9.3.1.1.2 • Se añadió *pDecodingCorrection* en 9.3.1.1.2 • Se añadió identificador de error en CHI en 9.3.1.1.2 • Se removió *vSyncFramesEven/Odd/A/B* en 9.3.1.3.4 • Se añadió indicador para *zLastDynTxSlot en 9.3.1.3.9* Introducción de variable de estado *vPOC!StartupState* en el canal CHI en 9.3.1.3.1. Se realizaron numerosas correcciones técnicas y aclaraciones a lo largo del documento. Modificación de figuras de acuerdo a la operación del protocolo.
2.1 Rev A 12/2005	Uso de entradas de prioridad para resolución de condiciones de carrera específicas. Reorganización de SDL para eliminar el uso de la estructura SDL *"enabling condition"*. Actualización de *zLastDynTxSlot*. Reorganización de la detección de inactividad. Introducción de nueva clase de variable *"a"* en la sección 1.6.1. Remplazo de contador de bits por un temporizador en la figura 3-23. Exportación explícita de todas las variables CHI. Extensión de la codificación de color a señales SDL. Reestructuración del apéndice B. Se realizaron numerosas correcciones técnicas y aclaraciones a lo largo del documento.
3.0 12/2009	La transmisión continua al final del segmento dinámico no conduce a un error fatal del protocolo. La recepción de una trama nula en *PPIndicator* está marcada como un error de contenido. Definición de comportamiento del pin TxD en caso de abortar la trasmisión. Soporte para 2.5 y 5 Mbits/s. *pKeySlotID* se puede configurar a cero para un nodo que no tenga una clave de ranura (*slot*). TxEN y TxD no pueden ser cambiadas a apagado simultáneamente. La recepción se ignora por un tiempo configurable después de la transmisión. Nuevo símbolo patrón de activación durante operación (WUDOP), que puede ser transmitido para apoyar en la activación durante la operación normal. El mecanismo de detección de inactividad se basa en la duración del tiempo en lugar de la duración del bit. Se agregaron dos nuevos métodos de sincronización, TT-L y TT-E. La multiplexación de ranuras (*slots*) en el segmento estático está permitida para todas las ranuras excepto para las ranuras clave. Reorganización del apéndice B (incluidas, pero no limitado a): • Precisión de fórmulas.

	• Error de distribución de *microtick*. • Eliminación de 100 ns *microtick*. • Retardos internos de controlador son parte del retardo de propagación. • Incremento en los rangos de los parámetros de corrección de reloj externo. • Restricción optimizada para el tamaño de la ranura estática. • Especificación de la desviación de reloj, *gClockDevationMax*. • *pdMaxDrift* fue remplazado por *pRateCorrectionOut*. • Se agregaron restricciones de configuración para los grupos TT-L y TT-E. Numerosas modificaciones al proceso MAC para mejorar la robustez contra el ruido. Introducción de nuevos parámetros CHI, *vDynResyncAttempt[A]* y *DynResyncAttempt[B]*. La configuración por defecto ahora evita transmisión y recepción. Actualizaciones al comportamiento de activación y arranque de dispositivos de un sólo canal cuando el temporizador expira. Tramas recibidas en ranuras de transmisión son marcadas como inválidas. Número de tramas de inicio validos son exportados a CHI. *PPIndicator* se eliminó de la información de estado de la ranura CHI. Se añadió el indicador de transmisión de trama al estado del búfer (*buffer*) de transmisión CHI. Eliminación del temporizador relativo opcional. El segundo temporizador absoluto se convirtió en obligatorio. Las interrupciones adicionales se convirtieron en obligatorias. Al menos un búfer FIFO es obligatorio y se especificaron los criterios de filtrado FIFO. Se añadió a los datos del estado de canal el indicador de estado de la ranura *vSS!TxConflict*. Aclaración de que un búfer de recepción configurado para ambos canales debe almacenar la fuente de los datos relacionados con la trama. Aclaración en el cual POC establece la configuración del búfer de mensajes y la lista de asignación de ranuras de transmisión, ya sea permitida o prohibida. Configuración o reconfiguración de limpieza del búfer, la carga de datos válidos y la actualización de los indicadores de estado de la ranura. Inicialización por defecto de los indicadores. Las nieves de voltaje y umbrales de tiempo se eliminación del capítulo 2. Transición fuera del *POC:ready* a uno de los estado asociados con el *startup* limpia la carga de los datos válidos y la actualización de los indicadores de estado de la ranura en el búfer de recepción. Las notas de aplicación *wakeup* se movieron del capítulo 7 al apéndice C. Se realizaron numerosas correcciones técnicas pequeñas, correcciones no técnicas y aclaraciones a lo largo del documento.
3.0.1 10/2010	Inicialización de los valores de medición de tiempo para la sincronización del reloj en el modo TT-E. Descripción más detallada del comportamiento básico de FIFO. Exportación de tramas a CHI extendido a tramas nulas. Descripción de los conceptos de búfer habilitado y búfer bloqueado en mayor detalle. Restablecimiento de las variables de control *vTransmitMTS_A, vTransmitWUDOP_A, vTransmitMTS_B, vTransmitWUDOP_B, vExternOffsetControl,* y *vExternRateControl* durante o después de la transición a *POC:ready* o *POC:halt*. Los requerimientos para exportar *vStartupPairs* fueron modificados en la dirección del modo TT-E Distinción entre señales internas y externas para TxD, TxEN y RxD fue añadida.

	Excepción añadida para nodo TT-E *colstart* para el manejo de datos y estado de las ranuras; la primera ranura en el primer ciclo después de la transición del nodo TT-E *coldstart* al estado *POC:external startup* del estado *POC:normal active.*

4.2. Conceptos Básicos de FlexRay

Los nodos de FlexRay son entidades lógicas conectadas en la red FlexRay, con la capacidad de recibir o transmitir tramas de datos. Cada nodo FlexRay cuenta con su propia ECU, cuya arquitectura (véase la Figura 4.1) integra cinco subcomponentes (González Salinas, 2008) y (Freescale Semiconductor, Inc., 2006):

- *Host*: es parte de la ECU y en él se ejecuta el software de aplicación; se encuentra separado del núcleo de protocolo FlexRay e interfazado mediante el controlador de interfaz con el *host* (CHI, *Controller Host Interface*), este último no se encuentra definido en las especificaciones del protocolo de comunicaciones FlexRay.
- Controlador de comunicaciones (CC, *Communication Controller*): es el componente electrónico encargado de la implementación de las especificaciones en las comunicaciones sobre FlexRay.
- Guardian de bus (BG, *Bus Gardian*): es el encargado de proteger de interferencias causadas por transmisiones no sincronizadas al esquema del clúster, sobre el bus de comunicaciones.
- Administrador de bus (BD, *Bus Driver*): componente electrónico conformado por un transmisor y un receptor, que interconecta al CC y un bus de comunicaciones FlexRay; se utiliza uno por cada canal.
- Fuente de energía: se encarga de proveer la energía eléctrica requerida por el sistema para su correcto funcionamiento.

Un clúster es un sistema de comunicaciones conformado por múltiples nodos conectados por lo menos a un canal, dentro de los cuales existen dos tipos de nodos FlexRay, los nodos normales y los nodos de arranque en frío (*coldstart node*), estos últimos son capaces de iniciar la comunicación en el clúster mediante

el envío de tramas de inicio de carga (*startup frames*) (FlexRay Consortium, 2010).

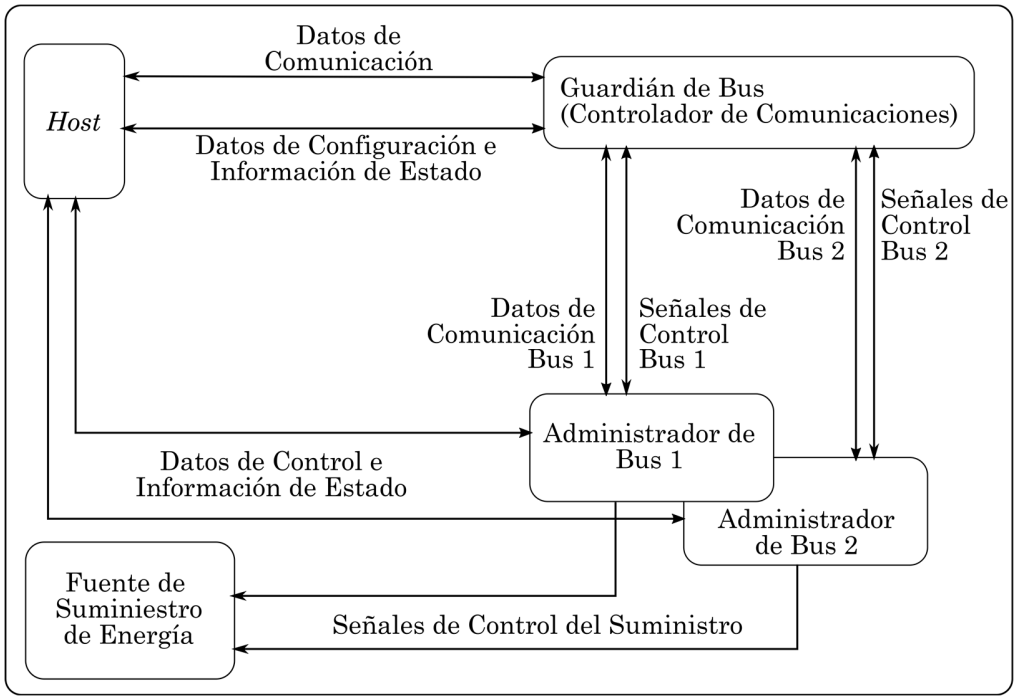


Figura 4.1. Arquitectura de nodo FlexRay (FlexRay Consortium, 2010).

4.3. Capa Física FlexRay

La capa física del protocolo de comunicaciones FlexRay define la manera en la cual se transmiten las señales entre los nodos y el bus, considerando los aspectos físicos como son nivel eléctrico de la señal, su representación (codificación) y sincronización, tiempos de bit e implementación de las topologías, entre otros (FlexRay Consortium, 2010).

Para el diseño de redes FlexRay es importante considerar que los nodos dentro de un clúster deben configurarse a la misma velocidad de transferencia de datos (sean 2.5, 5 o 10 Mbps), la cual puede variar entre distintos clústeres, además de que la topología utilizada es independiente del diseño.

4.3.1. Representación y sincronización de los bits

FlexRay utiliza el método de codificación y decodificación de no retorno a cero (NRZ, *Non Return to Zero*) para la cadena de bits denominada CE

(*Communication Element*) (véase Figura 4.2). Dado que el protocolo es independiente de la capa física fundamental no requiere de implementaciones adicionales (FlexRay Consortium, 2010).

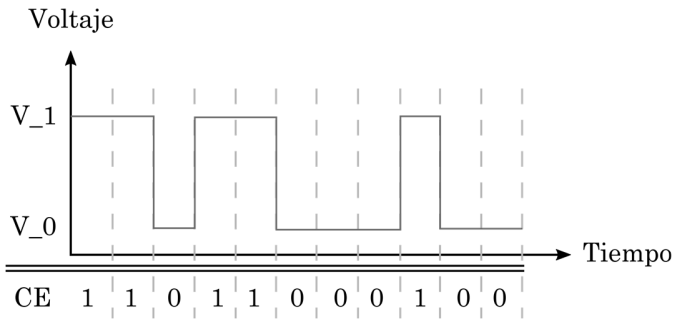


Figura 4.2. Codificación NRZ (González Salinas, 2008).

El CC se encarga de la sincronización de los bits en cada flanco de caída del byte de inicio de secuencia (BSS, *Byte Start Sequence*), como se ve en la Figura 4.3, insertando un BSS 1 (valor lógico 1) seguido de un BSS 0 (valor lógico 0). El nodo toma muestras por cada bit de duración nominal (gdBit), que depende de la velocidad de transferencia de datos y puede ser de 100, 200 o 400 ns, el tiempo de muestreo nominal está dado por (González Salinas, 2008):

$$T_{nom} = gdBit/8$$

El punto de muestreo en el que se toma el valor del bit se denomina *BitStrobing,* cuyo número de muestra es cinco. Para eliminar posibles rebotes y evitar que el valor sea tomado antes o después de tiempo, se permite una tolerancia de 3 T_{nom} para adelanto y de 4 T_{nom} para retardo (FlexRay Consortium, 2010).

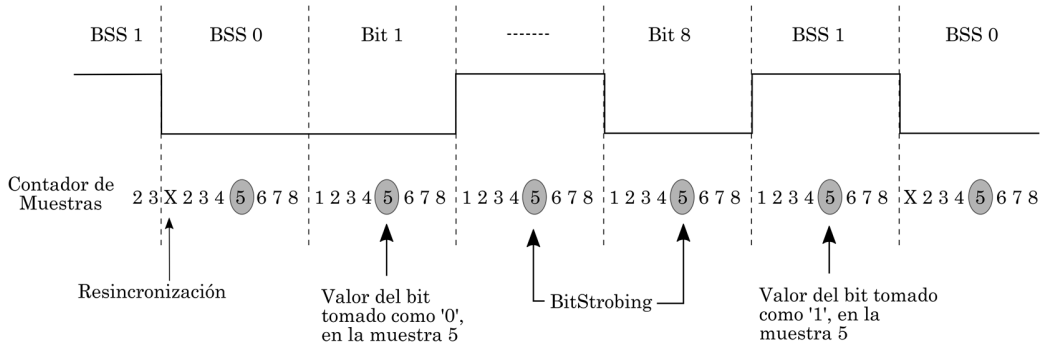


Figura 4.3. Sincronización, muestreo y BitStrobing (FlexRay Consortium, 2010).

4.3.2. Especificaciones eléctricas de la señal

Al estar diseñado para operar a velocidades de 10 Mbps, el gdBit mínimo es de 100 ns, por lo que el bus debe soportar las características de cableado y conectores para el bus que lista la Tabla 4.2. El nivel mínimo de voltaje diferencial para el transmisor es de 600 mV y para el receptor es de 400 mV. (FlexRay Consortium, 2010) y (González Salinas, 2008).

Tabla 4.2. Características del cableado y conectores para el bus.

Parámetro	Descripción	Min	Máx	Unidades
Z0	Impedancia en modo diferencial a 10 MHz	80	110	Ω
T'0	Retardo de línea especifico	-	10	ns/m
α5MHz	Atenuación del cable a 5 MHz	-	82	dB/Km
Protección contra interferencia		**Opcional**		
RDCC	Resistencia de contacto	-	50	mΩ
ZC	Impedancia del conector	70	200	Ω
LC	Longitud de acoplamiento del conector	-	150	mm

4.3.2.1. Transceptor FlexRay

El transceptor FlexRay TJA1080A es totalmente compatible con las especificaciones de la capa física de la revisión 2.1A del protocolo de comunicaciones FlexRay; las principales funciones como BD son: control del regulador de voltaje, interfaz de control del BG y adaptación de niveles lógicos, entre otras (NXP B.V, 2012).

Los voltajes entre las terminales BP (*Bus Plus*) y BM (*Bus Minus*) del BD respecto a la tierra (GND, *Ground*) del sistema, se denominan uBP y uBM respectivamente. El voltaje diferencial (uBus) está dado por:

$$uBus = uBP - uBM$$

El bus FlexRay puede tomar cuatro posibles estados (véase la Figura 4.4):

- *Idle_LP* (*Low Power*)*:* cuando no circula corriente por el bus y el transceptor fuerza un cero a la salida, por lo tanto, el bus se encuentra en baja energía.
- *Idle:* estado en el que no circula corriente por el bus, pero el transceptor fuerza un voltaje determinado para BP y BM

- *Dato_1:* estado de uBus positivo, con un valor lógico de 1.
- *Dato_0:* estado de uBus negativo, con un valor lógico de 0.

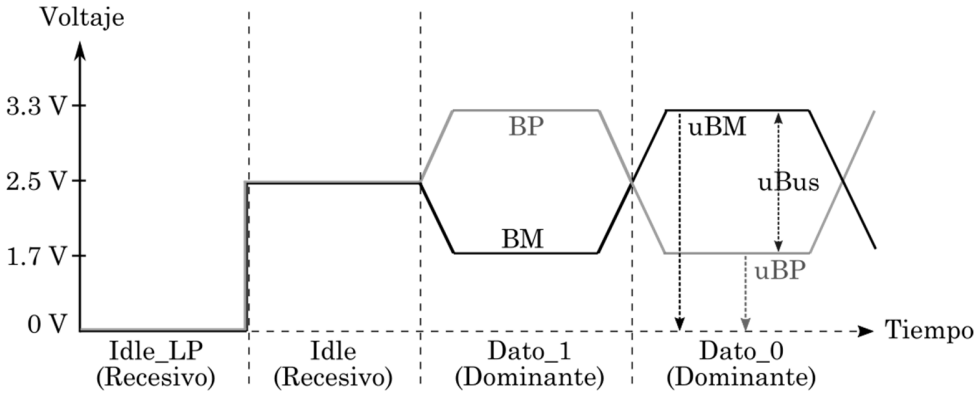


Figura 4.4. Estados en el bus FlexRay utilizando TJA1080A (FlexRay Consortium, 2010).

El BD no distingue entre los estados *Idle_LP* e *Idle,* debido a que en ambos casos no existe señal de datos en el bus; al entrar en los estados *Dato_1* y *Dato_0* por lo menos un BD genera una diferencia de potencial entre BP y BM que puede ser positiva o negativa, respectivamente. Esta relación se muestra en la Tabla 4.3.

Tabla 4.3. Nivel de señal en el bus en términos de los estados del BD (González Salinas, 2008).

Estado del BD	**TxEN***	**BGE****	**TxD*****	**Resultado de la señal en el bus**
Activo	Alto	X	X	*Idle*
	X	Bajo	X	*Idle*
	Bajo	Alto	Bajo	*Dato_0*
	Bajo	Alto	Alto	*Dato_1*
Baja Potencia (LP)	X	X	X	*Idle_LP*

*TxEN (*Transmit Data Enable Not signal from CC*).
**La señal BGE pertenece a la interfaz entre el BG y el BD.
***TxD (*Transmit Data signal from CC*).

4.3.2.1.1. Terminador de interfaz de conexión

El terminador más simple para el acoplamiento entre el nodo y el bus consiste en una resistencia entre las terminales BP y BM del BD; mientras que uno más complejo, que mejora el rendimiento de la conexión, consiste en arreglos resistivos, de capacitores, diodos de protección y bobinas de choque, con lo que se incrementa la protección respecto a descargas electroestáticas (ESD,

Electrostatic Discharge) e interferencia de campos electromagnéticos (EMI, *Electromagnetic Interference*) como se muestra en la Figura 4.5. En todas las conexiones punto a punto, sean nodo-estrella o estrella-estrella, la resistencia de terminación debe ser equivalente a la impedancia nominal del cable.

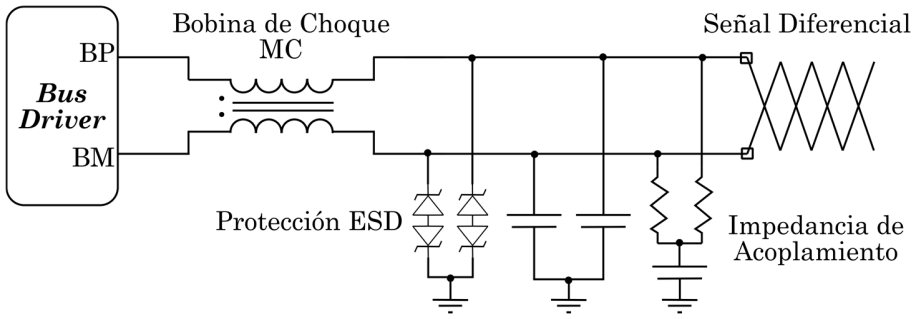


Figura 4.5. Esquema terminador de conexión (Xin He, 2018).

4.3.2.1.2. Bobina de choque

Idealmente, la utilización de voltaje diferencial evita el acoplamiento de ruidos externos en la señal transmitida, sin embargo, en la implementación se generan pequeñas asimetrías que hacen que la señal diferencial no esté perfectamente equilibrada. Al ocurrir esto, el componente diferencial ya no será constante y generará ruido dependiendo de los datos; para corregir este problema se utiliza como filtro una bobina de choque en modo común (CMC) que consiste en dos bobinas de alambre enrollado sobre un mismo núcleo magnético; se utiliza para dejar pasar las señales diferenciales o las señales que tienen la misma amplitud con una polaridad opuesta y para atenuar las señales que aparecen simultáneamente con el mismo nivel y polaridad (modo común) ya que en esta configuración dichas señales se anulan entre sí. Se suelen utilizar CMC de 51uF o 100uF (Robertson, 2016).

4.3.2.1.3. Protección por descarga electroestática

A pesar de reducir el ruido de modo común en la mayoría de las bandas de frecuencia, debido a que la CMC introduce una inductancia en serie, puede generar resonancias al combinarse con la capacitancia parásita de la red que aumenta la cantidad de ruido en las frecuencias en que resuene (Robertson,

2016). Una solución alternativa es colocar una protección por descarga electroestática (ESD). El PESD1CAN está diseñado para proteger dos líneas de bus CAN del daño causado por ESD y pulsos de sobretensión; FlexRay al estar basado en gran medida en CAN, es completamente compatible. El dispositivo se puede utilizar tanto para bus de alta velocidad como para protección de bus tolerante a fallos (Nexperia, 2008).

4.3.2.2. Topologías de red en FlexRay

Los clústeres de FlexRay permiten la utilización de topologías del tipo bus, estrella o una combinación híbrida; con canal único o canal dual, tanto como lo permitan los límites individuales de cada topología (véase la Tabla 4); por ejemplo, en la Figura 4.6 se muestra una topología híbrida con canal dual. Todas las topologías permitidas por FlexRay son del tipo lineal, por lo que FlexRay está libre de anillos cerrados, además de permitir conectar hasta 64 nodos en el clúster, dentro de los cuales como mínimo deben ser 2 nodos *coldstart* y máximo 15.

Tabla 4. Resumen de las restricciones de las topologías.

Descripción	Máximo	Unidad
Conexión punto a punto	24	m
Estructura tipo bus (Distancia eléctrica entre dos ECUs)	24	m
Red estrella activa	24	m

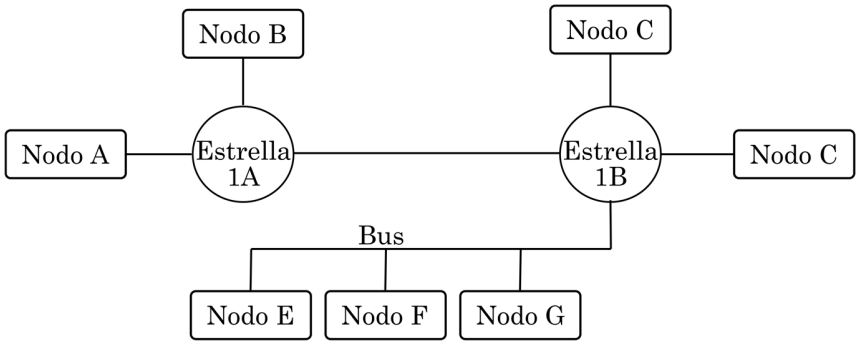


Figura 4.6. Ejemplo de topología híbrida con canal dual (Texas Instruments Incorporated, 2015).

4.3.3. Implementación del nodo

Existen tres tipos de arquitectura para FlexRay: a) microcontrolador (MCU, *Microcontroller Unit*)+CC+BD, b) MCU (con CC incorporado)+BD y c) MCU (con CC y BD incorporados). La arquitectura adoptada por Texas Instruments (TI) es la segunda, como se muestra en la Figura 4.7, por lo que requiere la implementación de un BD externo (Xin He, 2018).

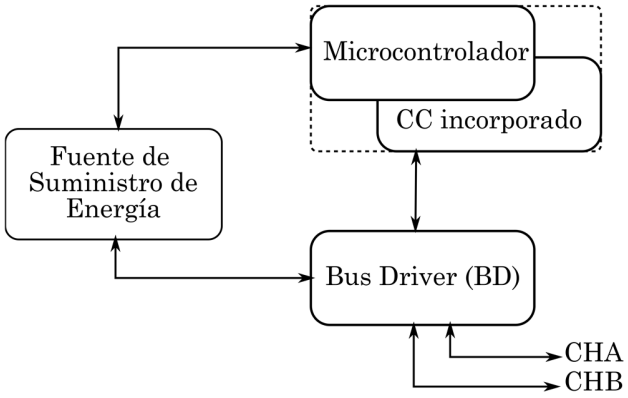


Figura 4.7. Estructura MCU+BD (Xin He, 2018).

La familia de MCU Hercules basados en ARM Cortex ofrece rendimiento escalable, conectividad, funciones de memoria y seguridad; además, implementan seguridad en hardware maximizando el rendimiento y reduciendo la sobrecarga del software. La familia TMS570 provee un alto rendimiento para aplicaciones automotrices (Texas Instruments Incorporated, 2011).

El sistema de desarrollo Hercules TMS570LC43x LaunchPad Development Kit (véase la Figura 4.8) es una plataforma de evaluación de alto rendimiento y bajo costo basada en la familia TMS570 de TI. Cuenta con diferentes características para el diagnóstico y protección de las diversas memorias internas, emplea un reloj de 300 MHz e integra una amplia variedad de periféricos como ADC de 12 bits, temporizadores programables de alta gama, periféricos de control de motor y de comunicaciones (Ethernet, FlexRay, CAN, MibSPI, EMIF y múltiples interfaces seriales) (Texas Instruments Incorporated, 2019).

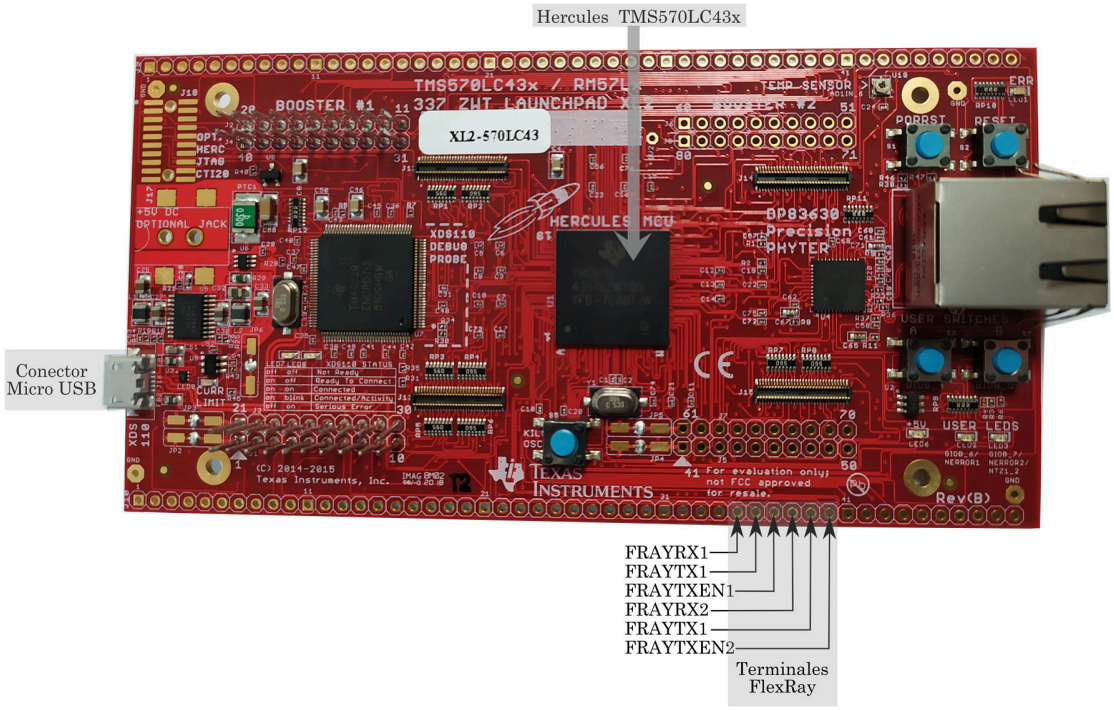


Figura 4.8. Tarjeta LAUNCHXL2-570LC43 (Texas Instruments Incorporated, 2019).

TI implementa el núcleo del protocolo de comunicaciones FlexRay (E-Ray), conforme a las especificaciones de la versión 2.1 Rev A (Texas Instruments Incorporated, 2015):

- Velocidades de transferencia de información de hasta 10 Mbps en ambos canales.
- Almacenamiento de 8 Kbytes en RAM para mensajes:
  - Búfer para 128 mensajes con sección de datos máxima de 48 bytes.
  - Búfer para 30 mensajes con sección de datos de 254 bytes.
  - Diferentes longitudes de carga de mensajes.
- Protección por paridad de mensajes en RAM.
- Controles del manejador de mensajes:
  - Mensaje de arbitraje de acceso a RAM.
  - Filtrado de aceptación.
  - Mantener el horario de transmisión.
  - Proporcionar información de estado.

- Cada búfer de mensajes puede ser configurado como:
  - Búfer de recepción o entrada (IBF, *Input Buffer*).
  - Búfer de transmisión o salida (OBF, *Output Buffer*).
- Cada Búfer puede ser asignado como:
  - Segmento estático del ciclo de comunicación.
  - Segmento dinámico del ciclo de comunicación.
  - Parte de un receptor FIFO (*First Input First Output*).
- Cuenta con dos módulos de interfaz:
  - Acceso directo del CPU a los búferes de mensajes vía búfer de entrada y de salida (VBUS IF).
  - Unidad de transferencia FlexRay (FTU, *FlexRay Transfer Unit*) para la transferencia automática de datos entre datos de memoria y búfer de mensaje, sin la interacción del CPU.
- Filtrado por identificador (ID) de trama, ID de canal y contador de ciclos.
- Módulo de interrupciones enmascarable.
- Compatible con gestión de red.

4.3.3.1. Tipos de memoria en la tarjeta Hercules

El Cortex-R5F utiliza direcciones de bus de 32 bits para acceder a un espacio de almacenamiento de hasta 4 GB, este espacio se encuentra dividido en una gran cantidad de regiones con direcciones de inicio (dirección base) propias. La memoria principal comienza en la dirección 0x00000000 por defecto al reiniciar el sistema. La memoria RAM de datos para el CPU comienza en la dirección 0x08000000. La memoria de instrucciones (*flash*) del CPU es de 4 MB y se implementan varios módulos SRAM para soportar las funcionalidades de los módulos incluidos (Texas Instruments Incorporated, 2018).

4.3.3.2. Módulo FlexRay

El módulo FlexRay de TI contiene los elementos mostrados en la Figura 4.9. La interfaz del búfer periférico con la arquitectura interna del microcontrolador (VBUS IF) proporciona a la CPU un acceso directo a los búferes de mensajes y le permite al módulo FlexRay actuar como nodo maestro o nodo esclavo.

El manejador de mensajes (MHD, *Menssage Handler*) se encarga de controlar la transferencia de datos entre el búfer de entrada/salida (IBF/OBF) y la RAM de mensajes, además, entre la RAM transitoria (TBF A/B, *Transient Buffer RAM*) y los dos controladores de canal del protocolo FlexRay (PRT A/B, *FlexRay Channel Protocol Controller*) y la RAM de mensajes, que almacena hasta 128 búferes de mensajes junto con los datos de configuración relacionados (encabezado y partición de datos). El TBF A/B almacena la sección de datos (dos mensajes completos), mientras que el PRT A/B consiste en registros de desplazamiento y el protocolo de máquina de estados finitos (FSM, *Finite State Machine*) de FlexRay, que conecta la RAM transitoria con la capa física del BD.

La unidad de tiempo global (GTU, *Global Time Unit*) genera microticks (μT) y macroticks (MT), y se encarga también de las siguientes tareas: sincronización de reloj tolerante a fallos por algoritmo FTM, gestión del contador de ciclos, control de tiempos y soporte de reloj externo.

El sistema de control universal (SUC, *System Universal Control*) controla las funciones de configuración, inicialización (*startup*), despertar (*wakeup*) y los modos de operación. El FPS (*Frame and Symbol Processing*) y el NEM (*Network Management*) se encargan del procesamiento de tramas y símbolos, y de administrar la red, respectivamente. Otros elementos que se integran al módulo FlexRay son los módulos de reloj y de control de interrupción (INT, *Interrupt Control*) (Texas Instruments Incorporated, 2018).

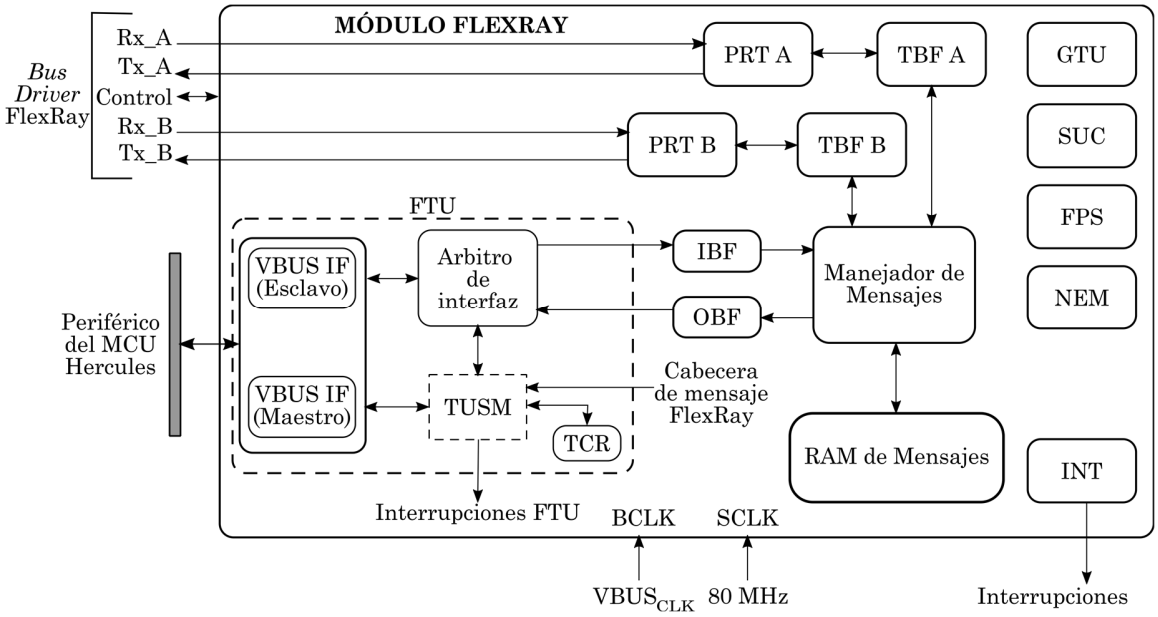


Figura 4.9. Diagrama de bloques del módulo FlexRay (Texas Instruments Incorporated, 2018).

4.3.3.2.1. Implementación del módulo FlexRay

El módulo FlexRay se conforma de dos segmentos de memoria RAM (véase la Figura 4.10) que se utiliza para el almacenamiento de los datos de control, configuración y monitoreo (registros de configuración) del CC y de la FTU respectivamente; la FTU se subdivide en dos segmento: el segmento (RAM de FTU) y el segmento de registros de configuración de la FTU, cuyas direcciones base se encuentran mapeadas en la memoria y se muestran en la Tabla 4.5.

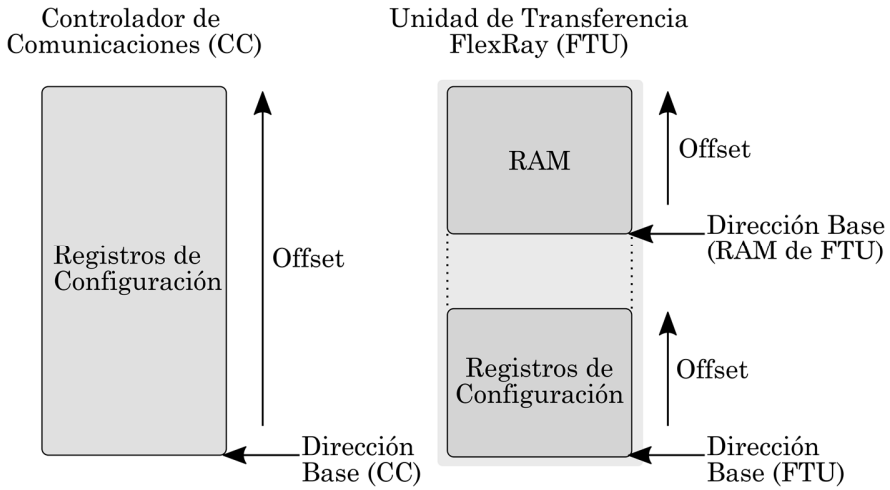


Figura 4.10. Diagrama de segmentos de memoria del módulo FlexRay (Texas Instruments Incorporated, 2018).

Tabla 4.5. Rango de direcciones del módulo FlexRay (Texas Instruments Incorporated, 2018).

Bloque	Registros	Rango de direcciones
CC	Controlador de comunicaciones FlexRay	0xFFF7_C800 – 0xFFF7_CFFF
FTU	Registros de la unidad de transferencia FlexRay	0xFFF7_A000 – 0xFFF7_A1FF
	RAM de la unidad de transferencia FlexRay	0xFF50_0000 – 0xFF51_FFFF

4.3.3.2.1. Controlador de comunicación del módulo FlexRay

La Figura 4.11 muestra el diagrama de estados del controlador de comunicaciones (CC) para el correcto funcionamiento de los nodos FlexRay.

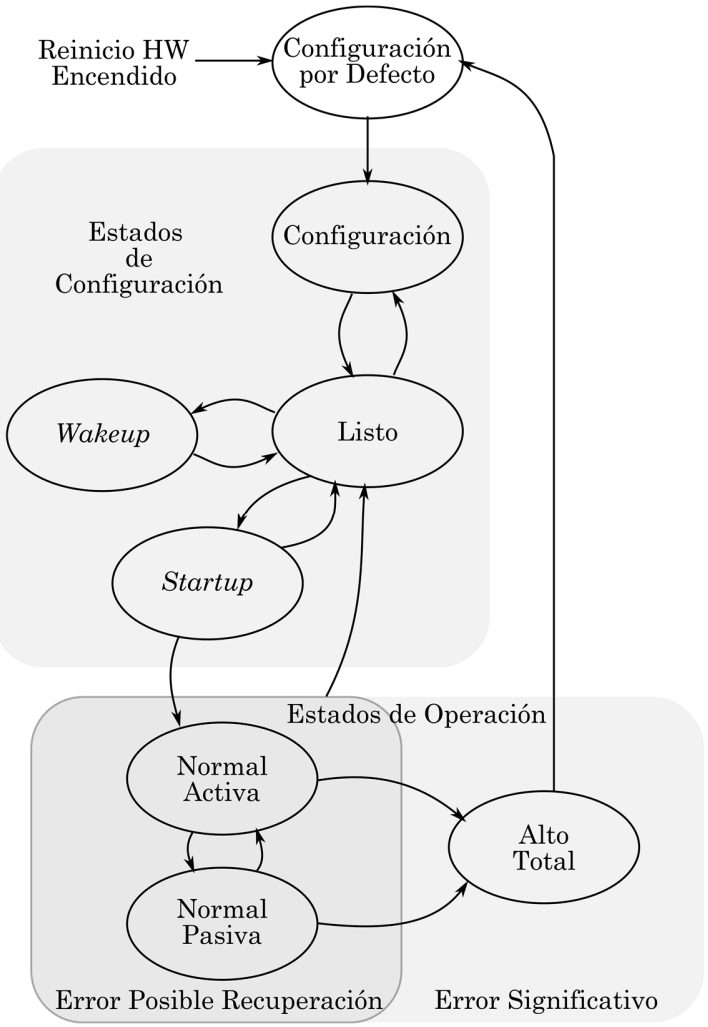


Figura 4.11. Diagrama de estados general del controlador de comunicaciones (Texas Instruments Incorporated, 2018).

El CC asigna un espacio de direcciones de 2 Kbytes (0x0000 a 0x07FF) en registros de 32 bits, divididos en bloques de acuerdo con su función como se

muestra en la Tabla 4.6. Los datos transitorios de entrada o salida (RAM de mensajes), consisten en un espacio de 128 búferes de mensajes individuales. El acceso de la CPU a la RAM de mensajes se realiza a través del IBF/OBF bajo el control del manejador de mensajes, para evitar conflictos entre la transmisión/recepción de mensajes y el acceso de la CPU (Texas Instruments Incorporated, 2018).

La transferencia de mensajes entre IBF/OBF y la RAM de mensajes, es activada por el CPU del *host* escribiendo el número del búfer destino u origen al que se accede en los registros de solicitud (IBCR/OBCR).

El filtrado se realiza mediante la comparación de la configuración de los búferes de mensajes asignados con los valores actuales del contador de ciclos, ranuras y el ID del canal (canal A, B), escribiendo la máscara de órdenes en los registros (IBCM/OBCM). Un búfer de mensajes sólo se recibe o se transmite al superar los filtros activados (Texas Instruments Incorporated, 2018).

Tabla 4.6. División a bloques de los registros del controlador de comunicaciones del módulo FlexRay (Texas Instruments Incorporated, 2018).

Bloque de registros del CC	**Desplazamiento (*offset*)**
Special Registers	0x00
Interrupt Registers	0x20
Communication Controller Control Registers	0x80
Communication Controller Status Registers	0x100
Message Buffer Control Registers	0x300
Message Buffer Status Registers	0x310
Identification Registers	0x3F0
Input Buffer (RAM de mensaje)	0x400
Input Buffer	0x500
Output Buffer (RAM de mensaje)	0x600
Output Buffer	0x700

Los búferes IBF/OBF se construyen con una estructura de doble búfer para el acceso del CPU *host* (IBF *host*/OBF *host*) (véase la Figura 4.12), mientras que la otra mitad es accesible para el manejador de mensajes (IBF *shadow*/OBF *shadow*) y estos intercambian los datos para realizar las transferencias. La configuración de los búferes de transmisión y de recepción se realiza por número de búfer, asignando la configuración deseada en los registros WRHS1, WRHS2, WRHS3; IBCM e IBCR (transmisiones del CPU *Host* a la RAM de mensajes) o en OBCM y OBCR (transmisión de la RAM de mensajes al CPU *Host*) (Texas Instruments Incorporated, 2018).

Para el manejador de mensajes, los ID de trama de los búferes de mensajes asignados al segmento estático deben estar en el rango de 1 a GTU7.NSS (9-0) y los ID de trama para los búferes de mensajes asignados al segmento dinámico deben estar en el rango de GTU7.NSS (9-0) + 1 a 2047.

Es indispensable que la información de la cabecera esté configurada correctamente antes de comenzar cualquier transferencia, por lo que se debe configurar una estructura de búfer de datos adecuada en la memoria del sistema (véase la Figura 4.13):

- 4 palabras: 3 palabras de cabecera y 1 palabra de estado de búfer, que se transfiere sólo por el CC FlexRay a la memoria del sistema, generalmente como información de estado de una trama recibida y debe considerarse en la definición de la estructura de datos sea o no utilizada la información. Para mayor información se recomienda consultar (Texas Instruments Incorporated, 2018).
- X medias palabras: para información de la carga útil con longitud idéntica a los mensajes del segmento estático.

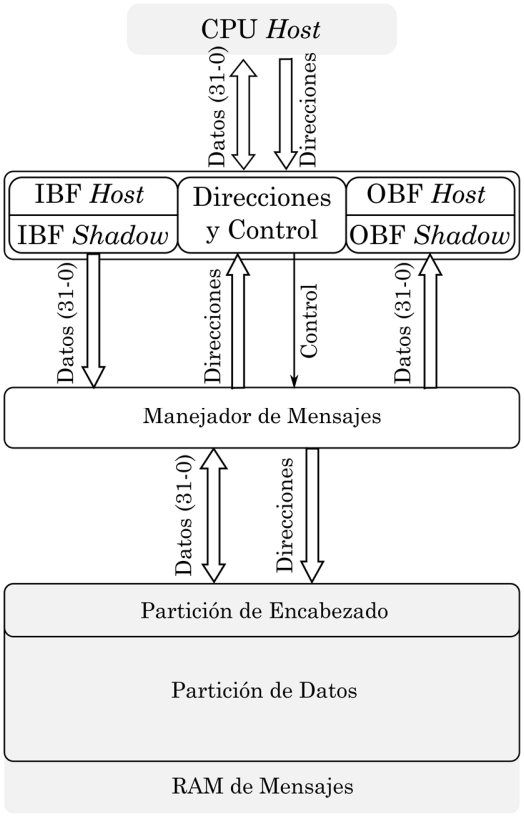


Figura 4.12. Diagrama de acceso entre CPU Host y RAM de mensajes (Texas Instruments Incorporated, 2018).

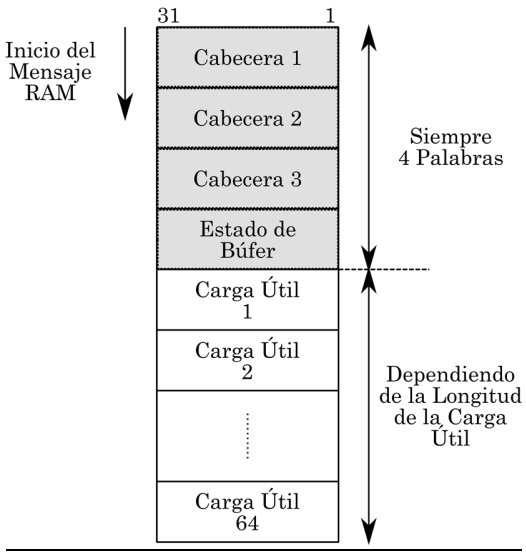


Figura 4.13. Estructura del búfer de mensajes (Texas Instruments Incorporated, 2018).

4.3.3.2.2. Unidad de transferencia FlexRay

La Unidad de Transferencia FlexRay (FTU) está dedicada para el controlador de comunicaciones FlexRay. Los búferes IBF/OBF pueden ser

accesados directamente por la CPU o mediante la interfaz nativa para la RAM de mensajes, que es una máquina de estados inteligente (TUSM, *Transfer Unit State Machine*) con la que se puede realizar la transferencia de datos entre los IBF/OBF del controlador de comunicaciones y la memoria del sistema sin la intervención del CPU. El árbitro de interfaz controla el acceso a los IBF/OBF por lo que el acceso directo del CPU no es posible si la TUSM está encendida. Con la RAM de configuración de transferencia (TCR, *Transfer Configuration RAM*) se configura la secuencia de transferencia ejecutada por la TUSM (Texas Instruments Incorporated, 2016).

Para el correcto funcionamiento de la FTU en la transferencia de datos, el módulo debe ser configurado considerando los siguientes elementos (Texas Instruments Incorporated, 2012):

- Adecuada configuración de la RAM de mensajes FlexRay.
- Adecuada generación y configuración de la estructura para el búfer de datos en la memoria del sistema (SM, *System Memory*) del dispositivo.
- Inicializar los búferes de mensajes FlexRay.
- Establecer la dirección base para transferencia de datos.
- Configurar la dirección de transferencia en la RAM de configuración (TCR) (opcional).
- Configurar interrupciones de la FTU (opcional).

4.3.3.2.3. Ciclo de comunicación

Los segmentos, estático y dinámico, y la ventana de símbolos, forman el tiempo de comunicación de red (NCT, *Network Communication Time*); el contador de ranuras comienza en 1 y cuenta hasta que se alcanza el final del segmento dinámico en cada canal de comunicación. Ambos canales utilizan el mismo MT sincronizado (National Instruments Corporation, 2019).

4.4. Capa de Enlace de Datos

La capa de enlace de datos (DLL, *Data Link Layer*) se considera el núcleo del protocolo FlexRay debido a que en ella se establecen los mecanismos para la detección de errores y se definen la temporización, la sincronización y la carga, entre otros; además, se establecen los mecanismos para el acceso al medio independientemente de la topología de red utilizada. La DLL se subdivide en dos subcapas: control de enlace lógico (LLC, *Logia Link Control*) y control de acceso al medio (MAC, *Medium Access Control*) como se muestra en la Figura 4.14 (González Salinas, 2008).

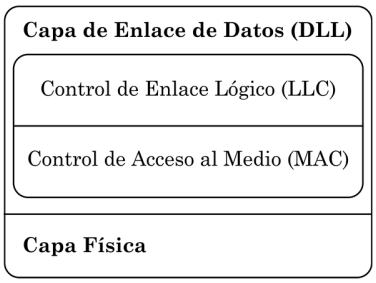


Figura 4.14. Capas del protocolo FlexRay (González Salinas, 2008).

4.4.1. Subcapa de control de enlace lógico

En la subcapa de control de enlace lógico (LLC) se definen dos tareas independientes del método de acceso al medio como son: controlador de interfaz con el *host* (CHI, *Controller Host Interface*) y control de operaciones del protocolo (POC, *Protocol Operation Control*) (González Salinas, 2008).

4.4.1.1. Controlador de interfaz con el host

El controlador de interfaz con el *host* (CHI) se encarga de administrar el flujo de los datos y el control entre el *host* y el motor del protocolo FlexRay en cada nodo; además, proporciona los medios para una interacción estructurada entre estos mediante las siguientes funciones: a) manejo de datos del protocolo, que controla el intercambio de datos relevantes para la operación del protocolo, como los datos de control, de configuración y de estado; b) manejo de mensajes de datos, que realiza tres tareas, controla el búfer para el intercambio de datos

y de mensajes a transmitir, realiza el filtrado de mensajes que es un medio para la selección de mensajes con base en su identificador que permite determinar si el mensaje le es o no útil al receptor dentro de la red, y manipula el vector de red; este último, permite enlazar dos nodos específicos para la comunicación en un mismo o en diferente clúster (González Salinas, 2008).

4.4.1.2. Control de operaciones del protocolo

Las funciones del control de operaciones del protocolo (POC) son iniciar, coordinar y apagar el núcleo del protocolo, así como también iniciar el clúster y proporcionar la detección de errores. La finalidad del POC es modificar los modos de operación del protocolo en el nodo en respuesta a las órdenes del *host* recibidas a través del CHI (González Salinas, 2008).

Para la sincronización del esquema TDMA, el POC realiza dos tareas primordiales: a) el inicio del clúster (*wakeup*) y b) el inicio del sistema (*startup*). El objetivo del *wakeup* es preparar a los nodos en el clúster para inicializar el sistema, esto sólo puede ser realizado por los nodos *coldstart* y debe realizarse por cada canal de forma individual para evitar interferencias entre ellos. El nodo *coldstart* debe asegurarse de que sólo un nodo transmita el patrón *wakeup* y encargarse de brindar soporte a los nodos restantes que intentan iniciar el clúster. El esquema TDMA requiere realizar la sincronización entre todos los nodos conectados al clúster que en FlexRay se define como *startup*, y que de acuerdo con su estrategia de tolerancia a fallos sólo se puede realizar cuando todos los nodos en el clúster se encuentran preparados (los canales del clúster deben estar activos) por el *wakeup*; sólo el nodo *coldstart* que comience la transmisión del símbolo CAS (*Collision Avoidance Symbol*) continuará transmitiendo tramas de *startup* y los primeros ocho ciclos son utilizados para la sincronización de los nodos.

La sincronización con el clúster del CC puede realizarse de tres formas distintas (FlexRay Consortium, 2010):

- Nodo *coldstart* principal: si no se detecta algún CE en el clúster se inicia la transmisión del símbolo CAS en el primer ciclo regular, a partir del cual se transmite la trama *startup*; debido a que cada nodo *coldstart* está habilitado para transmitir el símbolo CAS, pueden ocurrir transmisiones simultáneas que se resuelven durante los primeros cuatro ciclos después de dicha transmisión; y tan pronto se recibe el símbolo CAS o una cabecera de trama, se regresa al estado de detección, y en el cuarto ciclo el resto de nodos *coldstart* se inicia la transmisión de sus tramas *startup*.
- Nodo *coldstart* de seguimiento: al detectar un CE se intenta integrar al nodo *coldstart* transmisor si recibe un par de tramas *startup* válidas en los siguientes dos ciclos; si la corrección del reloj no genera ningún error y continua recibiendo tramas del mismo nodo se considera integrado, y procede con la transmisión de las tramas *startup*; si en los siguientes tres ciclos no se genera error en la corrección del reloj y es visible por lo menos para un nodo *coldstart*, el nodo entra en operación y en caso contrario regresa al estado de detección.
- Integración de nodos no *coldstart*: si detecta un CE intenta integrarse al nodo *coldstart* transmisor en los siguientes dos ciclos al tratar de encontrar por lo menos un par de nodos que transmitan tramas *startup* que se ajusten con su propio cronograma; en caso de error se aborta y realiza un nuevo intento. Después de recibir pares de tramas *startup* válidas durante cuatro ciclos, todos los nodos del clúster entran en operación.

Para responder a los errores, el POC cuenta con dos mecanismos básicos de acuerdo con los dos tipos de errores:

- Errores significativos: son las condiciones en los que el POC detendrá toda operación del nodo de manera inmediata, generadas por errores detectados por un producto específico (BIST, *Build In Self Test*),

errores detectados por el *host* que resultan en una orden de alto total y errores fatales detectados por el POC en algún mecanismo del núcleo del protocolo.

- Errores de posible recuperación en un tiempo limitado: se utiliza un modelo de degradación de errores de tres niveles diseñado para reaccionar a ciertas condiciones detectadas por el mecanismo de sincronización tolerante a fallos, con ello se puede corregir el error sin detener toda operación del nodo, el modelo se define como:
  - Operación normal activa: condiciones libres de errores o en límites tolerables en el nodo para la transmisión y recepción ya que no genera interferencias con otros nodos.
  - Operación normal pasiva: degradación en la sincronización con el clúster por lo que se deshabilita la transmisión debido a la posibilidad de colisión con transmisiones de otros nodos, pero se mantiene la recepción ya que es posible recuperar la sincronización y regresar al modo de operación normal activa.
  - Alto total (*Halt*): errores persistentes o graves, por los que se detiene toda operación del nodo y el POC prepara el mecanismo para reinicializar el nodo ya que no es posible recuperarse de estos errores y se notifica al *host* que actúa de acuerdo con el protocolo.

4.4.2. Control de acceso al medio

La subcapa MAC es un ciclo de comunicaciones periódico, en el que existen dos esquemas de modos de acceso al medio, TDMA y FTDMA, que consisten en un esquema de mini ranuras (*mini slots*).

4.4.2.1. Representación del tiempo en FlexRay

En FlexRay el tiempo se representa con base en ciclos (hasta 64) y utiliza como unidades *macroticks* (MT) y *microticks* (μT, típicamente 25 ns), cada ciclo se compone de MT conformado por un número específico de μT, que son la unidad

mínima de tiempo en el esquema de control de acceso al medio que derivan del oscilador del CC, como se muestra en la Figura 4.15 (FlexRay Consortium, 2010).

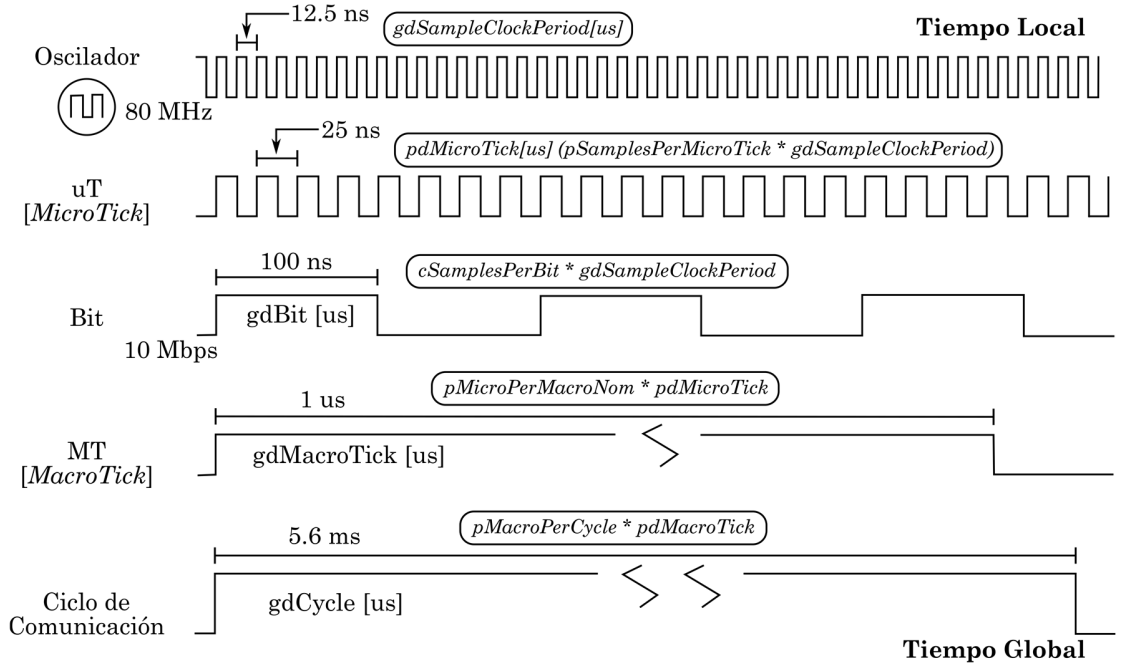


Figura 4.15. Representación de tiempos FlexRay (Paret, 2012).

4.4.2.1.1. Ciclo de comunicación

El ciclo de comunicación es el elemento fundamental en la comunicación FlexRay, en donde cada ciclo es de longitud idéntica e incluye una duración fija de segmentos estáticos y dinámicos (Iversen Huse, 2017).

El esquema de acceso al medio se define por la jerarquía de temporización (véase la Figura 4.16):

- El ciclo de comunicación: es la capa más alta, contiene los segmentos estáticos y dinámicos, la ventana de símbolo y el tiempo libre de red (NIT, *Network Idle Time*). El arbitraje de las transmisiones es por TDMA para segmentos estáticos y FTDMA para segmentos dinámicos con base en la manipulación de mini ranuras; en la ventana de símbolos se pueden transmitir símbolos y el NIT es un periodo libre de transmisiones.

- La capa de arbitraje: es el centro del control de acceso al medio y se construye en términos de MT; el segmento estático se forma por intervalos consecutivos de tiempo denominados ranuras estáticas y en el segmento dinámico se denominan mini ranuras a los intervalos de tiempo.
- Los puntos de acción (AP, *Action Point*): son instantes de tiempo específicos en los que el nodo realiza una acción específica en sincronización con su tiempo base local, como el inicio de la transmisión en el segmento estático y dinámico, o el fin de la transmisión en caso del segmento dinámico con las mini ranuras.

El segmento estático se genera al comienzo del ciclo de comunicación, consta de un número fijo de ranuras estáticas de una misma longitud y duración independientemente de la cantidad de datos a transmitir (máximo 1023, que es definido por el diseñador de la red) (González Salinas, 2008) y se asignan a los mensajes que serán transmitidos durante el segmento estático. Las ranuras se identifican por su número y el ID del mensaje, cada nodo cuenta con un contador de ranuras y dentro del clúster se incrementa sincrónicamente en todos los nodos al inicio de cada segmento estático, garantizando que todos los mensajes sean transmitidos oportuna y adecuadamente en cada ciclo de comunicación, por lo que el segmento estático está destinado a mensajes en tiempo real (Iversen Huse, 2017).

Después del segmento estático se genera el segmento dinámico, que tiene una duración fija dentro del ciclo de comunicación y consiste en varias mini ranuras dependiendo de la cantidad de datos en el segmento de carga útil a transmitir, de lo contrario la ranura será de longitud igual a una mini ranura, que puede variar en duración y longitud (González Salinas, 2008). Durante el segmento dinámico la transmisión no es obligatoria y se realiza de acuerdo con la activación por evento, cuando se transmite un mensaje dinámico, el contador de la ranura se incrementa al recibir dicho mensaje, si no se transmite una trama

durante una mini ranura, se incrementa el contador de ranura correspondiente a la mini ranura después de un tiempo definido. El segmento dinámico utiliza un sistema de priorización para el acceso al medio, dando prioridad al ID de mensaje de menor valor, además de garantizar que no existan colisiones de datos o arbitraje de bus durante el segmento, seguido de un delimitador inactivo de canal (CID, *Channel Idle Delimiter*) (Iversen Huse, 2017).

El CID se genera por diseño con 11 bits lógicos "1" para agregar un relleno de tiempo entre el final de la trama eléctrica y el final de las ranuras tanto estáticas como dinámicas.

La ventana de símbolo supervisa el rendimiento del BG y se genera de manera opcional justo después del segmento dinámico; se transmite un símbolo de prueba de acceso al medio (MTS, *Media acces Test Symbol*) de 30 bits para verificar el funcionamiento del BG local, seguido del CID (Iversen Huse, 2017).

El segmento NIT mantiene las líneas de transmisión inactivas (sin transmisión de datos) y permite que los nodos FlexRay calculen y apliquen los factores necesarios para sincronizar los relojes locales. Además, pueden extender su duración ya que contiene los MT que no se utilizaron en los segmentos anteriores, cuya duración no debe exceder los 767 macroticks, lo que produce una corrección de error por compensación al finalizar el NIT. Visto en el osciloscopio, no ocurre nada en la red y no se produce tráfico en sus líneas mientras dura debido a que la red está en modo de espera (Iversen Huse, 2017).

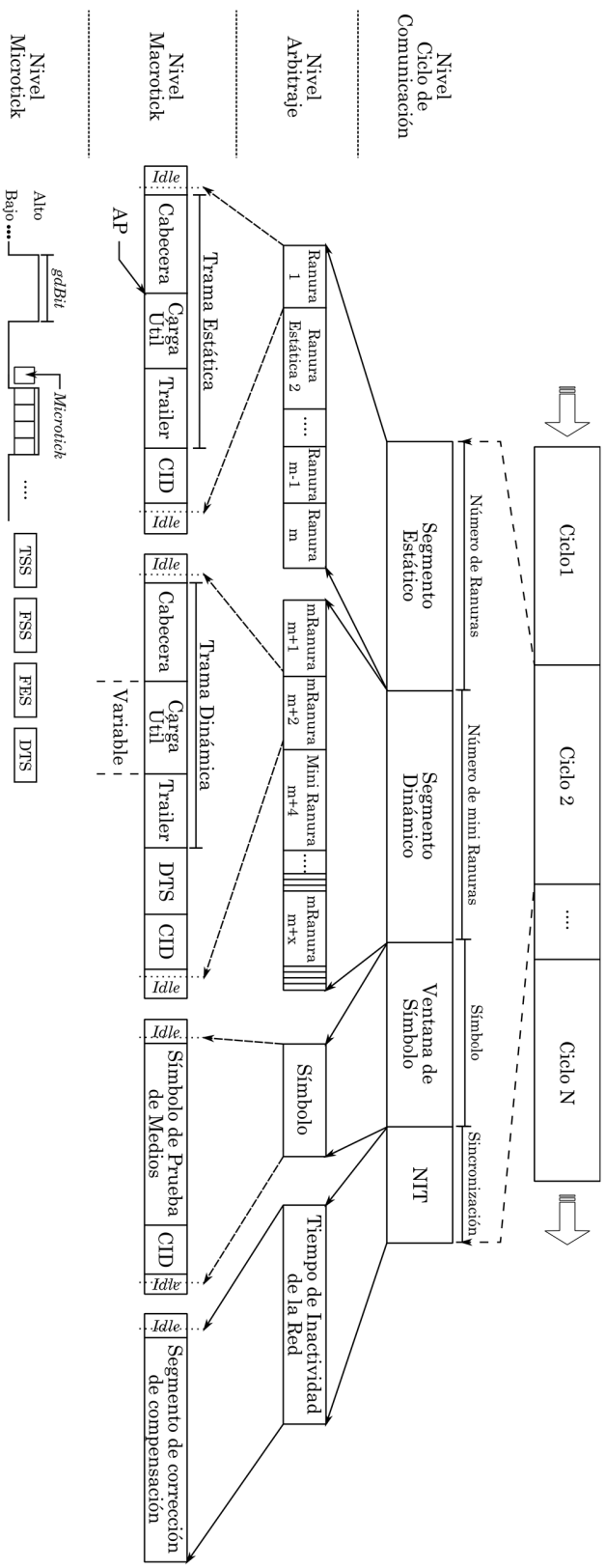


Figura 4.16. Estructura en ciclo de comunicación FlexRay (Paret, 2012).

4.4.2.2. Formato de trama

Las tramas de comunicación FlexRay se transmiten de la misma manera para segmentos estáticos y dinámicos. Las ranuras de los segmentos estáticos y las mini ranuras de los segmentos dinámicos están ocupadas por tramas de comunicación estáticas y dinámicas, respectivamente, siendo casi idénticas pero con algunas excepciones; ambas se componen principalmente por tres campos principales (Paret, 2012) (véase la Figura 4.17): a) cabecera, de una longitud de 5 bytes, b) carga útil, de longitud variable entre 0 y 254 bytes, y c) *trailer* que contiene el código de verificación por redundancia cíclica (CRC, *Cyclic Redundancy Check*) con longitud de 3 bytes, que es un código de tipo función que recibe un conjunto de datos y devuelve un valor de longitud fija como salida, su uso es común en redes digitales para detectar cambios accidentales en los datos al comparar el CRC recibido en la trama con el calculado con base en los datos de la trama en el receptor (FlexRay Consortium, 2010).

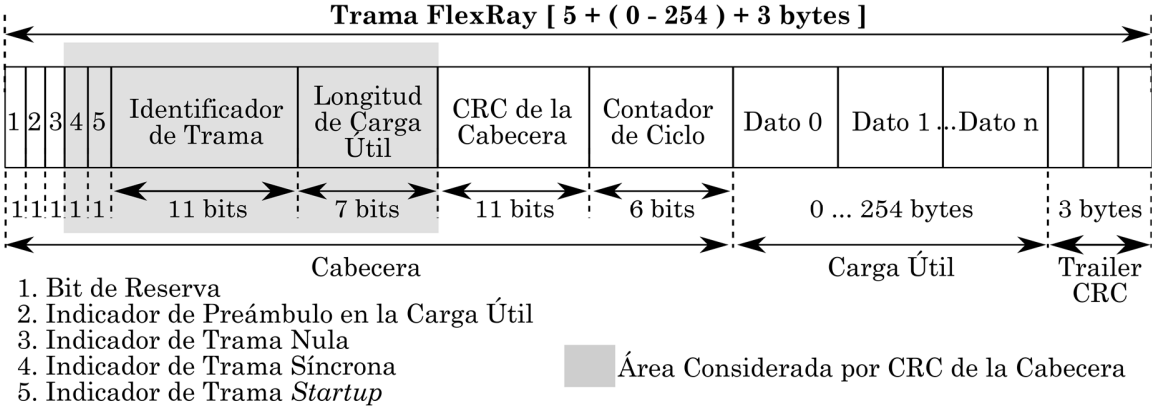


Figura 4.17. Trama del protocolo FlexRay (FlexRay Consortium, 2010).

Todos los segmentos son transmitidos de izquierda a derecha de acuerdo con la Figura 4.17, comenzando por el bit más significativo de cada elemento dentro de la trama, además, se incluyen BSS, FSS, TSS y FES que persisten en todos los mensajes transmitidos en el clúster FlexRay (Iversen Huse, 2017), como se muestra en la Figura 4.18.

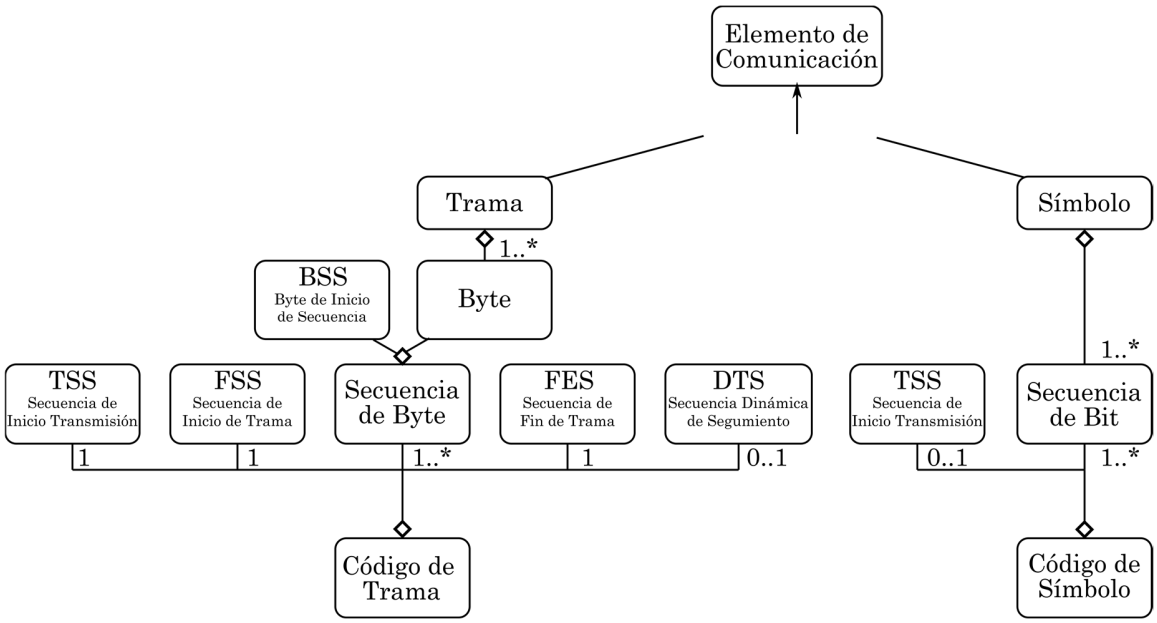


Figura 4.18. Encapsulamiento de código trama y código de símbolo (Paret, 2012).

El primer segmento es la cabecera, cuyos primeros cinco bits forman el campo *status*; dependiendo de los segmentos, estático o dinámico, tiene significados diferentes, los bits se encuentran en el siguiente orden (González Salinas, 2008) (Iversen Huse, 2017):

- Bit de reserva: este se destina para futuras aplicaciones, por lo que actualmente es ignorado por el protocolo y su valor lógico siempre es cero.
- Indicador de preámbulo en la carga útil: indica con un "1" lógico que se encuentra presente un vector de gestión de red en el segmento estático o un ID de mensaje en el segmento dinámico, y un "0" lógico indica que el segmento de carga útil no los contiene.
- Indicador de trama nula: un "1" lógico especifica un contenido válido en el segmento de carga útil y un "0" lógico indica si la trama transmitida es nula y establece todos los bytes de la carga útil en "0" lógico.
- Indicador de trama síncrona: si su valor lógico es "1", la trama transmitida será utilizada por los nodos receptores en los mecanismos

de sincronización de la red global, si el valor lógico es "0" no se utiliza la trama para ninguna tarea relacionada con la sincronización.

- Indicador de trama de inicio (*startup*): indica si es una trama *startup* o no. Un "1" lógico indica que es una trama *startup* que se envía por ciclo de comunicación y únicamente por nodos *coldstart,* por lo que solo estos son capaces de modificar este bit.
- Identificador de trama (*Frame ID*): es un campo de 11 bits que define la ranura (*slot*) por la que se transmite la trama, ésta no puede ser usada más de una vez en un ciclo de comunicación y tiene un rango valido del *slot* 1 al 2047.
- Longitud de carga útil: campo de 7 bits que indica la longitud del segmento de la carga útil; se codifica dividiendo el número de bytes entre 2, ya que se consideran palabras dobles.
- CRC de la cabecera: campo de 11 bits que contiene el código CRC calculado sobre los campos 4 al 7 de la trama FlexRay.
- Contador de ciclo: campo de 6 bits que indica el valor del contador de ciclos desde la perspectiva del nodo en el momento en que la trama fue transmitida.

El segundo segmento es la carga útil que puede contener de 0 a 254 bytes de datos (0 a 127 palabras dobles); tiene dos casos que se indican con un "1" lógico en el bit de preámbulo (González Salinas, 2008):

- Para las tramas transmitidas en el segmento estático, los primeros 12 bytes pueden ser usados como vector de gestión de red.
- Las tramas transmitidas en el segmento dinámico y los primeros dos bytes pueden ser usados como campo para el identificador de mensajes (ID).

El tercer segmento es el *trailer* CRC, contiene el código CRC calculado con los segmentos cabecera y carga útil; este es un campo de 3 bytes y utiliza una distancia Hamming de 6 para longitudes de hasta 248 bytes y de 4 para mayores

a 248 bytes, que es la efectividad de los códigos de bloque por lo que cuanto mayor sea la distancia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores (González Salinas, 2008). El nodo emplea un vector de inicialización diferente para cada canal, para el A utiliza 0xFEDCBA y para el canal B utiliza 0xABCDEF (Iversen Huse, 2017).

Además del formato de la trama FlexRay, el mensaje incluye secuencias que definen desde el inicio hasta el final de la transmisión, en una secuencia de empaquetamiento de bytes (véase la Figura 4.19).

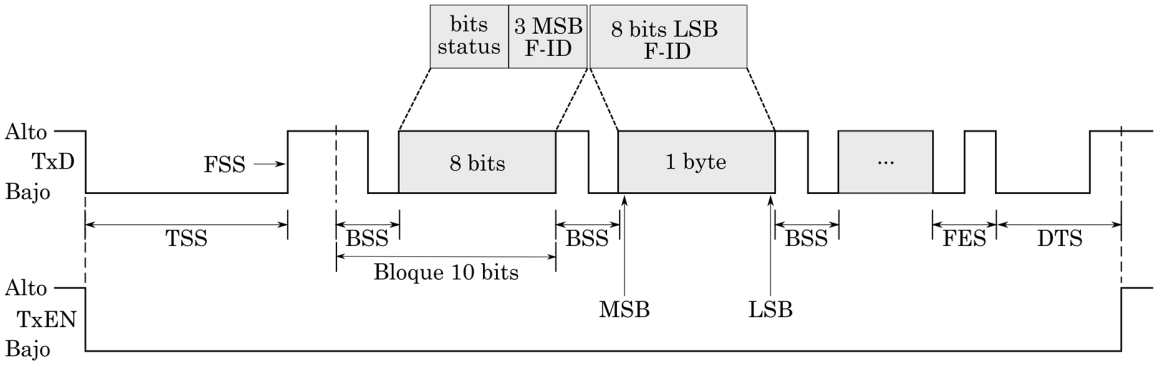


Figura 4.19. Secuencia de empaquetado de bytes de la trama FlexRay (Iversen Huse, 2017) y (González Salinas, 2008).

4.4.2.2.1. Secuencia de inicio de transmisión

La secuencia de inicio de transmisión (TSS, *Transmission Start Sequence*) es la primera en trasmitirse y se utiliza como indicador del inicio de la transmisión de la trama, tanto para los segmentos estático y dinámico, como para la ventana de símbolos. La longitud de la TSS depende del diseñador de la red, de la topología utilizada y de la distancia entre nodos, entre otros aspectos físicos; algunos valores válidos son de 3 a 15 bits de longitud que es un parámetro global en toda la red. Durante el tiempo válido del TSS (con duración de 1 bit al TSST (*TSS Transmitter*) + 1 bit), sólo se requiere que ocurra un bit en bajo para que el nodo se percate de una trama entrante, ya que la posibilidad de un retardo en el primer flanco de la trama respecto a los restantes es mayor, debido al retardo de la señal en los transceptores o por desviaciones en la red, haciendo

que la TSS sea percibida con menor longitud en los receptores, a esto se le denomina “truncamiento TSS” (Iversen Huse, 2017).

4.4.2.2.2. Secuencia de inicio de trama

La secuencia de inicio de trama (FSS, *Frame Start Sequence*) se indica con el bit lógico “1” al inicio de la trama y se coloca antes de la primera secuencia de inicio de byte (BSS, *Byte Start Sequence*); el flanco de subida del FSS indica el comienzo y presencia de una trama FlexRay (Iversen Huse, 2017).

4.4.2.2.3. Secuencia de inicio de bytes

En la secuencia de inicio de byte (BSS, *Byte Start Sequence*), la trama (cabecera+carga útil+trailer CRC) se subdivide en bytes, y para cada byte lógico se agrega un bit *start* (valor lógico de “0”) y un bit *stop* (valor lógico de “1”) al final, formando un bloque de 10 bits por cada byte que se envía, haciendo que en la sucesión de bytes se forme el patrón *stop-start* (valor lógico “10”) en el medio, que en FlexRay se denomina BSS; además la combinación del FSS y BSS genera un patrón distinto en el comienzo de la trama de valor lógico “110” (Iversen Huse, 2017).

4.4.2.2.4. Secuencia de finalización de trama

Después del segmento *trailer* CRC de la trama, el último byte es cerrado con una secuencia lógica específica de dos bits “01” denominada secuencia de fin de trama (FES, *Frame End Sequense*), los cuales conforman los últimos bits transmitidos y marcan el final de la trama FlexRay (Iversen Huse, 2017). Las Figuras 4.20 y 4.21 muestran las secuencias de finalización de trama estática y dinámica, respectivamente.

4.4.2.2.1. Secuencia de segmento estático y de segmento dinámico

En la ranura estática, adicionalmente a la trama FlexRay, se integra el delimitador de inactividad del canal (CID) y para la ranura dinámica la secuencia dinámica de seguimiento (DTS, *Dynamic Trailing Sequence*) entre el

trailer CRC y el CID, que soluciona el problema de desincronización debido al rango variable de la carga útil; el DTS que tiene una duración variable permite calibrar μT según la duración de las mini ranuras (Iversen Huse, 2017).

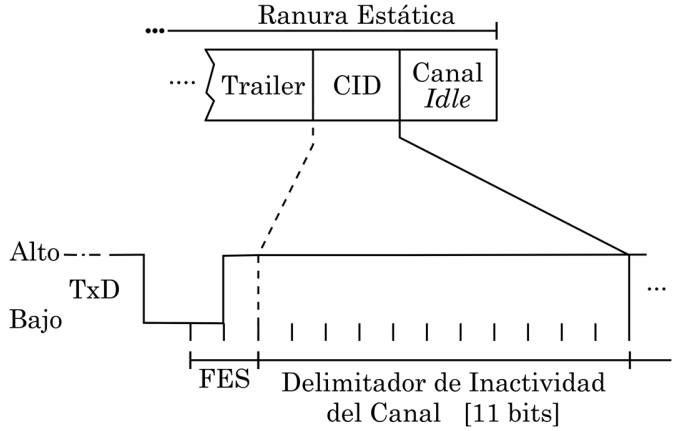


Figura 4.20. Fin de ranura estática (Paret, 2012).

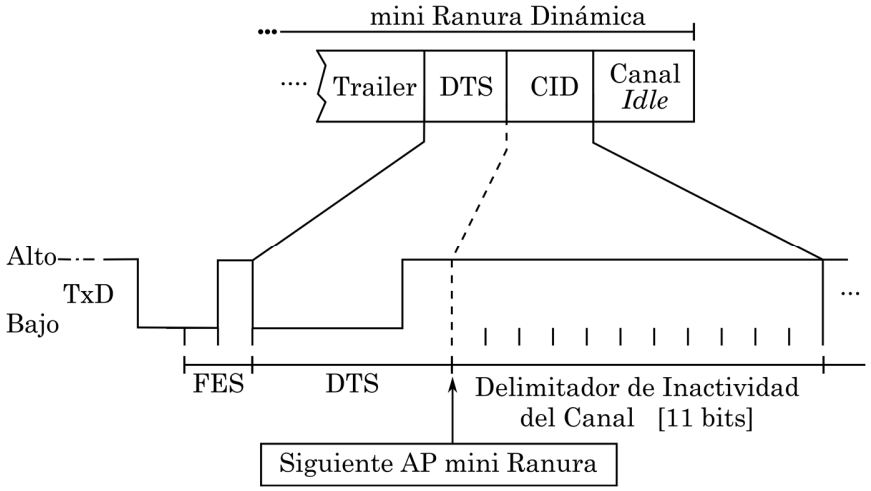


Figura 4.21. Fin de ranura dinámica (Paret, 2012).

4.4.2.3. Formato de símbolo

El protocolo de comunicaciones FlexRay define cuatro símbolos que se representan con tres patrones de bits distintos.

4.4.2.3.1. Patrón 1: Símbolo CAS y símbolo MTS

Ya que el nodo codificará el CAS (*Collision Avoidance Symbol)* y el MTS (*Media Access Test Symbol*) exactamente de la misma manera, los receptores distinguen entre estos símbolos según el estado del protocolo en el nodo.

El nodo transmitirá estos símbolos comenzando con el TSS, seguido de un nivel bajo con una duración de 30 bits, sincronizando los flancos de la señal TxEN con TxD al inicio de la transmisión, y con TxD en alto después de un retardo (*cdStaggerDelay*) con respecto al punto en el que TxEN regresa a alto al finalizar la transmisión, como se muestra en la Figura 4.22 (FlexRay Consortium, 2010).

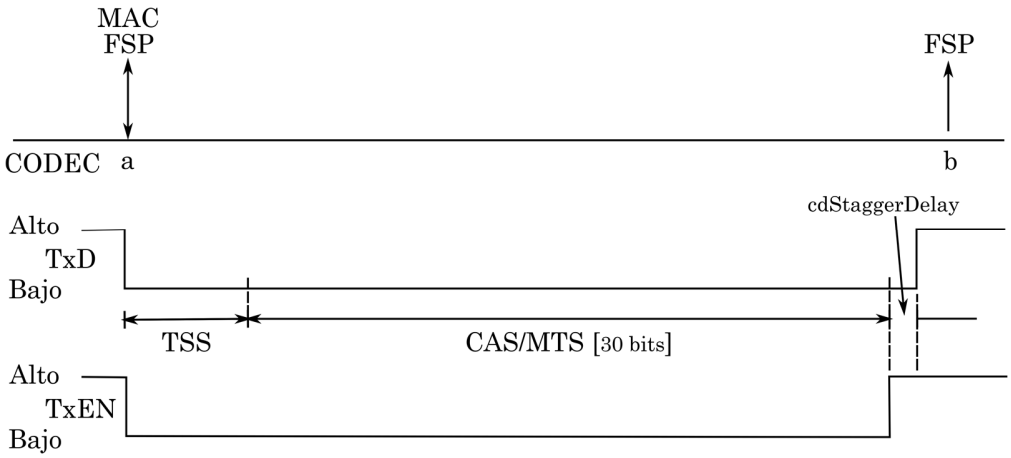


Figura 4.22. Codificación de símbolo CAS/MTS (FlexRay Consortium, 2010).

El segmento ventana de símbolo está dedicado a la inclusión del símbolo de prueba de acceso a los medios (MTS), que se utiliza para verificar que el guardián del bus esté operando correctamente. Este tiene la misma estructura que el CAS, terminando el segmento con el CID convencional (Paret, 2012).

4.4.2.3.2. Patrón 2: Símbolo WUS

Se incluyen dos tipos diferentes de activaciones (*wakeup*): a) *wakeup* local, cuando se debe al hecho de que una señal aplicada a un nodo a través de una entrada de activación separada despierta sólo este nodo, que una vez despierto puede ser capaz (si esto es parte de su tarea) de despertar al resto de los nodos del clúster; b) *wakeup* global; donde el nodo responsable de activar el clúster envía una señal particular llamada patrón de activación (WUP, *Wakeup Pattern*) en las líneas para activar el resto de los nodos del clúster a través del bus; consiste en una repetición de un símbolo llamado símbolo de activación (WUS, *wakeup Symbol*), con 2 a 63 WUS por WUP.

El símbolo WUS consta de una duración de entre 4 y 6 µs, de un número configurable de bits en Dato_0, y de 4 a18 µs, de un número de *Idle*. El nodo transmite el WUS sincronizando los flancos de bajada en TxEN y en TxD, al comienzo estando en bajo y volviendo a estado alto después de un retardo (*cdStaggerDelay*) en referencia al punto en que TxEN vuelve a alto, además no hay transmisión TSS, como se muestra en la Figura 4.23.

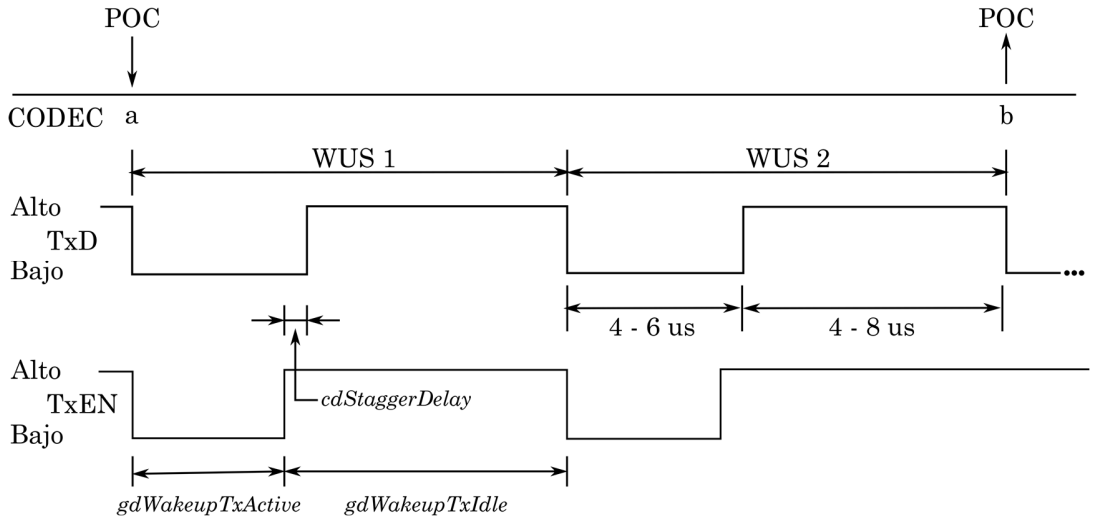


Figura 4.23. Patrón de activación que consta de dos o mas WUS (FlexRay Consortium, 2010).

4.4.2.3.3. Patrón 3: Símbolo WUDOP

El nodo admite la transmisión de un patrón de activación durante la operación (WUDOP, *Wakeup During Operation Pattern*), establecido para permitir activaciones durante la operación normal en el bus, permitiendo que un BD en estado de baja energía sea activado de manera remota. El WUDOP consta de una secuencia (baja, alta, baja, alta, baja) de duración gdWakeupTxActive, seguido de un gdBit en alto.

El nodo transmitirá un WUDOP sincronizando los flacos de bajada en TxEN y en TxD al comienzo en estado bajo, y con TxD volviendo a alto con un gdBit de tiempo antes que TxEN vuelva a alto al final del WUDOP, además no hay transmisión de TSS. Es importante considerar que antes y después de la transmisión del WUDOP, el receptor debe ver las fases de inactividad en el bus

mediante una configuración adecuada del AP y el tamaño de la ventana (FlexRay Consortium, 2010).

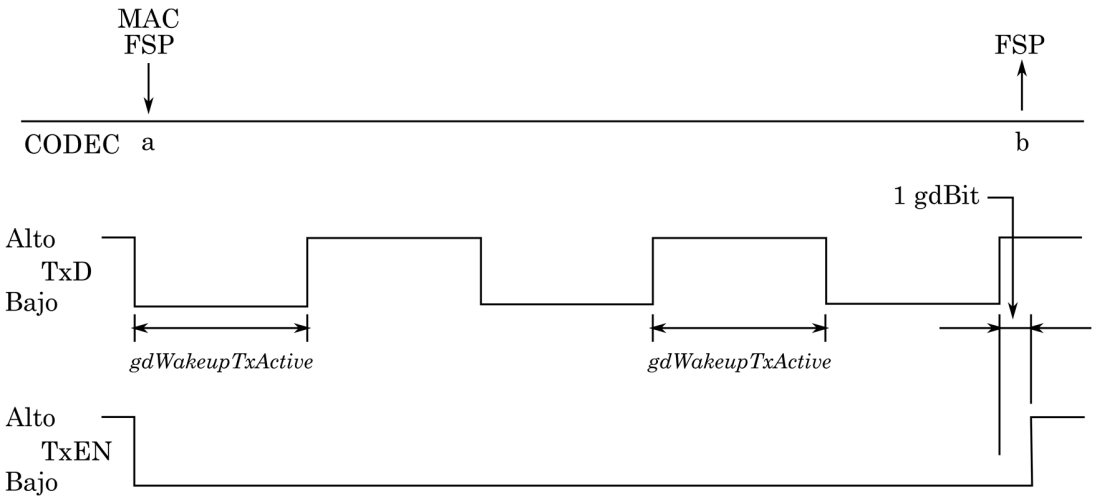


Figura 4.24. Patrón de activación WUDOP (FlexRay Consortium, 2010).

Capítulo 5. Desarrollo del Sistema FlexREY

En este capítulo se presenta el desarrollo del sistema FlexREY, siguiendo la metodología basada en el modelo de sistemas embebidos de Arnold Berger (2002). Cabe mencionar que únicamente se realizaron las primeras seis fases, debido a que la séptima fase está fuera del alcance de los objetivos del presente trabajo de tesis.

5.1. Especificaciones del Sistema FlexREY

El presente trabajo contempla el desarrollo de un sistema electrónico denominado FlexREY, que implementa una red basada en el protocolo FlexRay. Dicho sistema cuenta con dos nodos, dos paneles de pruebas y una GUI con capacidad de monitoreo de la red FlexRay, tal como se muestra en la Figura 5.1. El sistema permite validar el funcionamiento del protocolo de comunicaciones FlexRay ya que, mediante la GUI, se pueden enviar y recibir datos sobre la red, así como probar las condiciones de error preprogramadas para la tolerancia a fallos y con ayuda de los paneles se puede simular el funcionamiento del sistema *airbag* de un automóvil, en particular el sistema de protección contra impactos MRS III de la firma BMW.

5.1.1. Nodos FlexRay

Se implementaron dos nodos FlexRay, denominados Nodo A y Nodo B, ambos *coldstart*, utilizando una tarjeta LaunchPad Hercules TMS570LC43x, la cual brinda soporte al estándar FlexRay Rev 2.1 A en un módulo dedicado. Dicha tarjeta no integra el bus driver (BD), por lo que este se diseñó e implementó como un módulo acoplable basado en el transceptor TJA1080A (véase apartado 5.4.3) y se integró con el terminador de interfaz de conexión propuesto en el inciso 4.3.2.1 (véase la Figura 4.5).

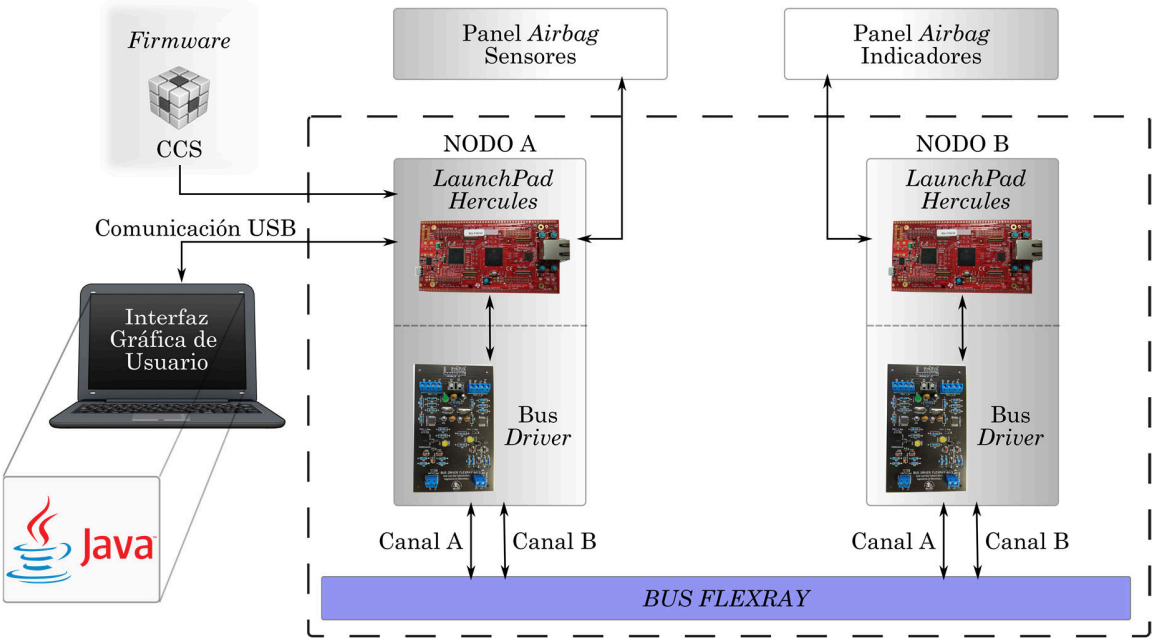


Figura 5.1. Diagrama a bloques del sistema FlexREY.

El nodo A se encarga de gestionar el panel de sensores, así como de la conexión serial con la interfaz de usuario, y de transmitir la información actual de los sensores al nodo B y recibir los datos de operación del MRS III.

El nodo B se encarga de simular el comportamiento del sistema MRS III y de controlar el panel de indicadores; además de recibir la información de los sensores del nodo A, también transmite los datos de estado y operación del MRS III.

La operación de los nodos A y B se basa en el esquema de sistemas operativos en tiempo real para sistemas embebidos, por lo que se asigna el tiempo de utilización de los recursos dividiéndolo en tres tareas controladas por TICK. operando de manera cíclica (véase la Figura 5.2).

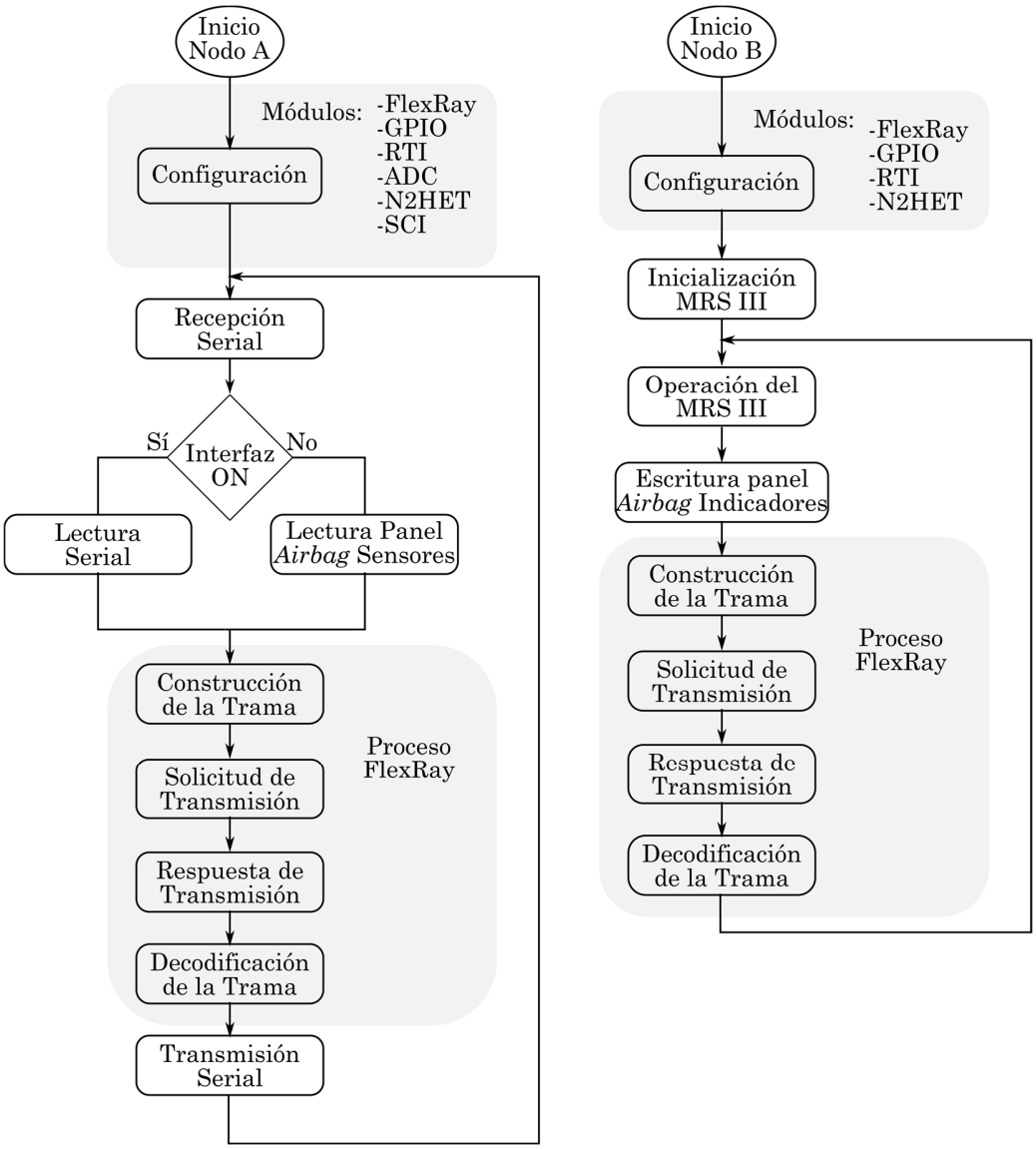


Figura 5.2. Diagrama de funcionamiento de los nodos A y B.

5.1.1.1. Comunicación FlexRay

La comunicación entre los nodos se basa en la implementación del núcleo E-Ray de TI, cuyas funciones se limitan a las requeridas por el sistema *airbag*. Se configuraron la unidad de tiempo global (GTU), los controladores de canal del protocolo (PRT), el manejador de mensajes (MHD), la configuración de mensajes en RAM (MRC, *Message RAM Configuration*) y el sistema de control universal (SUC); además, se emplea el acceso directo de la CPU a los búferes de entrada y de salida (VBUS IF), por lo que la configuración general es la siguiente:

- Duración de un ciclo de comunicación 5.6 ms, 224 000 *microticks* (μT) por ciclo y 5 600 *macrotick* (MT) por ciclo, 8 ranuras estáticas por ciclo con una duración de 86 MT, 346 mini ranuras por ciclo con una duración de 4 MT.
- Se establece la duración de la secuencia de inicio de transmisión (TSS, *Transmission start sequence transmitter*) en 10 gdBit (tiempo de bit), donde 1 gdBit = 4 μT = 100 ns, para una velocidad de transferencia de datos (*Baud rate*) de 10 Mbps, con un *BitStrobing* de 5, un periodo de muestreo de 12 ns, donde 1 μT = 25 ns, las muestras por μT = 2 para un reloj de muestra a 80 MHz.
- Una longitud de datos para trama estática de 9 dobles bytes y un máximo de 269 mini ranuras.
- 4 búferes para el segmento estático (búfer #0 al #3) y 24 búferes de mensajes.

Con lo anterior se consigue que la velocidad de comunicación sea de 10 Mbps, suficiente para cumplir con los requerimientos del sistema *airbag*, el cual gestiona tiempos de reacción menores a 25 ms.

El denominado proceso FlexRay consta de cuatro fases:

- Construcción de la trama: en la cual se asignan y configuran los identificadores, la carga útil, los CRC, el contador de ciclos y los datos de los segmentos estático y dinámico de la trama FlexRay (véase la Tabla 5.1). Para el nodo A, los datos son las lecturas de la GUI o de los sensores del panel, así como datos de configuración para el sistema MRS III, mientras que para el nodo B, son la información de estado y configuración del sistema MRS III y el estado panel *airbag*.
- Solicitud de transmisión: se escriben los registros necesarios para comenzar la transmisión de la trama vía FlexRay, hasta finalizar el envío.

- Respuesta de transmisión: se escriben los registros necesarios para iniciar la recepción de la trama de respuesta del nodo B.
- Decodificación de la trama: se toman y se procesan los datos recibidos para obtener la información de interés.

Tabla 5.1. Asignación de datos en las ranura y mini ranuras para construcción de la trama FlexRay.

Nodo	**Segmento Estático**		**Segmento Dinámico**	
A	Sensores de impacto	1	Sensores de ocupación (SBE)	10
	Sensores de seguridad	2	Interruptores del cinturón	11
	Aceleración	3	Valores de configuración MRS III	12
B	Estado del MRS III	4	Valores de configuración MRS III	13
			Estado del panel de indicadores	14

5.1.1.1.1. Software embebido del nodo

El software embebido de cada nodo se programó en lenguaje C para el MCU TMS570LC4357 de la tarjeta LaunchPad Hercules, utilizando las herramientas proporcionadas por TI, CCS (*Code Composer Studio*) y HALCoGen (*Hardware Abstraction Layer Code Generator*). Cada nodo cuenta con su propia secuencia de inicialización y configuración, así como de funciones específicas. El nodo A realiza las funciones: a) lectura para el panel *airbag* sensores y b) comunicación y empaquetamiento de tramas vía serial con la GUI. Por otro lado, el nodo B realiza las funciones: a) simulación comportamental del sistema MRS III (con base en la Figura 5.3, donde cada estado se comporta conforme a la Figura 5.4) y b) funciones de escritura para el panel *airbag* indicadores. Ambos nodos integran el sistema completo de simulación *airbag* al comunicarse mediante FlexRay (véase los Anexos H e I).

Ya que la firma TI no proporciona un API o biblioteca para el funcionamiento de su núcleo E-Ray TMS570LC4357, se desarrolló una biblioteca que contiene las funciones necesarias para implementar y utilizar el sistema como son: configuración y habilitación del módulo FlexRay, construcción y envío de tramas, recepción y decodificación de tramas, entre otras.

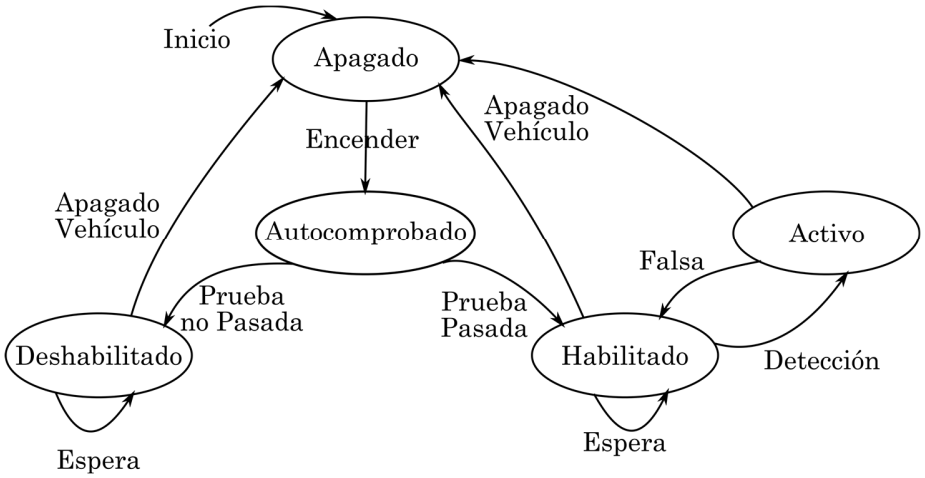


Figura 5.3. Diagrama de funcionamiento de sistema MRS III (BMW of North America, Inc., 2001).

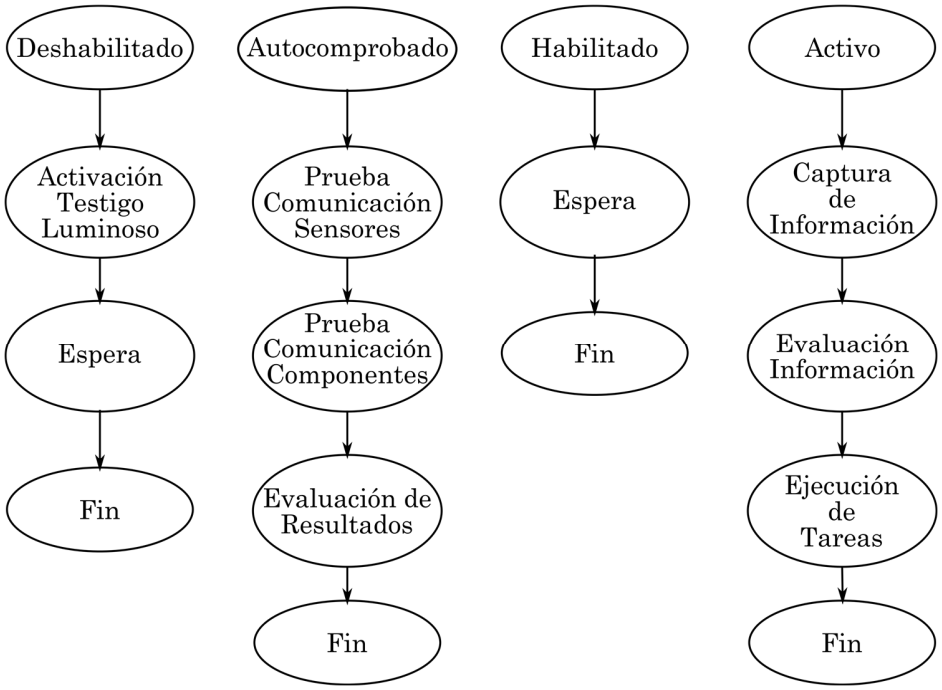


Figura 5.4. Principio de operación del sistema por estados.

5.1.2. Paneles de simulación del sistema airbag

Con base en el sistema MRS III de BMW, se desarrollaron dos paneles de prueba para simular el funcionamiento del sistema *airbag*; dichos paneles se conectan a los nodos A y B respectivamente, utilizando placas de circuito impreso (véase el Anexo A).

El primer panel se designa para los sensores del *airbag* que integran módulos de detección de impactos correspondientes a la distribución satelital frontal, lateral y trasera en el automóvil. Los sensores de seguridad Reed se sustituyeron por interruptores de palanca (1 polo 2 tiros), con lo que se puede probar cada sensor de impacto independientemente. Los sensores de ocupación de asientos (SBE) y los interruptores del cinturón de seguridad se implementan con interruptores de palanca. La velocidad de funcionamiento del automóvil se simula utilizando un potenciómetro como pedal de acelerador y luces LED para indicar los estados de la velocidad a los que corresponda la aceleración del pedal.

El segundo panel se utiliza para los indicadores de activación de los componentes del *airbag* de acuerdo con la respuesta del sistema MRS III, utilizando luces LED como indicadores de activación de algún elemento.

5.1.3. Interfaz gráfica de usuario

El desarrollo de la interfaz gráfica de usuario (GUI, *Graphic User Interface*) se realizó en lenguaje de programación Java mediante el entorno NetBeans, ya que este cuenta con una amplia gama de bibliotecas que ayudan a mejorar la velocidad de desarrollo y permiten optimizar las funciones de la interfaz, además de su soporte multiplataforma. La GUI cuenta con un estructurador de tramas que permite el envío y la recepción de mensajes personalizados, y la visualización de mensajes recibidos sobre el nodo A de la red FlexRay, al que se conecta mediante un puerto serie y se comunica por UART empaquetando los datos requeridos para la construcción de la trama; además permite alternar entre el monitoreo del estado actual del panel de sensores junto al panel de indicadores o simular el estado del panel de sensores para evaluar la respuesta del sistema.

5.2. Partición Hardware y Software

Las herramientas hardware utilizadas para cumplir con los objetivos del presente trabajo de tesis fueron las siguientes:

- Tarjeta de LaunchPad Hercules TMS570LC43x: se seleccionó por el soporte de periféricos de comunicaciones automotrices que brinda a los protocolos LIN, CAN y FlexRay; es uno de los pocos modelos de la familia LaunchPad Hercules que permite el acceso a los periféricos del módulo FlexRay del microcontrolador con salida a terminales en la tarjeta; además de su amplia gama de periféricos de comunicación, alta frecuencia de reloj, manejador de depuración integrado en la tarjeta, su compatibilidad con *boosterPack* y su bajo costo.
- Bus driver: se diseñó con base en las especificaciones de FlexRay Rev. 2.1 A como un módulo acoplable a la tarjeta Launchpad Hercules, anexando bloques de terminales para conexiones genéricas. El transceptor base para el desarrollo del BD es el TJA1080A que es utilizado en la literatura del tema. El desarrollo del módulo BD se llevó a cabo en tres etapas: a) diseño del circuito en esquemático y su correspondiente circuito impreso (PCB, *Printed Circuit Board*), b) fabricación del PCB y soldado de componentes electrónicos y c) pruebas de funcionamiento. Con esto se garantizó el cumplimiento de los requerimientos eléctricos en la capa física del protocolo FlexRay.
- Panel de *airbag*: se desarrolló para poder simular el sistema *airbag* de un automóvil con fines didácticos, utilizando sensores de vibración SW-4 e integrando como driver un microcontrolador ATMEGA328p, un potenciómetro resistivo, interruptores de palanca (1 polo 2 tiros) e indicadores de luces LED. El panel se utiliza en pruebas del protocolo FlexRay, pero no se limita a este ya que es compatible con los protocolos CAN o LIN.
- Computadora: ésta debe ser capaz de comunicarse con la tarjeta Launchpad Hercules, así como de soportar los programas de CCS, HALCoGen para el desarrollo del software embebido y de Java para

el desarrollo de la GUI, por lo que se seleccionó un equipo con procesador Intel Core i3 o superior, 4 o más GB de RAM, 50 GB libres para almacenamiento, puertos USB y sistema operativo MS Windows. Se procedió a instalar los controladores y programas requeridos dejando preparada la computadora para desarrollar el proyecto.

Respecto a las herramientas software utilizadas, se consideraron las siguientes:

- El software embebido para LaunchPad Hercules TMS570LC43x: se desarrolló en dos etapas, primeramente se utilizó HALCoGen para preestablecer los periféricos de entrada y salida, así como unidades de funcionamiento básico (reloj, temporizador, interrupciones, entre otros) del MCU, generando las bibliotecas preconfiguradas necesarias en CCS para el desarrollo del software embebido; a continuación, se implementaron manualmente las bibliotecas y las configuraciones requeridas por la unidad FlexRay integrada en el MCU, ya que ésta no es compatible con HALCoGen, con lo que el módulo FlexRay del MCU quedó habilitado permitiendo el envío y la recepción de datos de acuerdo con lo especificado en la capa de enlace de datos del protocolo FlexRay. La segunda etapa fue integrar las funcionalidades requeridas por el panel *airbag* y la GUI, utilizando las bibliotecas generadas por HALCoGen, logrando con ello que el MCU pueda adquirir y enviar datos externos, con lo que se puede simular una mayor cantidad de funciones para el nodo.
- GUI: se desarrolló utilizando el lenguaje de programación Java ya que permite simplificar la programación al contar con gran cantidad de herramientas y un fácil acceso a los recursos del sistema, además de ser portable para la ejecución en múltiples sistemas o plataformas. El desarrollo se llevó a cabo en tres etapas:

a) acceso y comunicación con los recursos hardware requeridos, b) generación del estructurador de tramas y c) diseño y programación de la GUI.

5.3. Iteración e Implementación

Con base en las especificaciones de las fases anteriores, se segmentaron los elementos del sistema FlexREY hasta formar módulos con funciones específicas, que al integrarse conforman el sistema completo, permitiendo un desarrollo modular con una mayor facilidad para realizar modificaciones.

En el diseño del HW se identifican los siguientes módulos funcionales:

- Módulos integrados en la tarjeta Hercules: módulo FlexRay, módulo GIO, módulo ADC, módulo SCI, módulo RTI y módulo N2HET.
- Módulos del panel *airbag* sensores: módulo de sensores de impacto, módulo de sensores de seguridad, módulo de sensores de SBE, módulo de sensores de cinturón y módulo de control de velocidad.
- Módulo del panel *airbag* indicadores: módulo de luces LED.
- Módulo bus driver.
- Fuente de suministro de energía.
- Computadora y cables de conexión.

En el diseño del SW se identifican los siguientes módulos funcionales:

- Software embebido: módulo de comunicación FlexRay, módulo de lectura de sensores, módulo de comunicación serial, módulo de simulación MRS III y módulo de control de indicadores.
- Software de la interfaz: módulo de comunicación serial y GUI.

5.4. Diseño Detallado Hardware y Software

Una vez segmentado el sistema en los módulos funcionales, se realizó el diseño a detalle de cada uno de ellos. Cabe destacar que para el funcionamiento correcto de los módulos se requiere de la integración del HW y del SW.

5.4.1. Programa principal del MCU

Debido a que cada nodo realiza diferentes tareas y utiliza diversos recursos, se requiere un software embebido especial para cada uno, los cuales se desarrollaron utilizando las herramientas *Code Composer Studio* y *HALCoGen* para generar dos proyectos independientes creando proyectos vacíos y utilizando el *HALCoGen* para programar las bibliotecas y código base de cada proyecto (Texas Instruments Incorporated, 2016).

5.4.1.1. Módulos integrados en tarjeta Hercules

Ya que estos módulos se encuentran integrados en el hardware del microcontrolador TMS570LC4357 y a que la tarjeta Hercules provee los periféricos requeridos para su utilización, dichos módulos requirieron ser habilitados y configurados vía software embebido, habilitando únicamente los componentes utilizados y configurando sus interrupciones con la herramienta HALCoGen.

Se configuró el módulo global de relojes utilizando el PLL1 para generar una señal de 300 MHz para el reloj global (GCLOK) y el PLL2 para generar una señal de 80 MHz para el reloj VCLKA2 requerido por el módulo FlexRay. Se habilitaron los módulos RTI, GPIO, SCI3, SCI1, ADC1 y HET1 para el nodo A y los módulos RTI, GPIO y HET1 para el nodo B. Se configuró el *PINMUX* para establecer la opción de salida de los terminales FlexRay, HET1, SCI1 y SCI3.

La configuración de los módulos para el nodo A se estableció como:

- Módulo RTI: Se configuró el comparador RTI1 y se configuraron los comparadores 0 y 1 a 80 ms y 6 ms, respectivamente.
- Módulo GIO: Se configuraron los bits 0, 1, 5, 6 y 7 del puerto A y los bits 3 y 7 del puerto B como terminales de salida y se habilitaron sus resistencias *pull-up*.
- Módulo ADC: Se configuró el grupo de conversión 1, habilitando una cola FIFO de 7 elementos y almacenamiento del ID del canal de

conversión, resolución 12 bits, el disparo de conversión por *timer* (RTI_COMP0) en flanco de subida habilitado por hardware, habilitación del capacitor de muestreo y selección de los canales de conversión (0, 7, 16, 18, 19, 20 y 21).

- Módulo SCI: Se habilitó la interrupción de entrada (Rx) y con una velocidad de transferencia de datos (*baud rate*) de 115,200 baudios para el SCI1 y de 9,600 baudios para SCI3; longitud de 8 bits, 2 bits de parada, sin paridad y se habilitó la terminal de salida (Tx) para SCI1 y SCI3, respectivamente.
- Módulo N2HET: Se configuraron las terminales 2, 16, 18 y 30 del HET1 con resistencias *pull-up* y se habilitó la salida de datos.

Por otro lado, la configuración de los módulos para el nodo B es la siguiente:

- Módulo RTI: Se configuró el comparador RTI1 y se configuraron los comparadores 0, 1 y 2 a 3,000 ms, 6 ms y 1,500 ms, respectivamente.
- Módulo GIO: Se configuró el bit 7 del puerto B como terminal de salida y se habilitó su resistencia *pull-down.*
- Módulo N2HET: Se configuraron las terminales 2, 10, 12, 14, 16, 18, 22, 25, 27, 28, 29 y 30 del HET1 con resistencias *pull-up* y se habilitó la salida de datos.

5.4.1.2. Proyecto nodo A

En el proyecto creado se identificó el archivo "HL_sys_main.c" (véase el Anexo H) que contiene la función *main*, en este archivo se desarrolló el código del programa principal para el MCU, con base en los requerimientos del sistema y conforme a la teoría de sistemas operativos en tiempo real, se programó la secuencia de un planificador con tres tareas principales: a) Lectura de datos para los sensores, b) Comunicación FlexRay y b) Comunicación con la GUI y se generó un TICK de 6 ms utilizando el *timer*.

Después de la configuración del hardware, el planificador lanza la primera tarea que verifica si la información de los sensores será tomada del panel de sensores (opción por defecto) o de la GUI; carga en una estructura de sensores los datos recibidos y espera a que el planificador envié la siguiente tarea con base en el TICK. A continuación, en la tarea 2 se realiza el proceso de transmisión y recepción de datos utilizando el protocolo FlexRay, cargando en el búfer de transmisión los datos de la estructura de sensores y después de recibir las respuestas obtiene la trama del búfer y almacena los datos en una estructura MRS_III; es importante destacar que la tarea 2 es una zona crítica de software. La tarea 3 se encarga de comunicarse con la interfaz y sí se recibió una trama serial dar una respuesta y verificar que sea uno de los siguientes casos válidos: a) cambio del modo de lectura, b) reconfiguración de umbrales del simulador MRS III, y c) transmisión de trama personalizada. A continuación, se encarga de estructurar una trama serial, con los datos actuales de los sensores y del estado de simulador MRS III, que se envían a la GUI para su visualización, como se muestra en la Figura 5.5.

5.4.1.3. Proyecto nodo B

En el proyecto creado se identificó el archivo "HL_sys_main.c" (véase el Anexo I) que contiene la función *main*, se generó un TICK de 6 ms utilizando el *timer* y se programó la secuencia del planificador sobre tres tareas principales: a) Comprobación de umbrales y sistema, b) Comunicación FlexRay y c) Configuración y ejecución del MRS III.

Al finalizar la configuración del hardware requerido y del simulador MRS III, el planificador inicia la primera tarea para comprobar los umbrales de activación y los carga en la estructura MRS_III sí los datos fueron solicitados, posteriormente, se verifica el estado del sistema y en caso de requerirse, lanza las correspondientes autocomprobaciones; finalmente, espera a que el planificador asigne la siguiente tarea con base en el TICK.

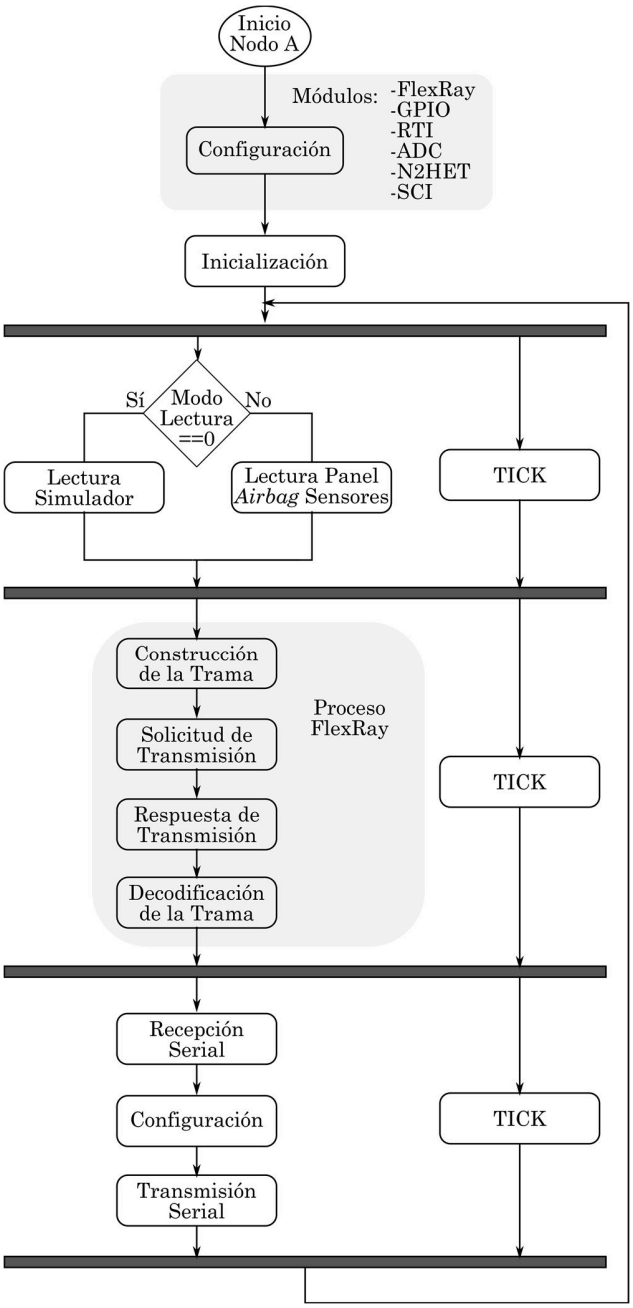


Figura 5.5. Diagrama de funcionamiento del nodo A.

La tarea 2, de igual manera que en el nodo A, se encarga de la comunicación utilizando el protocolo FlexRay. Finalmente, la tarea 3 reconfigura los umbrales de activación sí se ha enviado la orden y después evalúa que el estado del sistema sea el adecuado para comenzar la ejecución del simulador del MRS III para generar una respuesta apropiada y mostrarla utilizando el panel de indicadores (véase la Figura 5.6).

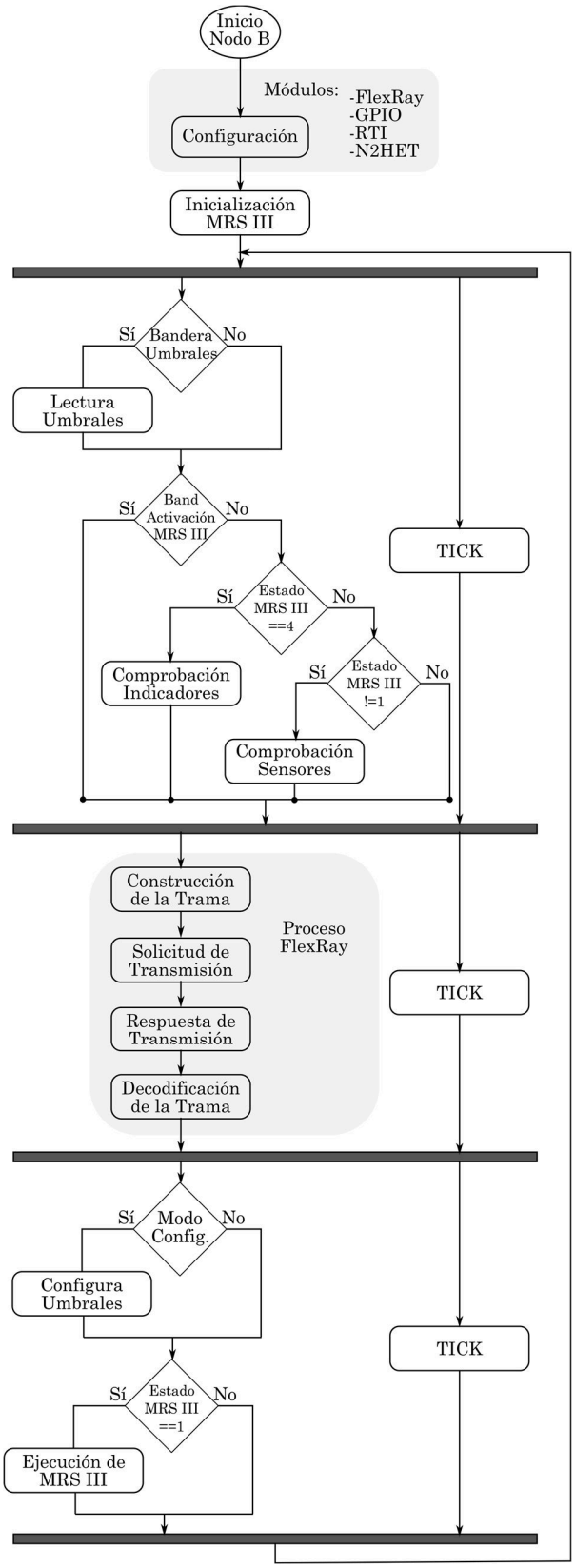


Figura 5.6. Diagrama de funcionamiento del nodo B.

5.4.2. Módulo FlexRay

El módulo FlexRay se encuentra integrado en el hardware del MCU por lo que requiere ser configurado mediante el software embebido; esta configuración se realizó de modo secuencial como se muestra en la Figura 5.7. Los Anexos B y C presentan las bibliotecas utilizadas.

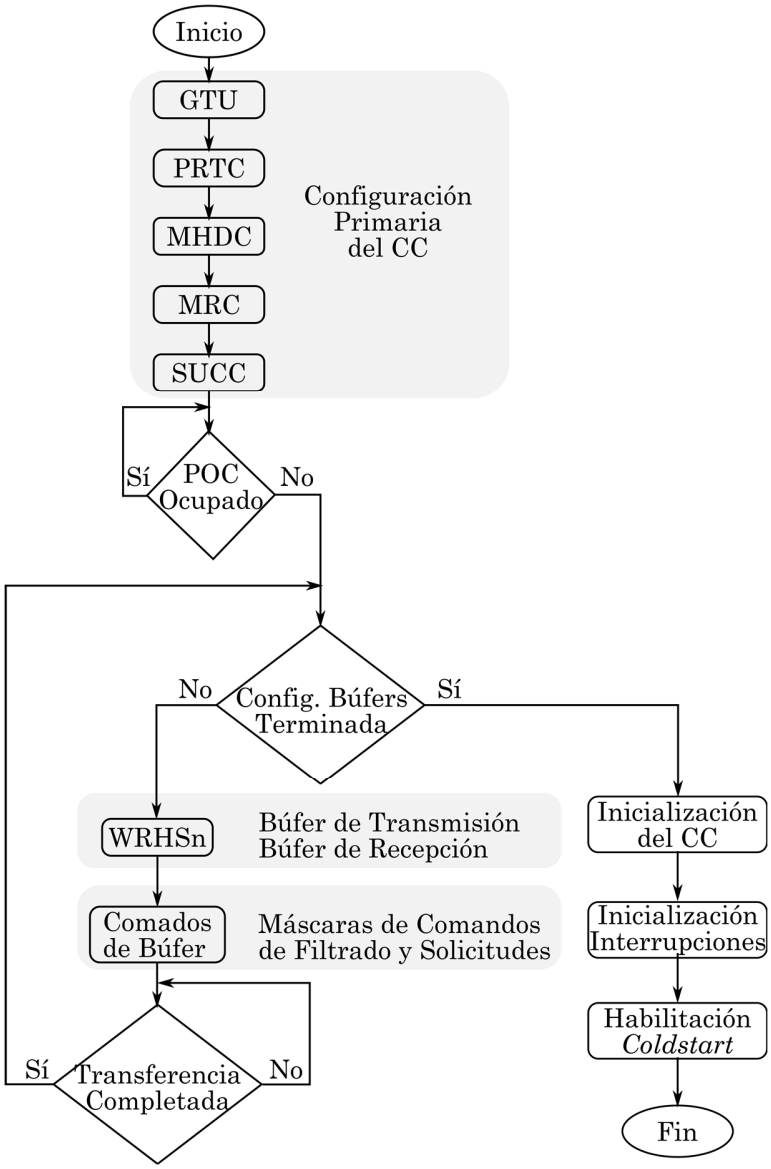


Figura 5.7. Diagrama de flujo de la configuración del módulo FlexRay.

5.4.2.1. Configuración por defecto del módulo FlexRay

La configuración primaria o, por defecto, se realizó escribiendo en los registros del CC los valores de configuración de acuerdo con la Tabla 5.2.

Tabla 5.2. Valores de configuración del CC.

Nombre	Registro	Valor (Hex)	Nombre	Registro	Valor (Hex)
Global Time Unit	GTU1	0x00036B00	-	GTU2	0x000F15E0
	GTU3	0x00061818		GTU4	0x0AE40AE3
	GTU5	0x33010303		GTU6	0x01510081
	GTU7	0x00080056		GTU8	0x015A0004
	GTU9	0x00010204		GTU10	0x015100CD
	GTU11	0x00000000	*Message Handler Configuration*	MHDC	0x010D000F
System Universal Control Configuration	SUCC1	0x0F1FFB00	-	SUCC2	0x0F036DA2
	SUCC3	0x000000FF	*Message RAM Configuration*	MRC	0x00174006
FlexRay Channel Protocol Controller Configuration	PRTC1	0x084C000A	-	PRTC2	0x3CB41212

5.4.2.2. Configuración de los búferes

Una vez terminada la configuración primaria se configuran globalmente los búferes escribiendo en los registros los valores que se muestran en la Tabla 5.3. Posteriormente, se configura cada búfer de manera individual, como se muestra en la Tabla 5.4, estableciéndose como un búfer de trasmisión (CFG=1) o de recepción (CFG=0), asignado al segmento estático o dinámico de la trama.

En primer lugar, en el segmento estático se configura el búfer de la ranura cero (IBRH=0), que también es el encargado de la transmisión de las tramas *startup* y de sincronización, por lo cual, se escribe a los bits CFG=1, RCI=1, SFI=1, SYN=1, CRC=calculado; también se asigna un identificador (FID) válido en el rango del segmento dinámico (1-8), el apuntador a la memoria (DP) y la carga útil configurada (PLC). Sólo un búfer puede transmitir las tramas *startup*

y de sincronización, por lo que los demás búferes escriben SFI=0 y SYN=0, además de asignar su propio identificador, apuntador de memoria y número de ranura de transmisión (en los bits IBRH) dentro del rango válido (0 a 5).

Tabla 5.3. Configuración global para búferes.

<table>
<tr><th colspan="4">WRHS1</th><th colspan="4">WRHS2</th></tr>
<tr><td>Bits</td><td>Valor</td><td>Bits</td><td>Valor</td><td>Bits</td><td>Valor</td><td>Bits</td><td>Valor</td></tr>
<tr><td>MBI</td><td>1</td><td>CHB</td><td>1</td><td rowspan="2">PLC</td><td rowspan="2">0</td><td rowspan="2">CRC</td><td rowspan="2">x</td></tr>
<tr><td>TXM</td><td>1</td><td rowspan="2">CYS</td><td rowspan="2">0</td></tr>
<tr><td>PPIT</td><td>0</td><td colspan="4">x: Calculado para la cabecera</td></tr>
<tr><td>CFG</td><td>0</td><td rowspan="2">FID</td><td rowspan="2">0</td><td colspan="4">WRHS3</td></tr>
<tr><td>CHA</td><td>1</td><td colspan="2">DP</td><td colspan="2">0</td></tr>
<tr><td colspan="4">IBCM</td><td colspan="4">IBCR</td></tr>
<tr><td colspan="3">STXRH</td><td>0</td><td colspan="3">IBRH</td><td>0</td></tr>
<tr><td colspan="3">LHSH</td><td>1</td><td colspan="3">IBSYH</td><td>1</td></tr>
<tr><td colspan="3">LDSH</td><td>0</td><td colspan="3">IBSYS</td><td>1</td></tr>
<tr><td colspan="4">OBCM</td><td colspan="4">OBCR</td></tr>
<tr><td colspan="3">RDSS</td><td>0</td><td colspan="3" rowspan="2">OBRS</td><td rowspan="2">0</td></tr>
<tr><td colspan="3">RHSS</td><td>0</td></tr>
</table>

Tabla 5.4. Configuración individual del búfer.

Número de Búfer	Registro	Valor (Hex)	Número de Búfer	Registro	Valor (Hex)
1 - Transmisión	WRHS1	0x27000001	2 - Transmisión	WRHS1	0x27000002
	WRHS2	0x0009029E		WRHS2	0x000907A2
	WRHS3	0x00000080		WRHS3	0x00000090
	IBCM	0x00010000		IBCM	0x00010001
	IBCR	0x00000000		IBCR	0x00010000
3 - Transmisión	WRHS1	0x27000003	4 - Recepción	WRHS1	0x23000004
	WRHS2	0x00090657		WRHS2	0x00090000
	WRHS3	0x00000100		WRHS3	0x00000200
	IBCM	0x00010001		IBCM	0x00010001
	IBCR	0x00020001		IBCR	0x00030001
10 - Transmisión	WRHS1	0x2500005A	11 - Transmisión	WRHS1	0x2500005B
	WRHS2	0x000904A9		WRHS2	0x0009055C

	WRHS3	0x00000110		WRHS3	0x00000120
	IBCM	0x00010001		IBCM	0x00010001
	IBCR	0x000A0001		IBCR	0x000B0001
12	WRHS1	0x2500005C	13	WRHS1	0x2100005D
	WRHS2	0x00090097		WRHS2	0x00090000
	WRHS3	0x00000130		WRHS3	0x00000210
	IBCM	0x00010001		IBCM	0x00010001
	IBCR	0x000C0001		IBCR	0x000D0001
14	WRHS1	0x2100005E			
	WRHS2	0x00090000			
	WRHS3	0x00000220			
	IBCM	0x00010001			
	IBCR	0x000E0001			

Para el segmento dinámico se escriben los ID en un rango de 90-100 y se asignan las ranuras en un rango de 10-15 y el apuntador a la memoria; además, se configura la transmisión/recepción sólo en el canal A, escribiendo en los bits CHA=1, CHB=0.

5.4.2.3. Inicialización del controlador de comunicaciones

Finalizada la configuración de los búferes, se inicializa el CC pasando del estado “Configuración por defecto” al estado “Configuración”, en donde se cargan las configuraciones del CC y se verifica que el POC no se encuentre ocupado o desconectado; una vez realizado lo anterior, se pasa al estado “Listo”.

5.4.2.4. Inicialización de interrupciones y habilitación coldstart

Se limpian los registros de interrupción (EIR, SIR, SILS), se reinician (*reset*) los estados de interrupción (SIER) y se habilita la interrupción de inicio del ciclo (SIES=0x4) para la interrupción 1 (int1) en el registro ILE.

Por otro lado, estando en el estado "Listo", el CC pasa al estado "*wakeup*" escribiendo CMD=0x9 en el vector de control de la interfaz *host,* para permitir la transmisión de las tramas *wakeup* y de sincronización en el estado "*startup*" que al finalizar regresa al estado "Listo".

5.4.2.5. Construcción de la trama

Con base en la trama del protocolo FlexRay mostrada en la Figura 4.17, se desarrolló una estructura de datos para asignar los campos requeridos y cumplir con los requerimientos del sistema *airbag*. De acuerdo con la Tabla 5.1, la comunicación ocupa cuatro ranuras para el segmento estático y cinco mini ranuras para el segmento dinámico, por lo el que sistema queda integrado con las funciones de ambos nodos como se muestra en la Figura 5.8.

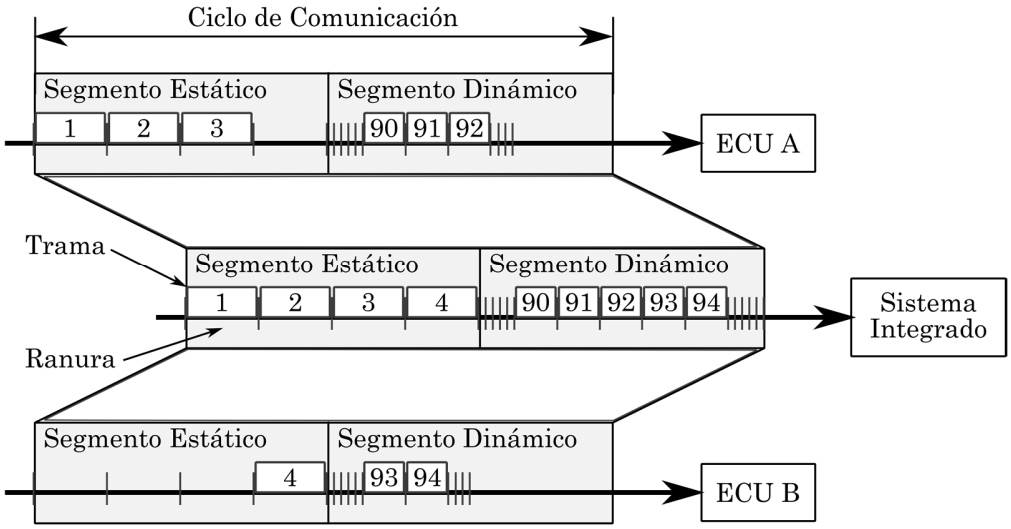


Figura 5.8. Integración de las funciones del sistema.

Mediante un arreglo de la estructura de la trama se consiguió modelar en código el sistema integrado para el manejo y monitoreo del sistema.

5.4.2.6. Transmisión de la trama

Se desarrolló la función de transmisión, que verifica la correcta comunicación con el POC, la cual escribe los bits STXRH=1, LDSH=1 para permitir la solicitud de transferencia, la carga de los datos de la "carga útil" en el búfer y asigna la ranura (IBRH) para la transmisión.

La carga de los datos en el búfer se realiza mediante una función que obtiene los datos de la trama y los escribe en el registro WRDSn conforme a la carga útil configurada y el segmento correspondiente. Al finalizar la carga de datos se escriben las órdenes de solicitud y las máscaras de búfer para comenzar la transmisión y espera hasta haber finalizado la transferencia.

5.4.2.7. Recepción de la trama

En la función de recepción, una vez realizada la comunicación con el POC, se verifica la recepción de tramas válidas, en caso de existir, se selecciona de acuerdo al número de ranura de transmisión su ranura de recepción (OBRS), se escriben los bits RDSS=1 y RHSS=1 para la lectura de la sección de cabeceras y de la carga útil.

La transmisión de los datos se realiza verificando que no existan transferencias en curso, se escriben las máscaras y órdenes del búfer para comenzar con la recepción de los datos en los registros RDHS, MBS y RDDS, que se leen y almacenan en la estructura de la trama en la memoria del arreglo de estructuras, con base en su ranura de recepción e identificador.

5.4.3. Módulo Bus Driver

El módulo bus driver (BD) se diseñó con base en las especificaciones establecidas utilizando las herramientas de EasyEDA, que proporcionan una interfaz para el diseño esquemático del circuito con una amplia variedad de bibliotecas de dispositivos, permitiendo generar el archivo fuente para el diseño del circuito impreso (PCB). El PCB se implementó como un diseño de dos capas utilizando dispositivos con tecnología de agujeros pasantes (THT, *Through-Hole*

Technology), a excepción de los dispositivos cuyas dimensiones sólo se encontraban con tecnología de montaje superficial (SMD, S*urface-Mount Technology*), contando con la implementación para utilizar tanto canal A y canal B, lo que implica la doble implementación del circuito.

5.4.3.1. Conexión y alimentación de la placa

Con base en los requerimientos del transceptor FlexRay TJA1080A se optó por integrar una etapa de alimentación para los múltiples voltajes requeridos para la operación del dispositivo, ya que requiere de +12 V, +5 V, +3.3 V, tomando como fuente principal la de +12 V; se añadieron reguladores de +5 V y +3.3 V con sus respectivos capacitores de filtrado, así como LEDs indicadores de alimentación.

Con base en la distribución de las terminales (*pinout*) de la tarjeta Hercules TMS570LC4357, se implementó una configuración de terminales para conectar la tarjeta en forma de *shield*; además, se agregaron "bloques de terminal" para permitir la conexión en caso de requerir conectar diferentes terminales y, de igual manera, se añadieron a las terminales de salida "bloques de terminal" y terminales para pruebas, y se incluyeron sus respectivos LEDs indicadores de error por canal de comunicación. Con base en el inciso 4.3.2.1, se implementó el circuito esquemático mostrado en la Figura 5.9.

5.4.4. Módulos del panel airbag sensores

El módulo del panel de sensores se diseñó con base en los requerimientos del sistema *airbag* para simular su funcionamiento, el cual consta principalmente de sensores de impacto, de seguridad, de ocupación (SBE), de cinturón de seguridad y de velocidad de movimiento, como se muestra en la Figura 5.10. Con base en la distribución del panel se diseñó una tarjeta PCB para montar todos los componentes requeridos por cada uno de los módulos, contando con las entradas de alimentación y las terminales de salida para la conexión con el nodo, como se muestra en la Tabla 5.5.

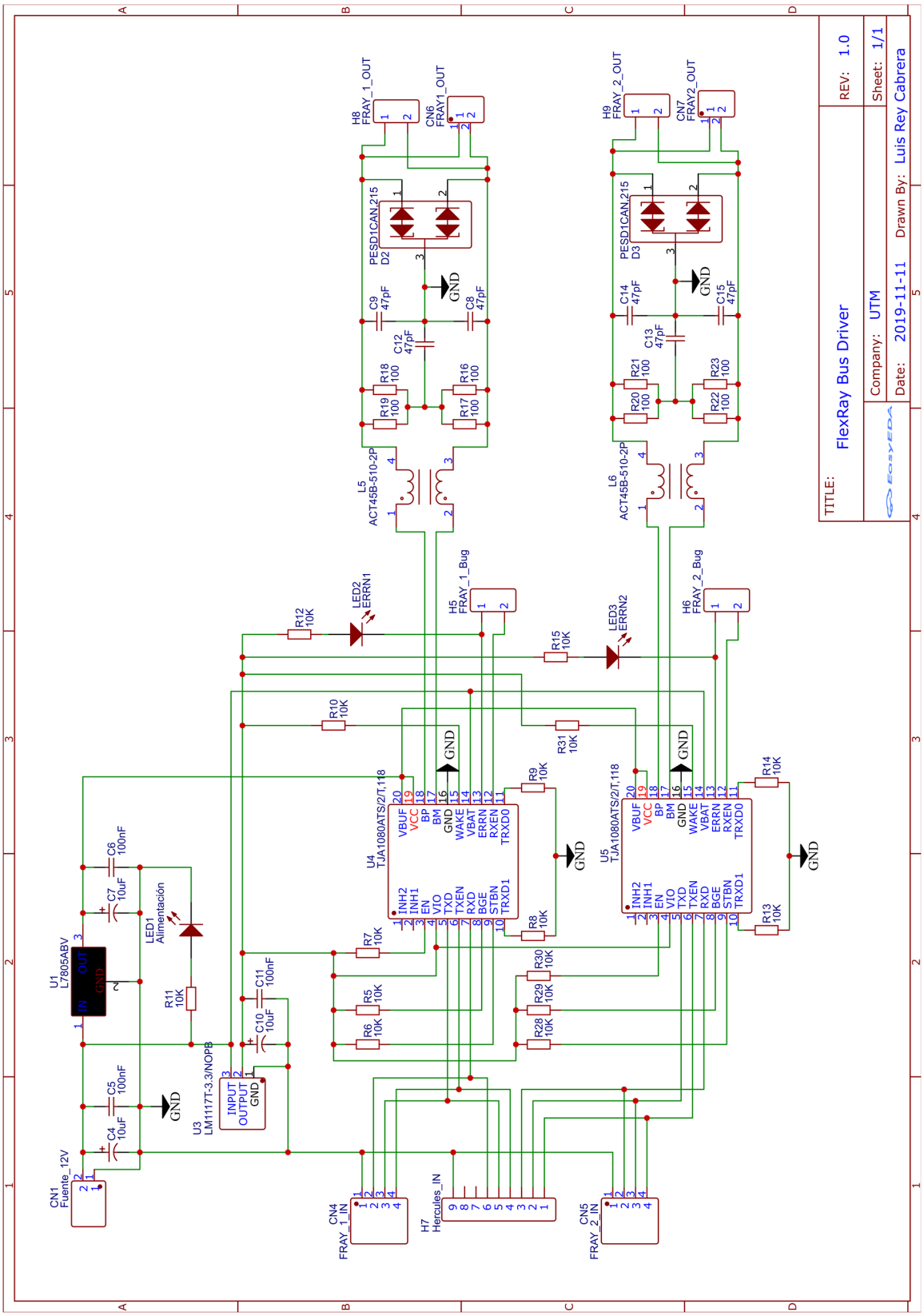


Figura 5.9. Diagrama eléctrico del Bus Driver.

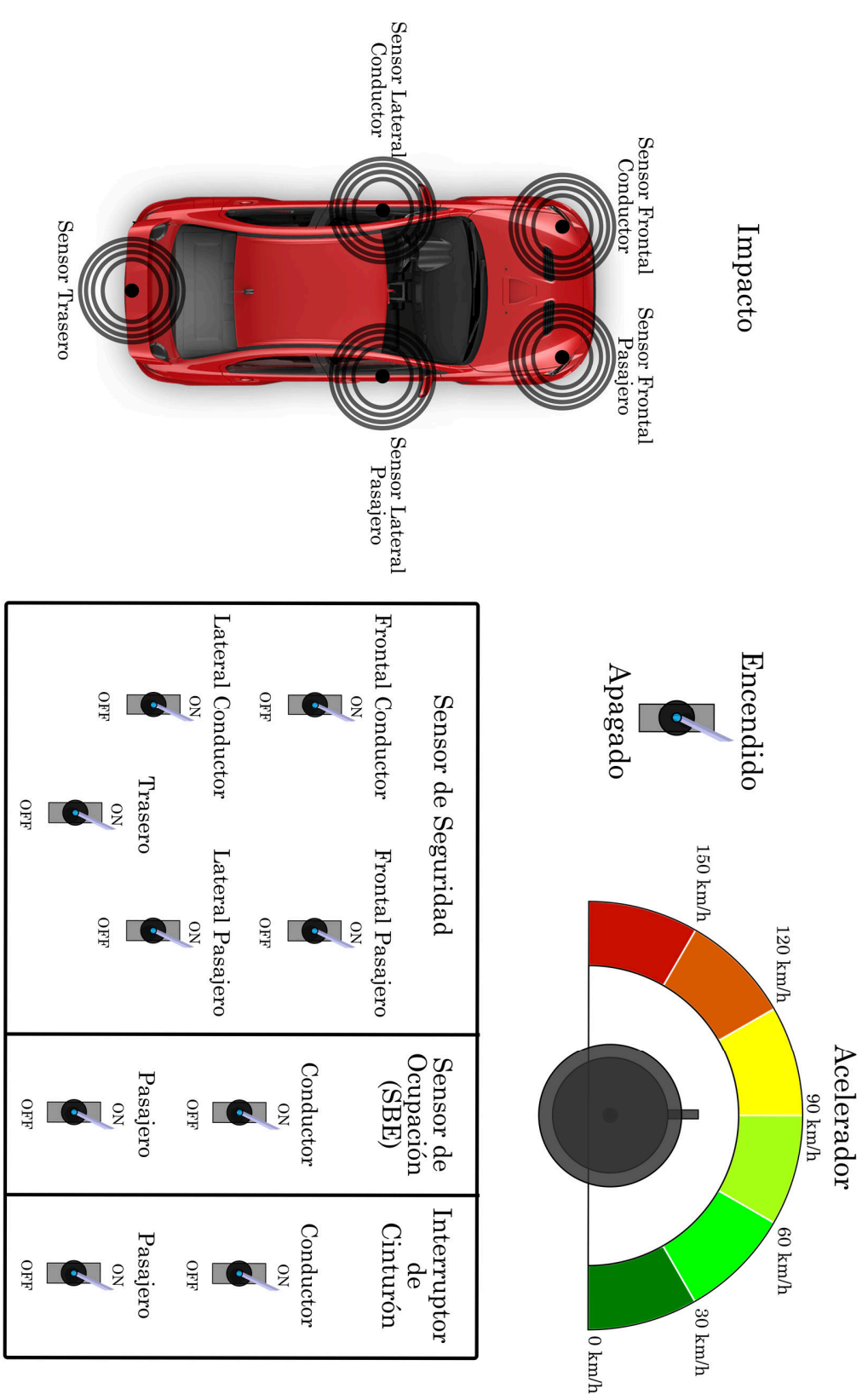


Figura 5.10. Módulo de panel sensores.

Tabla 5.5. Asignación de periféricos de los elementos del panel de sensores.

Elemento	Periférico	Terminal	
Sensores de Impacto	SCI3	SCI3RX	
Sensores de Seguridad	ADC1	Frontal conductor Lateral conductor Trasero Lateral pasajero Frontal pasajero	AD1[20] AD1[19] AD1[18] AD1[7] AD1[0]
Sensores SBE	N2HET1	SBE conductor SBE pasajero	N2HET1[30] N2HET1[16]
Interruptores de Cinturones	N2HET1	Cinturón conductor Cinturón pasajero	N2HET1[2] N2HET1[18]
Control de Velocidad	ADC1	AD1[21]	
	GPIO	LED_1 LED_2 LED_3 LED_4 LED_5 LED_6	GIOB[3] GIOA[0] GIOA[1] GIOA[5] GIOA[6] GIOA[7]

Para cumplir con los requerimientos se utilizaron tres sensores de impacto SW-420 como los sensores satelitales del automóvil, asignándolos como a) frontal conductor y frontal pasajero, b) lateral conductor y lateral pasajero, y c) trasero, además, se utilizó el microcontrolador ATMEGA328p como driver de control y comunicación de los sensores de impacto con el nodo.

El driver se programó para detectar y calcular la magnitud del impacto con base en el conteo de los flancos de la señal proporcionada por el sensor; este utiliza el reloj interno de 8 MHz y el periférico UART para la comunicación con el nodo; el *timer* 2 se configuró a 250 kHz así como su interrupción de desbordamiento, para generar un envió de datos cada 6 ms y el restablecimiento de los sensores cada 5 segundos, conectándose al nodo A vía SCI3 de la tarjeta Hercules.

Los cinco sensores de seguridad Reed se sustituyeron por interruptores de palanca, lo cual permite configurar distintos tipos de respuesta de seguridad al simular el impacto, asignándolos como sensor de seguridad frontal conductor,

frontal pasajero, lateral conductor, lateral pasajero y trasero, respectivamente. Se utilizó el módulo ADC1 de la tarjeta Hercules para la lectura de los interruptores, habilitando la función del capacitor de muestreo, lo anterior se llevó a cabo para aprovechar las terminales disponibles en dicha tarjeta.

Los sensores de ocupación SBE se sustituyeron con interruptores de palanca de 1 polo 2 tiros, asignándolos como SBE conductor y SBE pasajero; de igual manera se sustituyeron los sensores de los cinturones de seguridad por interruptores de palanca y se asignaron como cinturón conductor y cinturón pasajero. Para ello, se utilizó el periférico N2HET y los registros GPIO para la lectura digital de los interruptores en el nodo A.

Se añadió un interruptor de palanca como control de encendido del panel y se utilizó un potenciómetro resistivo de 100 kΩ, 6 LEDs como indicadores del estado del potenciómetro cuya función es simular el estado de la velocidad a la que se encuentra desplazándose el vehículo. La lectura del potenciómetro se realizó con el ADC1 de la tarjeta Hercules y se utilizaron las terminales GPIO para controlar los LEDs indicadores de la velocidad con base en la lectura del potenciómetro dentro del nodo A. La Figura 5.11 muestra la implementación electrónica del panel de sensores.

5.4.5. Módulo del panel airbag indicadores

El panel de indicadores se diseñó con base en el sistema MRS III y su función es proporcionar una respuesta visual del simulador como se muestra en la Figura 5.12, por lo que se contemplaron los siguientes indicadores:

- Testigo luminoso.
- Despliegue del *airbag* de dos etapas conductor.
- Despliegue del *airbag* de dos etapas pasajero.
- Activación ITS.
- Despliegue *thorax* conductor.
- Despliegue *thoras* pasajero.
- Activación de pretensor cinturón conductor.

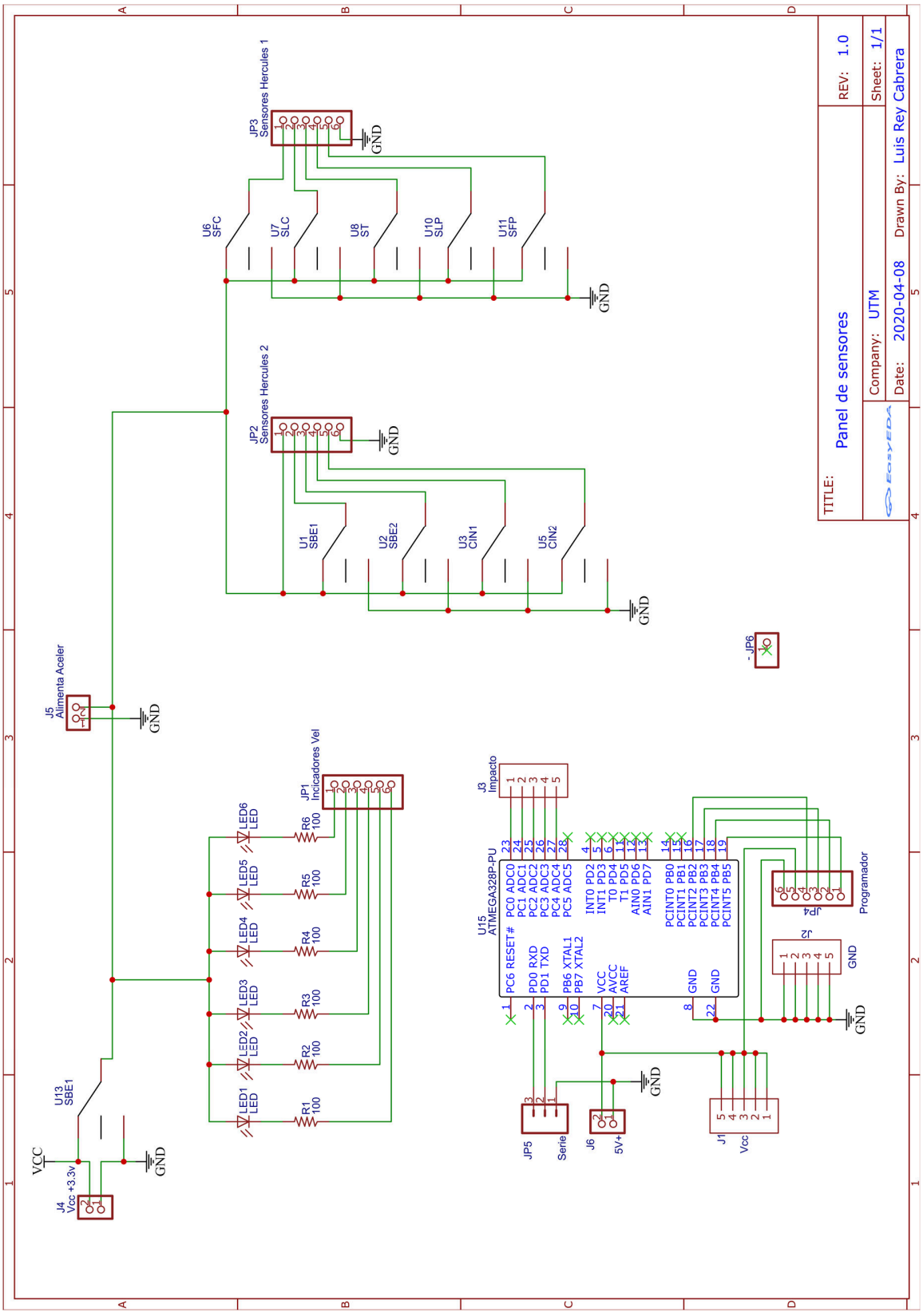


Figura 5.11. Diagrama eléctrico del panel de sensores.

- Activación de pretensor de pasajero.
- Activación de BST.

Los sistemas de activación mencionados se sustituyeron por indicadores LED, que representan la activación de dicho sistema en respuesta a la simulación de la operación del sistema MRS III. Con base en la distribución del panel se diseñó la placa PCB para el montaje de los indicadores, incluyendo su alimentación y terminales de conexión hacia el nodo B, para ello se utilizó el periférico N2HET y sus registros GIO para la escritura (véase la Tabla 5.6).

Tabla 5.6. Asignación de periféricos de los elementos del panel de indicadores.

Indicador	**Terminal del periférico N2HET1**
Testigo luminoso	N2HET1[2]
Despliegue *airbag* conductor E1	N2HET1[18]
Despliegue *airbag* conductor E2	N2HET1[16]
Despliegue *airbag* pasajero E1	N2HET1[30]
Despliegue *airbag* pasajero E2	N2HET1[14]
Activación ITS 1	N2HET1[12]
Activación ITS 2	N2HET1[22]
Thorax conductor	N2HET1[25]
Thorax pasajero	N2HET1[27]
Pretensor conductor	N2HET1[29]
Pretensor pasajero	N2HET1[10]
BST	N2HET1[28]

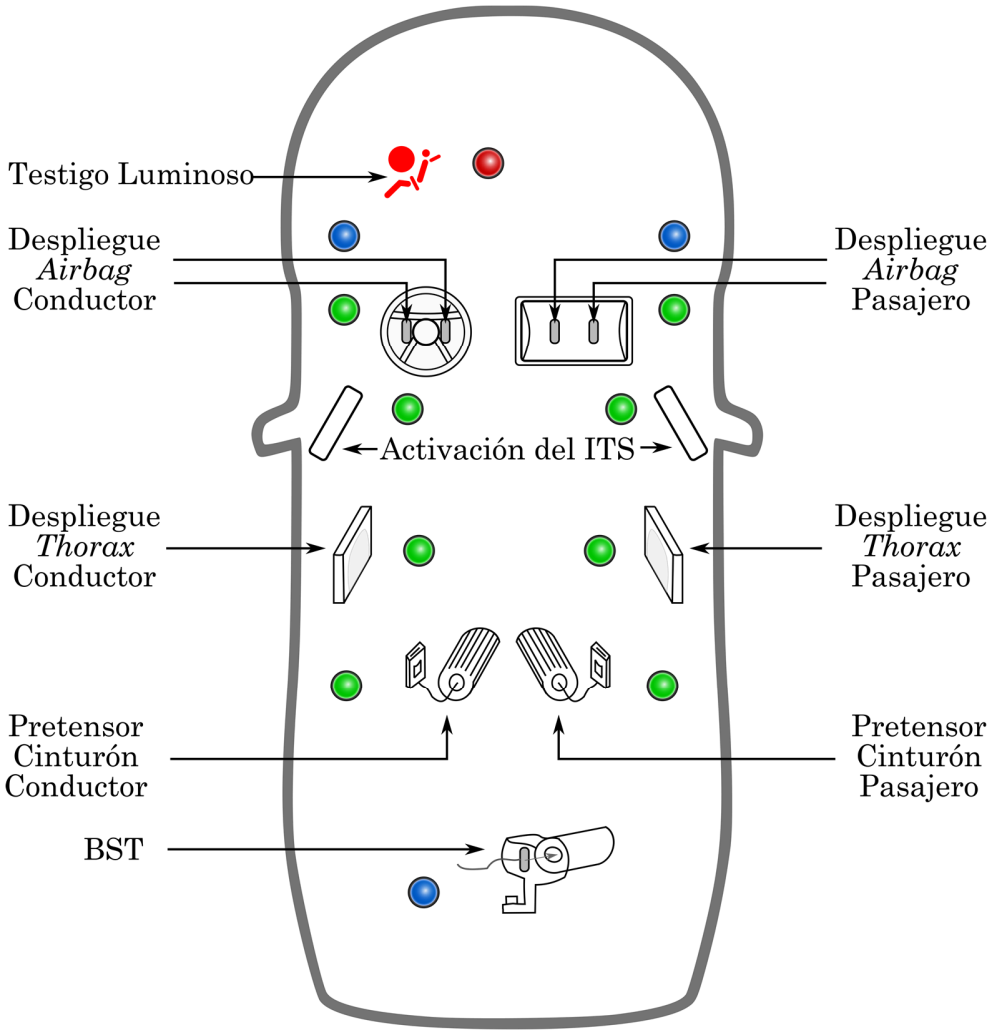


Figura 5.12. Diseño de panel indicadores (BMW of North America, Inc., 2001).

La Figura 5.13 muestra la implementación electrónica del panel de indicadores.

Para permitir el suministro de los diferentes voltajes requeridos por el sistema FlexREY, se optó por incluir una fuente de alimentación ATX de 250 Watts para alimentar las placas de los paneles de sensores, el panel de indicadores, y el bus driver FlexRay del nodo A y del nodo B.

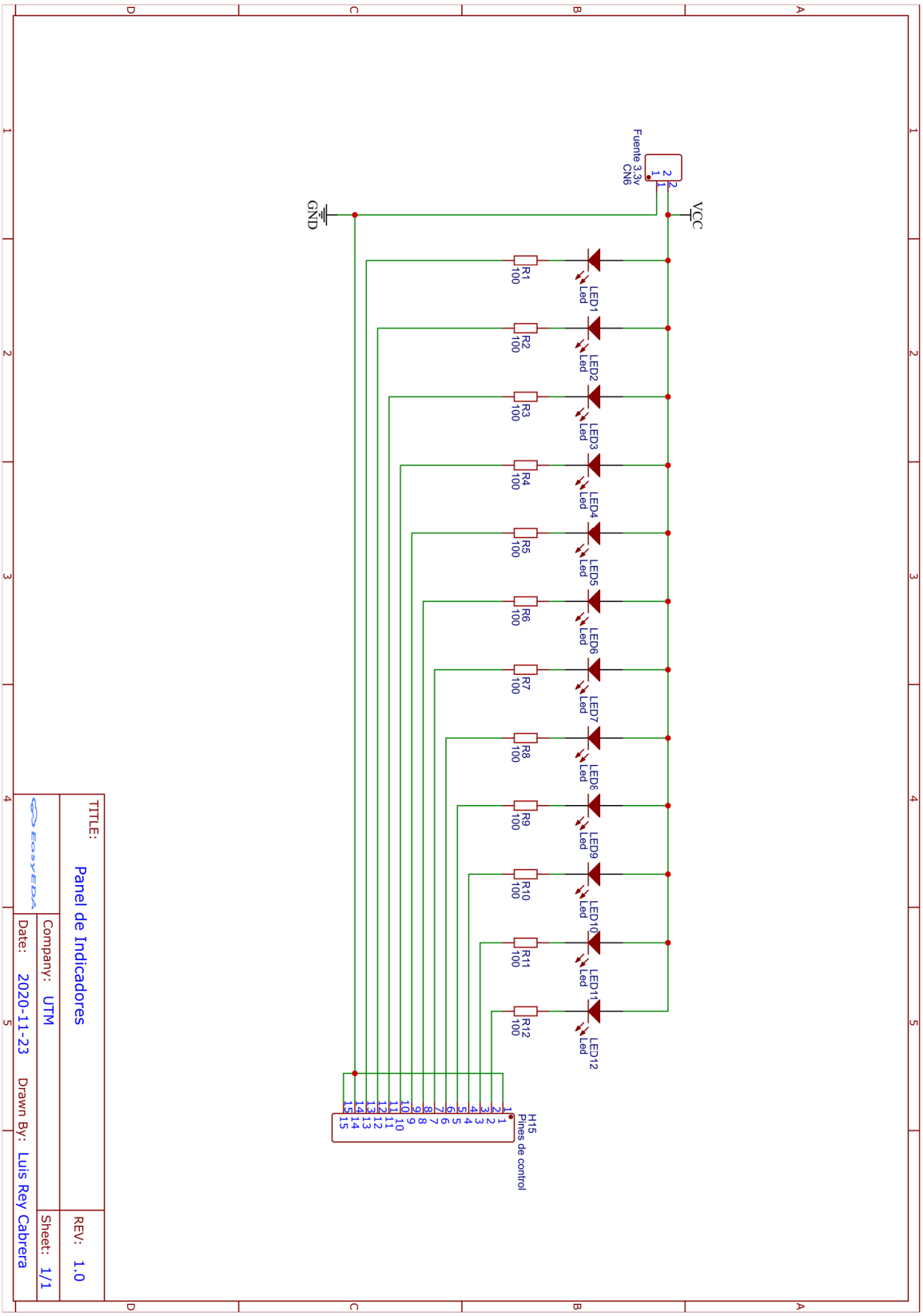


Figura 5.13. Diagrama electrónico del panel de indicadores.

5.4.6. Módulo de simulación MRS III

El sistema MRS III se diseñó como se especificó en el párrafo 5.1.1.1.1, implementándose en el nodo B de forma modular e independiente del programa principal (véase el Anexo I), lo que permite exportar con su respectivo archivos de código fuente (véase el Anexo E) y el archivo de biblioteca de funciones (véase el Anexo D), con las funciones disponibles (véase la Tabla 5.7) que integran la definición del tipo de dato *MRS_III* y contiene una estructura de datos con todos los elementos requeridos por el sistema.

Tabla 5.7. Funciones del simulador del sistema MRS III.

```
void config_Umbrales(trama_t* frame);
void set_Umbrales(MRS_t* umbrales);
void get_Umbrales(MRS_t* umbrales);
int autocomprobacion_panel_Indica();
int autocomprobacion_sensores();
void reset_Indicadores();
void set_Indicadores();
void ejecucion_MRSIII();
void ActivarIndica(panelact_t* panel, uint8_t retraso);
unsigned ActAirbag_Cond(uint8_t prioridad);
unsigned ActiAirbag_Pasj(uint8_t prioridad);
void whait(unsigned long value);
int verifica_Vel();
int desacelerador();
void initUmbrales();
```

Las funciones principales son: a) autocomprobación, b) lectura y configuración de umbrales y c) ejecución del sistema. La autocomprobación se encarga de la verificación de la conexión de los sensores y de los indicadores del sistema, para garantizar su correcto funcionamiento y en caso de no pasar las pruebas; el sistema debe mantenerse deshabilitado, como se muestra en la Figura 5.3. La lectura y escritura de los umbrales permite la configuración y comprobación del estado de los umbrales de operación del sistema que, de ser necesario, pueden ser reconfigurados para adecuar el funcionamiento del sistema a posibles variaciones. Finalmente, la ejecución del sistema se encarga de evaluar los datos proporcionados por los sensores y los umbrales de activación para determinar la respuesta adecuada del sistema que será visualizada en el panel de indicadores.

5.4.6.1. Determinación de la respuesta del sistema

La respuesta para la desaceleración se calcula al evaluar primeramente que la variación de velocidad sea mayor a 30 km/h; después, el estado del sensor de ocupación de asiento y finalmente el estado del interruptor de los cinturones que, en caso de ser positivos, cambia el estado en la estructura de datos MRS_III para la activación del pretensor correspondiente.

La respuesta para los choques se calcula verificando que la velocidad del vehículo sea mayor al mínimo de 30 km/h que corresponde a cada indicador del módulo de sensores como se muestra en la Figura 5.10, sí el sensor de ocupación se activa, la fuerza registrada en los sensores de impacto se compara con los umbrales de activación del *airbag*, así como el estado del interruptor del cinturón y se selecciona la respuesta indicada como se muestra en la Tabla 3.1 y se cambia el modo de operación a activo. Este proceso se realiza independientemente para el conductor y para el pasajero, con sus correspondientes sensores. Por otro lado, la respuesta para el BST se calcula al comparar el umbral de activación del BST con todos los sensores de impacto y de igual manera la activación del ITS con su respectivo umbral de activación.

Finalmente, teniendo todas respuestas de cada sistema ante el impacto, se manda la instrucción de activación de los correspondientes indicadores en el panel de indicadores. Se añadió además un *timer* para el reinicio del simulador con fines de pruebas, ya que la respuesta real del sistema dicta que el sistema una vez activado, se mantiene en ese estado hasta ser reemplazado completamente y reiniciado.

5.4.7. Diseño de la GUI

El desarrollo de la GUI se llevó a cabo utilizando el IDE NetBeans en lenguaje JAVA, partiendo de un proyecto del tipo Java *Application* vacío y generando un JFrame principal para la base de la interfaz. Se utilizaron JDialog para las ventanas emergentes requeridas, se añadió la biblioteca JSerialComm-

2.7.0 que integra los métodos de comunicación serial requeridos para la aplicación y se crearon las clases Panel y Monitor, cuyas funciones son proporcionar integración entre el diseño del panel de pruebas y el control de la interfaz en tiempo real mediante la utilización de hilos, respectivamente.

5.4.7.1. Operación

Con base en la clase Panel, se creó un objeto que contiene los elementos relacionados con el funcionamiento de los paneles, estructurando de esta manera los datos; además, se inicializan los componentes de la interfaz permitiendo múltiples procesos (hilos), cuyas funcionalidades integran la interfaz y pueden dividirse en cuatro procesos: a) Interfaz, b) Monitor, c) Comunicación y d) Salir, como se muestra en la Figura 5.14.

El proceso Interfaz se encarga de desplegar en pantalla los elementos gráficos de la GUI, permitiendo interactuar con los objetos de la interfaz para simular el comportamiento de los sensores del panel o generar una trama personalizada FlexRay, además de enviar esta información vía puerto serie, siempre que cumpla con los parámetros adecuados, y mostrar la información para su verificación en la interfaz, como se muestra en la Figura 5.15 inciso A.

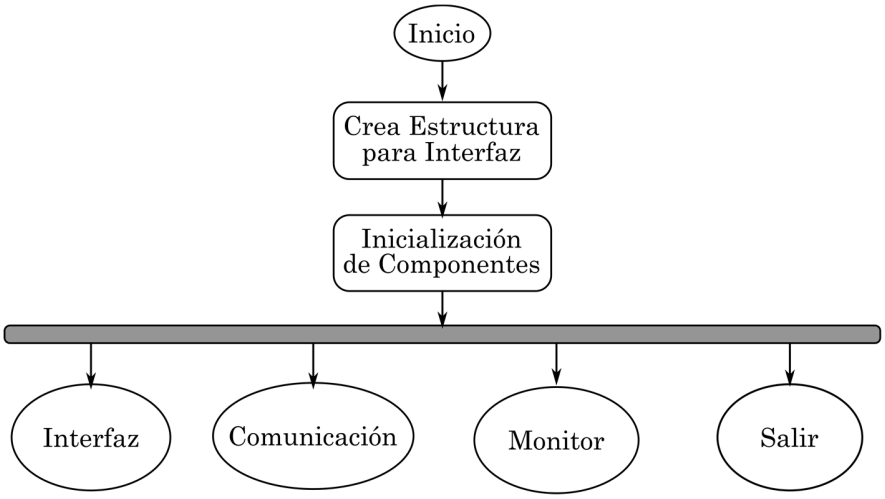


Figura 5.14. Diagrama de operación de interfaz.

Por otro lado, el proceso denominado Comunicación se encarga de esperar el inicio de la comunicación para iniciar el hilo encargado de ella y verificar que

la comunicación serial se encuentre activa con base en el *dataListener*; al ocurrir el evento de recepción de datos se lee el búfer para almacenar la información en la estructura (objeto tipo Panel) y se despliegan visualmente en la interfaz de acuerdo con la Figura 5.15 inciso B.

El tercer proceso denominado Monitor se encarga de iniciar un hilo cuya tarea es mantener actualizada la interfaz en tiempo real, por lo que, con base en el estado de la conexión, este habilita o deshabilita las funcionalidades de la interfaz. Además, actualiza la estructura del panel de control de acuerdo al modo de operación y los elementos de la GUI, con base en los datos de la estructura como se muestra en la Figura 5.16 inciso A.

Finalmente, el último proceso, denominado Salir, se encarga de terminar todos los procesos que se estén ejecutando y finalizar correctamente la ejecución del programa como se muestra en la Figura 5.16 inciso B.

5.4.7.2. Interfaz gráfica de usuario

La GUI cuenta con la ventana principal y su barra de menús que son: a) Conexión, b) Monitor y c) Ayuda; además, cuenta con dos paneles; el primero para la simulación de los sensores y el segundo para la construcción de la trama personalizada FlexRay, además de dos áreas de texto para visualizar los datos enviados y recibidos, respectivamente.

En el menú Conexión se encuentran las funciones Sincronizar que se utiliza en caso de requerir volver a sincronizar la conexión de la interfaz con el nodo A del sistema; Buscar, que permite desplegar una ventana con los puertos seriales disponibles para seleccionar y establecer el inicio de la comunicación; Desconectar, encargada de terminar la comunicación serie y cerrar el puerto utilizado y Salir, que termina la ejecución del software.

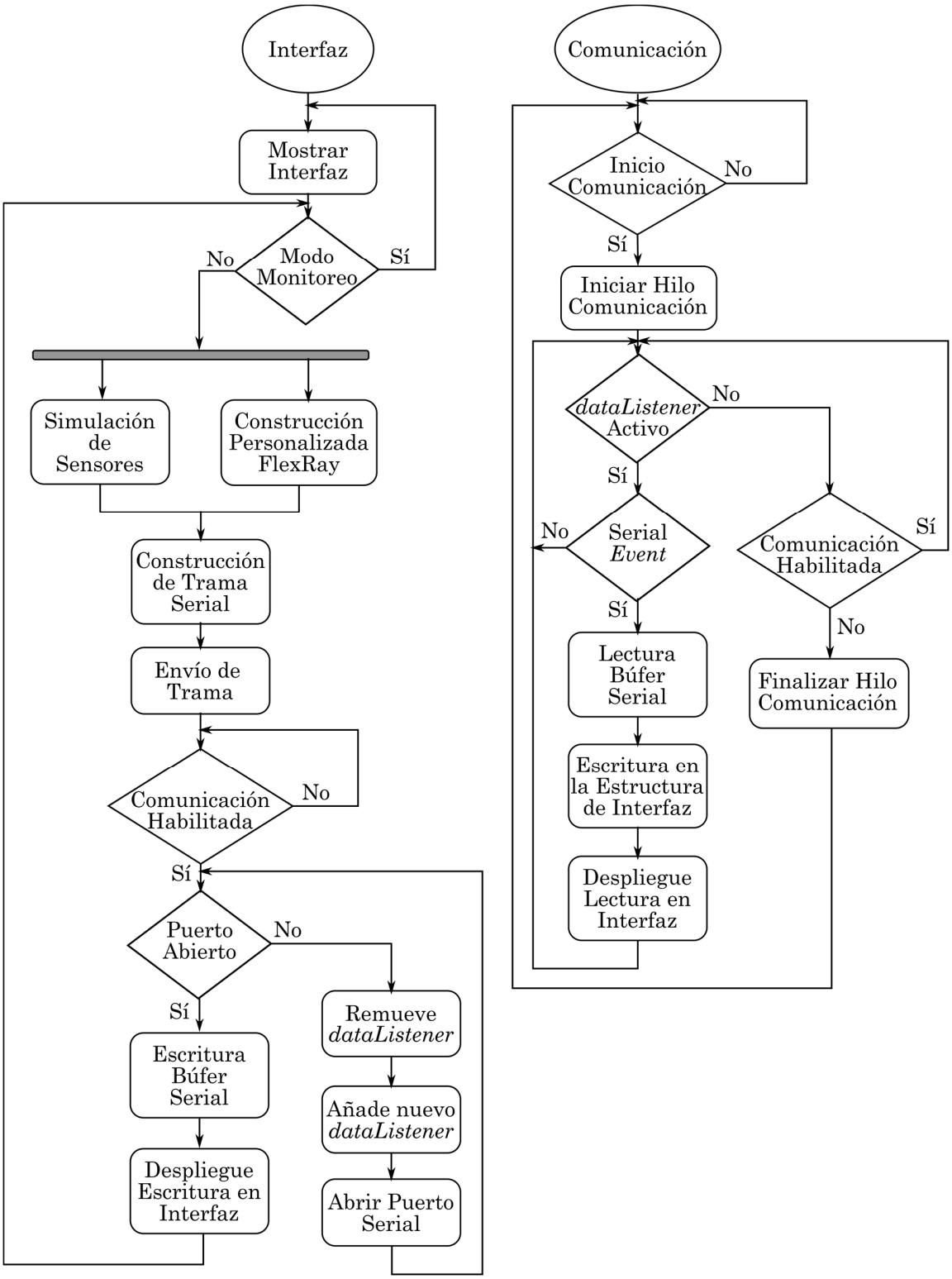


Figura 5.15. Diagrama de funcionamiento a) Proceso Interfaz, b) Proceso Comunicación.

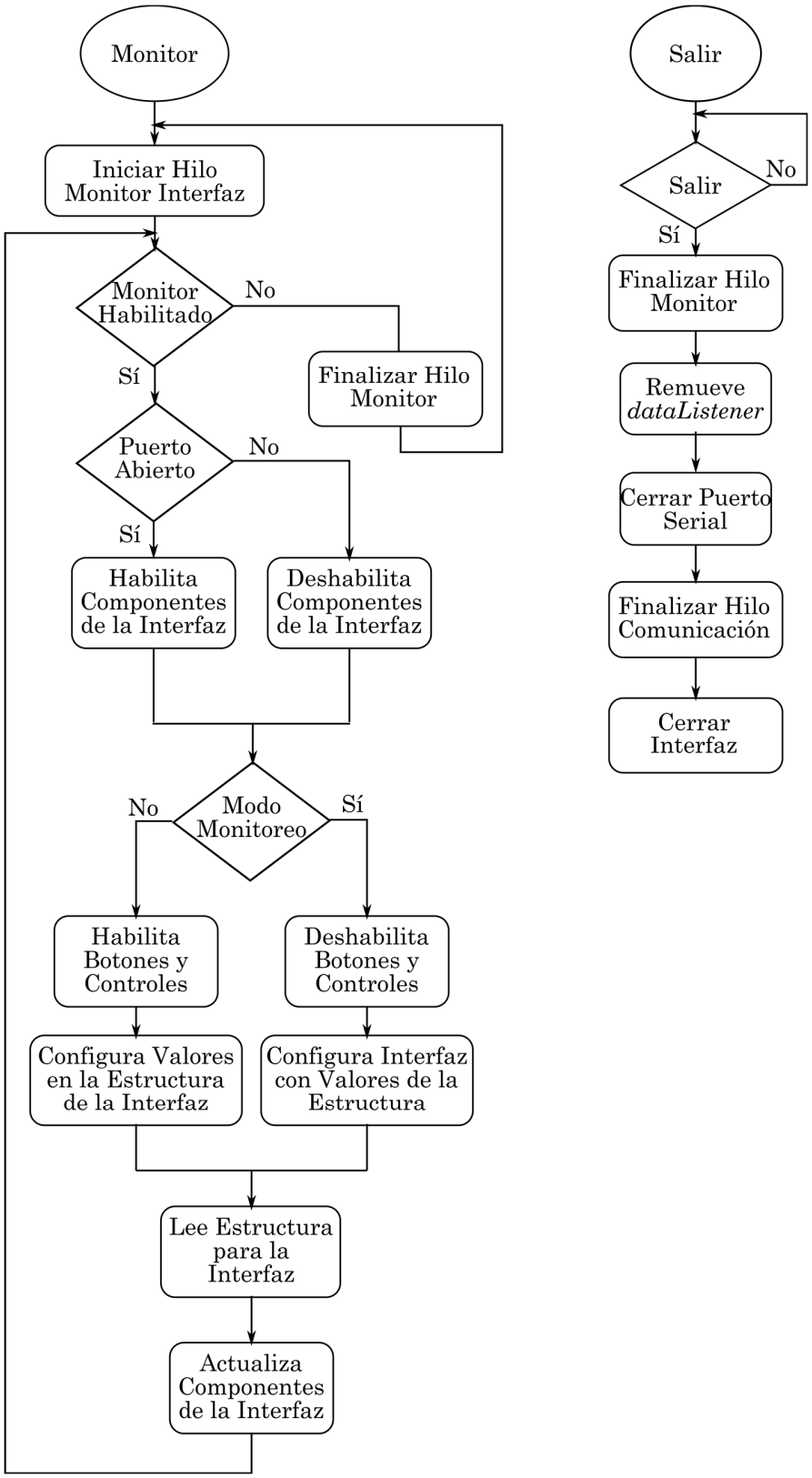


Figura 5.16. Diagrama de funcionamiento a) Proceso Monitor, b) Proceso Salir.

El menú Monitor permite alternar entre el modo Monitor de sensores y Simulador de sensores, además de desplegar la ventana de monitoreo del panel de indicadores y permite limpiar las áreas de texto de los datos en la ventana principal. El menú Ayuda permite desplegar ventanas de ayuda sobre la trama, el sistema e información del software.

5.4.7.3. Módulo de comunicación serial

El módulo de comunicación serial se basa en la biblioteca JSerialComm-2.7.0, la cual proporciona las funciones básicas para la utilización del periférico serial del equipo, pero requiere la implementación de la clase *MessageListener* para poder habilitar las funciones de lectura por eventos y mensajes delimitados, además, de la implementación de la interrupción respectiva.

La conexión se ejecuta en un nuevo hilo que es implementado automáticamente por la biblioteca una vez que se añade el *Listener* al puerto serial ya abierto y finaliza al removerlo. Las Tablas 5.8-10 listan las tramas implementadas.

5.5. Integración HW y SW del Sistema FlexREY

Debido a que la clave del diseño de sistemas embebidos es la integración del prototipo HW con el software embebido y la aplicación SW, una vez definidas las funciones de cada módulo fue necesario configurar los módulos para vincular los componentes HW con las rutinas desarrolladas en SW, como se muestra en la Figura 5.17 y la Figura 5.18, esto consistió en realizar las siguientes tareas:

- Descargar en la tarjeta Hercules el software embebido para habilitar los módulos HW.
- Verificar la conexión entre los componentes del HW.
- Ejecutar pruebas en periféricos y de conexión con la interfaz.
- Descarga del programa principal al MCU.
- Ejecutar el programa principal y validar su funcionamiento.

Tabla 5.8. Tramas del nodo A.

Tarea	Segmento	ID	Carga Útil	Trama
Sensores de Impacto	Estático	0x01	5 bytes	0x03 0x01 0x05 CRC Cont_Cic FrontC FrontP LatC LatP Tras Trailer_CRC
Sensores de Seguridad	Estático	0x02	5 bytes	0x00 0x02 0x05 CRC Cont_Cic FrontC FrontP LatC LatP Tras Trailer_CRC
Velocidad y Encendido	Estático	0x03	2 bytes	0x00 0x03 0x02 CRC Cont_Cic Velocidad Encendido Trailer_CRC
Sensores SBE	Dinámico	0x5A	2 bytes	0x00 0x5A 0x02 CRC Cont_Cic SBE_Conductor SBE_Pasajero Trailer_CRC
Sensores de Cinturón	Dinámico	0x5B	2 bytes	0x00 0x5B 0x02 CRC Cont_Cic Cinturón_Cond Cinturon_Pasj Trailer_CRC
Configuración MRS III	Dinámico	0x5C	4 bytes	0x00 0x5C 0x06 CRC Cont_Cic Estado Umb_Pretensor Umb_acti_front Umb_Trasero Trailer_CRC
Trama personalizada	Dinámico	0x5F - 0x64	n-bytes	00000 ID CRC Cont_Cic n-Bytes Trailer_CRC

Tabla 5.9. Tramas del nodo B.

Tarea	Segmento	ID	Carga Útil	Trama
Estado MRS III	Estático	0x04	2 bytes	0x03 0x04 0x02 CRC Cont_Cic Habilitación Estado Trailer_CRC
Configuración MRS III	Dinámico	0x5D	6 bytes	0x00 0x5D 0x06 CRC Cont_Cic Estado Umb_Pretensor Umb_acti_front Umb_Trasero Umb_BST Umb_ITS Trailer_CRC
Estado de Indicadores	Dinámico	0x5E	2 bytes	0x00 0x5E 0x02 CRC Indicadores_Bajo Indicadores_Alto Trailer_CRC

Tabla 5.10. Tramas para la GUI.

Tarea		ID	Carga Útil	Trama
Modo Simulación de Sensores	Envío	0x09	17 bytes	0x00 0x09 0x11 0x01 Encendido Velocidad Imp_FrontC Imp_FrontP Imp_LatC Imp_LatP Imp_Tras Seg _FrontC Seg _FrontP Seg _LatC Seg _LatP Seg_Tras SBE_Conductor SBE_Pasajero Cinturón_Cond Cinturon_Pasj
Modo Monitoreo	Envío	0x09	17 bytes	0x00 0x09 0x11 0x00 Encendido Velocidad Imp_FrontC Imp_FrontP Imp_LatC Imp_LatP Imp_Tras Seg _FrontC Seg _FrontP Seg _LatC Seg _LatP Seg_Tras SBE_Conductor SBE_Pasajero Cinturón_Cond Cinturon_Pasj
Configuración MRS III	Envío	0x5C	5 bytes	0x00 0x5C 0x03 Umb_Pretensor Umb_acti_front Umb_Trasero
Trama Personalizada	Envío	ID	n-bytes	Seg_Startup ID n-bytes
Monitoreo	Recepción	0x12	18 bytes	0x00 0x0A 0x12 Encendido Velocidad Imp_FrontC Imp_FrontP Imp_LatC Imp_LatP Imp_Tras Seg _FrontC Seg _FrontP Seg _LatC Seg _LatP Seg_Tras SBE_Conductor SBE_Pasajero Cinturón_Cond Cinturon_Pasj Indicadores_Bajo Indicadores_Alto
Response Interfaz	Recepción	0x0B	1 bytes	0x00 0x0B 0x01 0x2B

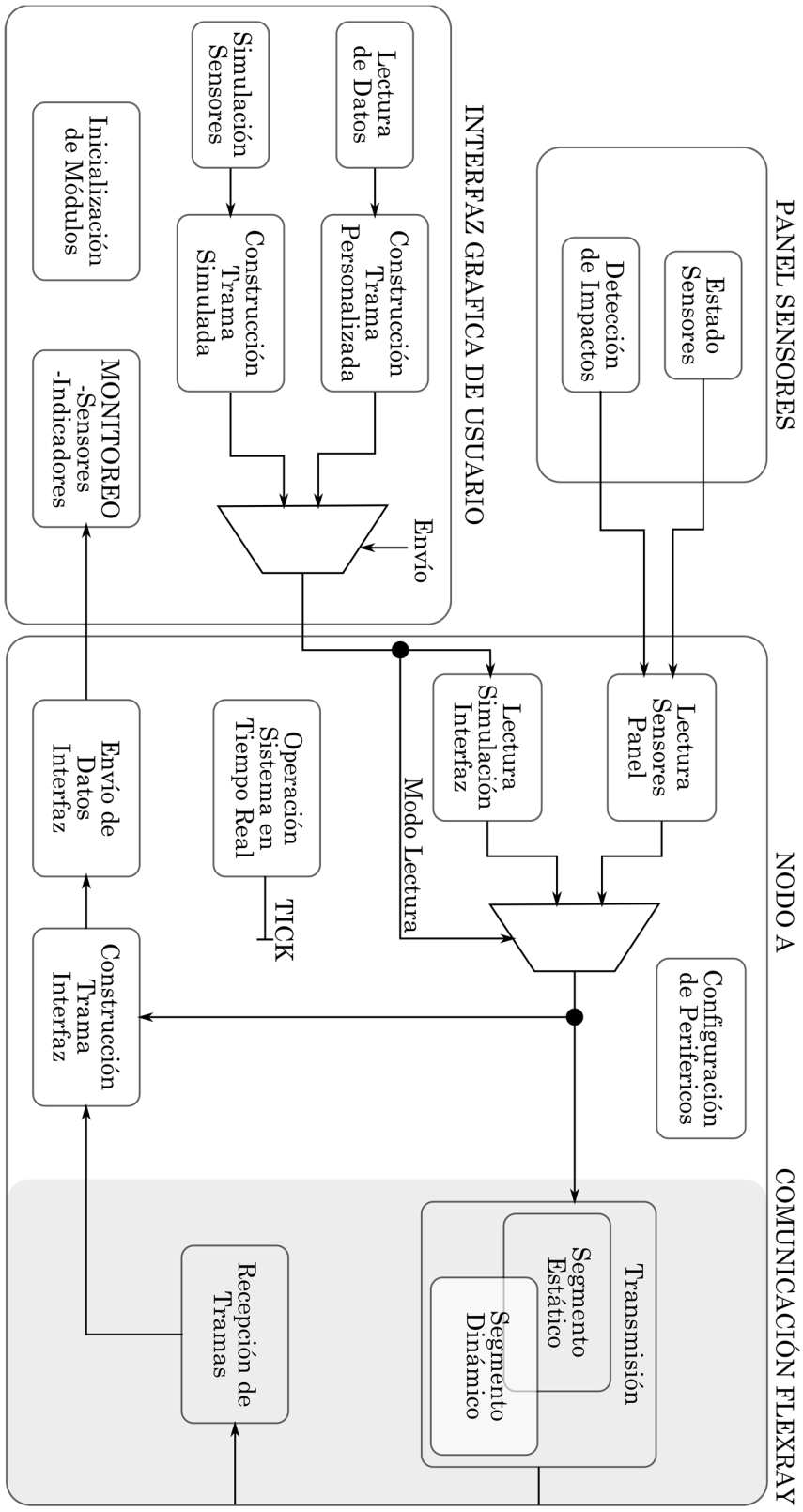


Figura 5.17. Diagrama a bloques del sistema integrado del nodo A, interfaz de usuario y panel de sensores.

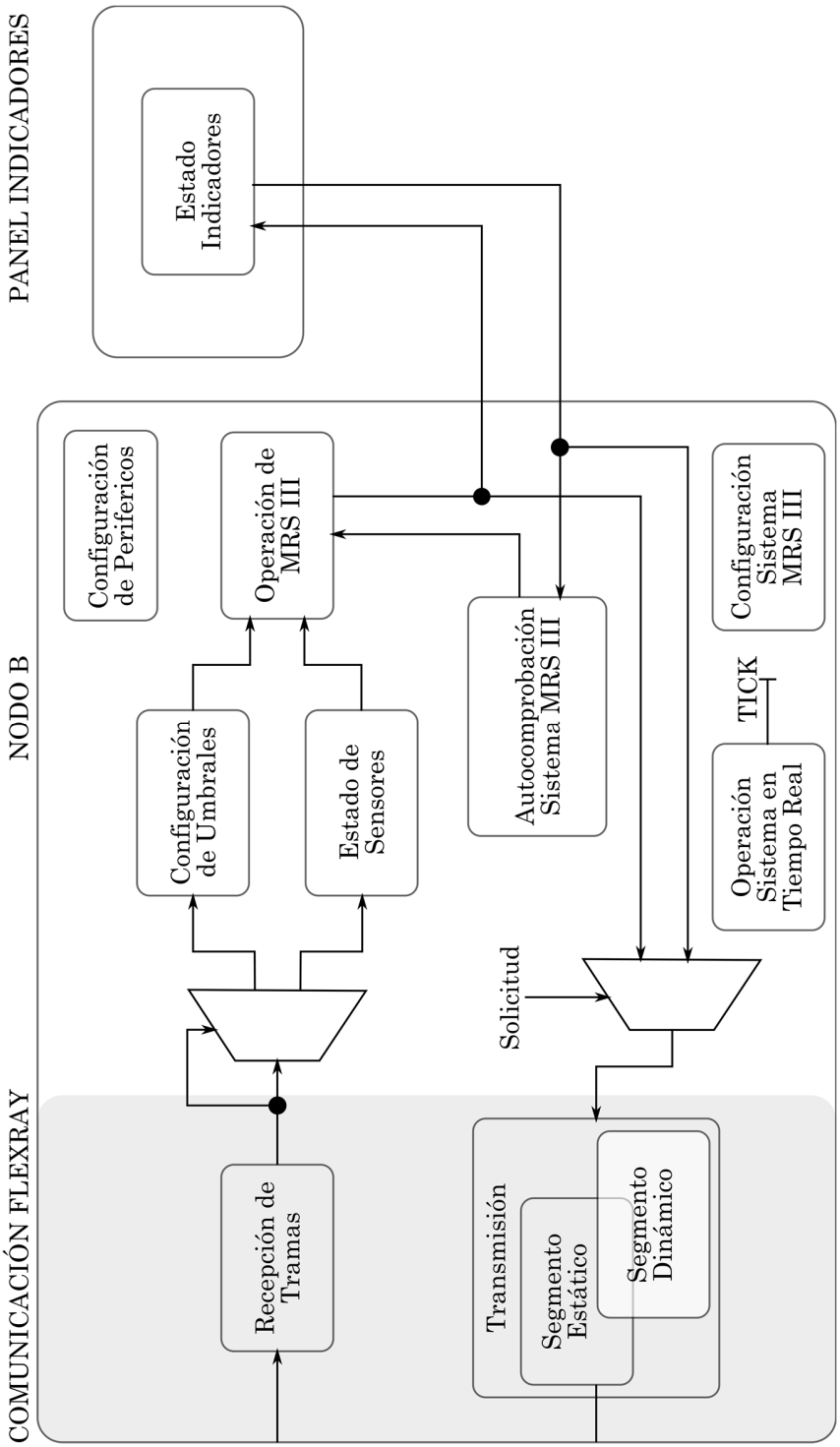


Figura 5.18. Diagrama a bloques del sistema integrado del nodo B y panel de indicadores.

Capítulo 6. Resultados

6.1. Sistema FlexREY

Al integrar el sistema completo, se obtuvo la plataforma de pruebas como se observa en la Figura 6.1, que contiene la red FlexRay con los nodos A y B utilizando las tarjetas Hercules y los BD fabricados (véase la Figura 6.2), fuente de suministro de energía, la placa de sensores (véase la Figura 6.3) y la placa de indicadores (véase la Figura 6.4); por otro lado, la GUI se puede observar en la Figura 6.5 y cuenta con cuatro secciones principales: a) Simulación de sensores, b) Construcción de trama FlexRay, c) Lectura de tramas seriales y d) Escritura de tramas seriales.

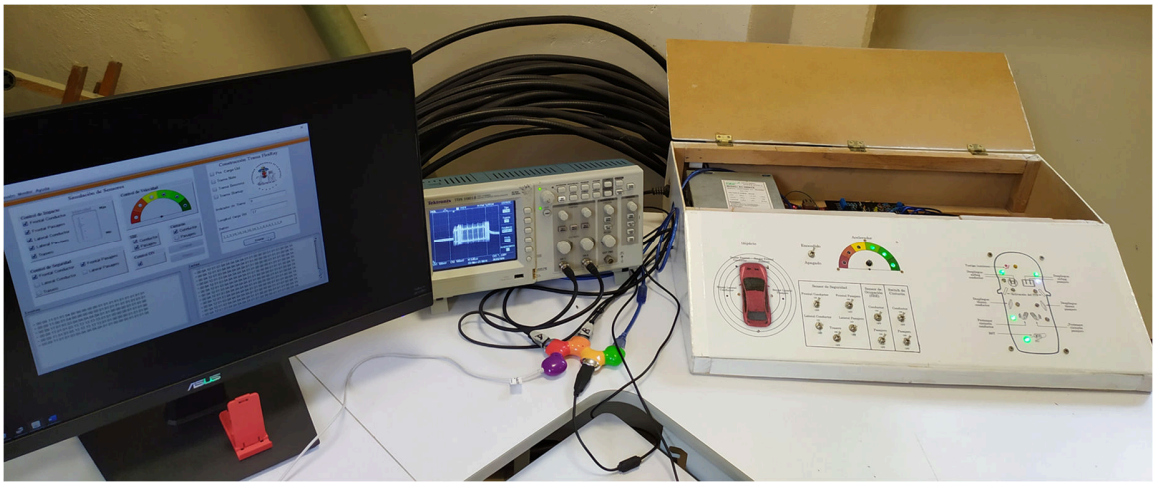


Figura 6.1. Sistema FlexREY.

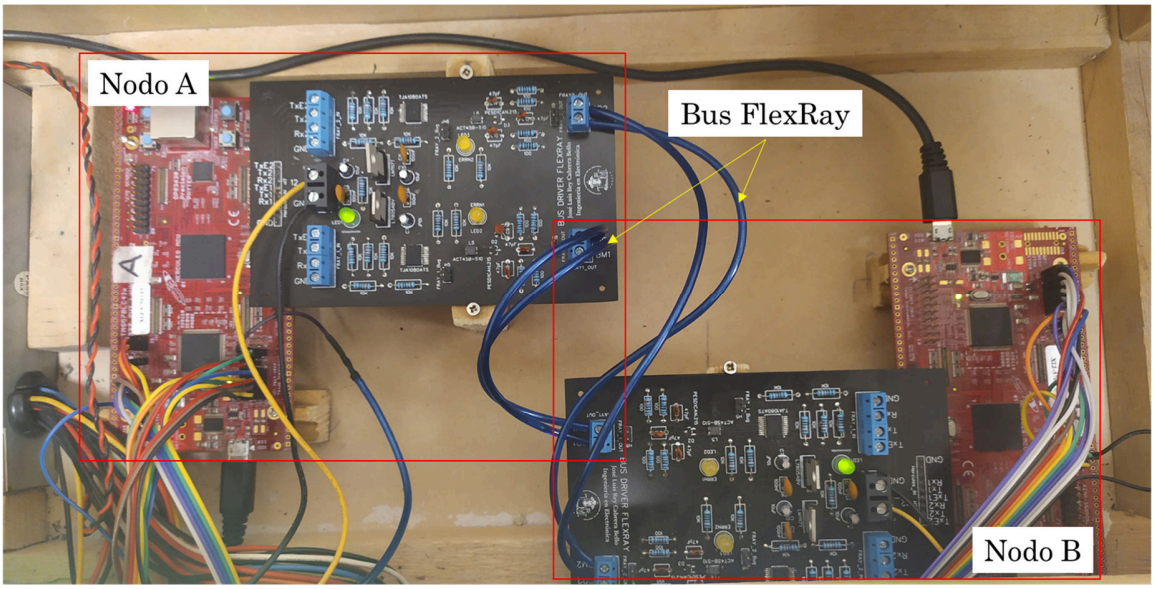


Figura 6.2.Red FlexRay.

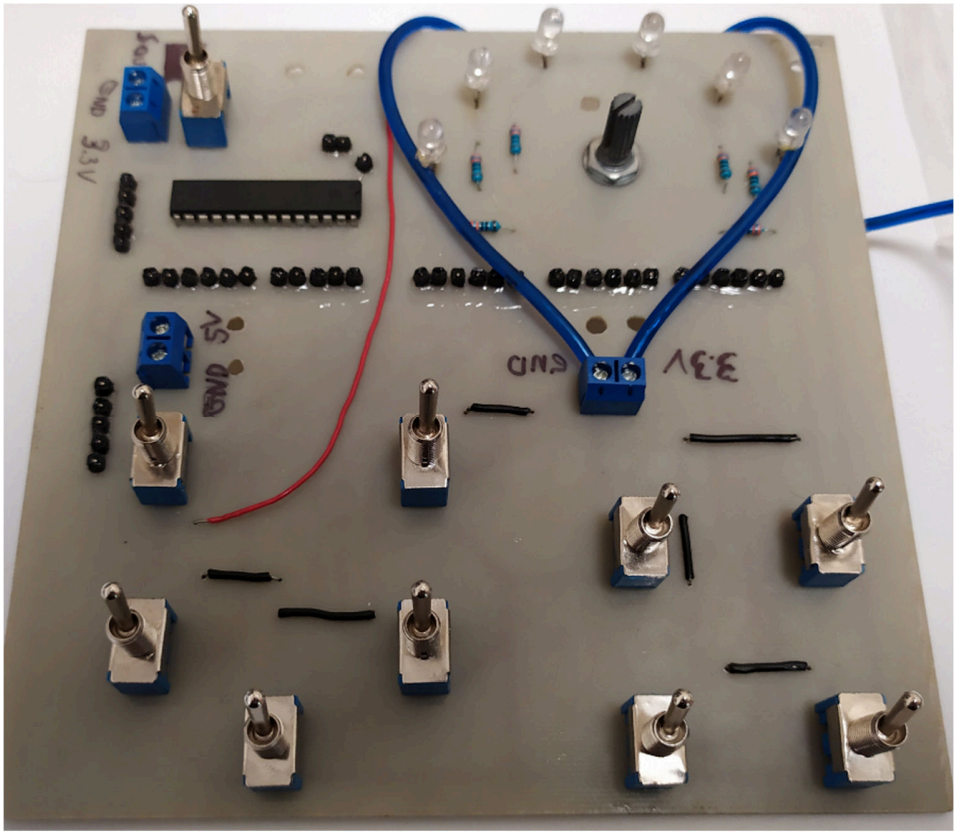


Figura 6.3. Placa de sensores.

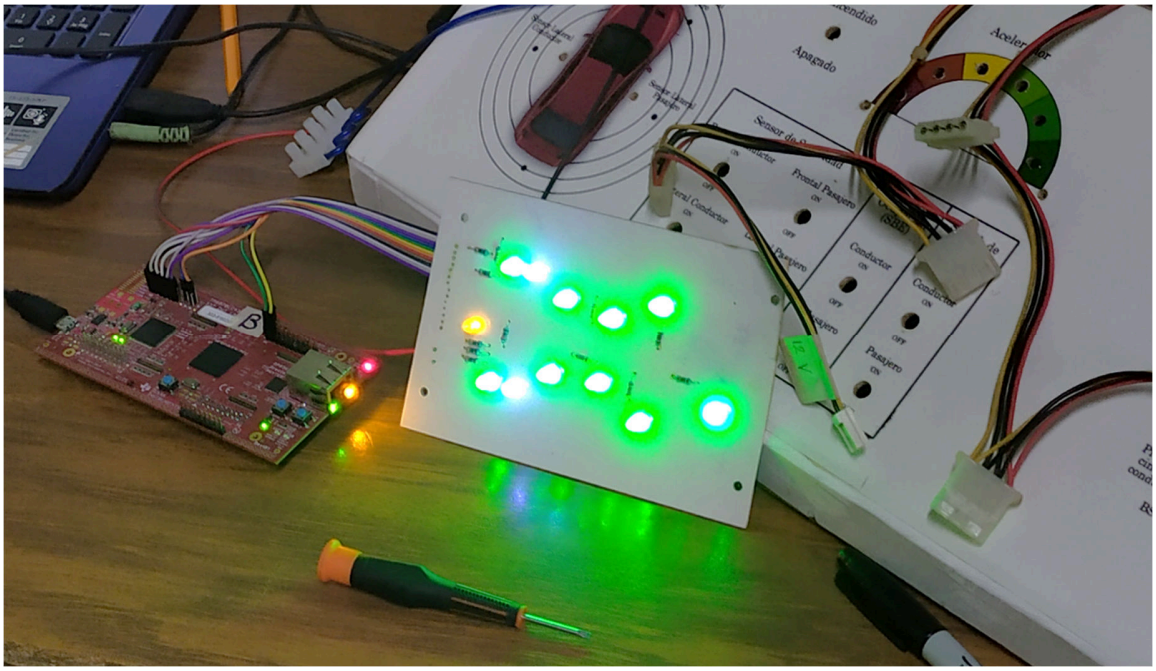


Figura 6.4. Placa de indicadores.

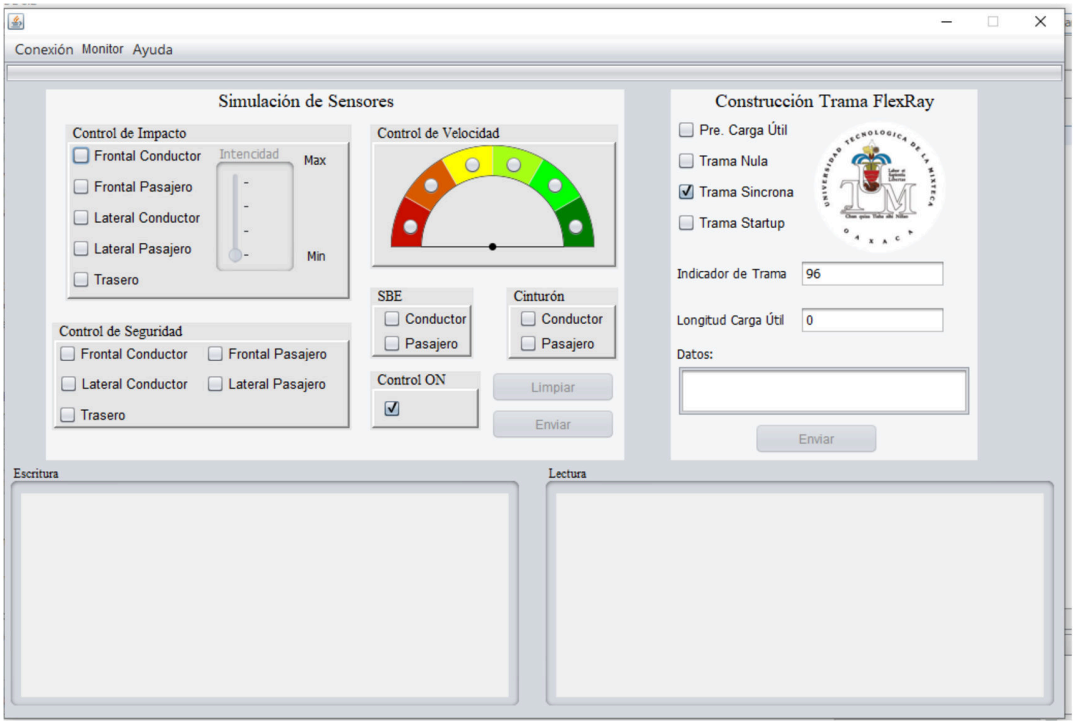


Figura 6.5. Interfaz gráfica de usuario GUI.

6.2. Encendido y Conexión del Sistema

Como se propuso en el diagrama de funcionamiento del párrafo 5.1.1.1.1, al encenderse, el sistema realiza una autocomprobación, por lo que se requiere habilitar todos los sensores e interruptores para que sean detectados correctamente. Una vez encendidos los nodos, la conexión de la interfaz requiere la búsqueda y selección del puerto correspondiente al nodo A en los periféricos seriales (COM) disponibles en el equipo, como se puede observar en la Figura 6.6.

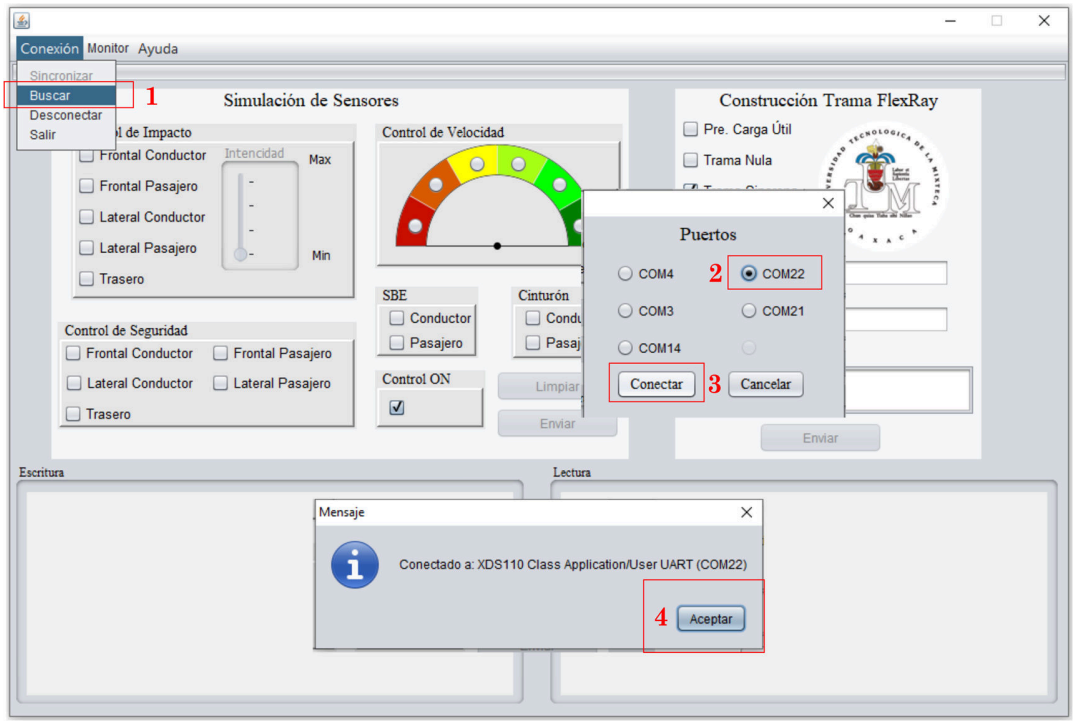


Figura 6.6. Conexión de la interfaz con el nodo A.

6.3. Pruebas de Aceptación

Para verificar el correcto funcionamiento del sistema se proponen dos tipos de pruebas: caja negra y caja gris; siendo las primeras únicamente comportamentales en las que se plantea una entrada específica y una salida determinada que debe coincidir con la entregada por el sistema, mientras que las de caja gris permiten evaluar el sistema dividiéndolo en subsistemas.

6.3.1. Pruebas de caja negra

Para estas pruebas se proponen los siguientes casos.

Encendido y conexión del sistema generando un error al dejar un interruptor apagado en la autocomprobación para mandarlo al estado deshabilitado, que debe poder ser visualizado al encender el testigo luminoso en los indicadores y en el monitor de la GUI (véase la Figura 6.7).

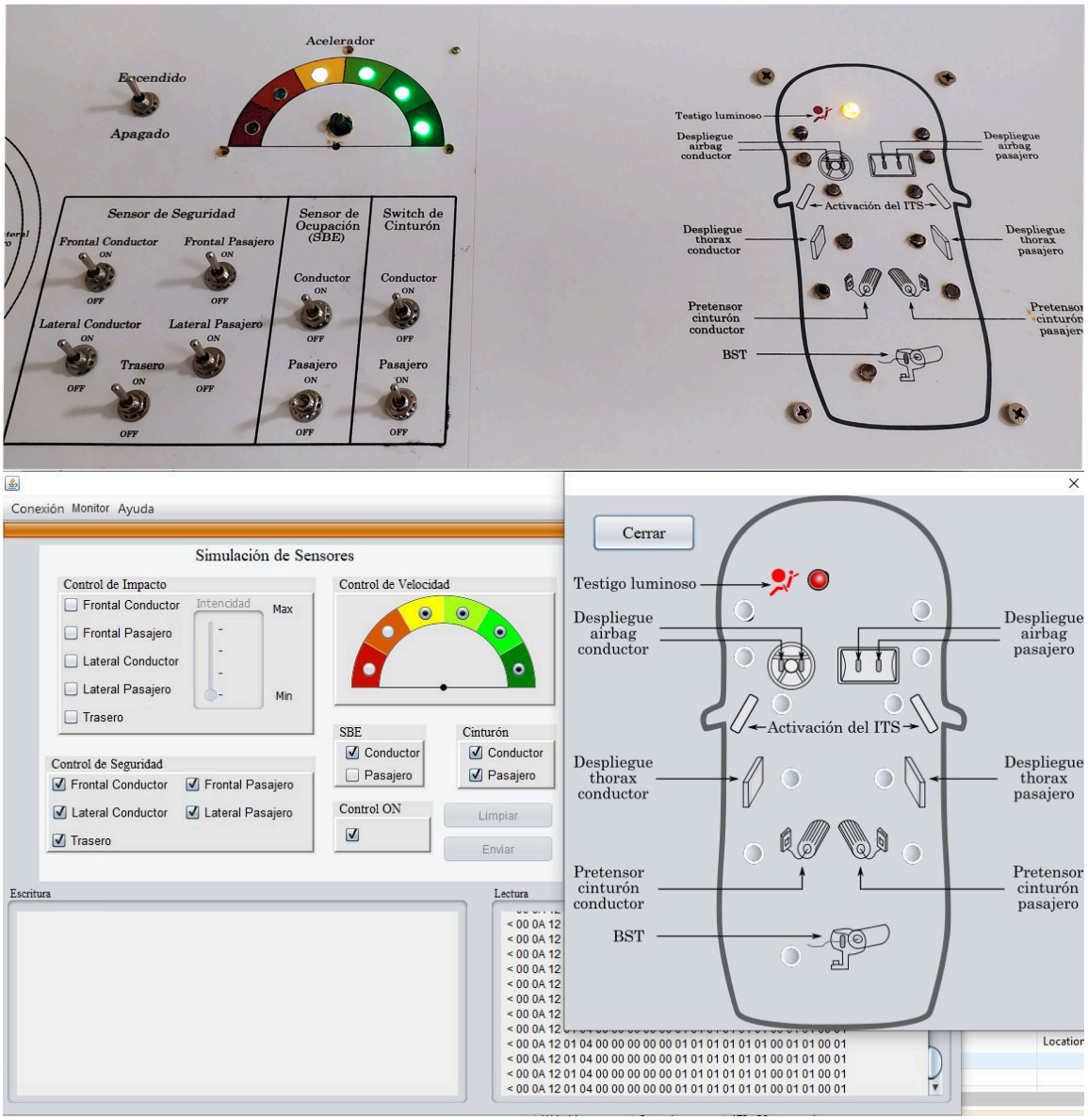


Figura 6.7. Prueba 1: Error de autocomprobación, sistema deshabilitado.

Solucionar el error en la autocomprobación permitiendo que el sistema cambie al estado habilitado, esto se debe visualizar al apagarse el testigo luminoso en los indicadores y en el monitor de la GUI (ver la Figura 6.8).

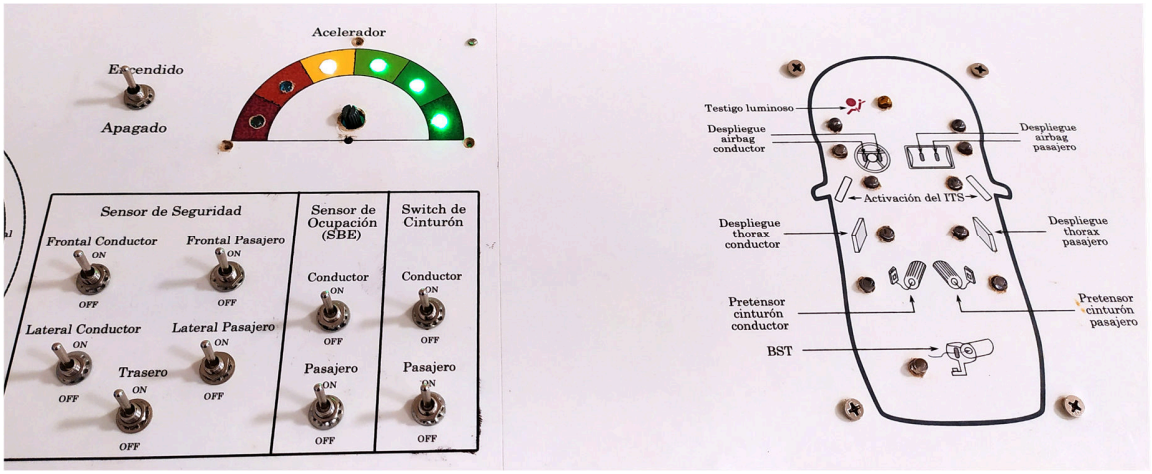


Figura 6.8. Prueba 2: Autocomprobación pasada, sistema habilitado.

Apagar sensores de seguridad y encender los sensores SBE de conductor y de pasajero, apagar el interruptor del cinturón del pasajero, colocar la lectura del acelerador a más de 90 km/h y disminuirlo drásticamente a menos de 30 km/h, esto debe encender el indicador del pretensor del conductor, pero no el del pasajero, como se muestra en la Figura 6.9.

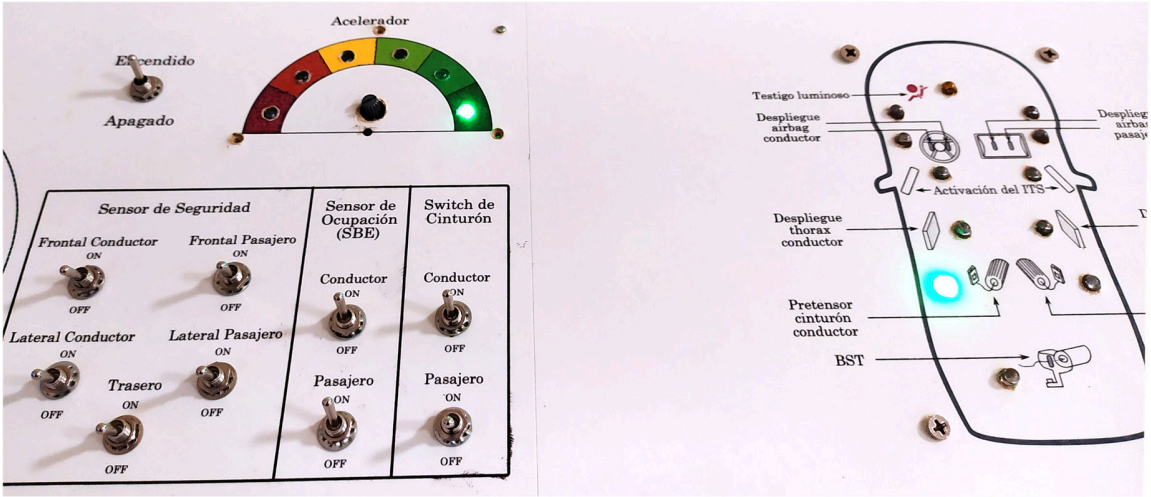


Figura 6.9. Prueba 3: Respuesta a desaceleraciones drásticas.

Encender los sensores de seguridad frontales, colocar la velocidad a más de 60 km/h y generar un impacto ligero; esto debe activar la primera etapa del *airbag* del pasajero y el pretensor del conductor (véase la Figura 6.10).

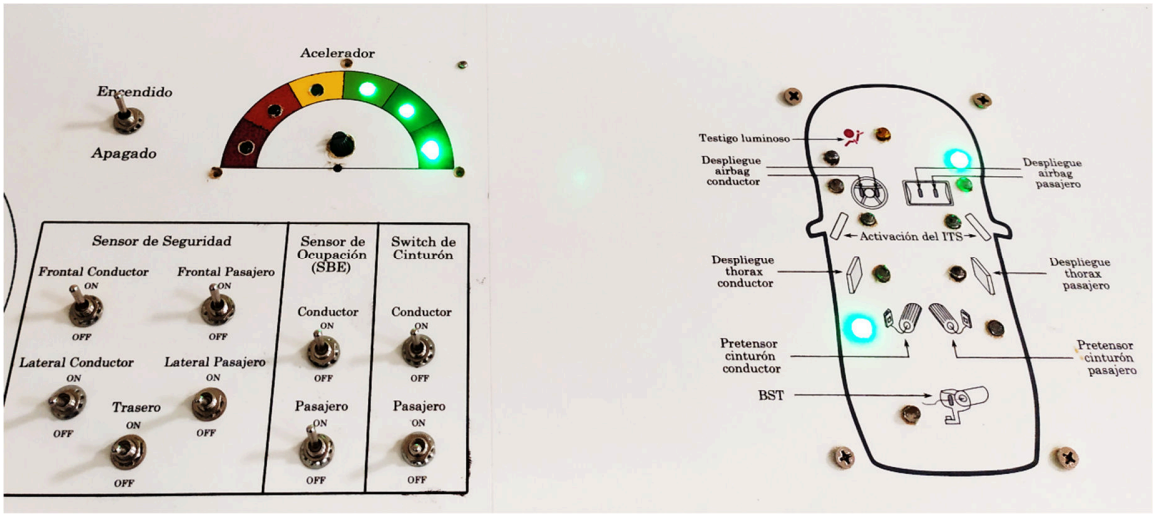


Figura 6.10. Prueba 4: Respuesta a impactos frontales ligeros.

Generar un impacto mediano, esto desplegará la primera etapa del *airbag* del conductor y su pretensor del cinturón, desplegará el *airbag* de dos etapas con retraso del pasajero y activará el BST, como se observa en la Figura 6.11.

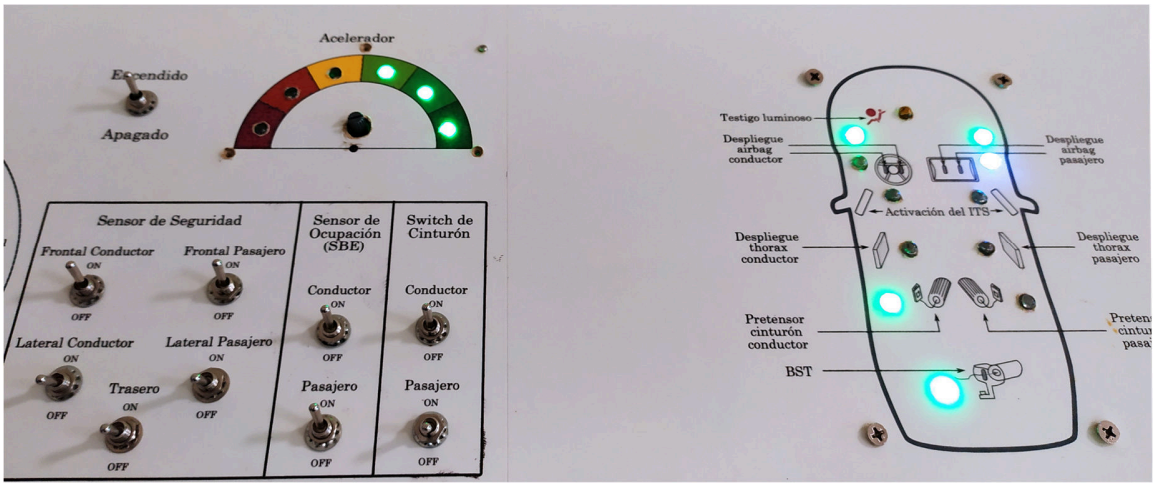


Figura 6.11. Prueba 5: Respuesta a impactos frontales medianos.

Generar un impacto fuerte, esto desplegará el *airbag* de dos etapas con retraso del conductor y su pretensor del cinturón, y también, desplegará el *airbag* de dos etapas del pasajero y activará el BST, como se observa en la Figura 6.12.

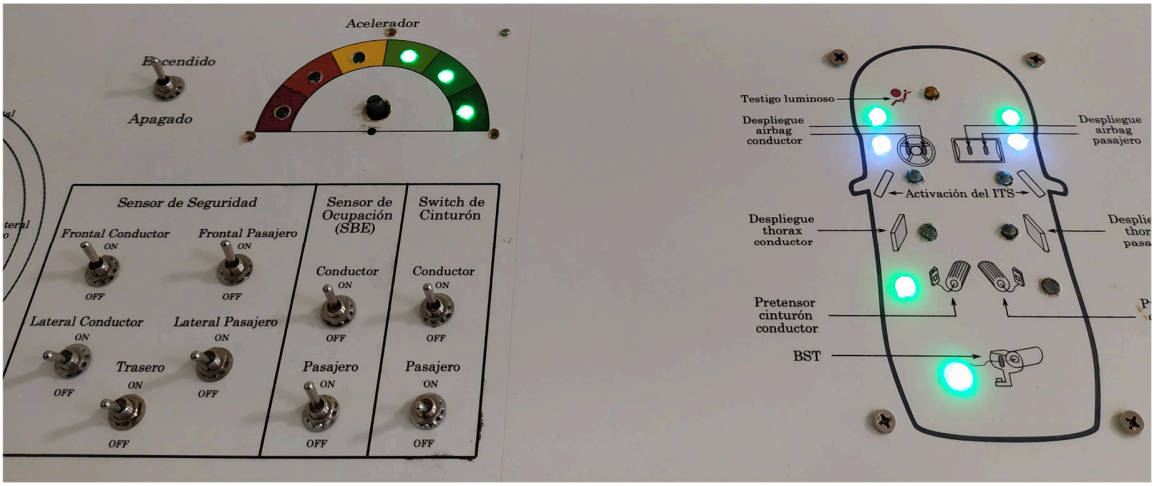


Figura 6.12. Prueba 6: Respuesta a impactos frontales fuertes.

Apagar los sensores de seguridad y encender sólo los laterales, apagar sensor SBE del pasajero y generar un impacto fuerte; esto desplegará el ITS y *thorax* del conductor y activará el BST, como se muestra en la Figura 6.13.

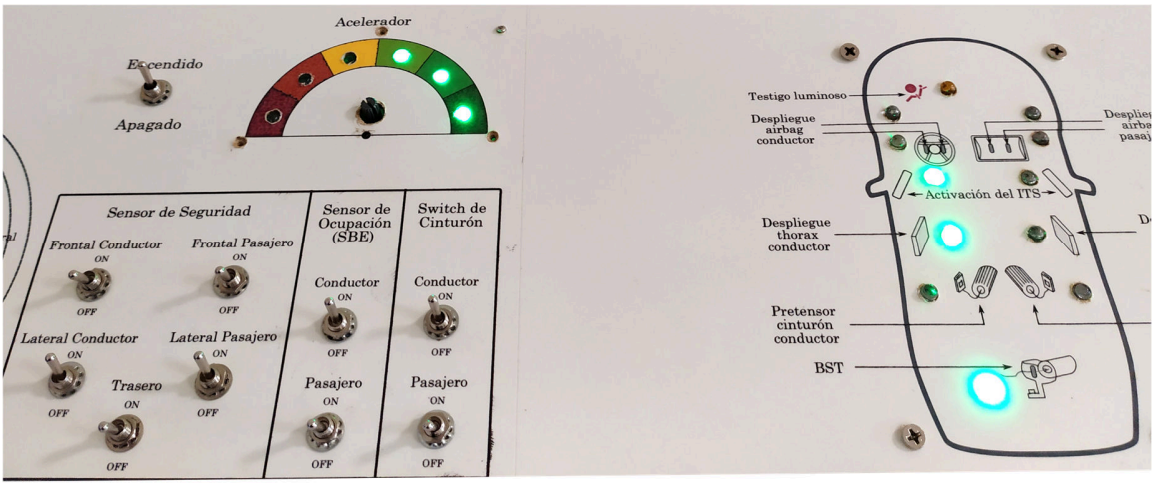


Figura 6.13. Prueba 7: Respuesta a impactos laterales fuertes.

Cambiar la interfaz al modo simulación de sensores en la pestaña de 'Monitor' y asegurar la sincronización con el nodo A, repetir las pruebas anteriores ahora desde la GUI, como se muestra en la Figura 6.14.

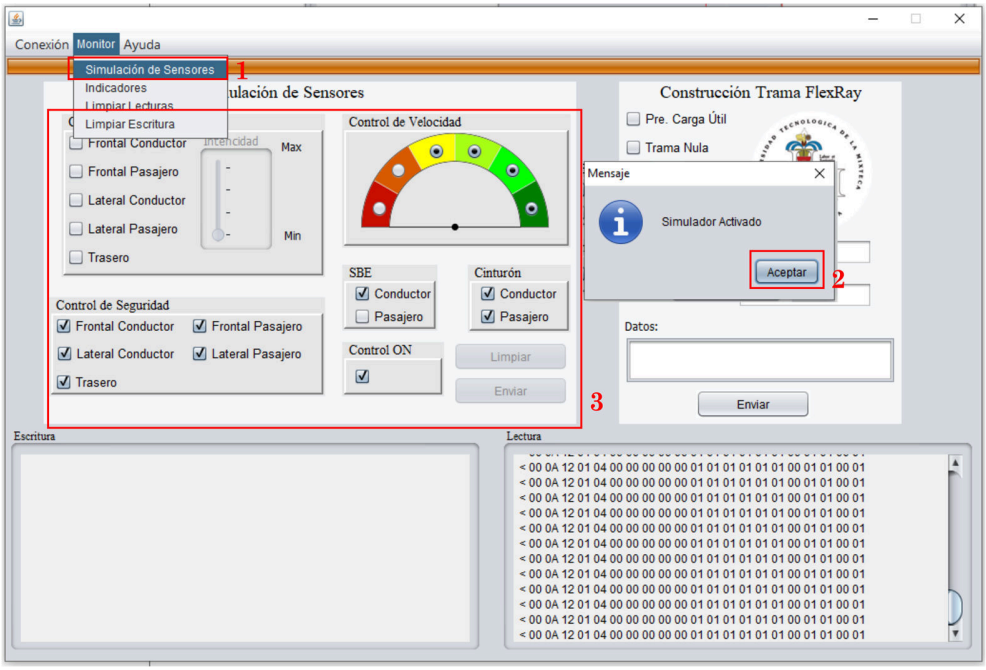


Figura 6.14. Cambio en la interfaz al modo de simulador de sensores.

Generar las tramas personalizadas desde la GUI para la prueba 3, como se muestra en la Figura 6.15.

0x00 09 11 01 01 04 00 00 00 00 00 00 00 00 00 00 01 01 01 00

0x00 09 11 01 01 01 00 00 00 00 00 00 00 00 00 00 01 01 01 00

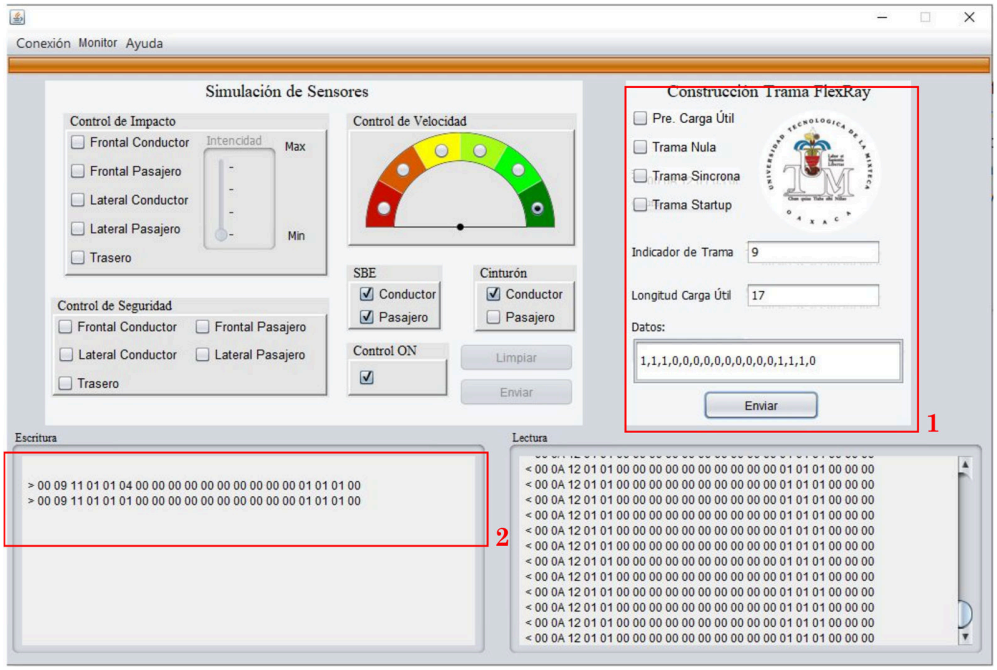


Figura 6.15. Construcción de trama personalizada.

Generar la trama personalizada desde la interfaz para la prueba 5, permitiendo ver el cambio en las lecturas de la trama recibida en respuesta a la trama enviada, como se muestra en la Figura 6.16.

0x00 09 11 01 01 03 0A 0A 0A 0A 0A 01 01 00 00 00 01 01 01 00

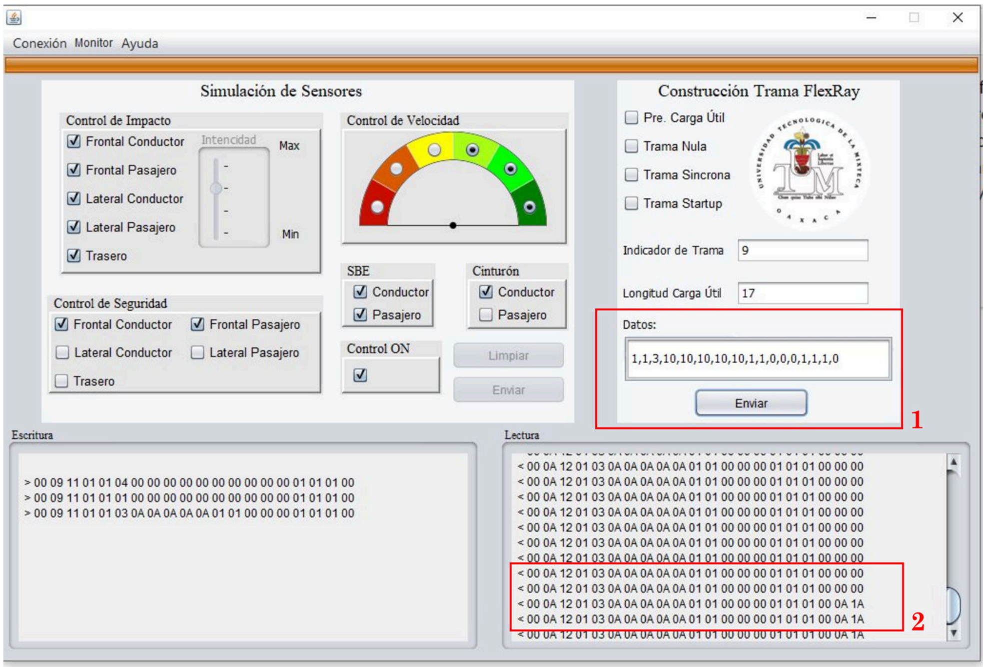


Figura 6.16. Respuesta a tramas personalizadas.

Monitoreando el bus FlexRay con el osciloscopio (véase la Figura 6.1) a una escala horizontal de 250 us, escala vertical de 1 volt por canal, atenuación de sonda x1, acoplamiento en cc, referencias centradas en -2.5 volts, y configurando la transmisión en modo cíclico, se puede visualizar el conjunto de tramas del sistema integrado como se planteó en el inciso 5.4.2.5 (véase la Figura 6.17).

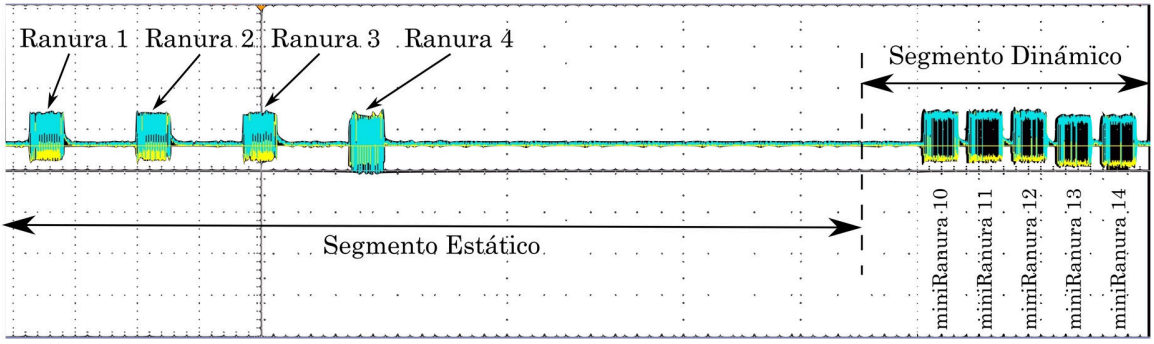


Figura 6.17. Captura en osciloscopio del sistema completo.

6.3.2. Pruebas de caja gris

Para la prueba de caja gis se propuso la transmisión de una trama personalizada desde la GUI de igual forma que en la prueba de caja negra (véase la Figura 6.16). La Figura 6.18 muestra el flujo de información a través del sistema dando seguimiento a la ejecución de la prueba.

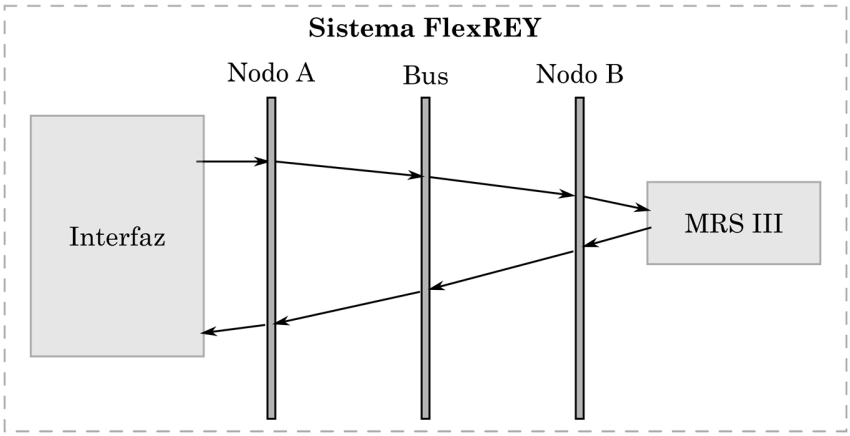


Figura 6.18. Diagrama de seguimiento para prueba caja gris.

Desde la GUI se ingresan los datos para construir la trama personalizada, al dar clic en Enviar, se leen los datos y se almacenan en los registros para ser cargados en el búfer serial como se muestra en la Figura 6.19, así el módulo serial se encarga de realizar la transmisión de los datos generando una trama serial UART. Ya que la velocidad de transferencia está establecida en 115,200 baudios, la transmisión de datos se codifica con: un bit de inicio, 6 bits datos reales y dos bits de parada, con una frecuencia de bit de 8.6 µs, además se agrega el símbolo 0x42 como inicio de empaquetamiento de la trama.

La trama decodificada que se muestra en la Figura 6.20 inicialmente se muestreó utilizando un osciloscopio, los datos obtenidos se importaron a Matlab con ayuda de un *script* encargado de acondicionar la señal mediante comparadores y generando una señal de reloj con un periodo de 9 µs. Finalmente, se graficaron los datos para facilitar la decodificación manual de la trama.

FrameSend	Trama_FlexRay	#2123
reserva	byte	0
pre_payload	byte	0
nula	byte	0
sincrona	byte	0
startup	byte	0
ID_trama	byte	9
longt_payload	byte	17
CRC_cabecera	byte	0
cont_cid	byte	0
dato	byte[]	#2126(length=17)
[0]	byte	1
[1]	byte	1
[2]	byte	3
[3]	byte	10
[4]	byte	10
[5]	byte	10
[6]	byte	10
[7]	byte	10
[8]	byte	1
[9]	byte	1
[10]	byte	0
[11]	byte	0
[12]	byte	0
[13]	byte	1
[14]	byte	1
[15]	byte	1
[16]	byte	0
trailer_CRC	byte	0

Figura 6.19. Registros de la interfaz gráfica.

Una vez recibida la trama serial de la interfaz, se lee del búfer y se almacena en la estructura de datos como se muestra en la Figura 6.21, con lo que se observa que la comunicación de la interfaz al nodo A sea realiza correctamente, al comparar los datos en los registros de envío y recepción.

Una vez recibidos y almacenados los datos, se lleva a cabo la transmisión FlexRay en cuando el planificador asigne el tiempo de ejecución de la segunda tarea, en la cual se cargan los datos en los búferes escribiendo los registros WRDS del CC FlexRay como se observa en la Figura 6.22 y asignando la ranura

o mini ranura correspondiente para la transmisión del segmento (como se estableció en el inciso 5.4.2.5).

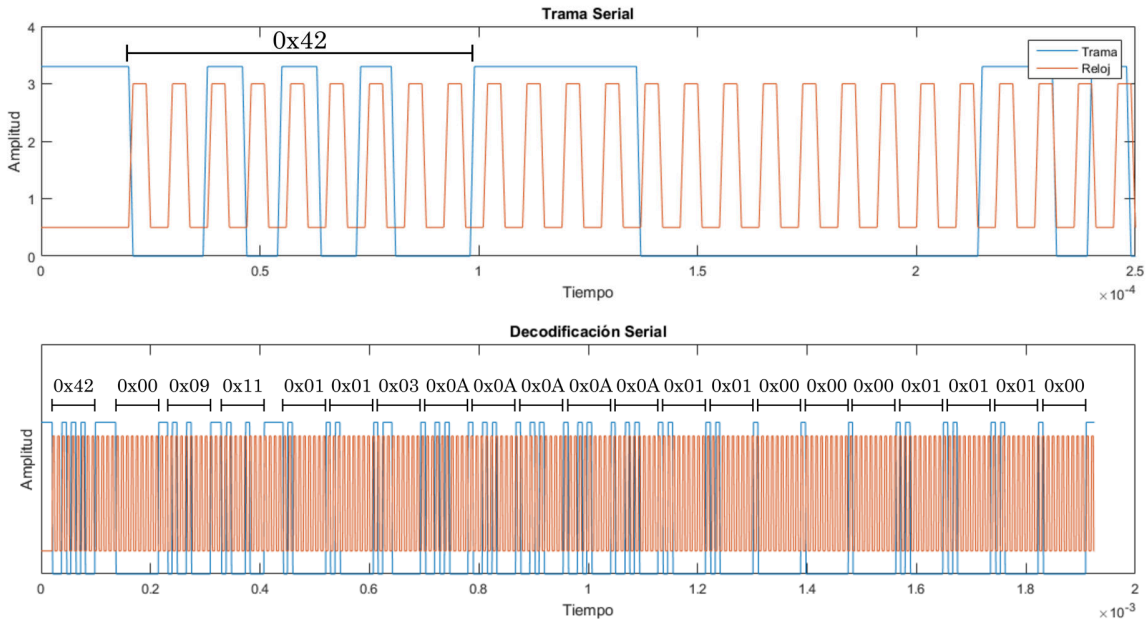


Figura 6.20. Trama enviada por puerto serial y decodificación con base en el reloj (Señal muestreada con osciloscopio y procesada digitalmente).

frame	0x08001500 {ranura=0,se
*(frame)	{ranura=0,seg_startup_U
ranura	0
seg_startup_UN	{seg_startup_UL=0,seg_s
seg_startup_UL	0
seg_startup_ST	{reserva=0,pre_payload=
reserva	0
pre_payload	0
nula	0
sincrona	0
startup	0
ID_trama	9
longt_payload	17
CRC_cabecera	0
cont_cicl	0
dato	[1,1,3,10,10...]
[0]	1
[1]	1
[2]	3
[3]	10
[4]	10
[5]	10
[6]	10
[7]	10
[8]	1
[9]	1
[10]	0
[11]	0
[12]	0
[13]	1
[14]	1
[15]	1
[16]	0

Figura 6.21. Registro de la estructura de datos de la trama serial recibida.

WRDS	[0x0000000A,0x0000000A,0x0000000...	0xFFF7CC00
[0]	0x0000000A (Hex)	0xFFF7CC00
[1]	0x0000000A (Hex)	0xFFF7CC04
[2]	0x0000000A (Hex)	0xFFF7CC08
[3]	0x0000000A (Hex)	0xFFF7CC0C
[4]	0x0000000A (Hex)	0xFFF7CC10

Figura 6.22. Carga de datos en búfer FlexRay.

Al realizar la transmisión sobre el bus FexRay y al utilizar el osciloscopio para monitorear el bus conectando el BP al canal 1 y el BM al canal 2 para realizar la lectura diferencial (véase la Figura 6.23), se observó la transmisión de las cuatro ranuras en el segmento estático y ya que el caso de prueba sólo requiere el envío del estado del panel que corresponde a la quinta mini ranura, solo se envía una trama en el segmento dinámico de manera periódica.

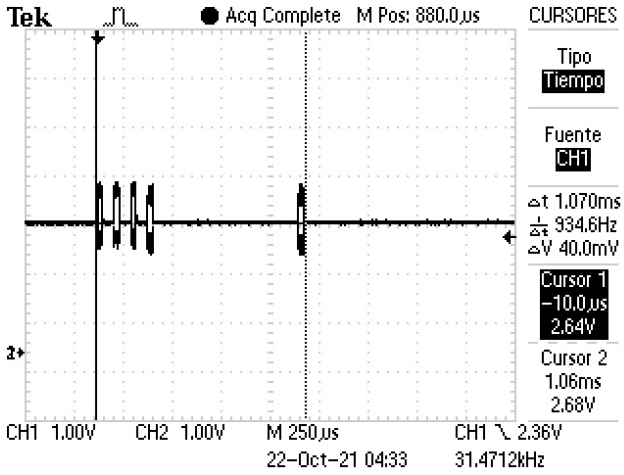


Figura 6.23. Lectura de osciloscopio de la transmisión sobre el bus FlexRay.

Por otro lado, para visualizar la primera ranura de la transmisión (véase la Figura 6.24), se procedió a muestrear la señal de la trama y posteriormente se programó un *script* en Matlab (véase el Anexo J) para realizar el filtrado y acondicionamiento de los datos, así mismo, se calculó el uBus y se generó la señal de reloj para la decodificación de la trama como se muestra en la Figura 6.25.

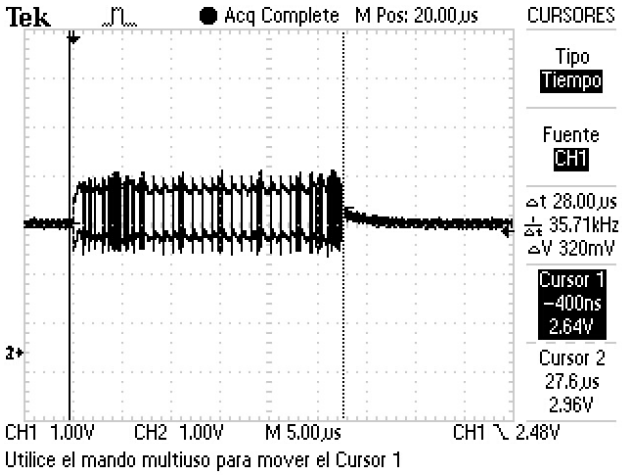


Figura 6.24. Captura de osciloscopio primer ranura de la transmisión.

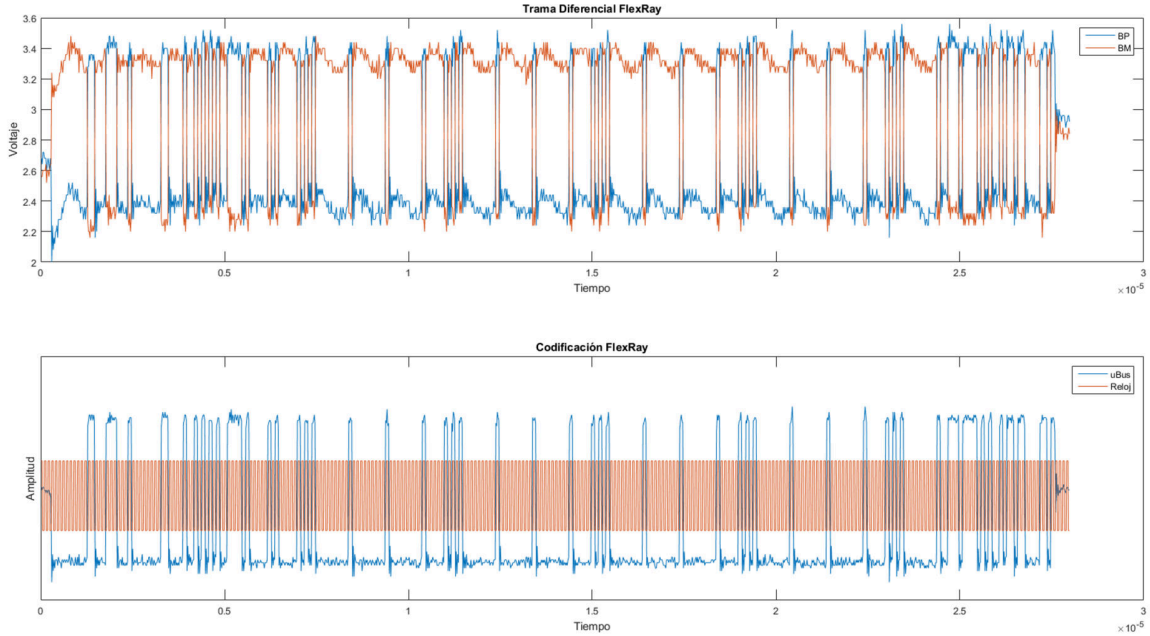


Figura 6.25. Trama diferencial FlexRay y uBus codificado (Señal muestreada con osciloscopio y procesada digitalmente).

Al realizar la decodificación de la trama, se observa que la lectura cumple con los parámetros establecidos por el protocolo (apartado 4.4.2). Se puede observar el inicio de la trama, así como la cabecera (véase la Figura 6.26) y la transmisión de los datos de la carga útil (véase la Figura 6.27).

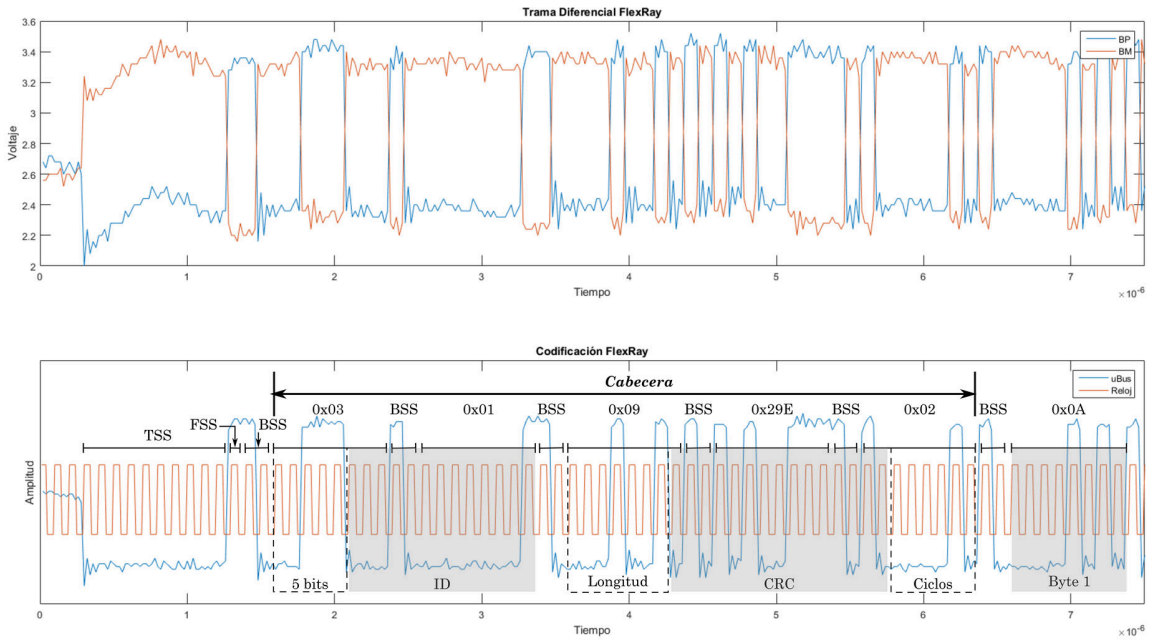


Figura 6.26. Inicio de trama FlexRay y decodificación por segmentos del uBus.

Ya que la carga útil fue establecida en 9 palabras dobles, se puede observar que tiene una longitud de 18 bytes, cada uno separado por BSS.

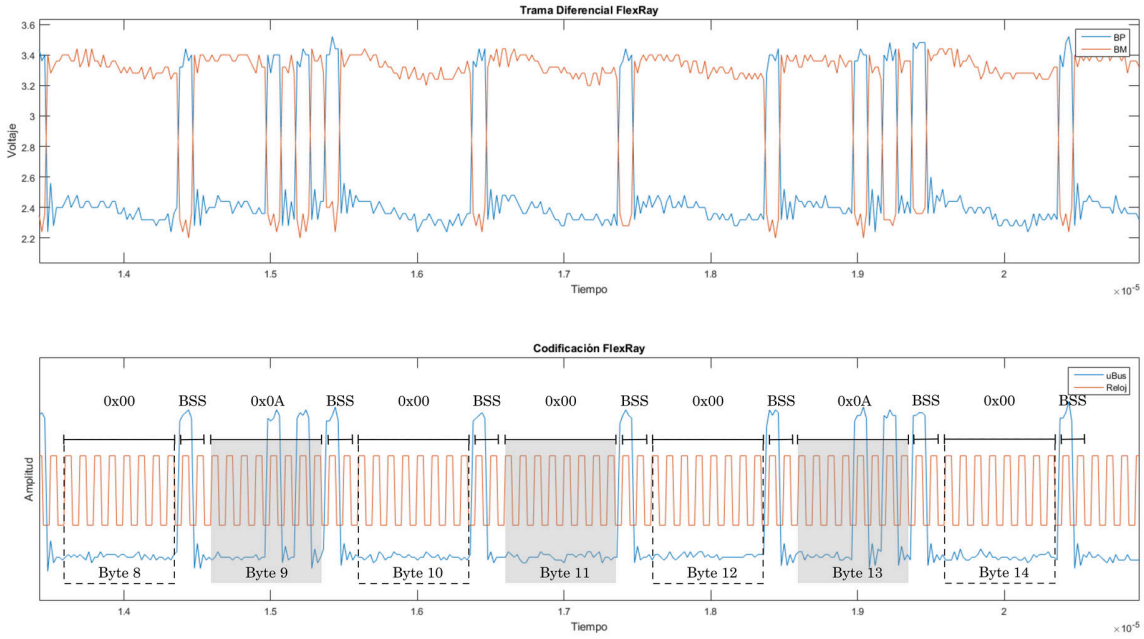


Figura 6.27. Carga útil en la trama FlexRay y decodificación por segmentos del uBus.

Finalmente se tiene el tráiler CRC de longitud de 3 bytes y la secuencia de fin de trama (FES) como se observa en la Figura 6.28.

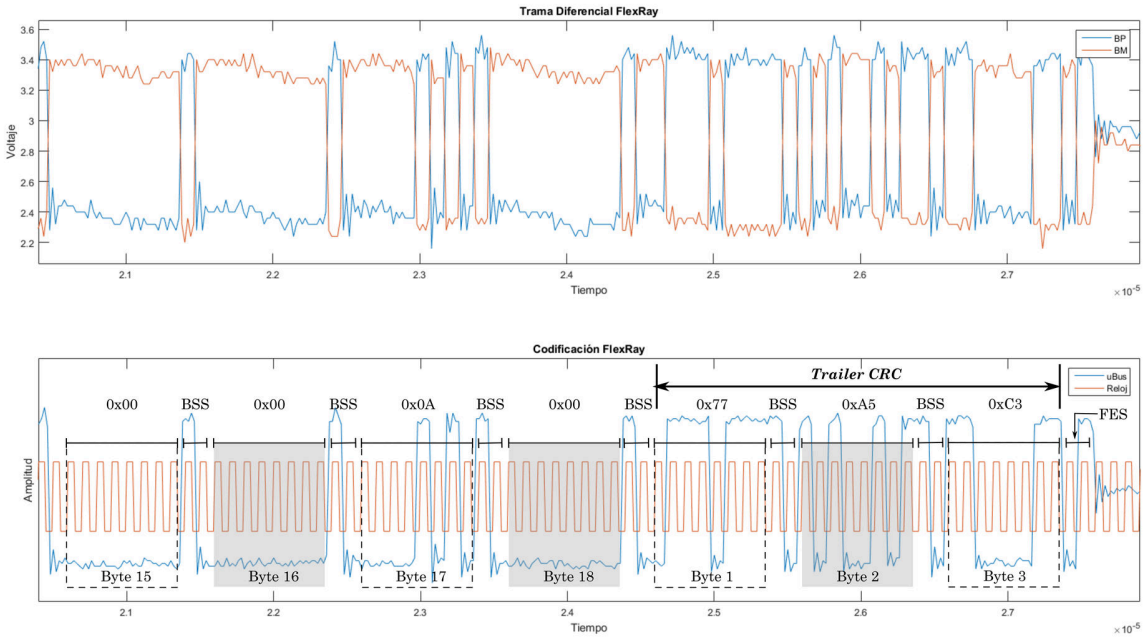
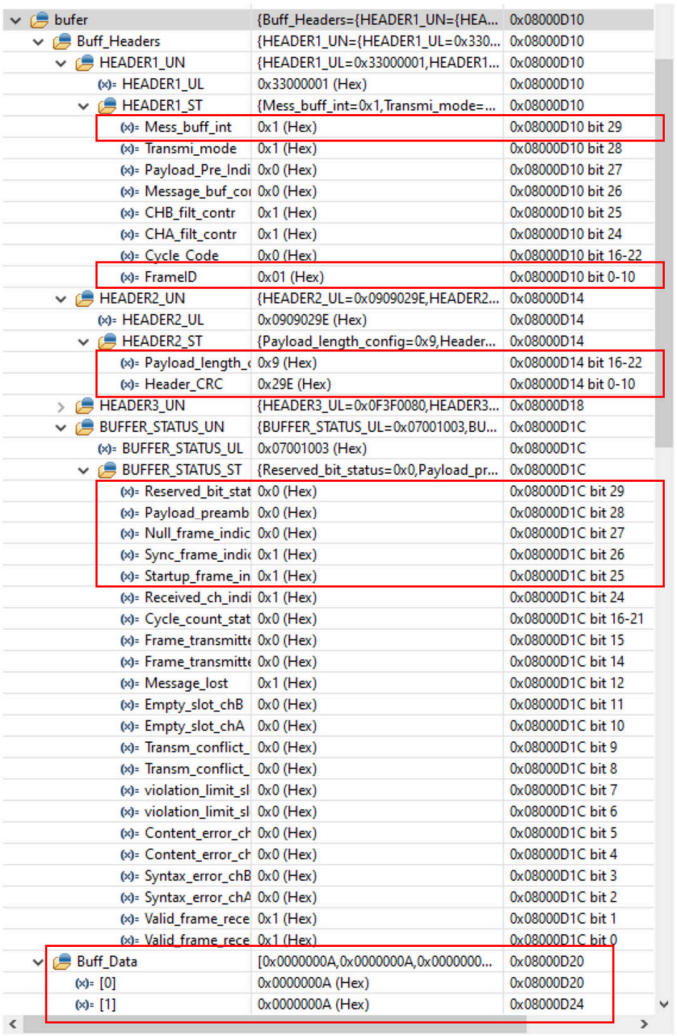


Figura 6.28. Tráiler CRC y secuencia de fin de trama.

Una vez recibidas las tramas FlexRay en el nodo B, se leen del búfer y se almacenan en la estructura de datos como se muestra en la Figura 6.29, con lo que se observa que la trasmisión y la recepción se realizan correctamente al comparar los datos en los registros de envío y recepción.

Con la información obtenida del nodo A, se ejecuta la simulación del sistema MRS III cuando el planificador inicia la ejecución de la tercera tarea, que obtiene una respuesta como se muestra en la Figura 6.30. Se despliega la primera etapa del *airbag* del conductor y su pretensor del cinturón, y el *airbag* de dos etapas con retraso del pasajero, así mismo, se activa el BST, como se observa en la Figura 6.11.

Una vez obtenida la respuesta del sistema, este entra en un periodo de espera de 3 segundos para posteriormente reestablecer el sistema al estado habilitado, mostrando una secuencia en el panel indicando el correcto restablecimiento.



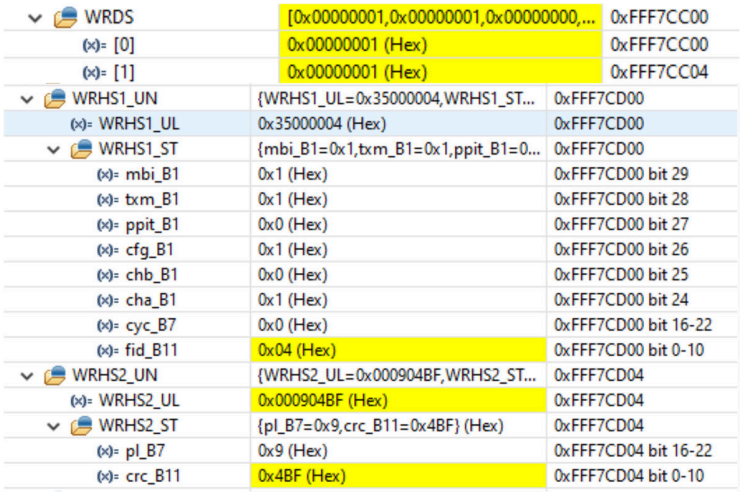
bufer	{Buff_Headers={HEADER1_UN={HEA...	0x08000D10
Buff_Headers	{HEADER1_UN={HEADER1_UL=0x330...	0x08000D10
HEADER1_UN	{HEADER1_UL=0x33000001,HEADER1...	0x08000D10
HEADER1_UL	0x33000001 (Hex)	0x08000D10
HEADER1_ST	{Mess_buff_int=0x1,Transmi_mode=...	0x08000D10
Mess_buff_int	0x1 (Hex)	0x08000D10 bit 29
Transmi_mode	0x1 (Hex)	0x08000D10 bit 28
Payload_Pre_Indi	0x0 (Hex)	0x08000D10 bit 27
Message_buf_coi	0x0 (Hex)	0x08000D10 bit 26
CHB_filt_contr	0x1 (Hex)	0x08000D10 bit 25
CHA_filt_contr	0x1 (Hex)	0x08000D10 bit 24
Cycle_Code	0x0 (Hex)	0x08000D10 bit 16-22
FrameID	0x01 (Hex)	0x08000D10 bit 0-10
HEADER2_UN	{HEADER2_UL=0x0909029E,HEADER2...	0x08000D14
HEADER2_UL	0x0909029E (Hex)	0x08000D14
HEADER2_ST	{Payload_length_config=0x9,Header...	0x08000D14
Payload_length_(	0x9 (Hex)	0x08000D14 bit 16-22
Header_CRC	0x29E (Hex)	0x08000D14 bit 0-10
HEADER3_UN	{HEADER3_UL=0x0F3F0080,HEADER3...	0x08000D18
BUFFER_STATUS_UN	{BUFFER_STATUS_UL=0x07001003,BU...	0x08000D1C
BUFFER_STATUS_UL	0x07001003 (Hex)	0x08000D1C
BUFFER_STATUS_ST	{Reserved_bit_status=0x0,Payload_pr...	0x08000D1C
Reserved_bit_stat	0x0 (Hex)	0x08000D1C bit 29
Payload_preamb	0x0 (Hex)	0x08000D1C bit 28
Null_frame_indic	0x0 (Hex)	0x08000D1C bit 27
Sync_frame_indi	0x1 (Hex)	0x08000D1C bit 26
Startup_frame_in	0x1 (Hex)	0x08000D1C bit 25
Received_ch_indi	0x1 (Hex)	0x08000D1C bit 24
Cycle_count_stat	0x0 (Hex)	0x08000D1C bit 16-21
Frame_transmitte	0x0 (Hex)	0x08000D1C bit 15
Frame_transmitte	0x0 (Hex)	0x08000D1C bit 14
Message_lost	0x1 (Hex)	0x08000D1C bit 12
Empty_slot_chB	0x0 (Hex)	0x08000D1C bit 11
Empty_slot_chA	0x0 (Hex)	0x08000D1C bit 10
Transm_conflict_	0x0 (Hex)	0x08000D1C bit 9
Transm_conflict_	0x0 (Hex)	0x08000D1C bit 8
violation_limit_sl	0x0 (Hex)	0x08000D1C bit 7
violation_limit_sl	0x0 (Hex)	0x08000D1C bit 6
Content_error_ch	0x0 (Hex)	0x08000D1C bit 5
Content_error_ch	0x0 (Hex)	0x08000D1C bit 4
Syntax_error_chB	0x0 (Hex)	0x08000D1C bit 3
Syntax_error_chA	0x0 (Hex)	0x08000D1C bit 2
Valid_frame_rece	0x1 (Hex)	0x08000D1C bit 1
Valid_frame_rece	0x1 (Hex)	0x08000D1C bit 0
Buff_Data	[0x0000000A,0x0000000A,0x0000000...	0x08000D20
[0]	0x0000000A (Hex)	0x08000D20
[1]	0x0000000A (Hex)	0x08000D24

Figura 6.29. Datos del búfer de recepción en la ranura 1 del nodo B.

panel_indicadores	{Indicadores_UN={Indicadores_UL=2...	0x08001644
Indicadores_UN	{Indicadores_UL=2586,Indicadores_S...	0x08001644
Indicadores_UL	0x00000A1A (Hex)	0x08001644
Indicadores_ST	{BST=1,Preten_pasj=0,Preten_cond=...	0x08001644
BST	1	0x08001644 bit 11
Preten_pasj	0	0x08001644 bit 10
Preten_cond	1	0x08001644 bit 9
Thorax_pasj	0	0x08001644 bit 8
Thorax_cond	0	0x08001644 bit 7
Acti_ITS_2	0	0x08001644 bit 6
Acti_ITS_1	0	0x08001644 bit 5
Desp_pasj_E2	1	0x08001644 bit 4
Desp_pasj_E1	1	0x08001644 bit 3
Desp_cond_E2	0	0x08001644 bit 2
Desp_cond_E1	1	0x08001644 bit 1
Testigo_Lum	0	0x08001644 bit 0

Figura 6.30. Registro de panel de indicadores.

Ya que el estado del sistema MRS III se asigna a la ranura 4 (véase la Figura 6.31), la respuesta del sistema se envía en el próximo ciclo de comunicación, que al ser recibida y leída en el nodo A (véase la Figura 6.32), se verifica la correcta transmisión de la información.



WRDS	[0x00000001,0x00000001,0x00000000,...	0xFFF7CC00
[0]	0x00000001 (Hex)	0xFFF7CC00
[1]	0x00000001 (Hex)	0xFFF7CC04
WRHS1_UN	{WRHS1_UL=0x35000004,WRHS1_ST...	0xFFF7CD00
WRHS1_UL	0x35000004 (Hex)	0xFFF7CD00
WRHS1_ST	{mbi_B1=0x1,txm_B1=0x1,ppit_B1=0...	0xFFF7CD00
mbi_B1	0x1 (Hex)	0xFFF7CD00 bit 29
txm_B1	0x1 (Hex)	0xFFF7CD00 bit 28
ppit_B1	0x0 (Hex)	0xFFF7CD00 bit 27
cfg_B1	0x1 (Hex)	0xFFF7CD00 bit 26
chb_B1	0x0 (Hex)	0xFFF7CD00 bit 25
cha_B1	0x1 (Hex)	0xFFF7CD00 bit 24
cyc_B7	0x0 (Hex)	0xFFF7CD00 bit 16-22
fid_B11	0x04 (Hex)	0xFFF7CD00 bit 0-10
WRHS2_UN	{WRHS2_UL=0x000904BF,WRHS2_ST...	0xFFF7CD04
WRHS2_UL	0x000904BF (Hex)	0xFFF7CD04
WRHS2_ST	{pl_B7=0x9,crc_B11=0x4BF} (Hex)	0xFFF7CD04
pl_B7	0x9 (Hex)	0xFFF7CD04 bit 16-22
crc_B11	0x4BF (Hex)	0xFFF7CD04 bit 0-10

Figura 6.31.Registros de escritura del búfer para ranura 4 en el nodo B.

Por último, al ejecutarse la tercera tarea de acuerdo con el planificador, se construye la trama serial que integra los estados de ambos paneles, para su monitoreo desde la interfaz como se muestra en la Figura 6.33.

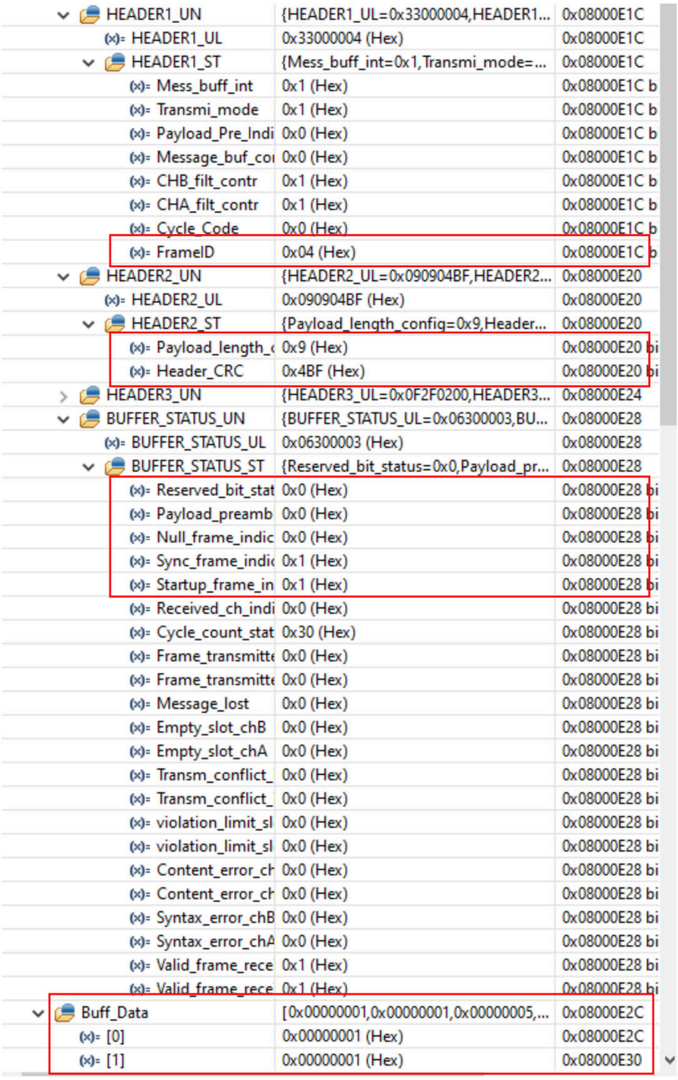


Figura 6.32. Registros de lectura del búfer de la ranura 4 en el nodo A.

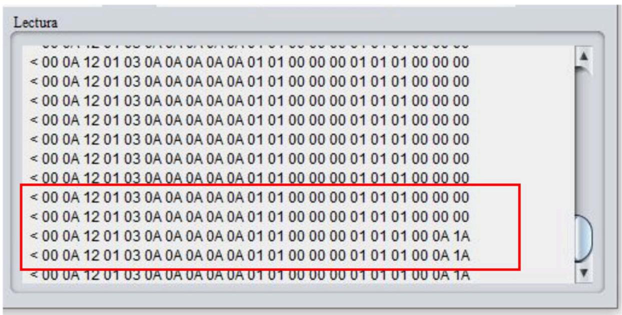


Figura 6.33. Lectura de trama de serial en la interfaz.

Conclusiones

Para el desarrollo del sistema FlexREY fue necesario identificar las especificaciones que debía cumplir con base en la Versión 2.1 Rev A del protocolo de comunicaciones FlexRay. Además de lo anterior, y con la finalidad de proporcionar una solución robusta, se realizó una investigación documental considerando artículos científicos, libros, manuales y normas, que permitieron segmentar y evaluar el sistema con base en la metodología de desarrollo (véase Capítulo 5). Dicha metodología propone dividir el sistema en sus partes HW y SW, y en este caso, facilitó la solución del problema y ayudó en la selección de herramientas.

Para implementar el protocolo de comunicaciones FlexRay, se optó por la tarjeta Hércules TMS570LC43 de la firma TI por ser la única, hasta ese momento, que permitía acceder a las terminales FlexRay a un precio accesible; además, proporciona herramientas de software libre (CCS y HALCoGen) para su programación. Al momento de empezar a conocer y estudiar dicha tarjeta de desarrollo, fue necesario implementar el HW del BD y programar manualmente una biblioteca que permitiera la utilización del módulo integrado FlexRay. Para ello se realizaron las siguientes tareas: a) aunque la información para la implementación de un BD FlexRay se encuentra poco desarrollada y en lenguajes como chino e inglés, se logró recabar la información suficiente para diseñar un prototipo basándose únicamente en la investigación teórica. Utilizando el transceptor TJA1080A e integrando un terminador de red al deseño, se implementó en PCB con un enfoque para pruebas, y b) se tomó como base una biblioteca para el módulo FlexRay de otra familia de MCU, la cual se modificó y adecuó al TMS570LC4357 para habilitar el núcleo E-Ray de la tarjeta Hercules.

Debido a los requerimientos del protocolo y a la complejidad del circuito del BD, se optó por mandar a fabricar su PCB a China, el proceso de fabricación y envío duró aproximadamente dos meses; la colocación y soldado de los

componentes se realizó manualmente. Con lo anterior se logró implementar una red FlexRay de dos nodos *coldstart*.

Con los nodos FlexRay operando, se procedió a implementar los paneles de prueba para el simulador del sistema MRS III; se eligió este sistema debido a que cumple con el enfoque de sistemas de tiempo real crítico y permite obtener un buen rendimiento del funcionamiento del protocolo de comunicaciones FlexRay.

Se configuró la tarjeta Hercules para utilizar sus periféricos y poder adquirir datos y controlar los indicadores. Los sensores de vibración seleccionados resultaron no ser eficientes y fue necesario establecer un mayor tiempo de muestreo, por lo que se optó por utilizar un MCU independiente como driver, el cual se encarga de muestrear y procesar los datos de las lecturas de impactos para enviar los datos finales de manera periódica al nodo A.

Se propusieron los diagramas eléctricos para las tarjetas de los paneles de sensores y de indicadores; se diseñó el esquema y distribución en el panel y se adaptó el diseño de las PCB en la que se montaron; para la alimentación se optó por acoplar una fuente ATX por sus diferentes niveles de voltaje y su disponibilidad. Se diseñó la estructura del panel con el tamaño necesario para contener la fuente de alimentación, las tarjetas de los nodos y las tarjetas de sensores y de indicadores; para su fabricación se escogió MDF, se pintó con pintura acrílica para protegerlo, se añadieron las ilustraciones correspondientes y se taladraron los espacios para los componentes. Con todos los elementos requeridos se montó el sistema colocando y asegurando cada tarjeta, finalmente, se procedió a realizar el cableado del sistema, como se describió en el Capítulo 6.

En paralelo al diseño de los diagramas eléctricos de las placas, se desarrolló el software embebido requerido para la operación de los módulos de HW requeridos (ADC, Timer, SCI y N2HET); además, se implementó el simulador del MRS III con base en el Capítulo 3 y la configuración de las comunicaciones con la GUI, todo ello siguiendo el esquema de sistemas

embebidos en tiempo real. Se asignaron las tres tareas principales a cada nodo y se configuró el planificador bajo este esquema.

El sistema presentó fallos debido al consumo de recursos y a la exigencia en los tiempos de respuesta. Por ello, se implementó la zona crítica de código para garantizar la correcta operación del módulo FlexRay y se configuró la transmisión FlexRay como se describe en el apartado 5.4.2.

Para el desarrollo de la GUI, se estableció que la implementación fuera amigable con el usuario. Se asignaron las funciones principales en hilos con el fin de permitir un mejor desempeño y mayor fluidez en su funcionamiento. Se integró la biblioteca de comunicación serial y se configuró para la recepción por evento de un paquete delimitado por un símbolo, el cual se ejecuta en un hilo independiente, permitiendo un monitoreo constante sin bloquear la operatividad de la interfaz. Además, el proceso de monitoreo se encargó de actualizar los datos de la GUI de acuerdo a los cambios realizados por el usuario o por los datos recibidos del panel, basándose en el modo de operación seleccionado. Cabe destacar que la velocidad de comunicación de la interfaz serial se estableció en 115 200 baudios ya que la transmisión no era completada en menos de 6 ms en velocidades inferiores, generando problemas de sincronización entre la interfaz y el nodo.

La construcción del paquete serial se basó en los parámetros de la trama FlexRay, pero sólo se utilizan el identificador, la longitud y la carga útil; para construir una trama personalizada completa de FlexRay se añadió la función de trama personalizada, pero para la aplicación del sistema FlexREY no se validó la trasmisión sobre el bus ya que no son necesarios más casos.

Finalmente, al realizar las pruebas del sistema y monitorear las tramas en el osciloscopio debido a la frecuencia de operación y a la resolución del equipo, la calidad de la imagen no pareció aceptable por lo que se decidió capturar los datos y procesarlos en Matlab para generar un resultado de mucha mejor calidad, además de permitir la decodificación manual de los datos que al ser

comparada con la información en los registros de depuración de las tarjetas e interfaz, estos se correspondieron totalmente (como se describe en el subcapítulo 6.3.2). Con ello se pudo comprobar que la implementación de una red basada en el protocolo FlexRay permite simular el funcionamiento del sistema de tiempo crítico como es el sistema MRS III de la firma BMW.

Es evidente que el mayor potencial y la importancia del trabajo desarrollado es la implementación de la red FlexRay, ya que con esto es posible conectar componentes automotrices comerciales o industriales permitiendo el estudio e investigación de múltiples sistemas que trabajen bajo el protocolo de comunicaciones FlexRay, siendo este la vanguardia en automoción.

Una de las mayores complicaciones para el desarrollo del trabajo fue la falta de equipo y herramientas enfocadas a comunicaciones automotrices para el diseño de la tarjeta del BD, ya que todo su diseño se basó únicamente en la teoría de la investigación realizada y no se encontró forma de probar su funcionamiento hasta que se fabricó y se puso en marcha el núcleo E-Ray de la tarjeta Hercules.

Por otro lado, los sensores de vibración utilizados generaron problemas ya que al conectarse en un plano inclinado del panel disminuyó la calidad de sus lecturas, que por ser económicos requerían de un acondicionamiento previo forzando que se utilizara el ATMEGA328p como driver para mejorar las lecturas. Además, resulta ser un poco complicado poder realizar las pruebas para distintos casos de impacto ya que los resultados varían mucho en las lecturas de estos sensores.

Un problema similar fue que los interruptores de palanca generaban mucho ruido en las lecturas, pero se logró solucionar al habilitar la opción Enable Sampling Capacitor Discharge en HALCoGen, lo cual permite retener el valor de la muestra durante el tiempo de lectura, sin embargo, no se consideró que también podían generar mucha carga en el sistema por lo que utilizarlos hizo que el potenciómetro varie su nivel de voltaje haciendo variar las lecturas de la velocidad, ya que faltó incorporar un seguidor de voltaje para evitar este

problema. Otro punto a considerar es que inicialmente resultó difícil utilizar el abstractor de hardware por la gran cantidad de periféricos y recursos que maneja, así como la utilización de las bibliotecas generadas para la programación, por lo que se recomienda leer a detalle las hojas de especificaciones del periférico utilizado , lo cual facilita el uso de los recursos, ayuda a definir claramente los requerimientos para configurar el hardware y sólo requiere invocar las secuencias de inicialización requeridas en el software embebido.

Futuros Trabajos de Investigación

Con los resultados obtenidos del sistema FlexREY se proponen los siguientes proyectos de investigación:

- Desarrollar un software embebido y un software más robusto para analizar en tiempo real las tramas con la finalidad de evaluar sistemas FlexRay semejantes a los utilizados por los escáneres CAN.
- Integración de una base de datos para el análisis y la evaluación de componentes comerciales que soporten el protocolo de comunicaciones FlexRay.
- Ampliación de la simulación de casos del sistema *airbag*, ya que este se restringe a condiciones lineales de prueba, dejando de lado condiciones con mayores variaciones como la inclinación o dirección del automóvil.
- Implementación y prueba de sistemas x-by-wire utilizando las prestaciones del protocolo FlexRay.
- Construcción de múltiples plataformas de pruebas para estudio de sistemas automotrices de manera práctica.

Bibliografía

Berger, A. S. (2002). *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. Lawrence, Kansas 66046 USA: CMP Books.

BMW of North America, Inc. (Junio de 2001). *Passive Safety Systems Technical Training E46*. Recuperado el 3 de Marzo de 2020, de http://v12.dyndns.org/BMW/BMW%203%20(E46)/7%20Passive%20Safety%20Systems.pdf

Carpio Guartambel, C. P. (2013). *Manual de Procedimientos Para Interactuar Entre Protocolos de Comunicación Automotriz*. Cuenca: Universidad del Azuay.

Chamú Morales, C. A. (2005). *Desarrollo de un Sistema Educativo para la Enseñanza del Protocolo de Comunicaciones CAN*. Huajuapan de León: UTM.

CiA. (15 de 11 de 2018). *CAN in Automation (CiA)*. Obtenido de History of the CAN technology: https://www.can-cia.org/can-knowledge/can/can-history/

Coronel, J. O., Blanes, F., Pérez, P., Albero, M., Benet, G., & Simó, J. E. (2006). SCoCAN: Un Protocolo de Comunicaciones de Tiempo Real para Sistemas Empotrados Distribuidos. Aplicación al Control de Robots. *Revista Iberoamericana de Automática e Informática Industrial*, 71-78.

FlexRay Consortium. (Octubre de 2010). *FlexRay™ Protocol Specification Version 3.0.1*. Recuperado el 19 de Agosto de 2019, de https://svn.ipd.kit.edu/nlrp/public/FlexRay/FlexRay™%20Protocol%20Specification%20Version%203.0.1.pdf

Freescale Semiconductor, Inc. (28-30 de Junio de 2006). *FlexRay – A Communications Network for Automotive Control Systems*. Recuperado el 19 de Agosto de 2019, de

https://pdfs.semanticscholar.org/3d2b/f9238ae96046eb687eec5ac1a37c2c5007ae.pdf

Galeano, G. (2009). *Programción de SISTEMAS EMBEBIDOS en C.* Alfaomega Grupo Editor.

González Salinas, R. (2008). *Especificación del Protocolo FlexRay utilizando un Lenguaje de Descripción Formal.* Huajuapan de León: UTM.

HELLA S.A. (2019). *ESTRUCTURA Y FUNCIONAMIENTO DEL AIRBAG*. Recuperado el 2 de Marzo de 2020, de HELLA TECH WORLD: https://www.hella.com/techworld/es/Informacion-Tecnica/Electricidad-y-electronica-del-automovil/Sistema-airbag-3083/

Iversen Huse, M. (Junio de 2017). *FlexRay Analysis, Configuration Parameter Estimation, and Adversaries.* Norwegian University of Science and Technology. Obtenido de Semantic Scholar: https://pdfs.semanticscholar.org/b3d1/c61fc0c4f8658589d88f1526b147f276bc5b.pdf

Jiménez García, V. (2008). *Estudio del nuevo bus de automoción Flexray y diseño de un prototipo ilustrativo de la tecnología.* Barcelona: Universidad Politécnica de Cataluña.

Martínez Requena, A. (2017). *Introducción a CAN bus: Descripción, ejemplos y aplicaciones de tiempo real.* Madrid: Universidad Politécnica de Madrid.

Murcia Educarm. (2019). *El airgbag*. Recuperado el 3 de Marzo de 2020, de Educarm.es: http://servicios.educarm.es/templates/portal/ficheros/websDinamicas/21/airbag.pdf

National Instruments Corporation. (28 de Mayo de 2019). *National Instruments*. Obtenido de FlexRay Automotive Communication Bus Overview: https://www.ni.com/es-mx/innovations/white-papers/06/flexray-automotive-communication-bus-overview.html

Nexperia. (2008). PESD1CAN: CAN bus ESD protection diode. *Nexperia*, 1-13.

NXP B.V. (28 de Noviembre de 2012). *TJA1080A FlexRay transceiver Rev. 6.* Obtenido de http://www.nxp.com: https://www.nxp.com/docs/en/data-sheet/TJA1080A.pdf

Paret, D. (2007). *Multiplexed Networks for Embedded systems.* Chichester: John Wiley & Sons Ltd.

Paret, D. (2012). *FlexRay and its Applications Real Time Multiplexed Network.* París: John Wiley & Sons Ltd.

Robertson, M. (2016). Simplify CAN bus implementations with chokeless transceivers. *Texas Instruments Incorporated*, 1-6.

Salazar Serna, C. A., & Correa Ortiz, L. C. (2011). Buses de Campo y Protocolos en Redes Industriales. *Ventana Informática*(25), 83-109.

Texas Instruments Incorporated. (2011). *Hercules™ Safety Microcontrollers.* Obtenido de Texas Instruments: http://www.ti.com/en/download/mcu/SPRB204.pdf?DCMP=hercules&HQS=hercules-mc

Texas Instruments Incorporated. (2012). *FlexRay Transfer Unit (FTU) Setup.* Obtenido de Texas Instruments: http://www.ti.com/lit/an/spna145/spna145.pdf

Texas Instruments Incorporated. (2015). *FlexRay Module Training.* Obtenido de Texas Instruments: http://www.ti.com/lit/ml/sprt718/sprt718.pdf

Texas Instruments Incorporated. (Agosto de 2016). *How to Create A HALCoGen-Based Project For CCS (Rev. B).* Recuperado el 19 de Agosto de 2019, de Texas Instruments: http://www.ti.com/lit/an/spna121b/spna121b.pdf

Texas Instruments Incorporated. (Junio de 2016). *TMS570LC4357 Hercules™ Microcontroller Based on the ARM® Cortex®-R Core datasheet (Rev. C).* Recuperado el 19 de Agosto de 2019, de Texas Instruments: http://www.ti.com/lit/ds/symlink/tms570lc4357.pdf

Texas Instruments Incorporated. (Marzo de 2018). *TMS570LC43x 16/32 RISC Flash Microcontroller Technical Reference Manual (Rev. A).* Recuperado el 20 de Agosto de 2019, de Texas Instruments: http://www.ti.com/lit/ug/spnu563a/spnu563a.pdf

Texas Instruments Incorporated. (29 de Octubre de 2019). *Hercules TMS570LC43x LaunchPad Development Kit*. Obtenido de www.ti.com: http://www.ti.com/tool/LAUNCHXL2-570LC43

Texas Instruments Incorporated. (2019). *TMS570LC43x and RM57Lx LaunchPad Schematic.* Recuperado el 20 de Agosto de 2019, de Texas Instruments: http://www.ti.com/lit/df/sprr397/sprr397.pdf

Universitatea Politehnica Timisoara. (06 de Octubre de 2019). *http://www.aut.upt.ro.* Obtenido de FlexRay protocol: http://www.aut.upt.ro/~pal-stefan.murvay/teaching/nes/Lecture_08_FlexRay.pdf

Vidal I., J. A., Zúñiga G., M. S., & Rojas A., O. A. (15 de Septiembre de 2019). *Implementacón de una red industrial CAN para un sistema SCADA.* Obtenido de academia.edu: https://www.academia.edu/3177876/Implementacion_de_una_red_industrial_CAN_para_una_sistema_SCADA

Xin He, J. W. (15 de Octubre de 2018). *Design Scheme of Communication Node Based on FlexRay Bus*. Obtenido de edpsciences: https://doi.org/10.1051/matecconf/201821401001

Anexo A. Circuitos Impresos

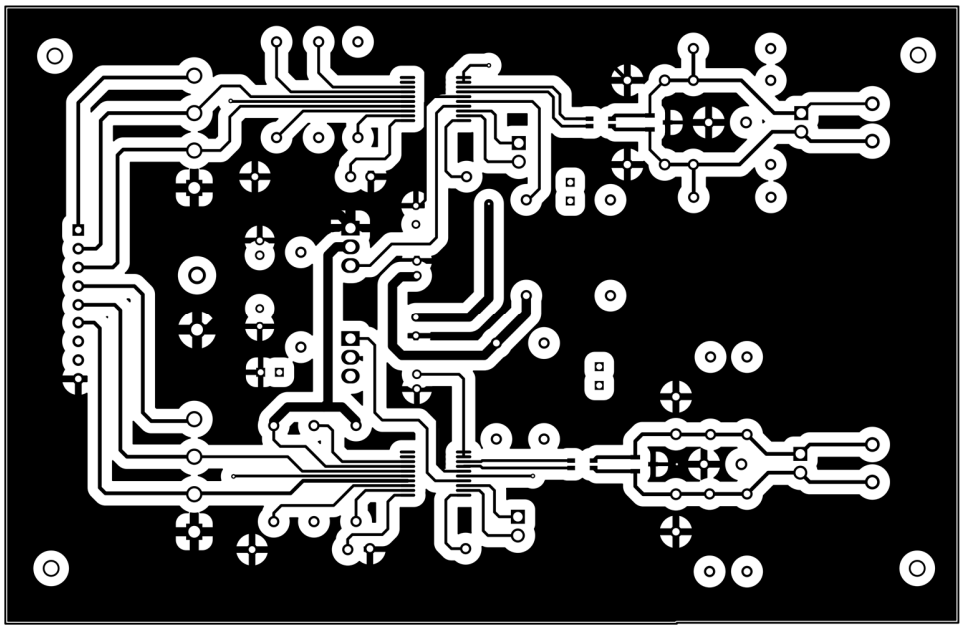


Figura A: 1. BD capa superior.

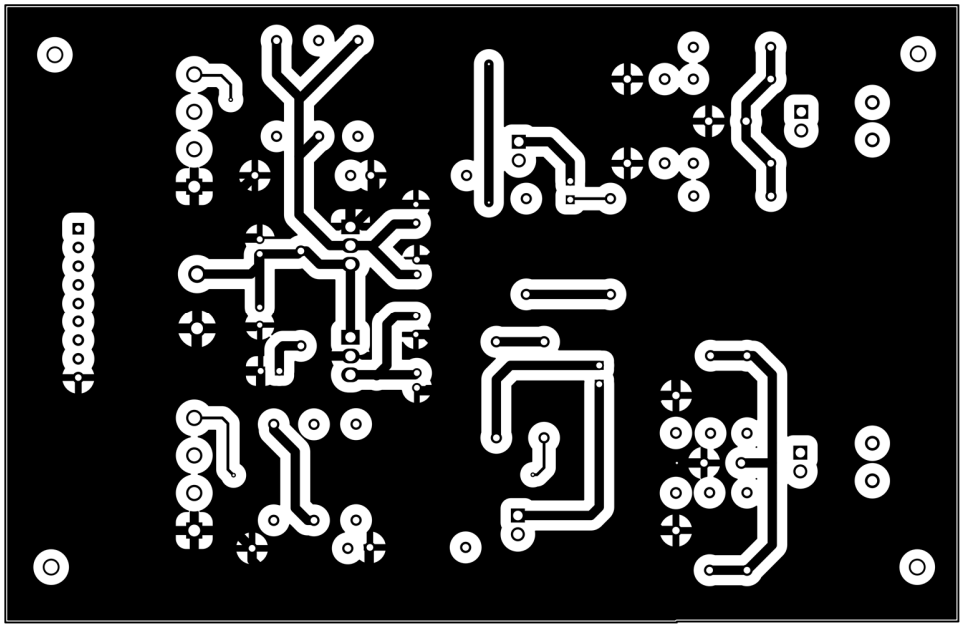


Figura A: 2. BD capa inferior.

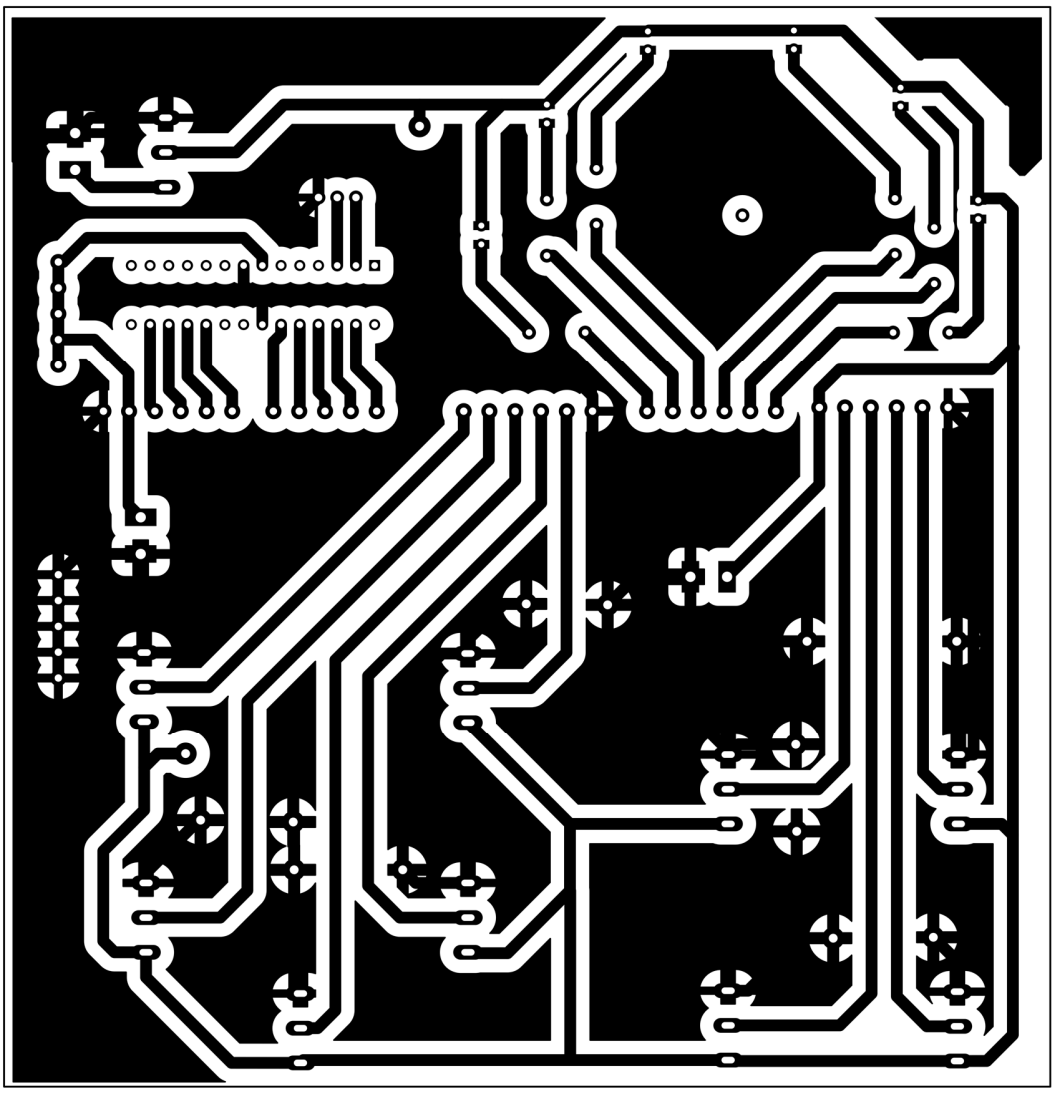


Figura A: 3. Panel de sensores.

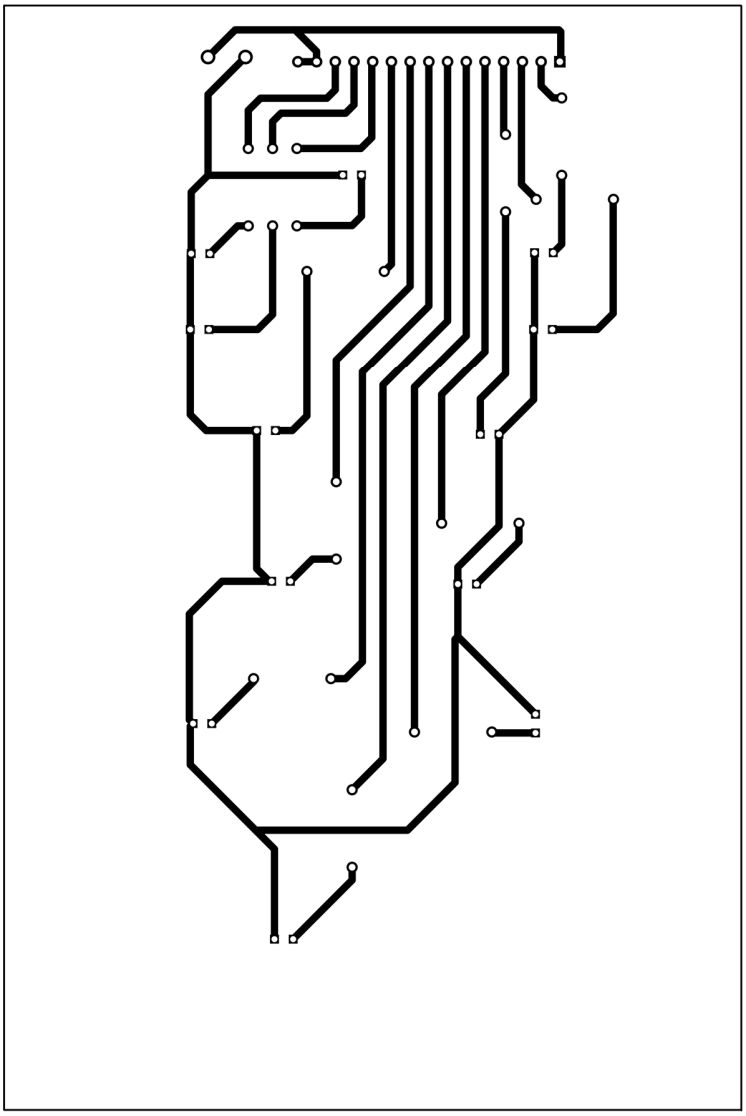


Figura A: 4. Panel de indicadores.

Anexo B. Código biblioteca FlexRay(fray.h)

```
#ifndef __FRAY_H__
#define __FRAY_H__

// CMD constants (SUCC1)

#define CMD_command_not_accepted          0x0
#define CMD_CONFIG                        0x1
#define CMD_READY                         0x2
#define CMD_WAKEUP                        0x3
#define CMD_RUN                           0x4
#define CMD_ALL_SLOTS                     0x5
#define CMD_HALT                          0x6
#define CMD_FREEZE                        0x7
#define CMD_SEND_MTS                      0x8
#define CMD_ALLOW_COLDSTART               0x9
#define CMD_RESET_STATUS_INDICATORS       0xA
#define CMD_MONITOR_MODE                  0xB
#define CMD_CLEAR_RAMS                    0xC
#define CMD_LOOPBACK MODE                 0xF

#include <stdint.h>
#define FLEX_PAYLOAD 64

#define slot_Stup 0;
#define slot1 1;
#define slot2 2;
#define slot3 3;
#define slot4 4;
#define mslot10 10;
#define mslot11 11;
#define mslot12 12;
#define mslot13 13;
#define mslot14 14;

typedef volatile struct fray_registers //Communication Controller Registers
{
   /* ---------------------------------------------------------------------- */
   /* FRAY_ST                                                                */
   /*  Definition of the FLEXRAY register map                                */

   /* ECC Control Register */
   /* 0x0 */
   union ECC_CTRL
   {
      unsigned long ECC_CTRL_UL;
      struct
      {
         unsigned :4;
         unsigned SBE_EVT_EN :4; //ECC Single-Bit Error Indication.
```

```
      unsigned :4;
      unsigned SBEL_B4 :4; //ECC Single-Bit Error Lock
      unsigned :12;
      unsigned DIAGSEL_B4 :4; //Diagnostic Mode select Key

   } ECC_CTRL_ST;
} ECC_CTRL_UN;

/* ECC Diagnostic Status Register */
/* 0x4 */
union ECCDSTAT
{
   unsigned long ECCDSTAT_UL;
   struct
   {
      unsigned :8;
      unsigned DEFH_B1 :1;
      unsigned DEFG_B1 :1;
      unsigned DEFF_B1 :1;
      unsigned DEFE_B1 :1;
      unsigned DEFD_B1 :1;
      unsigned DEFC_B1 :1;
      unsigned DEFB_B1 :1;
      unsigned DEFA_B1 :1;
      unsigned :8;
      unsigned SEFH_B1 :1;
      unsigned SEFG_B1 :1;
      unsigned SEFF_B1 :1;
      unsigned SEFE_B1 :1;
      unsigned SEFD_B1 :1;
      unsigned SEFC_B1 :1;
      unsigned SEFB_B1 :1;
      unsigned SEFA_B1 :1;

   } ECCDSTAT_ST;
} ECCDSTAT_UN;

/* ECC Test Register */
/* 0x8 */
union ECCTEST
{
   unsigned long ECCTEST_UL;
   struct
   {
      unsigned :9;
      unsigned RDECC_B7 :7;
      unsigned :9;
      unsigned WRECC_B7 :7;

   } ECCTEST_ST;
} ECCTEST_UN;

/* Single-Bit Error Status Register */
```

```
/* 0xC */
union SBESTAT
{
   unsigned long SBESTAT_UL;
   struct
   {
      unsigned SBE :1;
      unsigned :16;
      unsigned FMB :7;
      unsigned :1;
      unsigned MFMB :1;
      unsigned FMBD :1;
      unsigned STBF2 :1;
      unsigned STBF1 :1;
      unsigned SMR :1;
      unsigned SOBF :1;
      unsigned SIBF :1;
   } SBESTAT_ST;
} SBESTAT_UN;

/*  Special Registers                                          */
/* Test Register 1                                            */
/* 0x10 */
union test1
{
   unsigned long TEST1_UL;
   struct
   {
      unsigned CERB_B4 :4;   //Coding Error Report Channel B.
      unsigned CERA_B4 :4;   //Coding Error Report Channel A
      unsigned :2;
      unsigned txenb_B1 :1; /* Control of Channel B Transmit Enable Pin    */
      unsigned txena_B1 :1; /* Control of Channel A Transmit Enable Pin    */
      unsigned txb_B1 :1; /* Control of Channel B Transmit Pin          */
      unsigned txa_B1 :1; /* Control of Channel A Transmit Pin          */
      unsigned rxb_B1 :1; /* Monitor Channel B Receive Pin              */
      unsigned rxa_B1 :1; /* Monitor Channel A Receive Pin              */
      unsigned :6;
      unsigned AOB_B1 :1;   //Activity on B
      unsigned AOA_B1 :1;   //Activity on A
      unsigned :2;
      unsigned tmc_B2 :2; /* Test Mode Control                        */
      unsigned :2;
      unsigned ELBE_B1 :1;   //External Loop Back Enable.
      unsigned wrten_B1 :1; /* Write Test Register Enable               */
   } TEST1_ST;
} TEST1_UN;

/* Test Register 2                                            */
/* 0x14 */
union test2
{
   unsigned long TEST2_UL;
```

```
      struct
      {
         unsigned :16;
         unsigned rdpb_B1 :1; //When ECC mode is enabled, this bit is always read
as 0
         unsigned wrpb_B1 :1; //When ECC mode is enabled, this bit has no effect.
         unsigned :7;
         unsigned ssel_B3 :3;   //Segment select.
         unsigned :1;
         unsigned rs_B3 :3;   //RAM select
      } TEST2_ST;
   } TEST2_UN;

   unsigned :32;

   /* Lock Register                                              */
   /* 0x1C */
   union lck
   {
      unsigned long LCK_UL;
      struct
      {
         unsigned :16;  //Reserved
         unsigned tmk_B8 :8;   //Test mode key.
         unsigned clk_B8 :8;   //Configuration lock key
      } LCK_ST;
   } LCK_UN;

   /*  Interrupt Registers                                        */
   /* Error Input Register                                        */
   /* 0x20 */
   union eir
   {
      unsigned long EIR_UL;
      struct
      {
         unsigned :5;
         unsigned tabb_B1 :1;
         unsigned ltvb_B1 :1;
         unsigned edb_B1 :1;
         unsigned :5;
         unsigned taba_B1 :1;
         unsigned ltva_B1 :1;
         unsigned eda_B1 :1;
         unsigned :4;
         unsigned mhf_B1 :1;
         unsigned ioba_B1 :1;
         unsigned iiba_B1 :1;
         unsigned efa_B1 :1;
         unsigned rfo_B1 :1;
         unsigned perr_B1 :1;
         unsigned ccl_B1 :1;
         unsigned ccf_B1 :1;
```

```
      unsigned sfo_B1 :1;
      unsigned sfbm_B1 :1;
      unsigned cna_B1 :1;
      unsigned pemc_B1 :1;
   } EIR_ST;
} EIR_UN;

/* Status Interrupt Register                                    */
/* 0x24 */
union sir
{
   unsigned long SIR_UL;
   struct
   {
      unsigned :6;
      unsigned mtsb_B1 :1;
      unsigned wupb_B1 :1;
      unsigned :6;
      unsigned mtsa_B1 :1;
      unsigned wupa_B1 :1;
      unsigned sds_B1 :1;
      unsigned mbsi_B1 :1;
      unsigned sucs_B1 :1;
      unsigned swe_B1 :1;
      unsigned tobc_B1 :1;
      unsigned tibc_B1 :1;
      unsigned ti1_B1 :1;
      unsigned ti0_B1 :1;
      unsigned nmvc_B1 :1;
      unsigned rfcl_B1 :1;
      unsigned rfne_B1 :1;
      unsigned rxi_B1 :1;
      unsigned txi_B1 :1;
      unsigned cycs_B1 :1;
      unsigned cas_B1 :1;
      unsigned wst_B1 :1;
   } SIR_ST;
} SIR_UN;

/* Error Interrupt Line Select                                  */
/* 0x28 */
union eils
{
   unsigned long EILS_UL;
   struct
   {
      unsigned :5;
      unsigned tabbl_B1 :1;
      unsigned ltvbl_B1 :1;
      unsigned edbl_B1 :1;
      unsigned :5;
      unsigned tabal_B1 :1;
      unsigned ltval_B1 :1;
```

```
      unsigned edal_B1 :1;
      unsigned :4;
      unsigned mhfl_B1 :1;
      unsigned iobal_B1 :1;
      unsigned iibal_B1 :1;
      unsigned efal_B1 :1;
      unsigned rfol_B1 :1;
      unsigned perrl_B1 :1;
      unsigned ccll_B1 :1;
      unsigned ccfl_B1 :1;
      unsigned sfol_B1 :1;
      unsigned sfbml_B1 :1;
      unsigned cnal_B1 :1;
      unsigned pemcl_B1 :1;
   } EILS_ST;
} EILS_UN;

/* Status Interrupt Line Select                                    */
/* 0x2C */
union sils
{
   unsigned long SILS_UL;
   struct
   {
      unsigned :6;
      unsigned mtsbl_B1 :1;
      unsigned wupbl_B1 :1;
      unsigned :6;
      unsigned mtsal_B1 :1;
      unsigned wupal_B1 :1;
      unsigned sdsl_B1 :1;
      unsigned mbsil_B1 :1;
      unsigned sucsl_B1 :1;
      unsigned swel_B1 :1;
      unsigned tobcl_B1 :1;
      unsigned tibcl_B1 :1;
      unsigned ti1l_B1 :1;
      unsigned ti0l_B1 :1;
      unsigned nmvcl_B1 :1;
      unsigned rffl_B1 :1;
      unsigned rfnel_B1 :1;
      unsigned rxil_B1 :1;
      unsigned txil_B1 :1;
      unsigned cycsl_B1 :1;
      unsigned casl_B1 :1;
      unsigned wstl_B1 :1;
   } SILS_ST;
} SILS_UN;

/* Error Interrupt Enable Set                                      */
/* 0x30 */
union eies
{
```

```
    unsigned long EIES_UL;
    struct
    {
       unsigned :5;
       unsigned tabbe_B1 :1;
       unsigned ltvbe_B1 :1;
       unsigned edbe_B1 :1;
       unsigned :5;
       unsigned tabae_B1 :1;
       unsigned ltvae_B1 :1;
       unsigned edae_B1 :1;
       unsigned :4;
       unsigned mhfe_B1 :1;
       unsigned iobae_B1 :1;
       unsigned iibae_B1 :1;
       unsigned efae_B1 :1;
       unsigned rfoe_B1 :1;
       unsigned perre_B1 :1;
       unsigned ccle_B1 :1;
       unsigned ccfe_B1 :1;
       unsigned sfoe_B1 :1;
       unsigned sfbme_B1 :1;
       unsigned cnae_B1 :1;
       unsigned pemce_B1 :1;
    } EIES_ST;
} EIES_UN;

/* Error Interrupt Enable Reset                                    */
/* 0x34 */
union eier
{
    unsigned long EIER_UL;
    struct
    {
       unsigned :5;
       unsigned tabbe_B1 :1;
       unsigned ltvbe_B1 :1;
       unsigned edbe_B1 :1;
       unsigned :5;
       unsigned tabae_B1 :1;
       unsigned ltvae_B1 :1;
       unsigned edae_B1 :1;
       unsigned :4;
       unsigned mhfe_B1 :1;
       unsigned iobae_B1 :1;
       unsigned iibae_B1 :1;
       unsigned efae_B1 :1;
       unsigned rfoe_B1 :1;
       unsigned perre_B1 :1;
       unsigned ccle_B1 :1;
       unsigned ccfe_B1 :1;
       unsigned sfoe_B1 :1;
       unsigned sfbme_B1 :1;
```

```
      unsigned cnae_B1 :1;
      unsigned pemce_B1 :1;
   } EIER_ST;
} EIER_UN;

/* Status Interrupt Enable Set                                    */
/* 0x38 */
union sies
{
   unsigned long SIES_UL;
   struct
   {
      unsigned :6;
      unsigned mtsbe_B1 :1;
      unsigned wupbe_B1 :1;
      unsigned :6;
      unsigned mtsae_B1 :1;
      unsigned wupae_B1 :1;
      unsigned sdse_B1 :1;
      unsigned mbsie_B1 :1;
      unsigned sucse_B1 :1;
      unsigned swee_B1 :1;
      unsigned tobce_B1 :1;
      unsigned tibce_B1 :1;
      unsigned ti1e_B1 :1;
      unsigned ti0e_B1 :1;
      unsigned nmvce_B1 :1;
      unsigned rffe_B1 :1;
      unsigned rfnee_B1 :1;
      unsigned rxie_B1 :1;
      unsigned txie_B1 :1;
      unsigned cycse_B1 :1;
      unsigned case_B1 :1;
      unsigned wste_B1 :1;
   } SIES_ST;
} SIES_UN;

/* Status Interrupt Enable Reset                                   */
/* 0x3C */
union sier
{
   unsigned long SIER_UL;
   struct
   {
      unsigned :6;
      unsigned mtsbe_B1 :1;
      unsigned wupbe_B1 :1;
      unsigned :6;
      unsigned mtsae_B1 :1;
      unsigned wupae_B1 :1;
      unsigned sdse_B1 :1;
      unsigned mbsie_B1 :1;
      unsigned sucse_B1 :1;
```

```
      unsigned swee_B1 :1;
      unsigned tobce_B1 :1;
      unsigned tibce_B1 :1;
      unsigned ti1e_B1 :1;
      unsigned ti0e_B1 :1;
      unsigned nmvce_B1 :1;
      unsigned rffe_B1 :1;
      unsigned rfnee_B1 :1;
      unsigned rxie_B1 :1;
      unsigned txie_B1 :1;
      unsigned cycse_B1 :1;
      unsigned case_B1 :1;
      unsigned wste_B1 :1;
   } SIER_ST;
} SIER_UN;

/* Interrupt Line Enable                                    */
/* 0x40 */
union ile
{
   unsigned long ILE_UL;
   struct
   {
      unsigned :30;
      unsigned eint1_B1 :1;
      unsigned eint0_B1 :1;
   } ILE_ST;
} ILE_UN;

/* Timer 0 Configuration                                    */
/* 0x44 */
union t0c
{
   unsigned long T0C_UL;
   struct
   {
      unsigned :2;
      unsigned t0mo_B14 :14;
      unsigned :1;
      unsigned t0cc_B7 :7;
      unsigned :6;
      unsigned t0ms_B1 :1;
      unsigned t0rc_B1 :1;
   } T0C_ST;
} T0C_UN;

/* Timer 1 Configuration                                    */
/* 0x48 */
union t1c
{
   unsigned long T1C_UL;
   struct
   {
```

```
      unsigned :2;
      unsigned t1mc_B14 :14;
      unsigned :14;
      unsigned t1ms_B1 :1;
      unsigned t1rc_B1 :1;
   } T1C_ST;
} T1C_UN;

/* Stop Watch Register 1                                    */
/* 0x4C */
union stpw1
{
   unsigned long STPW1_UL;
   struct
   {
      unsigned :2;
      unsigned smtv_B14 :14;
      unsigned :2;
      unsigned sccv_B6 :6;
      unsigned :1;
      unsigned eint1_B1 :1;
      unsigned eint0_B1 :1;
      unsigned eetp_B1 :1;
      unsigned sswt_B1 :1;
      unsigned edge_B1 :1;
      unsigned swms_B1 :1;
      unsigned eswt_B1 :1;
   } STPW1_ST;
} STPW1_UN;

/* Stop Watch Register 2                                    */
/* 0x50 */
union stpw2
{
   unsigned long STPW2_UL;
   struct
   {
      unsigned :5;
      unsigned SSCVB :11;
      unsigned :5;
      unsigned SSCVA :11;
   } STPW2_ST;
} STPW2_UN;

unsigned long RES1[11]; //Reserved Interrupt Registers

/*  CC Control Registers                                   */
/* SUC Configuration Register 1                             */
/* 0x80 */
union succ1
{
   unsigned long SUCC1_UL;
   struct
```

```
    {
      unsigned :4; //MSB    Reserved
      unsigned cchb_B1 :1; //Connected to channel B
      unsigned ccha_B1 :1; //Connected to channel A
      unsigned mtsb_B1 :1; //Select channel A for MTS Transmission B
      unsigned mtsa_B1 :1; //Select channel A for MTS Transmission A
      unsigned hcse_B1 :1; //Halt due to clock sync error
      unsigned tsm_B1 :1; //Transmission slot mode
      unsigned wucs_B1 :1; //Wakeup channel select
      unsigned pta_B5 :5; //Passive to active
      unsigned csa_B5 :5; //Cold start attempts
      unsigned :1; //Reserved
      unsigned txsy_B1 :1; //Transmit sync frame in key slot
      unsigned txst_B1 :1; //Transmit startup frame in key slot
      unsigned pbsy_B1 :1; //POC busy
      unsigned :3; //Reserved
      unsigned cmd_B4 :4; //LSB The controller host interface command vector
    } SUCC1_ST;
  } SUCC1_UN;

  /* SUC Configuration Register 2                                    */
  /* 0x84 */
  union succ2
  {
    unsigned long SUCC2_UL;
    struct
    {
      unsigned :4;
      unsigned ltn_B4 :4;
      unsigned :3;
      unsigned lt_B21 :21;
    } SUCC2_ST;
  } SUCC2_UN;

  /* SUC Configuration Register 3                                    */
  /* 0x88 */
  union succ3
  {
    unsigned long SUCC3_UL;
    struct
    {
      unsigned :24;
      unsigned wcf_B4 :4;
      unsigned wcp_B4 :4;
    } SUCC3_ST;
  } SUCC3_UN;

  /* NEM Configuration Register                                      */
  /* 0x8C */
  union nemc
  {
    unsigned long NEMC_UL;
    struct
```

```
    {
      unsigned :28;
      unsigned nml_B4 :4;
    } NEMC_ST;
} NEMC_UN;

/* PRT Configuration Register 1                                    */
/* 0x90 */
union prtc1
{
    unsigned long PRTC1_UL;
    struct
    {
      unsigned rwp_B6 :6;
      unsigned :1;
      unsigned rxw_B9 :9;
      unsigned brp_B2 :2;
      unsigned spp_B2 :2;
      unsigned :1;
      unsigned casm_B7 :7;
      unsigned tsst_B4 :4;
    } PRTC1_ST;
} PRTC1_UN;

/* PRT Configuration Register 2                                    */
/* 0x94 */
union prtc2
{
    unsigned long PRTC2_UL;
    struct
    {
      unsigned :2;
      unsigned txl_B6 :6;
      unsigned txi_B8 :8;
      unsigned :2;
      unsigned rxl_B6 :6;
      unsigned :2;
      unsigned rxi_B6 :6;
    } PRTC2_ST;
} PRTC2_UN;

/* MHD Configuration Register                                      */
/* 0x98 */
union mhdc
{
    unsigned long MHDC_UL;
    struct
    {
      unsigned :3;
      unsigned slt_B13 :13;
      unsigned :9;
      unsigned sfdl_B7 :7;
    } MHDC_ST;
```

```
} MHDC_UN;

unsigned :32;

/* GTU Configuration Register 1                                    */
/* 0xA0 */
union gtuc1
{
   unsigned long GTUC1_UL;
   struct
   {
      unsigned :12;
      unsigned ut_B20 :20;
   } GTUC1_ST;
} GTUC1_UN;

/* GTU Configuration Register 2                                    */
/* 0xA4 */
union gtuc2
{
   unsigned long GTUC2_UL;
   struct
   {
      unsigned :12;
      unsigned snm_B4 :4; //Sync node max (in frames).
      unsigned :2;
      unsigned mpc_B14 :14; //Macrotick per cycle (in macroticks).
   } GTUC2_ST;
} GTUC2_UN;

/* GTU Configuration Register 3                                    */
/* 0xA8 */
union gtuc3
{
   unsigned long GTUC3_UL;
   struct
   {
      unsigned :1;
      unsigned miob_B7 :7;
      unsigned :1;
      unsigned mioa_B7 :7;
      unsigned uiob_B8 :8;
      unsigned uioa_B8 :8;
   } GTUC3_ST;
} GTUC3_UN;

/* GTU Configuration Register 4                                    */
/* 0xAC */
union gtuc4
{
   unsigned long GTUC4_UL;
   struct
   {
```

```
      unsigned :2;
      unsigned ocs_B14 :14;
      unsigned :2;
      unsigned nit_B14 :14;
   } GTUC4_ST;
} GTUC4_UN;

/* GTU Configuration Register 5                    */
/* 0xB0 */
union gtuc5
{
   unsigned long GTUC5_UL;
   struct
   {
      unsigned dec_B8 :8;
      unsigned :3;
      unsigned cdd_B5 :5;
      unsigned dcb_B8 :8;
      unsigned dca_B8 :8;
   } GTUC5_ST;
} GTUC5_UN;

/* GTU Configuration Register 6                    */
/* 0xB4 */
union gtuc6
{
   unsigned long GTUC6_UL;
   struct
   {
      unsigned :5;
      unsigned mod_B11 :11;
      unsigned :5;
      unsigned asr_B11 :11;
   } GTUC6_ST;
} GTUC6_UN;

/* GTU Configuration Register 7                    */
/* 0xB8 */
union gtuc7
{
   unsigned long GTUC7_UL;
   struct
   {
      unsigned :6;
      unsigned nss_B10 :10;
      unsigned :6;
      unsigned ssl_B10 :10;
   } GTUC7_ST;
} GTUC7_UN;

/* GTU Configuration Register 8                    */
/* 0xBC */
union gtuc8
```

```
{
   unsigned long GTUC8_UL;
   struct
   {
      unsigned :3;
      unsigned nms_B13 :13;
      unsigned :10;
      unsigned msl_B6 :6;
   } GTUC8_ST;
} GTUC8_UN;

/* GTU Configuration Register 9                                    */
/* 0xC0 */
union gtuc9
{
   unsigned long GTUC9_UL;
   struct
   {
      unsigned :14;
      unsigned dsi_B2 :2;
      unsigned :3;
      unsigned mapo_B5 :5;
      unsigned :2;
      unsigned apo_B6 :6;
   } GTUC9_ST;
} GTUC9_UN;

/* GTU Configuration Register 10                                   */
/* 0xC4 */
union gtuc10
{
   unsigned long GTUC10_UL;
   struct
   {
      unsigned :5;
      unsigned mrc_B11 :11;
      unsigned :2;
      unsigned moc_B14 :14;
   } GTUC10_ST;
} GTUC10_UN;

/* GTU Configuration Register 11                                   */
/* 0xC8 */
union gtuc11
{
   unsigned long GTUC11_UL;
   struct
   {
      unsigned :5;
      unsigned erc_B3 :3;
      unsigned :5;
      unsigned eoc_B3 :3;
      unsigned :6;
```

```
        unsigned ercc_B2 :2;
        unsigned :6;
        unsigned ecc_B2 :2;
      } GTUC11_ST;
   } GTUC11_UN;

   /*    // BGS Configuration Register
    // 0xCC
    union bgs
    {
    unsigned long BGS_UL;
    struct
    {
    unsigned :22;
    unsigned dse_B1 :1;
    unsigned bgd_B1 :1;
    unsigned :2;
    unsigned bgt_B6 :6;
    } BGS_ST;
    } BGS_UN;

    unsigned long RES2[12]; // Reserved
    */
   unsigned long RES2[13]; // Reserved Communication Controller Status
Registers

   /* CC Status Registers                                          */
   /* CC Status Vector                                            */
   /* 0x100 */
   union ccsv
   {
      unsigned long CCSV_UL;
      struct
      {
         unsigned :2;
         unsigned psl_B6 :6;
         unsigned rca_B5 :5;
         unsigned wsv_B3 :3;
         unsigned :1;
         unsigned csi_B1 :1;
         unsigned csai_B1 :1;
         unsigned csni_B1 :1;
         unsigned :2;
         unsigned slm_B2 :2;
         unsigned hrq_B1 :1;
         unsigned fsi_B1 :1;
         unsigned pocs_B6 :6;
      } CCSV_ST;
   } CCSV_UN;

   /* CC Error Vector                                             */
   /* 0x104 */
   union ccev
```

```
{
   unsigned long CCEV_UL;
   struct
   {
      unsigned :19;
      unsigned ptac_B5 :5;
      unsigned errm_B2 :2;
      unsigned :2;
      unsigned ccfc_B4 :4;
   } CCEV_ST;
} CCEV_UN;

unsigned :32;
unsigned :32;

/* Slot Counter Value                                    */
/* 0x110 */
union scv
{
   unsigned long SCV_UL;
   struct
   {
      unsigned :5;
      unsigned sccb_B11 :11;
      unsigned :5;
      unsigned scca_B11 :11;
   } SCV_ST;
} SCV_UN;

/* Macrotick and Cycle Counter Value                          */
/* 0x114 */
union mtccv
{
   unsigned long MTCCV_UL;
   struct
   {
      unsigned :10;
      unsigned ccv_B6 :6;
      unsigned :2;
      unsigned mtv_B14 :14;
   } MTCCV_ST;
} MTCCV_UN;

/* Rate Correction Value                                 */
/* 0x118 */
union rcv
{
   unsigned long RCV_UL;
   struct
   {
      unsigned :20;
      unsigned rcv_B12 :12;
   } RCV_ST;
```

```
} RCV_UN;

/* Offset Correction Value                                    */
/* 0x11C */
union ocv
{
   unsigned long OCV_UL;
   struct
   {
      unsigned :12;
      unsigned ocv_B14 :20;
   } OCV_ST;
} OCV_UN;

/* Sync Frame Status                                          */
/* 0x120 */
union sfs
{
   unsigned long SFS_UL;
   struct
   {
      unsigned :12;
      unsigned rclr_B1 :1;
      unsigned mrcs_B1 :1;
      unsigned olcr_B1 :1;
      unsigned mocs_B1 :1;
      unsigned vsbo_B4 :4;
      unsigned vsbe_B4 :4;
      unsigned vsao_B4 :4;
      unsigned vsae_B4 :4;
   } SFS_ST;
} SFS_UN;

/* Symbol Windows and NIT Status                              */
/* 0x124 */
union swnit
{
   unsigned long SWNIT_UL;
   struct
   {
      unsigned :20;
      unsigned sbnb_B1 :1;
      unsigned senb_B1 :1;
      unsigned sbna_B1 :1;
      unsigned sena_B1 :1;
      unsigned mtsb_B1 :1;
      unsigned mtsa_B1 :1;
      unsigned tcsb_B1 :1;
      unsigned sbsb_B1 :1;
      unsigned sesb_B1 :1;
      unsigned tcsa_B1 :1;
      unsigned sbsa_B1 :1;
      unsigned sesa_B1 :1;
```

```
   } SWNIT_ST;
} SWNIT_UN;

/* Aggregated Channel Status                                    */
/* 0x128 */
union acs
{
   unsigned long ACS_UL;
   struct
   {
      unsigned :19;
      unsigned sbvb_B1 :1;
      unsigned cib_B1 :1;
      unsigned cedb_B1 :1;
      unsigned sedb_B1 :1;
      unsigned vfrb_B1 :1;
      unsigned :3;
      unsigned sbva_B1 :1;
      unsigned cia_B1 :1;
      unsigned ceda_B1 :1;
      unsigned seda_B1 :1;
      unsigned vfra_B1 :1;
   } ACS_ST;
} ACS_UN;

unsigned :32;

/* Even Sync ID [1..15]                                        */
/* 0x130 .. 0x168 */
unsigned long ESID_UL[15];

unsigned :32;

/* Odd Sync ID [1..15]                                         */
/* 0x170 .. 0x1A8 */
unsigned long OSID_UL[15];

unsigned :32;

/* Network Management Vector 1                                  */
/* 0x230 */
union nmv1
{
   unsigned long NMV1_UL;
} NMV1_UN;

/* Network Management Vector 2                                  */
/* 0x234 */
union nmv2
{
   unsigned long NMV2_UL;
} NMV2_UN;
```

```
/* Network Management Vector 3                                    */
/* 0x238 */
unsigned long NMV3_UL;

unsigned long RES3[81]; /* Reserved                               */

/* Message Buffer Control Registers                               */
/* Message RAM Configuration                                      */
/* 0x300 */
union mrc
{
   unsigned long MRC_UL;
   struct
   {
      unsigned :5;
      unsigned splm_B1 :1;
      unsigned sec_B2 :2;
      unsigned lcb_B8 :8;
      unsigned ffb_B8 :8;
      unsigned fdb_B8 :8;
   } MRC_ST;
} MRC_UN;

/* FIFO Rejection Filter                                        */
/* 0x304 */
union frf
{
   unsigned long FRF_UL;
   struct
   {
      unsigned :7;
      unsigned rnf :1;
      unsigned rss :1;
      unsigned cyf_B7 :7;
      unsigned :3;
      unsigned fid_B11 :11;
      unsigned ch_B2 :2;
   } FRF_ST;
} FRF_UN;

/* FIFO Rejection Filter Mask                                     */
/* 0x308 */
union frfm
{
   unsigned long FRFM_UL;
   struct
   {
      unsigned :19;
      unsigned mfid_B11 :11;
      unsigned :2;
   } FRFM_ST;
} FRFM_UN;
```

```
   unsigned long FCL;

   /* Message Buffer Status Registers                                    */

   /* Message Handler Status                                            */
   /* 0x310 */
   union mhds
   {
      unsigned long MHDS_UL;
      struct
      {
         unsigned :1;
         unsigned mbu_B7 :7;
         unsigned :1;
         unsigned mbt_B7 :7;
         unsigned :1;
         unsigned fmb_B7 :7;
         unsigned cram_B1 :1;
         unsigned mfmb_B1 :1;
         unsigned fmbd_B1 :1;
         unsigned ptbf2_B1 :1;
         unsigned ptbf1_B1 :1;
         unsigned pmr_B1 :1;
         unsigned pobf_B1 :1;
         unsigned pibf_B1 :1;
      } MHDS_ST;
   } MHDS_UN;

   unsigned :32;

   /* FIFO Status Register                                          */
   /* 0x318 */
   union fsr
   {
      unsigned long FSR_UL;
      struct
      {
         unsigned :16;
         unsigned rffl_B8 :8;
         unsigned :5;
         unsigned rfo_B1 :1;
         unsigned rfcl_B1 :1;
         unsigned rfne_B1 :1;
      } FSR_ST;
   } FSR_UN;

   /* Message Handler Constraints Flags                                  */
   /* 0x31C */
   union mhdf
   {
      unsigned long MHDF_UL;
      struct
      {
```

```
      unsigned :23;
      unsigned wahp_B1 :1;
      unsigned :2;
      unsigned tbfb_B1 :1;
      unsigned tbfa_B1 :1;
      unsigned fnfb_B1 :1;
      unsigned fnfa_B1 :1;
      unsigned snub_B1 :1;
      unsigned snua_B1 :1;
   } MHDF_ST;
} MHDF_UN;

/* Transmission Request Register 1                    */
/* 0x320 */
union txrq1
{
   unsigned long TXRQ1_UL;
} TXRQ1_UN;

/* Transmission Request Register 2                    */
/* 0x324 */
union txrq2
{
   unsigned long TXRQ2_UL;
} TXRQ2_UN;

/* Transmission Request Register 3                    */
/* 0x328 */
union txrq3
{
   unsigned long TXRQ3_UL;
} TXRQ3_UN;

/* Transmission Request Register 4                    */
/* 0x32C */
union txrq4
{
   unsigned long TXRQ4_UL;
} TXRQ4_UN;

/* New Data Register 1                              */
/* 0x330 */
union ndat1
{
   unsigned long NDAT1_UL;
} NDAT1_UN;

/* New Data Register 2                              */
/* 0x334 */
union ndat2
{
   unsigned long NDAT2_UL;
} NDAT2_UN;
```

```
/* New Data Register 3                                    */
/* 0x338 */
union ndat3
{
   unsigned long NDAT3_UL;
} NDAT3_UN;

/* New Data Register 4                                    */
/* 0x33c */
union ndat4
{
   unsigned long NDAT4_UL;
} NDAT4_UN;

/* Message Buffer Status Changed 1                            */
/* 0x340 */
union mbsc1
{
   unsigned long MBSC1_UL;
} MBSC1_UN;

/* Message Buffer Status Changed 2                            */
/* 0x344 */
union mbsc2
{
   unsigned long MBSC2_UL;
} MBSC2_UN;

/* Message Buffer Status Changed 3                            */
/* 0x348 */
union mbsc3
{
   unsigned long MBSC3_UL;
} MBSC3_UN;

/* Message Buffer Status Changed 4                            */
/* 0x34C */
union mbsc4
{
   unsigned long MBSC4_UL;
} MBSC4_UN;

unsigned long RES4[44]; /* Reserved                       */

/* Input Buffer                                         */
/* Write Data Section [1..64]                             */
/* 0x400 .. 0x4FC */
unsigned long WRDS[64];

/* Write Header Section 1                                 */
/* 0x500 */
union wrhs1
```

```
{
   unsigned long WRHS1_UL;
   struct
   {
      unsigned :2;
      unsigned mbi_B1 :1;
      unsigned txm_B1 :1;
      unsigned ppit_B1 :1;
      unsigned cfg_B1 :1;
      unsigned chb_B1 :1;
      unsigned cha_B1 :1;
      unsigned :1;
      unsigned cyc_B7 :7;
      unsigned :5;
      unsigned fid_B11 :11;
   } WRHS1_ST;
} WRHS1_UN;

/* Write Header Section 2                                    */
/* 0x504 */
union wrhs2
{
   unsigned long WRHS2_UL;
   struct
   {
      unsigned :9;
      unsigned pl_B7 :7;
      unsigned :5;
      unsigned crc_B11 :11;
   } WRHS2_ST;
} WRHS2_UN;

/* Write Header Section 3                                    */
/* 0x508 */
union wrhs3
{
   unsigned long WRHS3_UL;
   struct
   {
      unsigned :21;
      unsigned dp_B11 :11;
   } WRHS3_ST;
} WRHS3_UN;

unsigned :32;

/* Input Buffer Command Mask                                    */
/* 0x510 */
union ibcm
{
   unsigned long IBCM_UL;
   struct
   {
```

```
      unsigned :13;
      unsigned stxrs_B1 :1;
      unsigned ldss_B1 :1;
      unsigned lhss_B1 :1;
      unsigned :13;
      unsigned stxrh_B1 :1;
      unsigned ldsh_B1 :1;
      unsigned lhsh_B1 :1;
   } IBCM_ST;
} IBCM_UN;

/* Input Buffer Command Request                                    */
/* 0x514 */
union ibcr
{
   unsigned long IBCR_UL;
   struct
   {
      unsigned ibsys_B1 :1;
      unsigned :8;
      unsigned ibrs_B7 :7;
      unsigned ibsyh_B1 :1;
      unsigned :8;
      unsigned ibrh_B7 :7;
   } IBCR_ST;
} IBCR_UN;

unsigned long RES5[58]; /* Reserved                          */

/* Output Buffer                                             */
/* Read Data Section [1..64]                                  */
/* 0x600 .. 0x6FC */
unsigned long RDDS[64];

/* Read Header Section 1                                      */
/* 0x700 */
union rdhs1
{
   unsigned long RDHS1_UL;
   struct
   {
      unsigned :2;
      unsigned mbi_B1 :1;
      unsigned txm_B1 :1;
      unsigned ppit_B1 :1;
      unsigned cfg_B1 :1;
      unsigned chb_B1 :1;
      unsigned cha_B1 :1;
      unsigned :1;
      unsigned cyc_B7 :7;
      unsigned :5;
      unsigned fid_B11 :11;
   } RDHS1_ST;
```

```
} RDHS1_UN;

/* Read Header Section 2                                    */
/* 0x704 */
union rdhs2
{
   unsigned long RDHS2_UL;
   struct
   {
      unsigned :1;
      unsigned plr_B7 :7;
      unsigned :1;
      unsigned plc_B7 :7;
      unsigned :5;
      unsigned crc_B11 :11;
   } RDHS2_ST;
} RDHS2_UN;

/* Read Header Section 3                                    */
/* 0x708 */
union rdhs3
{
   unsigned long RDHS3_UL;
   struct
   {
      unsigned :2;
      unsigned res_B1 :1;
      unsigned ppi_B1 :1;
      unsigned nfi_B1 :1;
      unsigned syn_B1 :1;
      unsigned sfi_B1 :1;
      unsigned rci_B1 :1;
      unsigned :2;
      unsigned rcc_B6 :6;
      unsigned :5;
      unsigned dp_B11 :11;
   } RDHS3_ST;
} RDHS3_UN;

/* Message Buffer Status                                    */
/* 0x70C */
union mbs
{
   unsigned long MBS_UL;
   struct
   {
      unsigned :2;
      unsigned ress_B1 :1;
      unsigned ppis_B1 :1;
      unsigned nfis_B1 :1;
      unsigned syns_B1 :1;
      unsigned sfis_B1 :1;
      unsigned rcis_B1 :1;
```

```
      unsigned :2;
      unsigned ccs_B6 :6;
      unsigned ftb_B1 :1;
      unsigned fta_B1 :1;
      unsigned :1;
      unsigned mlst_B1 :1;
      unsigned esb_B1 :1;
      unsigned esa_B1 :1;
      unsigned tcib_B1 :1;
      unsigned tcia_B1 :1;
      unsigned svob_B1 :1;
      unsigned svoa_B1 :1;
      unsigned ceob_B1 :1;
      unsigned ceoa_B1 :1;
      unsigned seob_B1 :1;
      unsigned seoa_B1 :1;
      unsigned vfrb_B1 :1;
      unsigned vfra_B1 :1;
   } MBS_ST;
} MBS_UN;

/* Output Buffer Command Mask                                    */
/* 0x710 */
union obcm
{
   unsigned long OBCM_UL;
   struct
   {
      unsigned :14;
      unsigned rdsh_B1 :1;
      unsigned rhsh_B1 :1;
      unsigned :14;
      unsigned rdss_B1 :1;
      unsigned rhss_B1 :1;
   } OBCM_ST;
} OBCM_UN;

/* Output Buffer Command Request                                 */
/* 0x714 */
union obcr
{
   unsigned long OBCR_UL;
   struct
   {
      unsigned :9;
      unsigned obrh_B7 :7;
      unsigned obsys_B1 :1;
      unsigned :5;
      unsigned req_B1 :1;
      unsigned view_B1 :1;
      unsigned :1;
      unsigned obrs_B7 :7;
   } OBCR_ST;
```

```
   } OBCR_UN;
} FRAY_ST;

//*************************************************************
// Structure for initializing CC - Fr_Init - Fr_ConfigPtr
typedef volatile struct cfg
{
   int mrc;
   int prtc1;
   int prtc2;
   int mhdc;
   int gtu1;
   int gtu2;
   int gtu3;
   int gtu4;
   int gtu5;
   int gtu6;
   int gtu7;
   int gtu8;
   int gtu9;
   int gtu10;
   int gtu11;
   int succ2;
   int succ3;
} cfg;

// Structure for configuring buffer
typedef volatile struct wrhs
{
   //Header 1
   unsigned mbi;
   unsigned txm;
   unsigned ppit;
   unsigned cfg;
   unsigned chb;
   unsigned cha;
   unsigned cyc;
   unsigned fid;

   //Header 2
   unsigned plr;   //RDHS2
   unsigned plc;
   unsigned crc;

   //Header 3
   unsigned res;   //RDHS3
   unsigned ppi;   //RDHS3
   unsigned nfi;   //RDHS3
   unsigned syn;   //RDHS3
   unsigned sfi;   //RDHS3
   unsigned rci;   //RDHS3
   unsigned rcc;   //RDHS3
   unsigned dp;       //R/W
```

```
   //Message Buffer Status MBS (all R)
   unsigned long mbs_UL;

   //unsigned long Data[FLEX_PAYLOAD];

} wrhs;

// Structure for initializing buffer
typedef volatile struct bc
{
   //IBCM
   unsigned stxrs; //R
   unsigned ldss;  //R
   unsigned lhss;  //R
   unsigned stxrh;
   unsigned ldsh;
   unsigned lhsh;

   //IBCR
   unsigned ibsys; //R
   unsigned ibrs;  //R
   unsigned ibsyh; //R
   unsigned ibrh;

   // OBCM
   unsigned rdsh;  //R
   unsigned rhsh;  //R
   unsigned rdss;
   unsigned rhss;

   //OBCR
   unsigned obrh;
   unsigned obsys; //R
   unsigned req;
   unsigned view;
   unsigned rsvd;  //R
   unsigned obrs;
} bc; //Buffer command

//================================================================
================
typedef volatile struct FRAY_BUFF
{
   struct Buff_Headers
   {
      union h1
      {
         unsigned long HEADER1_UL;
         struct
         {
            unsigned :2;
            unsigned Mess_buff_int :1;
```

```
            unsigned Transmi_mode :1;
            unsigned Payload_Pre_Indi :1; //ppit_B1 :1;  Payload preamble
indicator transmit
            unsigned Message_buf_config_bit :1;
            unsigned CHB_filt_contr :1;
            unsigned CHA_filt_contr :1;
            unsigned :1;
            unsigned Cycle_Code :7;
            unsigned :5;
            unsigned FrameID :11;
         } HEADER1_ST;
      } HEADER1_UN;
      union h2
      {
         unsigned long HEADER2_UL;
         struct
         {
            unsigned :9;
            unsigned Payload_length_config :7;
            unsigned :5;
            unsigned Header_CRC :11;
         } HEADER2_ST;
      } HEADER2_UN;
      union h3
      {
         unsigned long HEADER3_UL;
         struct
         {
            unsigned :21;
            unsigned Data_pointer :11;
         } HEADER3_ST;
      } HEADER3_UN;
      union h4
      {
         unsigned long BUFFER_STATUS_UL;
         struct
         {
            unsigned :2;
            unsigned Reserved_bit_status :1;
            unsigned Payload_preamble_indic :1;
            unsigned Null_frame_indic :1;
            unsigned Sync_frame_indicator :1;
            unsigned Startup_frame_indic :1;
            unsigned Received_ch_indic :1;
            unsigned :2;
            unsigned Cycle_count_status :6;
            unsigned Frame_transmitted_chB :1;
            unsigned Frame_transmitted_chA :1;
            unsigned :1; ///////////
            unsigned Message_lost :1;
            unsigned Empty_slot_chB :1;
            unsigned Empty_slot_chA :1;
            unsigned Transm_conflict_indication_chB :1;
```

```
            unsigned Transm_conflict_indication_chA :1;
            unsigned violation_limit_slot_chB :1;
            unsigned violation_limit_slot_chA :1;
            unsigned Content_error_chB :1;
            unsigned Content_error_chA :1;
            unsigned Syntax_error_chB :1;
            unsigned Syntax_error_chA :1;
            unsigned Valid_frame_received_chB :1;
            unsigned Valid_frame_received_chA :1;
         } BUFFER_STATUS_ST;
      } BUFFER_STATUS_UN;
   } Buff_Headers;
   unsigned long Buff_Data[FLEX_PAYLOAD];
   unsigned ranura :7;
} fray_buff;

typedef volatile struct trama
{
   unsigned ranura :1;
   union segmento_startup
   {
      unsigned seg_startup_UL :5;
      struct
      {
         unsigned reserva :1;
         unsigned pre_payload :1;
         unsigned nula :1;
         unsigned sincrona :1;
         unsigned startup :1;
      } seg_startup_ST;
   } seg_startup_UN;

   unsigned ID_trama :11;
   unsigned longt_payload :7;
   unsigned CRC_cabecera :11;
   unsigned cont_cicl :6;
   unsigned long dato[FLEX_PAYLOAD];
   unsigned long trailer_CRC :24;

} trama_t; //Buffer command

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

#define frayREG     (FRAY_ST *)     0xFFF7C800 //FlexRay Communication
Controller

// Functions
void Fray_buffConfig(FRAY_ST *Fray_PST, wrhs *Fr_LPduPtr);
int header_crc_calc(wrhs *Fr_LPduPtr);
void Fr_Init(FRAY_ST *Fray_PST, cfg *Fr_ConfigPtr);
int Fr_ControllerInit(FRAY_ST *Fray_PST);
int Fr_AllowColdStart(FRAY_ST *Fray_PST);
int Fr_StartCommunication(FRAY_ST *Fray_PST);
```

```
int Fr_ReStartCommunication(FRAY_ST *Fray_PST);

void Fray_Transfer_Input(FRAY_ST *Fray_PST, bc *Fr_LSduPtr);
void Fray_Transfer_Output(FRAY_ST *Fray_PST, bc *Fr_LSduPtr);

void Config_CC_Nodo_A(FRAY_ST *Fray_PST);
void Config_Custom_mSlot(FRAY_ST *Fray_PST,trama_t* frame);
void configure_initialize_node_b(FRAY_ST *Fray_PST);

void Escritura_trama(trama_t* frame, unsigned seg_startup, unsigned fid,
unsigned plc, unsigned cont_cycl);
void Lectura_trama(trama_t* frame, fray_buff* bufer);

#endif
```

Anexo C. Código biblioteca FlexRay(fray.c)

```
#include <Codigos/fray.h>

void Fray_buffConfig(FRAY_ST *Fray_PST, wrhs *Fr_LPduPtr)
/*****************************************************************
****
 The function Fr_PrepareLPdu shall perform the following tasks on FlexRay
 CC Fr_CtrIdx:
 1. Figure out the physical resource (e.g., a buffer) mapped to the processing of
 the FlexRay frame identified by Fr_LPduIdx.
 2. Configure the physical resource (a buffer) appropriate for Fr_LPduPtr
 operation (SlotId, Cycle filter, payload length, header CRC, etc.) if the MCG
 uses the reconfiguration feature.

*****************************************************************
****/
{
   int wrhs1;
   int wrhs2;
   wrhs1 = (Fr_LPduPtr->mbi) & 0x1) << 29;
   wrhs1 |= (Fr_LPduPtr->txm & 0x1) << 28;
   wrhs1 |= (Fr_LPduPtr->ppit & 0x1) << 27;
   wrhs1 |= (Fr_LPduPtr->cfg & 0x1) << 26;
   wrhs1 |= (Fr_LPduPtr->chb & 0x1) << 25;
   wrhs1 |= (Fr_LPduPtr->cha & 0x1) << 24;
   wrhs1 |= (Fr_LPduPtr->cyc & 0x7F) << 16;
   wrhs1 |= (Fr_LPduPtr->fid & 0x7FF);
   Fray_PST->WRHS1_UN.WRHS1_UL = wrhs1;

   wrhs2 = (Fr_LPduPtr->plc & 0x7F) << 16) | (Fr_LPduPtr->crc & 0x7FF);
   Fray_PST->WRHS2_UN.WRHS2_UL = wrhs2;

   Fray_PST->WRHS3_UN.WRHS3_UL = (Fr_LPduPtr->dp & 0x7FF);
}

void Fr_Init(FRAY_ST *Fray_PST, cfg *Fr_ConfigPtr)
/*****************************************************************
****
 The function Fr_Init shall internally store the configuration address to enable
 subsequent API calls to access the configuration.

*****************************************************************
****/
{
   Fray_PST->MRC_UN.MRC_UL = Fr_ConfigPtr->mrc;
   Fray_PST->PRTC1_UN.PRTC1_UL = Fr_ConfigPtr->prtc1;
   Fray_PST->PRTC2_UN.PRTC2_UL = Fr_ConfigPtr->prtc2;
   Fray_PST->MHDC_UN.MHDC_UL = Fr_ConfigPtr->mhdc;
   Fray_PST->GTUC1_UN.GTUC1_UL = Fr_ConfigPtr->gtu1;
```

```
   Fray_PST->GTUC2_UN.GTUC2_UL = Fr_ConfigPtr->gtu2;
   Fray_PST->GTUC3_UN.GTUC3_UL = Fr_ConfigPtr->gtu3;
   Fray_PST->GTUC4_UN.GTUC4_UL = Fr_ConfigPtr->gtu4;
   Fray_PST->GTUC5_UN.GTUC5_UL = Fr_ConfigPtr->gtu5;
   Fray_PST->GTUC6_UN.GTUC6_UL = Fr_ConfigPtr->gtu6;
   Fray_PST->GTUC7_UN.GTUC7_UL = Fr_ConfigPtr->gtu7;
   Fray_PST->GTUC8_UN.GTUC8_UL = Fr_ConfigPtr->gtu8;
   Fray_PST->GTUC9_UN.GTUC9_UL = Fr_ConfigPtr->gtu9;
   Fray_PST->GTUC10_UN.GTUC10_UL = Fr_ConfigPtr->gtu10;
   Fray_PST->GTUC11_UN.GTUC11_UL = Fr_ConfigPtr->gtu11;
   Fray_PST->SUCC2_UN.SUCC2_UL = Fr_ConfigPtr->succ2;
   Fray_PST->SUCC3_UN.SUCC3_UL = Fr_ConfigPtr->succ3;
}

void Fray_Transfer_Input(FRAY_ST *Fray_PST, bc *Fr_LSduPtr)
/*****************************************************************
****
 Transferencia de datos desde el búfer de entrada a la RAM de mensajes

*****************************************************************
****/
{
   //Espere hasta que se complete la transferencia en curso
   while (Fray_PST->IBCR_UN.IBCR_ST.ibsyh_B1 != 0)
      ;    // wait for completion on host registers
   while (Fray_PST->IBCR_UN.IBCR_ST.ibsys_B1 != 0)
      ;    // wait for completion on shadow registers

   //Escribir los datos en buffer (WRDSn)
   //Echo en la funcion CargaBuffer

   //Escriba la sección de encabezado en WRHS1,2,3
   //Echo en la funcion Fray_buffConfig

   //Máscara de comando de escritura
   Fray_PST->IBCM_UN.IBCM_ST.stxrh_B1 = Fr_LSduPtr->stxrh;
   Fray_PST->IBCM_UN.IBCM_ST.lhsh_B1 = Fr_LSduPtr->lhsh;
   Fray_PST->IBCM_UN.IBCM_ST.ldsh_B1 = Fr_LSduPtr->ldsh;

   //Solicitar transferencia de datos al búfer de mensajes de destino (ibrh = numero
de buffer)
   Fray_PST->IBCR_UN.IBCR_ST.ibrh_B7 = Fr_LSduPtr->ibrh; //Fr_LSduPtr->ibrh &
0x3F;

   while (Fr_LSduPtr->ibsyh != 0 && Fray_PST->IBCR_UN.IBCR_ST.ibsyh_B1 != 0)
      ;    // wait for completion on host registers
   while (Fr_LSduPtr->ibsys != 0 && Fray_PST->IBCR_UN.IBCR_ST.ibsys_B1 != 0)
      ;    // wait for completion on shadow registers
}

void Fray_Transfer_Output(FRAY_ST *Fray_PST, bc *Fr_LSduPtr)
/*****************************************************************
****
```

```
 Transferencia de datos desde la RAM de mensajes al búfer de salida

*****************************************************************
****/
{
   // Garantizar que no haya transferencias en curso
   while (Fray_PST->OBCR_UN.OBCR_ST.obsys_B1 != 0)
      ;

   //Escribir máscara de comando de búfer de salida
   Fray_PST->OBCM_UN.OBCM_ST.rhss_B1 = Fr_LSduPtr->rhss & 0x1;
   Fray_PST->OBCM_UN.OBCM_ST.rdss_B1 = Fr_LSduPtr->rdss & 0x1;

   //Solicitud de la transferencia del búfer de mensajes
   Fray_PST->OBCR_UN.OBCR_ST.obrs_B7 = Fr_LSduPtr->obrs; //Fr_LSduPtr-
>obrs & 0x3F; //req=1, view=0
   Fray_PST->OBCR_UN.OBCR_ST.req_B1 = 1;

   // wait for completion on shadow registers
   while (Fray_PST->OBCR_UN.OBCR_ST.obsys_B1 != 0)
      ;

   Fray_PST->OBCR_UN.OBCR_ST.view_B1 = 1;

   //Máscara de comando de búfe salida para segundo mensaje
   Fray_PST->OBCM_UN.OBCM_ST.rhss_B1 = Fr_LSduPtr->rhss & 0x1;
   Fray_PST->OBCM_UN.OBCM_ST.rdss_B1 = Fr_LSduPtr->rdss & 0x1;

   //Alterna OBF Shadow y OBF Host e inicia la transferencia del segundo búfer de
mensaje
   Fray_PST->OBCR_UN.OBCR_ST.obrs_B7 = Fr_LSduPtr->obrs; //req=0, view=1
   Fray_PST->OBCR_UN.OBCR_ST.req_B1 = 1;

   // wait for completion on shadow registers
   while (Fray_PST->OBCR_UN.OBCR_ST.obsys_B1 != 0)
      ;

   Fray_PST->OBCR_UN.OBCR_ST.view_B1 = 1;
}

int Fr_ControllerInit(FRAY_ST *Fray_PST)
/*****************************************************************
****
 Fr_ControllerInit

*****************************************************************
****/
{
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_UL = 0x0F1FFB00;
   if (Fray_PST->CCSV_UN.CCSV_ST.pocs_B6 != 0xF) //CONFIG state
      Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_CONFIG;
   // Check if POC has accepted last command
```

```
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   // unlock CONFIG and enter READY state
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0xCE;
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0x31;
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_READY;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   return 1;
}

int Fr_AllowColdStart(FRAY_ST *Fray_PST)
/*****************************************************************
****
 Fr_AllowColdStart

*****************************************************************
****/
{
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_ALLOW_COLDSTART;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   return 1;
}

int Fr_StartCommunication(FRAY_ST *Fray_PST)
/*****************************************************************
****
 Fr_StartCommunication

*****************************************************************
****/
{
   // write SUCC1 configuration
   (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_RUN);
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   return 1;
}
```

```
int Fr_ReStartCommunication(FRAY_ST *Fray_PST)
{

   // write SUCC1 configuration
   (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_FREEZE);
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   Fray_PST->SUCC1_UN.SUCC1_UL = 0x0F1FFB00;
   if (Fray_PST->CCSV_UN.CCSV_ST.pocs_B6 != 0xF) //CONFIG state
      Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_CONFIG;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   // unlock CONFIG and enter READY state
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0xCE;
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0x31;
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_READY;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;

   // write SUCC1 configuration
   (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_RUN);
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   return 1;
}

int header_crc_calc(wrhs *Fr_LPduPtr)
/*****************************************************************
****
 This function calculates the header CRC.

*****************************************************************
****/
{
   unsigned int header;

   int CrcInit = 0x1A;
   int length = 20;
   int CrcNext;
   unsigned long CrcPoly = 0x385;
   unsigned long CrcReg_X = CrcInit;
```

```
   unsigned long header_temp, reg_temp;

   header = (Fr_LPduPtr->syn & 0x1) << 19) | (Fr_LPduPtr->sfi & 0x1) << 18);
   header |= (Fr_LPduPtr->fid & 0x7FF) << 7) | (Fr_LPduPtr->plc & 0x7F);

   header <<= 11;
   CrcReg_X <<= 21;
   CrcPoly <<= 21;

   while (length--)
   {
      header <<= 1;
      header_temp = header & 0x80000000;
      reg_temp = CrcReg_X & 0x80000000;

      if (header_temp ^ reg_temp)
      {  // Step 1
         CrcNext = 1;
      }
      else
      {
         CrcNext = 0;
      }

      CrcReg_X <<= 1;            // Step 2

      if (CrcNext)
      {
         CrcReg_X ^= CrcPoly;      // Step 3
      }
   }

   CrcReg_X >>= 21;

   return CrcReg_X;
}

void Escritura_trama(trama_t* frame, unsigned seg_startup, unsigned fid,
                     unsigned plc, unsigned cont_cycl)
/*****************************************************************
****
 frame
 segment startup => (ppi)(nfi)(syn)(sfi)
 frame ID
 payload length configured (0xFF <=> 1 byte)
 Receive cycle count
 Datos

*****************************************************************
****/
{
   frame->seg_startup_UN.seg_startup_UL = seg_startup;
   frame->ID_trama = fid;                //frame ID
```

```
   frame->longt_payload = plc;    //payload length configured (0xFF <=> 1 byte)

   wrhs header; //Calculo del CRC
   header.syn = frame->seg_startup_UN.seg_startup_ST.sincrona;
   header.sfi = frame->seg_startup_UN.seg_startup_ST.startup;
   header.fid = frame->ID_trama;
   header.plc = frame->longt_payload;
   frame->CRC_cabecera = header_crc_calc(&header); // calculated  by the host

   frame->cont_cicl = cont_cycl;                //Receive cycle count
   frame->trailer_CRC = 0;
}

void Lectura_trama(trama_t* frame, fray_buff* bufer)
/*****************************************************************
****
 Leé trama Flexray

*****************************************************************
****/
{
   frame->ranura = bufer->ranura;
   frame->seg_startup_UN.seg_startup_ST.reserva =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Reserved_bit_status;
   frame->seg_startup_UN.seg_startup_ST.pre_payload =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Payload_preamble_indic;
   frame->seg_startup_UN.seg_startup_ST.nula =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Null_frame_indic;
   frame->seg_startup_UN.seg_startup_ST.sincrona =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Sync_frame_indicator;
   frame->seg_startup_UN.seg_startup_ST.startup =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Startup_frame_indic;

   frame->ID_trama = bufer->Buff_Headers.HEADER1_UN.HEADER1_ST.FrameID;
   frame->longt_payload =
         bufer->Buff_Headers.HEADER2_UN.HEADER2_ST.Payload_length_config;
   frame->CRC_cabecera = bufer-
>Buff_Headers.HEADER2_UN.HEADER2_ST.Header_CRC;
   frame->cont_cicl = bufer-
>Buff_Headers.HEADER1_UN.HEADER1_ST.Cycle_Code;
   int j;
   for (j = 0; j < frame->longt_payload; j++)
   {
      frame->dato[j] = bufer->Buff_Data[j];
   }
}
```

Anexo D. Código biblioteca MRSIII (MRSIII.h)

```
/*
 * MRSIII.h
 *
 *  Created on: 24/06/2021
 *      Author: Luis Rey
 */

#ifndef CODIGOS_MRSIII_H_
#define CODIGOS_MRSIII_H_

#include <Codigos/Panel_Airbag.h>
#include <Codigos/fray.h>

typedef volatile struct MRS_III
{
   unsigned Activacion    : 1;
   unsigned Estado;
   unsigned Umbral_pretensor   : 5;
   unsigned Umbral_activacion_frontal   : 5;
   unsigned Umbral_choque_trasero   : 5;
   unsigned Umbral_BST       : 5;
   unsigned Umbral_ITS       : 5;
   unsigned Umbral_airbag_1   : 5;
   unsigned Umbral_airbag_2   : 5;
   unsigned Umbral_airbag_3   : 5;
   unsigned Umbral_airbag_4   : 5;
} MRS_t;

void config_Umrales(trama_t* frame);
void set_Umbrales(MRS_t* umbrales);
void get_Umbrales(MRS_t* umbrales);
int autocomprobacion_panel_Indica();
int autocomprobacion_sensores();
void reset_Indicadores();
void set_Indicadores();
void ejecucion_MRSIII();
void ActivarIndica(panelact_t* panel, uint8_t retraso);
unsigned ActAirbag_Cond(uint8_t prioridad);
unsigned ActiAirbag_Pasj(uint8_t prioridad);
void whait(unsigned long value);
int verifica_Vel();
int desacelerador();
void initUmbrales();

#endif /* CODIGOS_MRSIII_H_ */
```

Anexo E. Código biblioteca MRSIII (MRSIII.c)

```
/*
 * MRSIII.c
 *
 *  Created on: 16/11/2020
 *      Author: Luis Rey
 */

#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>

unsigned Umbral_pretensor = 0x02;
unsigned Umbral_activacion_frontal = 0x05;
unsigned Umbral_choque_trasero = 0x07;
unsigned Umbral_BST_FRONT;
unsigned Umbral_BST_LAT;
unsigned Umbral_BST_TRA;
unsigned Umbral_ITS;

unsigned Umbral_airbag_1;
unsigned Umbral_airbag_2;
unsigned Umbral_airbag_3;
unsigned Umbral_airbag_4;

unsigned Vel_ant;

int dxlay = 400;
hetBASE_t* het1_debug = hetREG1;

extern panelsen_t panel_sensores;
extern panelact_t panel_indicadores;
extern unsigned modo_config;
extern FRAY_ST* FlexRay_CC;
extern MRS_t MRSIII;
extern volatile boolean dinaseg_1;
extern volatile boolean dinaseg_2;
extern volatile boolean dinaseg_3;
extern volatile boolean dinaseg_4;
extern volatile boolean dinaseg_5;

void initUmbrales()
{
   Umbral_airbag_1 = Umbral_activacion_frontal;
   Umbral_airbag_2 = Umbral_activacion_frontal * 2;
   Umbral_airbag_3 = Umbral_activacion_frontal * 3;
   Umbral_airbag_4 = Umbral_activacion_frontal * 4;

   Umbral_BST_FRONT = Umbral_airbag_2;
```

```
   Umbral_BST_LAT = Umbral_airbag_1;
   Umbral_BST_TRA = Umbral_airbag_2;
   Umbral_ITS = Umbral_airbag_2;


}

void config_Umrales(trama_t* frame)
{
   Umbral_pretensor = frame->dato[0];
   Umbral_activacion_frontal = frame->dato[1];
   Umbral_choque_trasero = frame->dato[2];

   initUmbrales();

   modo_config = 0;
}

void set_Umbrales(MRS_t* umbrales)
{
   Umbral_pretensor = umbrales->Umbral_pretensor;
   Umbral_activacion_frontal = umbrales->Umbral_activacion_frontal;
   Umbral_choque_trasero = umbrales->Umbral_choque_trasero;
   initUmbrales();
}

void get_Umbrales(MRS_t* umbrales)
{
   umbrales->Umbral_pretensor = Umbral_pretensor;
   umbrales->Umbral_activacion_frontal = Umbral_activacion_frontal;
   umbrales->Umbral_choque_trasero = Umbral_choque_trasero;
   umbrales->Umbral_BST = Umbral_BST_FRONT;
   umbrales->Umbral_ITS = Umbral_ITS;
   umbrales->Umbral_airbag_1 = Umbral_airbag_1;
   umbrales->Umbral_airbag_2 = Umbral_airbag_2;
   umbrales->Umbral_airbag_3 = Umbral_airbag_3;
   umbrales->Umbral_airbag_4 = Umbral_airbag_4;
}

int autocomprobacion_panel_Indica()
{
   panelact_t* indicadores = &panel_indicadores;
   int i;
   for (i = 0; i < 0x3FFF; i = (i << 1) + 1)
   {
      indicadores->Indicadores_UN.Indicadores_UL = i;
      whait(250);
      ActivarIndica(indicadores, 0);
   }
   whait(400);
   indicadores->Indicadores_UN.Indicadores_UL = 0;
   ActivarIndica(indicadores, 0);
   whait(400);
```

```
   indicadores->Indicadores_UN.Indicadores_UL = 0x3FFF;
   ActivarIndica(indicadores, 0);
   whait(300);
   indicadores->Indicadores_UN.Indicadores_UL = 0;
   ActivarIndica(indicadores, 0);
   whait(300);
   indicadores->Indicadores_UN.Indicadores_UL = 0x3FFF;
   ActivarIndica(indicadores, 0);
   whait(700);
   indicadores->Indicadores_UN.Indicadores_UL = 0;
   ActivarIndica(indicadores, 0);
   return 3;
}

int autocomprobacion_sensores()
{
   panelsen_t* sensores = &panel_sensores;
   panelact_t* indicadores = &panel_indicadores;

   if (sensores->encendido)
   {
      if (sensores->seg_Front_Cond & sensores->seg_Front_Pasj
            & sensores->seg_Late_Cond & sensores->seg_Late_Pasj
            & sensores->seg_Trasero)
      {
         if (sensores->SBE_Cond & sensores->SBE_Pasj)
         {
            if (sensores->scint_Cond & sensores->scint_Pasj)
            {
               indicadores->Indicadores_UN.Indicadores_UL = 0;
               ActivarIndica(indicadores, 0);
               return 1;
            }
         }
      }
      indicadores->Indicadores_UN.Indicadores_UL = 0;
      indicadores->Indicadores_UN.Indicadores_ST.Testigo_Lum = 1;
      ActivarIndica(indicadores, 0);
      return 0;
   }
   indicadores->Indicadores_UN.Indicadores_UL = 0;
   ActivarIndica(indicadores, 0);
   return 2;
}

void reset_Indicadores()
{
   panelact_t* indicadores = &panel_indicadores;
   int x;
   x=150;
   indicadores->Indicadores_UN.Indicadores_UL = 0x1;
   ActivarIndica(indicadores, 0);
   whait(x);
```

```
   indicadores->Indicadores_UN.Indicadores_UL = 0xA;
   ActivarIndica(indicadores, 0);
   whait(x);
   indicadores->Indicadores_UN.Indicadores_UL = 0x14;
   ActivarIndica(indicadores, 0);
   whait(x);
   indicadores->Indicadores_UN.Indicadores_UL = 0x60;
   ActivarIndica(indicadores, 0);
   whait(x);
   indicadores->Indicadores_UN.Indicadores_UL = 0x180;
   ActivarIndica(indicadores, 0);
   whait(x);
   indicadores->Indicadores_UN.Indicadores_UL = 0x600;
   ActivarIndica(indicadores, 0);
   whait(x);
   indicadores->Indicadores_UN.Indicadores_UL = 0xFFF;
   ActivarIndica(indicadores, 0);
   whait(x);
   indicadores->Indicadores_UN.Indicadores_UL = 0;
   ActivarIndica(indicadores, 0);
}

void set_Indicadores()
{
   panelact_t* indicadores = &panel_indicadores;
   indicadores->Indicadores_UN.Indicadores_UL = 0x1FFF;
   ActivarIndica(indicadores, 0);
}

int verifica_Vel()
{
   if (panel_sensores.velocidad > 1)
      return 1;
   return 0;
}

int desacelerador()
{
   if (Vel_ant - panel_sensores.velocidad >= Umbral_pretensor
         && Vel_ant - panel_sensores.velocidad <= 6)
      return 1;
   return 0;
}

void ejecucion_MRSIII()
{
   panelsen_t* sensores = &panel_sensores;
   panelact_t* indicadores = &panel_indicadores;
   unsigned modo = 0x00;

   //Control de pretensores del cinturon
   if (desacelerador() == 1)
   {
```

```
      if (sensores->SBE_Cond == 1)
         if (sensores->scint_Cond == 1)
            indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
      if (sensores->SBE_Pasj == 1)
         if (sensores->scint_Pasj == 1)
            indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
      MRSIII.Activacion = 1;
   }

   //Vel_ant=panel_sensores.velocidad;

   if (verifica_Vel() == 1)
   {

      //control activacion frontal del airbag conductor
      if (sensores->seg_Front_Cond == 1)
      {
         if (sensores->imp_Front_Cond >= Umbral_airbag_4)
         {
            modo = ActAirbag_Cond(4);
            MRSIII.Activacion = 1;
         }
         else if (sensores->imp_Front_Cond >= Umbral_airbag_3)
         {
            modo = ActAirbag_Cond(3);
            MRSIII.Activacion = 1;
         }
         else if (sensores->imp_Front_Cond >= Umbral_airbag_2)
         {
            modo = ActAirbag_Cond(2);
            MRSIII.Activacion = 1;
         }
         else if (sensores->imp_Front_Cond >= Umbral_airbag_1)
         {
            modo = ActAirbag_Cond(1);
            MRSIII.Activacion = 1;
         }
      }

      //control activacion frontal del airbag pasajero
      if (sensores->seg_Front_Pasj == 1)
      {
         if (sensores->imp_Front_Pasj >= Umbral_airbag_4)
         {
            modo = (modo & 0xC) + ActiAirbag_Pasj(4);
            MRSIII.Activacion = 1;
         }
         else if (sensores->imp_Front_Pasj >= Umbral_airbag_3)
         {
            modo = (modo & 0xC) + ActiAirbag_Pasj(3);
            MRSIII.Activacion = 1;
         }
         else if (sensores->imp_Front_Pasj >= Umbral_airbag_2)
```

```
        {
            modo = (modo & 0xC) + ActiAirbag_Pasj(2);
            MRSIII.Activacion = 1;
        }
        else if (sensores->imp_Front_Pasj >= Umbral_airbag_1)
        {
            modo = (modo & 0xC) + ActiAirbag_Pasj(1);
            MRSIII.Activacion = 1;
        }
    }

    //control activacion trasera del airbag
    if (Umbral_choque_trasero < sensores->imp_Trasero
            && sensores->seg_Trasero == 1)
    {
        modo = 0;
        if (sensores->imp_Trasero > Umbral_airbag_4)
        {
            modo = ActAirbag_Cond(4) + ActiAirbag_Pasj(4);
            if (sensores->scint_Cond == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
            if (sensores->scint_Pasj == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
            MRSIII.Activacion = 1;
        }
        else if (sensores->imp_Trasero > Umbral_airbag_3)
        {
            modo = ActAirbag_Cond(3) + ActiAirbag_Pasj(3);
            if (sensores->scint_Cond == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
            if (sensores->scint_Pasj == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
            MRSIII.Activacion = 1;
        }
    }

    //Control del BST
    if (Umbral_BST_FRONT <= sensores->imp_Front_Cond
            && sensores->seg_Front_Cond == 1)
            || (Umbral_BST_FRONT <= sensores->imp_Front_Pasj
                    && sensores->seg_Front_Pasj == 1)
            || (Umbral_BST_TRA <= sensores->imp_Trasero
                    && sensores->seg_Trasero == 1)
            || (Umbral_BST_LAT <= sensores->imp_Late_Cond
                    && sensores->seg_Late_Cond == 1)
            || (Umbral_BST_LAT <= sensores->imp_Late_Pasj
                    && sensores->seg_Late_Pasj == 1)
    {
        indicadores->Indicadores_UN.Indicadores_ST.BST = 1;
        if (sensores->SBE_Cond == 1)
            if (sensores->scint_Cond == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
        if (sensores->SBE_Pasj == 1)
```

```
            if (sensores->scint_Pasj == 1)
               indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
         MRSIII.Activacion = 1;
      }

      //Control de ITS
      if (Umbral_ITS < sensores->imp_Late_Cond
            || Umbral_ITS < sensores->imp_Late_Pasj)
            && (sensores->seg_Late_Cond == 1 || sensores->seg_Late_Pasj == 1)
      {
         if (sensores->SBE_Cond == 1)
         {
            indicadores->Indicadores_UN.Indicadores_ST.Acti_ITS_1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Thorax_cond = 1;
            if (sensores->scint_Cond == 1)
               indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
         }
         if (sensores->SBE_Pasj == 1)
         {
            indicadores->Indicadores_UN.Indicadores_ST.Acti_ITS_2 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Thorax_pasj = 1;
            if (sensores->scint_Pasj == 1)
               indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
         }

         MRSIII.Activacion = 1;
      }
   }
   if (MRSIII.Activacion==1)
   {
      rtiStartCounter(rtiREG1, 1);
      ActivarIndica(indicadores, modo);
      dinaseg_5=true;
   }
}

void ActivarIndica(panelact_t* panel, uint8_t convinacion)
{
   HET1_digitalWrite(PIN_HET_2,
                  panel->Indicadores_UN.Indicadores_ST.Testigo_Lum);
   HET1_digitalWrite(PIN_HET_18,
                  panel->Indicadores_UN.Indicadores_ST.Desp_cond_E1);
   HET1_digitalWrite(PIN_HET_30,
                  panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E1);
   HET1_digitalWrite(PIN_HET_22,
                  panel->Indicadores_UN.Indicadores_ST.Acti_ITS_1);
   HET1_digitalWrite(PIN_HET_12,
                  panel->Indicadores_UN.Indicadores_ST.Acti_ITS_2);
   HET1_digitalWrite(PIN_HET_25,
                  panel->Indicadores_UN.Indicadores_ST.Thorax_cond);
   HET1_digitalWrite(PIN_HET_27,
                  panel->Indicadores_UN.Indicadores_ST.Thorax_pasj);
   HET1_digitalWrite(PIN_HET_29,
```

```
                panel->Indicadores_UN.Indicadores_ST.Preten_cond);
    HET1_digitalWrite(PIN_HET_10,
                panel->Indicadores_UN.Indicadores_ST.Preten_pasj);
    HET1_digitalWrite(PIN_HET_28, panel->Indicadores_UN.Indicadores_ST.BST);
    switch (convinacion)
    {
    case 0:
    {
       HET1_digitalWrite(PIN_HET_16,
                   panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       HET1_digitalWrite(PIN_HET_14,
                   panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       break;
    }
    case 1:
    {
       HET1_digitalWrite(PIN_HET_16,
                   panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_14,
                   panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);

       break;
    }
    case 2:
    {
       HET1_digitalWrite(PIN_HET_16,
                   panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       whait(dxlay);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_14,
                   panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       break;
    }
    case 4:
    {
       HET1_digitalWrite(PIN_HET_14,
                   panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_16,
                   panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);

       break;
    }
    case 8:
    {
       HET1_digitalWrite(PIN_HET_14,
                   panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       whait(dxlay);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_16,
                   panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       break;
```

```
   }
   case 5:
   {
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                  panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      HET1_digitalWrite(PIN_HET_14,
                  panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      break;
   }
   case 9:
   {
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                  panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_14,
                  panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      break;
   }
   case 6:
   {
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                  panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_14,
                  panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      break;
   }
   case 10:
   {
      whait(dxlay);
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                  panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      HET1_digitalWrite(PIN_HET_14,
                  panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      break;
   }
   }
}

unsigned ActAirbag_Cond(uint8_t prioridad)
//prioridad 4->maxima 1->minima,
{
   panelsen_t* sensores = &panel_sensores;
   panelact_t* indicadores = &panel_indicadores;

   if (sensores->SBE_Cond == 1)
   {
      if (sensores->scint_Cond == 1)
      {
```

```
        indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
        switch (prioridad)
        {
        case 1:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 0;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 0;
           return 0;
        }
        case 2:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 0;
           return 0;
        }
        case 3:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
           return 4;
        }
        case 4:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
           return 0;
        }
        }
     }
     else
     {
        switch (prioridad)
        {
        case 1:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 0;
           return 0;
        }
        case 2:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
           return 8;
        }
        case 3:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
           return 4;
        }
        case 4:
        {
```

```
            indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
            return 0;
         }
         }
      }
   }
   return 0;
}

unsigned ActiAirbag_Pasj(uint8_t prioridad)
//prioridad 4->maxima 1->minima,
{
   panelsen_t* sensores = &panel_sensores;
   panelact_t* indicadores = &panel_indicadores;

   if (sensores->SBE_Pasj == 1)
   {
      if (sensores->scint_Pasj == 1)
      {
         indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
         switch (prioridad)
         {
         case 1:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 0;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 0;
            return 0;
         }
         case 2:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 0;
            return 0;
         }
         case 3:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
            return 1;
         }
         case 4:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
            return 0;
         }
         }
      }
      else
      {
         switch (prioridad)
         {
```

```
        case 1:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 0;
           return 0;
        }
        case 2:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
           return 2;
        }
        case 3:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
           return 1;
        }
        case 4:
        {
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
           indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
           return 0;
        }
        }
     }
   }
   return 0;
}

void whait(unsigned long value)
{
   while (value)
   {
      long x = value * 180;
      while (x)
      {
         x--;
      }
      value--;
   }
}

void HET1_digitalWrite(unsigned HET_pin, unsigned value)
{
   uint32 mask = 0xFFFFFFFF - (value << HET_pin);
   het1_debug->DSET = 1 << HET_pin;
   het1_debug->DOUT = (uint32) hetREG1->DOUT & mask;
}
```

Anexo F. Código módulo FlexRay A

```
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>

wrhs header_reg; //Variables globales
cfg Fray_Config;
bc Fray_buffcomand;
int uh;
int custon12 = 0;

extern panelsen_t panel_sensores;
extern panelact_t panel_indicadores;
extern MRS_t MRSIII;
extern unsigned datos_carga_util[64];
extern volatile boolean dinaseg_1;
extern volatile boolean dinaseg_2;
extern volatile boolean dinaseg_3;
extern volatile boolean dinaseg_4;
extern volatile boolean dinaseg_5;
extern unsigned modo_config;
extern unsigned frameMOD;
extern trama_t interfaz_trama_in;

void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,
                unsigned plc, unsigned long* datos);

void Config_CC_Nodo_A(FRAY_ST *Fray_PST)
{
   //Variables locales apuntan a las varibles globales
   cfg *Fr_ConfigPtr = &Fray_Config; // Structure for initializing CC - Fr_Init - Fr_ConfigPtr

   // GTUs (Global Time Unit ), PRTC configuration
   Fr_ConfigPtr->gtu1 = 0x00036B00; // pMicroPerCycle = 224000d = 36B00h (has to be x40 of MacroPerCyle)
                           // [19:0]: These bits configure the duration of the communication cycle in microticks

   Fr_ConfigPtr->gtu2 = 0x000F15E0; // gSyncodeMax = Fh, gMacroPerCycle = 5600d = 15E0h  (cycle period, 5.6us)
                           //[13:0]: Macrotick per cycle (in macroticks). These bits configure the duration of one communication cycle
                           //     in macroticks. The cycle length must be identical in all nodes of a cluster.
                           //[19:16]: Sync node max (in frames). These bits configure the maximum number of frames within a cluster
                           //      with sync frame indicator bit SYN set. The number of frames must be identical in all nodes of a cluster.
```

```
   Fr_ConfigPtr->gtu3 = 0x00061818; // gMacroInitialOffset = 6h,
pMicroInitialOffset = 24d = 18h
   Fr_ConfigPtr->gtu4 = 0x0AE40AE3; // gOffsetCorrectionStart - 1 = 2788d = AE4h
(OCS)  , gMacroPerCycle - gdNIT - 1 = 2787d = AE3h (NIT)
   Fr_ConfigPtr->gtu5 = 0x33010303; // pDecodingCorrection = 51d = 33h,
pClusterDriftDamping = 1h, pDelayCompensation = 3h
   Fr_ConfigPtr->gtu6 = 0x01510081; // pdMaxDrift = 337d = 151h,
pdAcceptedStartupRange = 129d = 81h

   Fr_ConfigPtr->gtu7 = 0x00080056; // gNumberOfStaticSlots = 8h, gdStaticSlot =
86d = 56h
                           // [25:16]: These bits configure the number of static slots
in a cycle.
                           // [9:0]: These bits configure the duration of a static slot
(macroticks).

   Fr_ConfigPtr->gtu8 = 0x015A0004; // gNumberOfMinislots = 346d = 15Ah,
gdMinislot = 4h
                           // [28:16]:These bits configure the number of minislots in
the dynamic segment of a cycle
                           // [5:0]: These bits configure the duration of a minislot

   Fr_ConfigPtr->gtu9 = 0x00010204; // gdDynamicSlotIdlePhase = 1,
gdMinislotActionPointOffset = 2, gdActionPointOffset = 4h
   Fr_ConfigPtr->gtu10 = 0x015100CD; // pRateCorrectionOut = 337d = 151h,
pOffsetCorrectionOut = 205d = CDh
   Fr_ConfigPtr->gtu11 = 0x00000000; // pExternRateCorrection = 0,
pExternOffsetCorrection = 0, no ext. clk. corr.

   Fr_ConfigPtr->succ2 = 0x0F036DA2; // gListenNoise = Fh, pdListenTimeout =
224674d = 36DA2h
                           //LTN [27:24]: Listen timeout noise. Configures the upper
limit for the startup and wakeup listen timeout in the
                           //presence of noise. Must be identical in all nodes of a
cluster.
                           //The wakeup / startup noise timeout is calculated as
follows: LT[20:0] · (LTN[3:0] + 1)
                           // LT[20:0]: Listen timeout. Configures the upper limit of
the startup and wakeup listen timeout.

   Fr_ConfigPtr->succ3 = 0x000000FF; // gMaxWithoutClockCorrectionFatal = Fh ,
passive = Fh
                           //WCF[7:4]: Maximum without clock correction fatal.
These bits define the number of consecutive even/odd
                           //cycle pairs with missing clock correction terms that will
cause a transition from
                           //NORMAL_ACTIVE or NORMAL_PASSIVE state.

   //WCP[3:0]: Maximum without clock correction passive. These bits define the
number of consecutive
   //even/odd cycle pairs with missing clock correction terms that will cause a
transition from
```

```
   //NORMAL_ACTIVE to NORMAL_PASSIVE to HALT state.

   Fr_ConfigPtr->prtc1 = 0x084C000A; // pWakeupPattern = 2h,
gdWakeupSymbolRxWindow = 76d, BRP = 0, gdTSSTransmitter = Ah
                           //BRP[15:14]; Baud rate prescaler. These bits configure
the baud rate on the FlexRay bus. The baud rates
                           //listed below are valid with a sample clock of 80 MHz.
One bit time always consists of 8 samples
                           //independent of the configured baud rate.  =0 ->10Mb/s

   Fr_ConfigPtr->prtc2 = 0x3CB41212; // gdWakeupSymbolTxLow = 60d,
gdWakeupSymbolTxIdle = 180d, gdWakeupSymbolRxLow = 18d,
gdWakeupSymbolRxIdle = 18d

   Fr_ConfigPtr->mhdc = 0x010D0009; // pLatestTransmit = 269d = 010Dh,
gPayloadLengthStatic = 9h
                           //Start of latest transmit (in minislots). These bits
configure the maximum minislot value allowed
                           //minislots before inhibiting new frame transmissions in the
Dynamic Segment of the cycle.
                           //[7:0]: Static frame data length.

   Fr_ConfigPtr->mrc = 0x00174006; // LCB=23d, FFB=64d, FDB=4d (0..3 static,
4..23 dyn., 0 fifo)

   // Wait for PBSY bit to clear - POC not busy.
   // 1: Signals that the POC is busy and cannot accept a command from the host.
CMD(3-0) is locked against write accesses.
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0)
      ;

   // Initialize
   Fr_Init(Fray_PST, Fr_ConfigPtr);

   //Configuración por defecto slots

   wrhs *Fray_header = &header_reg; // Structure for configuring buffers -
Fray_buffConfig - Fray_header
   bc *Fray_buff_comand = &Fray_buffcomand; // Structure for initializing buffer -
Fray_Transfer_Input, Fr_ReceiveRxLPdu - Fray_buff_comand

   //Header Section Global Config
   //Header Section Register 1 (WRHS1)
   Fray_header->mbi = 1;   // message buffer interrupt disabled
   Fray_header->txm = 1;   // transmission mode - continuous mode
   Fray_header->ppit = 0;  // Payload Preamble Indicator is not set
   Fray_header->cfg = 0; // message buffer configuration bit (0= receive buffer, 1 =
transmit buffer)
   Fray_header->chb = 1;   // Ch B
   Fray_header->cha = 1;   // Ch A
   Fray_header->cyc = 0;   // Cycle Filtering Code (no cycle filtering)
   Fray_header->fid = 0;   // Frame ID
```

```
   //Header Section 2 (WRHS2)
   Fray_header->plc = 0;   // Payload Length configured
   Fray_header->crc = 0;

   //Header Section 3 (WRHS3)
   Fray_header->dp = 0;   // Pointer to start of data in message RAM

   //Buffer Command Global Config
   //Output Buffer
   Fray_buff_comand->rdss = 0;  // read data section                OBCM
   Fray_buff_comand->rhss = 0;  // read header section               OBCM
   Fray_buff_comand->obrs = 0;  // output buffer number               OBCR

   //Input Buffer
   Fray_buff_comand->stxrh = 0; // set transmission request          IBCM  0
   Fray_buff_comand->ldsh = 0; // load data section                 IBCM  0
   Fray_buff_comand->lhsh = 1; // load header section               IBCM  1
   Fray_buff_comand->ibrh = 0; // input buffer number               IBCR  0
   Fray_buff_comand->ibsyh = 1; // check for input buffer busy host     IBCR
   Fray_buff_comand->ibsys = 1; // check for input buffer busy shadow    IBCR

   // Message buffers Particular Config

//=================================================
===Static segment
   // Buffer #1 Transmit buffer
   Fray_header->fid = 1;      // frame ID
   Fray_header->cfg = 1;      // Transmit buffer
   Fray_header->plc = 9; // 18 byte payload length configured (0xFF <=> 1 byte)
   Fray_header->dp = 0x80;     // Pointer to start of data in message RAM
   Fray_header->rcc = 0;      //Receive cycle count
   Fray_header->rci = 0;      //Received on channel indicator
   Fray_header->sfi = 1;      //Startup frame indicator
   Fray_header->syn = 1;      //Sync frame indicator
   Fray_header->nfi = 0;      //Null frame indicator
   Fray_header->ppi = 0;      //Payload preamble indicator
   Fray_header->res = 0;      //Reserved bit
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 0;  // input buffer number               IBCR
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #2 Transmit buffer
   Fray_header->fid = 2;   // frame ID
   Fray_header->dp = 0x90; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Transmission on Ch A
   Fray_header->chb = 1;    // Transmission on Ch B
   Fray_header->plc = 9;  // 254 byte payload
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 1;  // input buffer number
```

```
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #3 Transmit buffer
   Fray_header->fid = 3;   // frame ID
   Fray_header->dp = 0x100; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Transmission on Ch A
   Fray_header->chb = 1;    // Transmission on Ch B
   Fray_header->plc = 9;  // 254 byte payload
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 2;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #4 Receive buffer
   Fray_header->fid = 4;    // frame ID
   Fray_header->dp = 0x200; // Pointer to start of data in message RAM
   Fray_header->cfg = 0;    // Receive buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Receive on Ch A
   Fray_header->chb = 1;    // Receive on Ch B
   Fray_header->plc = 9;    // 18 byte payload
   Fray_header->crc = 0;
   Fray_buff_comand->ibrh = 3;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);


//================================================
===Dynamic segment
   // buffer #10 Transmit buffer
   Fray_header->fid = 90;   // frame ID
   Fray_header->dp = 0x110; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Transmission on Ch A
   Fray_header->chb = 0;    // No transmission on Ch B
   Fray_header->plc = 9;  // 254 byte payload
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 10;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #11 Transmit buffer
   Fray_header->fid = 91;   // frame ID
   Fray_header->dp = 0x120; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
```

```
Fray_header->sfi = 0;    // startup frame indicator
Fray_header->cha = 1;    // Transmission on Ch A
Fray_header->chb = 0;    // No transmission on Ch B
Fray_header->plc = 9;  // 254 byte payload
Fray_header->crc = header_crc_calc(Fray_header);
Fray_buff_comand->ibrh = 11;  // input buffer number
Fray_buffConfig(Fray_PST, Fray_header);
Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

// buffer #12 Transmit buffer
Fray_header->fid = 92;   // frame ID
Fray_header->dp = 0x130; // Pointer to start of data in message RAM
Fray_header->cfg = 1;    // Transmit buffer
Fray_header->syn = 0;    // sync frame indicator
Fray_header->sfi = 0;    // startup frame indicator
Fray_header->cha = 1;    // Transmission on Ch A
Fray_header->chb = 0;    // No transmission on Ch B
Fray_header->plc = 9;  // 254 byte payload
Fray_header->crc = header_crc_calc(Fray_header);
Fray_buff_comand->ibrh = 12;  // input buffer number
Fray_buffConfig(Fray_PST, Fray_header);
Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

// buffer #13 Receive buffer
Fray_header->fid = 93;    // frame ID
Fray_header->dp = 0x210; // Pointer to start of data in message RAM
Fray_header->cfg = 0;    // Receive buffer
Fray_header->syn = 0;    // sync frame indicator
Fray_header->sfi = 0;    // startup frame indicator
Fray_header->cha = 1;    // Receive on Ch A
Fray_header->chb = 0;    // No receive on Ch B
Fray_header->plc = 9;    // 18 byte payload
Fray_header->crc = 0;
Fray_buff_comand->ibrh = 13;  // input buffer number
Fray_buffConfig(Fray_PST, Fray_header);
Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

// buffer #14 Receive buffer
Fray_header->fid = 94;    // frame ID
Fray_header->dp = 0x220; // Pointer to start of data in message RAM
Fray_header->cfg = 0;    // Receive buffer
Fray_header->syn = 0;    // sync frame indicator
Fray_header->sfi = 0;    // startup frame indicator
Fray_header->cha = 1;    // Receive on Ch A
Fray_header->chb = 0;    // No receive on Ch B
Fray_header->plc = 9;    // 18 byte payload
Fray_header->crc = 0;
Fray_buff_comand->ibrh = 14;  // input buffer number
Fray_buffConfig(Fray_PST, Fray_header);
Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

uh = Fr_ControllerInit(Fray_PST);
```

```
   // Initialize Interrupts
   Fray_PST->EIR_UN.EIR_UL = 0xFFFFFFFF; // Clear Error Interrupt Register
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF; // Clear Status Interrupt Register
   Fray_PST->SILS_UN.SILS_UL = 0x00000000; // all Status Interrupt Line Select
Register Interrupt is assigned to interrupt line CC_int0.
   Fray_PST->SIER_UN.SIER_UL = 0xFFFFFFFF; // Disable all Status Int.
   Fray_PST->SIES_UN.SIES_UL = 0x00000004; // Enable Cycle start interrupt is
enabled (CYCSE)
   Fray_PST->ILE_UN.ILE_ST.eint1_B1 = 1; // enable eray_int1

   Fr_AllowColdStart(Fray_PST);
}


void comunicacion_Nodo_A(FRAY_ST *Fray_PST)
{
   panelsen_t* psensores = &panel_sensores;
   panelact_t* pindicadores = &panel_indicadores;
   trama_t sistema;
   int i, plr, plc;
   bc buff_comand_output;
   unsigned long ndat1;
   fray_buff bufer;
   unsigned long* datos = (unsigned long*) Fray_PST->WRDS;

   bc buff_comand_input;

   buff_comand_input.stxrh = 1;  // set transmission request          IBCM
   buff_comand_input.ldsh = 1;  // load data section                 IBCM
   buff_comand_input.lhsh = 0;  // load header section               IBCM
   buff_comand_input.ibrh = 0;  // input buffer number               IBCR
   buff_comand_input.ibsys = 0; // check for input buffer busy shadow  IBCR
   buff_comand_input.ibsyh = 1; // check for input buffer busy host    IBCR

   //wait for cycle start interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;        // clear all status int. flags
   while (Fray_PST->SIR_UN.SIR_ST.cycs_B1 == 0x0)
      ;    // wait for CYCS (Cycle Start Interrupt) interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;        // clear all status int. flags

   buff_comand_input.ibrh = slot_Stup
   ;
   datos[0] = psensores->imp_Front_Cond;
   datos[1] = psensores->imp_Front_Pasj;
   datos[2] = psensores->imp_Late_Cond;
   datos[3] = psensores->imp_Late_Pasj;
   datos[4] = psensores->imp_Trasero;
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);

   buff_comand_input.ibrh = slot1
   ;
   datos[0] = psensores->seg_Front_Cond;
   datos[1] = psensores->seg_Front_Pasj;
```

```
datos[2] = psensores->seg_Late_Cond;
datos[3] = psensores->seg_Late_Pasj;
datos[4] = psensores->seg_Trasero;
Fray_Transfer_Input(Fray_PST, &buff_comand_input);

buff_comand_input.ibrh = slot2
;
datos[0] = psensores->velocidad;
datos[1] = psensores->encendido;
Fray_Transfer_Input(Fray_PST, &buff_comand_input);


if (dinaseg_1==true)
{
   buff_comand_input.ibrh = mslot10
   ;
   datos[0] = psensores->SBE_Cond;
   datos[1] = psensores->SBE_Pasj;
   Escritura_Trama_Dinamica(0x0, 90, 2, datos);    //Seg Dinamico 1
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);
   dinaseg_1 = false;
}

if (dinaseg_2==true)
{
   buff_comand_input.ibrh = mslot11
   ;
   datos[0] = psensores->scint_Cond;
   datos[1] = psensores->scint_Pasj;
   Escritura_Trama_Dinamica(0x0, 91, 2, datos);    //Seg Dinamico 2
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);
   dinaseg_2=false;
}

if (dinaseg_3 == true)
   {
      buff_comand_input.ldsh = 1; // load data section                IBCM  0
      buff_comand_input.lhsh = 1; // load header section
      buff_comand_input.ibrh = mslot12;
      get_Umbrales(&MRSIII);
      datos[0] = MRSIII.Estado;
      datos[1] = MRSIII.Umbral_pretensor;
      datos[2] = MRSIII.Umbral_activacion_frontal;
      datos[3] = MRSIII.Umbral_choque_trasero;
      Escritura_Trama_Dinamica(0x0, 92, 4, datos);    //Seg Dinamico 3
      Fray_Transfer_Input(Fray_PST, &buff_comand_input);
      dinaseg_3 = false;
   }
else if (frameMOD == 0x1)
{
   buff_comand_input.ldsh = 1; // load data section                IBCM  0
   buff_comand_input.lhsh = 1; // load header section
   buff_comand_input.ibrh = mslot12;
```

```
      int i;
      for (i = 0; i < interfaz_trama_in.longt_payload; i++)
      {
         datos[i] = interfaz_trama_in.dato[i];
      }
      Escritura_Trama_Dinamica(interfaz_trama_in.seg_startup_UN.seg_startup_UL,
                  interfaz_trama_in.ID_trama,
                  interfaz_trama_in.longt_payload, datos); //Seg Dinamico 3
      Fray_Transfer_Input(Fray_PST, &buff_comand_input);
      frameMOD=0;
   }


// check received frames
   ndat1 = Fray_PST->NDAT1_UN.NDAT1_UL;

   if (ndat1 != 0)
   {
      buff_comand_output.rdss = 1;  // read data section
      buff_comand_output.rhss = 1;  // read header section

      if (ndat1 & 0x8)  //buffer #4 Estado del MRS III
      {
         buff_comand_output.obrs = slot3; // output buffer number
         Fray_Transfer_Output(Fray_PST, &buff_comand_output);
         //Leer el búfer de mensajes
         bufer.Buff_Headers.HEADER1_UN.HEADER1_UL =
               Fray_PST->RDHS1_UN.RDHS1_UL;
         bufer.Buff_Headers.HEADER2_UN.HEADER2_UL =
               Fray_PST->RDHS2_UN.RDHS2_UL;
         bufer.Buff_Headers.HEADER3_UN.HEADER3_UL =
               Fray_PST->RDHS3_UN.RDHS3_UL;
         bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL =
               Fray_PST->MBS_UN.MBS_UL;
         bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

         plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
         plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
         if (plr == plc)
            for (i = 0; i < plr; i++)
            {
               bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
            }
         Lectura_trama(&sistema, &bufer);

         MRSIII.Activacion = sistema.dato[0];
         MRSIII.Estado = sistema.dato[1];

      }
      if (ndat1 & 0x2000)    //buffer #13 Valores de configuración MRS III
      {
         buff_comand_output.obrs = mslot13; // output buffer number
```

```
        Fray_Transfer_Output(Fray_PST, &buff_comand_output); //Leer el búfer de
mensajes
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL =
              Fray_PST->RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL =
              Fray_PST->RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL =
              Fray_PST->RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL =
              Fray_PST->MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
           for (i = 0; i < plr; i++)
           {
              bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
           }
        Lectura_trama(&sistema, &bufer);

        MRSIII.Estado = sistema.dato[0];
        MRSIII.Umbral_pretensor = sistema.dato[1];
        MRSIII.Umbral_activacion_frontal = sistema.dato[2];
        MRSIII.Umbral_choque_trasero = sistema.dato[3];
//        MRSIII.Umbral_BST = sistema.dato[4];
//        MRSIII.Umbral_ITS = sistema.dato[5];
        set_Umbrales(&MRSIII);

      }
      if (ndat1 & 0x4000)    //buffer #14 Estado del panel indicadores
      {
        buff_comand_output.obrs = mslot14;  // output buffer number
        Fray_Transfer_Output(Fray_PST, &buff_comand_output);
        //Leer el búfer de mensajes
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL =
              Fray_PST->RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL =
              Fray_PST->RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL =
              Fray_PST->RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL =
              Fray_PST->MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
           for (i = 0; i < plr; i++)
           {
              bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
           }
        Lectura_trama(&sistema, &bufer);
```

```
        pindicadores->Indicadores_UN.Indicadores_UL = (sistema.dato[1] << 8)
              + sistema.dato[0];
      }
   }
}

void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,
                unsigned plc, unsigned long* datos)
/*****************************************************************
****
 segment startup => (ppi)(nfi)(syn)(sfi)
 frame ID
 payload length configured (0xFF <=> 1 byte)
 Datos

*****************************************************************
****/
{
   trama_t sistema;
   sistema.seg_startup_UN.seg_startup_UL = seg_startup;

   wrhs header;
   header.cfg = 1;    // Transmit buffer
   header.syn = sistema.seg_startup_UN.seg_startup_ST.sincrona;
   header.sfi = sistema.seg_startup_UN.seg_startup_ST.startup;
   header.fid = fid;
   header.plc = plc;
   header.rcc = 0;      //Receive cycle count
   header.cha = 1;    // Transmission on Ch A
   header.chb = 0;    // No transmission on Ch B

   switch(fid){
   case 90:
      header.dp = 0x110; // Pointer to start of data in message RAM
      break;
   case 91:
      header.dp = 0x120; // Pointer to start of data in message RAM
      break;
   case 92:
      header.dp = 0x130; // Pointer to start of data in message RAM
      break;
   case 93:
      header.dp = 0x210; // Pointer to start of data in message RAM
      break;
   case 94:
      header.dp = 0x220; // Pointer to start of data in message RAM
      break;
   default:
      header.dp = 0x240; // Pointer to start of data in message RAM
   }
   sistema.CRC_cabecera = header_crc_calc(&header); // calculated  by the host
   Fray_buffConfig(frayREG, &header);
}
```

Anexo G. Código módulo FlexRay B

```
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>

wrhs header_reg; //Variables globales
cfg Fray_Config;
bc Fray_buffcomand1;
bc Fray_buffcomand2;
int uh;

extern panelsen_t panel_sensores;
extern panelact_t panel_indicadores;
extern MRS_t MRSIII;
extern unsigned datos_carga_util[64];
extern volatile boolean dinaseg_1;
extern volatile boolean dinaseg_2;
extern volatile boolean dinaseg_3;
extern volatile boolean dinaseg_4;
extern volatile boolean dinaseg_5;
extern volatile unsigned modo_config;
extern volatile unsigned frameMOD;
extern trama_t interfaz_trama_in;
extern unsigned EnableMRSIII;


void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,unsigned
plc, unsigned long* datos);

void configure_initialize_node_b(FRAY_ST *Fray_PST)
{
   wrhs *Fray_header = &header_reg;
   cfg *Fr_ConfigPtr = &Fray_Config;
   bc *Fray_buff_comand = &Fray_buffcomand1;

   // GTUs, PRTC configuration
   Fr_ConfigPtr->gtu1 = 0x00036B00; // pMicroPerCycle = 224000d = 36B00h
   Fr_ConfigPtr->gtu2 = 0x000F15E0; // gSyncodeMax = Fh, gMacroPerCycle =
5600d = 15E0h
   Fr_ConfigPtr->gtu3 = 0x00061818; // gMacroInitialOffset = 6h,
pMicroInitialOffset = 24d = 18h
   Fr_ConfigPtr->gtu4 = 0x0AE40AE3; // gOffsetCorrectionStart - 1 = 2788d =
AE4h, gMacroPerCycle - gdNIT - 1 = 2787d = AE3h
   Fr_ConfigPtr->gtu5 = 0x33010303; // pDecodingCorrection = 51d = 33h,
pClusterDriftDamping = 1h, pDelayCompensation = 3h
   Fr_ConfigPtr->gtu6 = 0x01510081; // pdMaxDrift = 337d = 151h,
pdAcceptedStartupRange = 129d = 81h
   Fr_ConfigPtr->gtu7 = 0x00080056; // gNumberOfStaticSlots = 8h, gdStaticSlot =
86d = 56h
```

```
   Fr_ConfigPtr->gtu8 = 0x015A0004; // gNumberOfMinislots = 346d = 15Ah,
gdMinislot = 4h
   Fr_ConfigPtr->gtu9 = 0x00010204; // gdDynamicSlotIdlePhase = 1,
gdMinislotActionPointOffset = 2, gdActionPointOffset = 4h
   Fr_ConfigPtr->gtu10 = 0x015100CD; // pRateCorrectionOut = 337d = 151h,
pOffsetCorrectionOut = 205d = CDh
   Fr_ConfigPtr->gtu11 = 0x00000000; // pExternRateCorrection = 0,
pExternOffsetCorrection = 0, no ext. clk. corr.
   Fr_ConfigPtr->succ2 = 0x0F036DA2; // gListenNoise = Fh, pdListenTimeout =
224674d = 36DA2h
   Fr_ConfigPtr->succ3 = 0x000000FF; // gMaxWithoutClockCorrectionFatal = Fh ,
passive = Fh
   Fr_ConfigPtr->prtc1 = 0x084C000A; // pWakeupPattern = 2h,
gdWakeupSymbolRxWindow = 76d, BRP = 0, gdTSSTransmitter = Ah
   Fr_ConfigPtr->prtc2 = 0x3CB41212; // gdWakeupSymbolTxLow = 60d,
gdWakeupSymbolTxIdle = 180d, gdWakeupSymbolRxLow = 18d,
gdWakeupSymbolRxIdle = 18d
   Fr_ConfigPtr->mhdc = 0x010D0009; // pLatestTransmit = 269d = 010Dh,
gPayloadLengthStatic = 9h
   Fr_ConfigPtr->mrc = 0x00174006; // LCB=23d, FFB=64d, FDB=4d (0..3 static,
4..23 dyn., 0 fifo)

   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0)
      ;

   // Initialize
   Fr_Init(Fray_PST, Fr_ConfigPtr);

   //Header Section Global Config
   //Header Section Register 1 (WRHS1)
   Fray_header->mbi = 1;   // message buffer interrupt disabled
   Fray_header->txm = 1;   // transmission mode - continuous mode
   Fray_header->ppit = 0;  // Payload Preamble Indicator is not set
   Fray_header->cfg = 0; // message buffer configuration bit (0= receive buffer, 1 =
transmit buffer)
   Fray_header->chb = 1;   // Ch B
   Fray_header->cha = 1;   // Ch A
   Fray_header->cyc = 0;   // Cycle Filtering Code (no cycle filtering)
   Fray_header->fid = 0;   // Frame ID

   //Header Section 2 (WRHS2)
   Fray_header->plc = 0;   // Payload Length configured
   Fray_header->crc = 0;  // calculated  by the host

   //Header Section 3 (WRHS3)
   Fray_header->dp = 0;   // Pointer to start of data in message RAM
   Fray_header->sfi = 0;   // startup frame indicator
   Fray_header->syn = 0;   // sync frame indicator

   //Buffer Command Global Config
   //Output Buffer
   Fray_buff_comand->rdss = 0;  // read data section                OBCM
```

```
   Fray_buff_comand->rhss = 0;  // read header section                OBCM
   Fray_buff_comand->obrs = 0;  // output buffer number                OBCR

   //Input Buffer
   Fray_buff_comand->stxrh = 0; // set transmission request           IBCM
   Fray_buff_comand->ldsh = 0; // load data section                  IBCM
   Fray_buff_comand->lhsh = 1; // load header section                IBCM
   Fray_buff_comand->ibrh = 0; // input buffer number                IBCR
   Fray_buff_comand->ibsyh = 1; // check for input buffer busy host      IBCR
   Fray_buff_comand->ibsys = 1; // check for input buffer busy shadow    IBCR

   // Message buffers Particular Config
//=================================================
===Static segment
   // Buffer #4 Transmit buffer
   Fray_header->fid = 4;      // frame ID
   Fray_header->cfg = 1;      // Transmit buffer
   Fray_header->plc = 9; // 18 byte payload length configured (0xFF <=> 1 byte)
   Fray_header->dp = 0x200;     // Pointer to start of data in message RAM
   Fray_header->rcc = 0;      //Receive cycle count
   Fray_header->rci = 0;      //Received on channel indicator
   Fray_header->sfi = 1;      //Startup frame indicator
   Fray_header->syn = 1;      //Sync frame indicator
   Fray_header->nfi = 0;      //Null frame indicator
   Fray_header->ppi = 0;      //Payload preamble indicator
   Fray_header->res = 0;      //Reserved bit
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 0;  // input buffer number                IBCR
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #1 Receive buffer
   Fray_header->fid = 1;    // frame ID
   Fray_header->dp = 0x80; // Pointer to start of data in message RAM
   Fray_header->cfg = 0;    // Receive buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Receive on Ch A
   Fray_header->chb = 1;    // Receive on Ch B
   Fray_header->plc = 9;    // 18 byte payload
   Fray_header->crc = 0;
   Fray_buff_comand->ibrh = 1;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #2 Receive buffer
   Fray_header->fid = 2;    // frame ID
   Fray_header->dp = 0x90; // Pointer to start of data in message RAM
   Fray_header->cfg = 0;    // Receive buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Receive on Ch A
   Fray_header->chb = 1;    // Receive on Ch B
```

```
   Fray_header->plc = 9;    // 18 byte payload
   Fray_header->crc = 0;
   Fray_buff_comand->ibrh = 2;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #3 Receive buffer
   Fray_header->fid = 3;    // frame ID
   Fray_header->dp = 0x100; // Pointer to start of data in message RAM
   Fray_header->cfg = 0;    // Receive buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Receive on Ch A
   Fray_header->chb = 1;    // Receive on Ch B
   Fray_header->plc = 9;    // 18 byte payload
   Fray_header->crc = 0;
   Fray_buff_comand->ibrh = 3;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);


//=================================================
===Dynamic segment
   // buffer #13 Transmit buffer
   Fray_header->fid = 93;   // frame ID
   Fray_header->dp = 0x210; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Transmission on Ch A
   Fray_header->chb = 0;    // No transmission on Ch B
   Fray_header->plc = 9;  // 254 byte payload
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 13;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #14 Transmit buffer
   Fray_header->fid = 94;   // frame ID
   Fray_header->dp = 0x220; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Transmission on Ch A
   Fray_header->chb = 0;    // No transmission on Ch B
   Fray_header->plc = 9;  // 254 byte payload
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 14;  // input buffer number
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #10 Receive buffer
   Fray_header->fid = 90;    // frame ID
```

```
    Fray_header->dp = 0x110; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 10;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #11 Receive buffer
    Fray_header->fid = 91;    // frame ID
    Fray_header->dp = 0x120; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 11;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #12 Receive buffer
    Fray_header->fid = 92;    // frame ID
    Fray_header->dp = 0x130; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 12;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    Fr_ControllerInit(Fray_PST);

    // Initialize Interrupts
    Fray_PST->EIR_UN.EIR_UL = 0xFFFFFFFF; // Clear Error Int.
    Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF; // Clear Status Int.
    Fray_PST->SILS_UN.SILS_UL = 0x00000000; // all Status Int. to eray_int0
    Fray_PST->SIER_UN.SIER_UL = 0xFFFFFFFF; // Disable all Status Int.
    Fray_PST->SIES_UN.SIES_UL = 0x00000004; // Enable CYCSE Int.
    Fray_PST->ILE_UN.ILE_UL = 0x00000002; // enable eray_int1

    Fr_AllowColdStart(Fray_PST);
}
```

```
void comunicacion_Nodo_B(FRAY_ST *Fray_PST)
{
   panelsen_t* psensores = &panel_sensores;
   panelact_t* pindicadores = &panel_indicadores;
   trama_t sistema;
   int i, plr, plc;
   bc buff_comand_output;
   unsigned long ndat1;
   fray_buff bufer;
   unsigned long* datos = (unsigned long*) Fray_PST->WRDS;

   bc buff_comand_input;
   buff_comand_input.stxrh = 1;  // set transmission request          IBCM
   buff_comand_input.ldsh = 1;  // load data section                 IBCM
   buff_comand_input.lhsh = 0;  // load header section               IBCM
   buff_comand_input.ibrh = 0;  // input buffer number               IBCR
   buff_comand_input.ibsys = 0; // check for input buffer busy shadow  IBCR
   buff_comand_input.ibsyh = 1; // check for input buffer busy host    IBCR

   //wait for cycle start interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;        // clear all status int. flags
   while (Fray_PST->SIR_UN.SIR_ST.cycs_B1 == 0x0)
      ;    // wait for CYCS (Cycle Start Interrupt) interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;        // clear all status int. flags

   buff_comand_input.ibrh = slot_Stup;
   datos[0] = MRSIII.Activacion;
   datos[1] = MRSIII.Estado;
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);

   if (dinaseg_4==true) //Configuracion MRS III
   {
      buff_comand_input.ibrh = mslot13
      ;
      get_Umbrales(&MRSIII);
      datos[0] = MRSIII.Estado;
      datos[1] = MRSIII.Umbral_pretensor;
      datos[2] = MRSIII.Umbral_activacion_frontal;
      datos[3] = MRSIII.Umbral_choque_trasero;
      datos[4] = MRSIII.Umbral_BST;
      datos[5] = MRSIII.Umbral_ITS;
      Escritura_Trama_Dinamica(0x0, 93, 6, datos);    //Seg Dinamico 4
      Fray_Transfer_Input(Fray_PST, &buff_comand_input);
      dinaseg_4=false;
   }

   if (dinaseg_5==true)    //Estado panel
   {
      buff_comand_input.ibrh = mslot14
      ;
      datos[0] = pindicadores->Indicadores_UN.Indicadores_UL & 0xFF;
      datos[1] = (pindicadores->Indicadores_UN.Indicadores_UL >> 8) & 0xFF;
```

```
      Escritura_Trama_Dinamica(0x0, 94, 2, datos);    //Seg Dynamic 5
      Fray_Transfer_Input(Fray_PST, &buff_comand_input);
      dinaseg_5=false;
   }

// check received frames
   ndat1 = Fray_PST->NDAT1_UN.NDAT1_UL;

   if (ndat1 != 0)
   {
      buff_comand_output.rdss = 1;  // read data section
      buff_comand_output.rhss = 1;  // read header section

      if (ndat1 & 0x2)    //buffer #1
      {
         buff_comand_output.obrs = slot1
         ;  // output buffer number
         Fray_Transfer_Output(Fray_PST, &buff_comand_output);
         //Leer el búfer de mensajes
         bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
         bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
         bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
         bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
         bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

         plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
         plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
         if (plr == plc)
            for (i = 0; i < plr; i++)
            {
               bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
            }
         Lectura_trama(&sistema, &bufer);

         psensores->imp_Front_Cond = sistema.dato[0];
         psensores->imp_Front_Pasj = sistema.dato[1];
         psensores->imp_Late_Cond = sistema.dato[2];
         psensores->imp_Late_Pasj = sistema.dato[3];
         psensores->imp_Trasero = sistema.dato[4];

      }
      if (ndat1 & 0x04)    //buffer #2
      {
         buff_comand_output.obrs = slot2
         ;
         // output buffer number
         Fray_Transfer_Output(Fray_PST, &buff_comand_output);
         //Leer el búfer de mensajes
```

```
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
           for (i = 0; i < plr; i++)
           {
              bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
           }
        Lectura_trama(&sistema, &bufer);

        psensores->seg_Front_Cond = sistema.dato[0];
        psensores->seg_Front_Pasj = sistema.dato[1];
        psensores->seg_Late_Cond = sistema.dato[2];
        psensores->seg_Late_Pasj = sistema.dato[3];
        psensores->seg_Trasero = sistema.dato[4];

     }
     if (ndat1 & 0x08)    //buffer #3
     {
        buff_comand_output.obrs = slot3
        ;
        // output buffer number
        Fray_Transfer_Output(Fray_PST, &buff_comand_output);
        //Leer el búfer de mensajes
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
           for (i = 0; i < plr; i++)
           {
              bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
           }
        Lectura_trama(&sistema, &bufer);

        psensores->velocidad = sistema.dato[0];
```

```
        psensores->encendido = sistema.dato[1];

     }
     if (ndat1 & 0x400)    //buffer #10
     {
        buff_comand_output.obrs = mslot10
        ;  // output buffer number
        Fray_Transfer_Output(Fray_PST, &buff_comand_output);
        //Leer el búfer de mensajes
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
           for (i = 0; i < plr; i++)
           {
              bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
           }
        Lectura_trama(&sistema, &bufer);

        psensores->SBE_Cond = sistema.dato[0];
        psensores->SBE_Pasj = sistema.dato[1];

     }
     if (ndat1 & 0x0800)    //buffer #11
     {
        buff_comand_output.obrs = mslot11
        ;  // output buffer number
        Fray_Transfer_Output(Fray_PST, &buff_comand_output);
        //Leer el búfer de mensajes
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
           for (i = 0; i < plr; i++)
           {
```

```
            bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
         }
      Lectura_trama(&sistema, &bufer);

      psensores->scint_Cond = sistema.dato[0];
      psensores->scint_Pasj = sistema.dato[1];
   }
   if (ndat1 & 0x1000)    //buffer #12
   {
      buff_comand_output.obrs = mslot12
      ;  // output buffer number
      Fray_Transfer_Output(Fray_PST, &buff_comand_output);
      //Leer el búfer de mensajes
      bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
      bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
      bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
      bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
      bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

      plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
      plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
      if (plr == plc)
         for (i = 0; i < plr; i++)
         {
            bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
         }
      Lectura_trama(&sistema, &bufer);

      if(sistema.ID_trama==92){
         modo_config = 1;
         MRSIII.Estado = sistema.dato[0];
         MRSIII.Umbral_pretensor = sistema.dato[1];
         MRSIII.Umbral_activacion_frontal = sistema.dato[2];
         MRSIII.Umbral_choque_trasero = sistema.dato[3];
         set_Umbrales(&MRSIII);
      }
   }
  }
}

void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,
              unsigned plc, unsigned long* datos)
/*****************************************************************
****
 segment startup => (ppi)(nfi)(syn)(sfi)
 frame ID
 payload length configured (0xFF <=> 1 byte)
 Datos
```

```
**********************************************************************
****/
{
   trama_t sistema;
   sistema.seg_startup_UN.seg_startup_UL = seg_startup;

   wrhs header;
   header.cfg = 1;    // Transmit buffer
   header.syn = sistema.seg_startup_UN.seg_startup_ST.sincrona;
   header.sfi = sistema.seg_startup_UN.seg_startup_ST.startup;
   header.fid = fid;
   header.plc = plc;
   header.rcc = 0;      //Receive cycle count
   header.cha = 1;    // Transmission on Ch A
   header.chb = 0;    // No transmission on Ch B

   switch(fid){
   case 90:
      header.dp = 0x110; // Pointer to start of data in message RAM
      break;
   case 91:
      header.dp = 0x120; // Pointer to start of data in message RAM
      break;
   case 92:
      header.dp = 0x130; // Pointer to start of data in message RAM
      break;
   case 93:
      header.dp = 0x210; // Pointer to start of data in message RAM
      break;
   case 94:
      header.dp = 0x220; // Pointer to start of data in message RAM
      break;
   }
   sistema.CRC_cabecera = header_crc_calc(&header); // calculated  by the host
   Fray_buffConfig(frayREG, &header);
}
```

Anexo H. Código proyecto nodo A (HL_sys_main.c)

```
/** @file HL_sys_main.c
*   @brief Application main file
*   @date 11-Dec-2018
*   @version 04.07.01
*
*   This file contains an empty main function,
*   which can be used for the application.
*/

/*
* Copyright (C) 2009-2018 Texas Instruments Incorporated - www.ti.com
*
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*    Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
*    Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the
*    distribution.
*
*    Neither the name of Texas Instruments Incorporated nor the names of
*    its contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
*  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
*  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*/
```

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include "HL_sys_common.h"

/* USER CODE BEGIN (1) */
#include "HL_sys_core.h"
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>
#include "HL_sci.h"
/* USER CODE END */

/** @fn void main(void)
*   @brief Application main function
*   @note This function is empty by default.
*
*   This function is called after startup.
*   The user can use this function to implement the application.
*/

/* USER CODE BEGIN (2) */
void comunicacion_Nodo_A(FRAY_ST *Fray_PST);
int lectura_Interfaz(trama_t* frame);
void envio_Interfaz(trama_t* frame);
void trama_Interfaz(trama_t* frame);
void response_Interfaz(trama_t* frame);


volatile int taskSist = 0;
volatile unsigned modo_lectura;
volatile boolean dinaseg_1;
volatile boolean dinaseg_2;
volatile boolean dinaseg_3;
volatile boolean dinaseg_4;
volatile boolean dinaseg_5;
int posicion;

trama_t interfaz_trama_in;
trama_t interfaz_trama_out;
panelsen_t panel_sensores;
panelact_t panel_indicadores;
MRS_t MRSIII;
FRAY_ST* FlexRay_CC = frayREG;
adcData_t adc_data[7];

unsigned datos_carga_util[64];
unsigned modo_config;
unsigned interfaz_in;
unsigned interfaz_out;
```

```
unsigned band_Vib;
unsigned frameMOD;
int response=0;

/* USER CODE END */

int main(void)
{
/* USER CODE BEGIN (3) */
   _enable_IRQ_interrupt_();

//=====================================================
==========GPIO
   gioInit();

//=====================================================
==========RTI
   rtiInit();
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE1);
   rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);

//=====================================================
==========FlexRay
   Config_CC_Nodo_A(FlexRay_CC);
   Fr_StartCommunication(FlexRay_CC);

//=====================================================
==========ADC
   adcInit();
   adcStartConversion(adcREG1, adcGROUP1);

//=====================================================
==========HET
   hetInit();

//=====================================================
==========UART
   sciInit();

//=====================================================
==========Inicialización
   modo_lectura = 0;
   posicion = 0;
   dinaseg_1 = false;
   dinaseg_2 = false;
   dinaseg_3 = false;
   dinaseg_4 = false;
   dinaseg_5 = false;
   interfaz_in = 0;

   while (1)
   {
```

```
      switch (taskSist) //Planificador
      {
      case 1:
      {
         if (modo_lectura == 0)
            lectura_sensores(&panel_sensores);
         else if(interfaz_in == 0)
            interfaz_sensores(&panel_sensores);
         while(taskSist==1);
         break;
      }
      case 2:
      {
         _disable_IRQ_interrupt_();
         comunicacion_Nodo_A(FlexRay_CC);
         _enable_IRQ_interrupt_();
         while(taskSist==2);
         break;
      }
      case 3:
      {
         if (interfaz_in)
         {
            response = lectura_Interfaz(&interfaz_trama_in);
            if(response==1){
               response_Interfaz(&interfaz_trama_out);
               envio_Interfaz(&interfaz_trama_out);
               response=0;
            }

            if (interfaz_trama_in.ID_trama == 0x9)
               modo_lectura = interfaz_trama_in.dato[0] & 0x1;

            if (interfaz_trama_in.ID_trama == 92)
            {
               dinaseg_3=true;
               config_Umrales(&interfaz_trama_in);
            }
            else if(interfaz_trama_in.ID_trama > 94 && interfaz_trama_in.ID_trama
< 101)
               frameMOD = 0x1;
            interfaz_in = 0;
         }
         if (interfaz_out){
            trama_Interfaz(&interfaz_trama_out);
            envio_Interfaz(&interfaz_trama_out);
            interfaz_out=0;
         }
         while(taskSist==3);
         break;
      }
      }
   }
```

```
   /* USER CODE END */

   //return 0;
}


/* USER CODE BEGIN (4) */
/* USER CODE END */
```

Anexo I. Código proyecto nodo B (HL_sys_main.c)

```
/** @file HL_sys_main.c
*   @brief Application main file
*   @date 11-Dec-2018
*   @version 04.07.01
*
*   This file contains an empty main function,
*   which can be used for the application.
*/

/*
* Copyright (C) 2009-2018 Texas Instruments Incorporated - www.ti.com
*
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*    Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
*    Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the
*    distribution.
*
*    Neither the name of Texas Instruments Incorporated nor the names of
*    its contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
*  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
*  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*/
```

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include "HL_sys_common.h"

/* USER CODE BEGIN (1) */
#include "HL_sys_core.h"
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>
/* USER CODE END */

/** @fn void main(void)
*   @brief Application main function
*   @note This function is empty by default.
*
*   This function is called after startup.
*   The user can use this function to implement the application.
*/

/* USER CODE BEGIN (2) */
void comunicacion_Nodo_B(FRAY_ST *Fray_PST);
void lectura_estados();
void Operacion_MRSIII();


volatile int taskSist = 0;
volatile boolean dinaseg_1;
volatile boolean dinaseg_2;
volatile boolean dinaseg_3;
volatile boolean dinaseg_4;
volatile boolean dinaseg_5;
volatile unsigned modo_config;
volatile unsigned modo_lectura;
int posicion;

panelsen_t panel_sensores;
panelact_t panel_indicadores;
MRS_t MRSIII;
FRAY_ST* FlexRay_CC = frayREG;


/* USER CODE END */

int main(void)
{
/* USER CODE BEGIN (3) */
   _enable_IRQ_interrupt_();
```

```
//=====================================================
==========GPIO
   gioInit();

//=====================================================
==========RTI
   rtiInit();
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE1);
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE2);
   rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
   rtiStopCounter(rtiREG1, 1);

//=====================================================
==========FlexRay
   configure_initialize_node_b(FlexRay_CC);
   Fr_StartCommunication(FlexRay_CC);

//=====================================================
==========HET
   hetInit();

//=====================================================
==========Inicialización
   modo_lectura = 0;
   posicion = 0;
   dinaseg_1 = false;
   dinaseg_2 = false;
   dinaseg_3 = false;
   dinaseg_4 = false;
   dinaseg_5 = false;
   modo_lectura = 0;
   MRSIII.Estado = 0;
   MRSIII.Activacion = 0;
   posicion = 0;
   initUmbrales();
   reset_Indicadores();
   MRSIII.Estado = autocomprobacion_panel_Indica();
   dinaseg_4 = true;
   dinaseg_5 = true;

   while (1)
   {
      switch (taskSist) //Planificador
      {
      case 1:
      {
         if (dinaseg_4==true)
            get_Umbrales(&MRSIII);

         if (MRSIII.Activacion == 0)
         {
```

```
            if (MRSIII.Estado == 4)
            {
               MRSIII.Estado = autocomprobacion_panel_Indica();
               dinaseg_5=true;
            }
            else if (MRSIII.Estado != 1)
            {
               MRSIII.Estado = autocomprobacion_sensores();
               dinaseg_5=true;
            }
         }
         while (taskSist == 1);
         break;
      }
      case 2:
      {
         _disable_IRQ_interrupt_();
         comunicacion_Nodo_B(FlexRay_CC);
         _enable_IRQ_interrupt_();
         while (taskSist == 2);
         break;
      }
      case 3:
      {
         if (modo_config == 1)
         {
            set_Umbrales(&MRSIII);
            dinaseg_4=true;
            modo_config = 0;
         }


         if (MRSIII.Estado == 1)
         {
            ejecucion_MRSIII();
            dinaseg_5=true;
         }
         while (taskSist == 3);
         break;
      }
      }
   }
   /* USER CODE END */

   //return 0;
}


/* USER CODE BEGIN (4) */
/* USER CODE END */
```

Anexo J. Script Matlab del procesamiento digital de la trama FlexRay muestreada.

```
%% Import data from spreadsheet
% Script for importing data from the following spreadsheet:
%
%    Workbook: D:\Tesis\imagenes\tramas\Trama FlexRay_v2.xlsm
%    Worksheet: F0002CH1
%
% To extend the code for use with different selected data or a different
% spreadsheet, generate a function instead of a script.

% Auto-generated by MATLAB on 2021/10/17 15:28:33

%% Import the data
clear
clc
[~, ~, raw0_0] = xlsread('D:\Tesis\imagenes\tramas\Trama
FlexRay_v2.xlsm','F0002CH1','D1:E1400');
[~, ~, raw0_1] = xlsread('D:\Tesis\imagenes\tramas\Trama
FlexRay_v2.xlsm','F0002CH1','G1:G1400');
raw = [raw0_0,raw0_1];
raw(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw) = {''};

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Create output variable
data = reshape([raw{:}],size(raw);

%% Allocate imported array to column variable names
BPH = data(:,1);
BMH = data(:,2);
%Tiempo = data(:,3);

N=size(BMH)*2;
Tiempo=zeros();
BP=zeros();
BM=zeros();
uBus=zeros();
Reloj=zeros();
Tiempo(1)=1e-8;
for i=2:1:N-2
   Tiempo(i,1)=Tiempo(i-1,1)+1e-8;
end

x=1;
band=1;
offset =0;
```

```
for i=1:1:N-2
   if(mod(i,2)==0)
      BP(i,1)=BPH(x,1)+offset;
      BM(i,1)=BMH(x,1)+offset;

   else
      BP(i,1)=(BPH(x,1)+BPH(x+1,1)/2+offset;
      BM(i,1)=(BMH(x,1)+BMH(x+1,1)/2+offset;
      Reloj(i,1)=-1;
      x=x+1;
   end
   if(mod(i,5)==0)
      band=band*-1;
   end
   if(band==1)
      Reloj(i,1)=.5;
   else
      Reloj(i,1)=-.5;
   end
   uBus(i,1)=BP(i,1)-BM(i,1);

end

figure

subplot(2,1,1)
plot(Tiempo,BP,Tiempo,BM,[0],[3.6])

title('Trama Diferencial FlexRay')
xlabel('Tiempo')
ylabel('Voltaje')
legend('BP','BM')
%set(gca,'xtick',[],'ytick',[])

subplot(2,1,2)
grid off;
plot(Tiempo,uBus,Tiempo,Reloj,[0],[2])

title('Codificación FlexRay')
xlabel('Tiempo')

ylabel('Amplitud')
legend('uBus','Reloj')
set(gca,'ytick',[])

%% Clear temporary variables
clearvars data raw raw0_0 raw0_1 R;
```