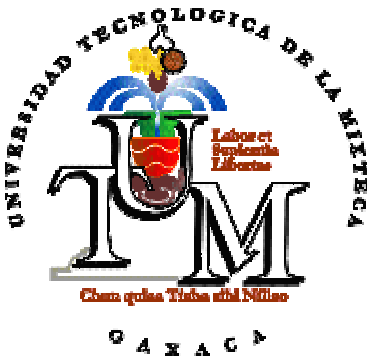


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"ESPECIFICACIÓN DEL PROTOCOLO FLEXRAY
UTILIZANDO UN LENGUAJE DE DESCRIPCIÓN FORMAL"

TESIS

PARA OBTENER EL TÍTULO DE
INGENIERO EN ELECTRÓNICA

PRESENTA

RENÉ GONZÁLEZ SALINAS

DIRECTORES DE TESIS

ING. HERIBERTO ILDEFONSO HERNÁNDEZ MARTÍNEZ

HUAJUAPAN DE LEÓN, OAX.; JULIO DE 2008

Tesis presentada el día 11 de julio de 2008
Ante los sinodales:

M.C. Maribel Tello Bello
M.C. Alejandro E. Ramírez González
M.C. Fermin H. Ramírez Leyva

Director de Tesis:
Ing. Heriberto I. Hernández Martínez

Dedicatoria

Con mucho amor y cariño a la memoria de mamá Justina Salinas Avelino (Q.E.P.D.).

A mis hermanos Ody, Cundy, Jorge, Ever, Mely, Miky, Aure y Misa.

A mi padre.

René

"Sólo los que miran más allá del horizonte saben dar la inspiración y la motivación a sí mismos y a los demás; sólo los que luchan saben el valor de la meta… el paso grande de llegar al éxito. Sea este esfuerzo homenaje para los que han estado, en mucho y por mucho, dando todo moral e incondicional apoyo"

Agradecimientos

Agradezco a Heriberto I. Hernández Martínez, por permitirme trabajar bajo su dirección, compartiéndome su experiencia y conocimientos, así como su amistad y confianza haciendo posible finalizar este presente proyecto de tesis.

Al M.C. José A. Moreno Espinosa, cuyas aportaciones contribuyeron para mejorar el presente documento.

A mi familia, por que a pesar de lo difícil de las situaciones siempre han estado conmigo, especialmente a Miky por que sin su apoyo no me hubiera sido posible lograrlo.

A los sinodales Hugo, Alejandro y Maribel, por su aportación a este trabajo.

A mi amigo y profesor Alejandro (mouse) por sus consejos y apoyo a lo largo de estos años.

A mis amigos Abraham, Humberto, Edgar, Kena y Aleyda, por todos esos momentos y experiencias que compartimos.

A mis compañeros de la carrera, Miguelito, Crilin, Ulises, Pepe, putla y todos aquellos que se me escapan en esta lista.

A la familia Arango, por lo hermoso de su amistad, los sabios consejos y la motivación para continuar que me brindó tan bella señora Azarel.

Al consorcio FlexRay™, por facilitar la información relacionada a la especificación del protocolo y la firma FleeScale por el soporte proporcionado a lo largo del desarrollo del proyecto de tesis.

Reno

Índice

Índice de figuras

Índice de tablas

Introducción

Las técnicas de descripción formal (FDTs, *Formal Description Technique*) surgieron como resultado del trabajo realizado durante dos décadas encaminado al desarrollo de lenguajes de especificación formal que proporcionen métodos rigurosos para la descripción de sistemas basados en computadora; en esencia todos estos lenguajes se fundamentan en una teoría matemática que asegura su precisión y manejabilidad [8, 11].

Las principales organizaciones interesadas en la elaboración de FDTs son: el CCITT (*International Consultative Committee on Telegraphy and Telephony*) y la ISO (*International Organization for Standardization*). La ISO normativizó las FDTs ESTELLE (*Extended Finite State Machine Language*) y LOTOS (*Language of Temporal Ordering Specification*), mientras que el CCITT elaboró la norma SDL (*Specification and Description Language*) [2, 8, 10].

El uso de las FDTs en la aplicación de sistemas, equipos, protocolos o normas beneficia a los profesionales que trabajan en la definición de nuevas normas, los fabricantes, los grupos que realizan pruebas y verificación de protocolos y sistemas, y a los usuarios finales.

En la actualidad, se hace uso de las FDTs en el diseño de sistemas, equipos y normas en una diversidad de aplicaciones. Al resultado de describir y diseñar un sistema con un FDT se llama especificación; para un mismo sistema puede haber más de una especificación posible con diferentes niveles de abstracción. Las especificaciones formales sirven como base para extraer realizaciones del sistema, aunque el método de diseño tiene que empezar con especificaciones muy abstractas que ocultan detalles de realización y proporcionan una visión general del sistema; y es mediante sucesivos pasos de refinamiento que se logran obtener especificaciones menos abstractas y poco a poco se van incluyendo las decisiones de realización.

SDL es un estándar para especificar y diseñar sistemas de tiempo real. Fue desarrollado por el CCITT, actualmente ITU-T (*International Telecommunications Union, Telecommunications Area*), y está detallado en la recomendación Z.100 [2, 9, 13, URL2, URL5].

Como antecedente se mencionan los siguientes trabajos utilizando FDTs como herramientas de diseño para validar, comprobar y verificar el correcto funcionamiento de la especificación:

- Comprobación de una terminal de servicios bancaria ATM (*Automatic Teller Machine*) con SDL [URL 11].
- Desarrollo de un banco de pruebas de tiempo real para telefonía inalámbrica DECT (*Digital Enhanced Cordless Telecommunications*) con SDL [URL 10].
- Descripción formal del protocolo de comunicaciones industriales WorldFIP con LOTOS [URL 9].

- Descripción del protocolo de comunicaciones industriales PROFIBUS con SDL y LOTOS [25, URL 9]

Por otro lado, en la actualidad se cuenta con una gran variedad de sistemas de comunicaciones automotrices, de los cuales los más utilizados son CAN (*Controller Area Network*), J-1850, DSI (*Distributed Systems Interface*), LIN (*Local Interconnect Network*) e IEEE-1394 [1, 3]. Muchos de estos sistemas están orientados a eventos, es decir que sólo proporcionan una respuesta cuando ocurre un evento.

Flexray es un protocolo de comunicaciones desarrollado por el Consorcio Flexray ™ y soportado por las firmas: BMW AG, Daimler Chrysler AG, Freescale Halbleiter Deutschland GmbH, General Motors corporation, Philips GmbH, Robert Bosch GmbH y Volkswagen AG [URL3]. Flexray implementa una arquitectura con tiempos de respuesta conocidos, utilizando el esquema de acceso múltiple por división en el tiempo (TDMA, *Time Division Multiple Access*) y TDMA flexible (FTDMA, *Flexible TDMA*), además soporta una velocidad máxima de hasta 10 Mbps, con los que se pueden enviar y recibir mensajes prácticamente[1] en tiempo real [URL3], respecto a las aplicaciones automotrices.

Flexray soporta diferentes topologías de red, donde el bus de datos puede ser configurado como canal único o canal doble, topología de estrella o bus híbrido. Un *cluster* está compuesto por dos canales, A y B, y cada nodo en el *cluster* (hasta 64 nodos) puede estar conectado a uno o ambos canales. Flexray es un sistema tolerante a fallos (*fault tolerance*) gracias a su capacidad de soportar dos canales de datos y usar algoritmos para corrección de fase y frecuencia, esto último para sincronizar al nodo con el *cluster*.

Un ciclo de comunicación está compuesto por un segmento estático (TDMA) y un segmento dinámico (FTDMA); en el segmento estático, todas las ranuras de comunicación y todas las tramas son idénticas en duración y longitud de trama, mientras que en el segmento dinámico éstas pueden ser de duración y longitud diferente, dependiendo de los datos a transmitir y sus prioridades de transmisión.

Con base en lo anterior, el objetivo principal del presente trabajo de tesis es realizar la especificación del protocolo Flexray utilizando el lenguaje de descripción formal SDL.

Para cumplir el objetivo principal, se plantean los siguientes objetivos secundarios:

- Realizar una investigación de las FDTs existentes.
- Estudiar el funcionamiento de la norma SDL.
- Estudiar el funcionamiento del protocolo de comunicaciones Flexray.
- Realizar la especificación del protocolo Flexray

Para realizar una descripción formal de un sistema o protocolo con una FDT, es necesario planificar el desarrollo de la especificación en una serie de fases o etapas[2] [10]. La metodología de desarrollo a utilizar es orientada a estados y orientada a restricciones, debido a que en la especificación de un sistema en SDL, el aspecto dinámico (*procesos*) se realiza mediante máquinas de estados (*orientada a estados*), los cuales estarán bien definidos y estructurados en bloques o módulos (*orientada a restricciones*). La Figura I.1 muestra las etapas a seguir para la elaboración de la especificación formal a desarrollar en esta tesis:

[1] En el campo de las aplicaciones automotrices, la máxima velocidad de transferencia era de 2.5 Mbps con TTCAN (Time Triggered CAN), Flexray supera los requerimientos de velocidad para el tiempo de respuesta de los sistemas *x-by-wire* que ofrece TTCAN.

[2] Las dos últimas etapas no se ejecutan debido a que la presente tesis, como caso de estudio, no está desarrollada para una aplicación específica y las primeras cuatro etapas cubren los requerimientos de diseño formal con una FDT.

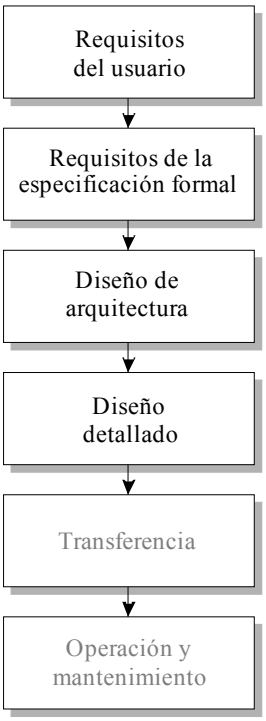


Figura I. 1. Metodología de desarrollo para especificar el protocolo de comunicaciones Flexray.

- *Requisitos de usuario*: En esta etapa se definen, de forma clara y concisa, los requisitos que debe cumplir la especificación mediante el uso del lenguaje natural. En este caso los requisitos de usuario serán las especificaciones del protocolo Flexray.
- *Requisitos de la especificación formal*: Se definen los requisitos necesarios para escribir la especificación SDL, así como validar, verificar y simular su comportamiento.
- *Diseño de arquitectura*: En esta etapa se descompone el sistema global en módulos que interaccionan entre sí, el número de módulos y/o sub-módulos depende de la profundidad de abstracción con la que se desea realizar la especificación.
- *Diseño detallado*: Cada uno de los módulos que componen al sistema será especificado formalmente con SDL, incluyendo la depuración, la simulación y el refinamiento mediante la herramienta de desarrollo.
- *Transferencia*: Cuando finaliza la especificación se transfiere a los usuarios (*no ejecutada en este trabajo de tesis*).
- *Operación y mantenimiento*: Se usa la especificación formal obtenida y se hacen las modificaciones necesarias para adaptarla a los usuarios finales (*no ejecutada en este trabajo de tesis*).

La metodología descrita debe permitir:

- Escribir especificaciones sin ambigüedad, claras, precisas y concisas.
- Analizar y corregir en su plenitud una especificación.
- Determinar si un diseño cumple con determinada especificación.
- Utilizar herramientas para crear, mantener, analizar y simular especificaciones.

Para especificar con SDL se eligió la herramienta Cinderella SDL [9, URL1], herramienta comercial que incorpora las modificaciones más recientes al estándar y que permite crear sistemas SDL de forma amigable, proporcionando las siguientes características:

- Análisis incremental.

- Integración de SDL con el estándar ASN.1.
- Edición, análisis y simulación integral.
- Importar y exportar especificaciones con otras herramientas de SDL.
- Soporte con SDL de acuerdo al estándar ITU-T.

La herramienta Cinderella SDL soporta notación GR (gráfica) y notación PR (texto), esta última es poco recomendable para especificar sistemas robustos. Una desventaja de esta herramienta es que no soporta macros en notación GR, las cuales son de gran ayuda en sistemas con muchos estados.

A continuación se detalla el contenido del documento de tesis (Figura I.2):

- El Capítulo 1 describe las características de las principales FDTs.
- El Capítulo 2 presenta un estudio de la FDT SDL.
- En el Capítulo 3 se presenta el estudio del protocolo de comunicaciones Flexray, basado en la arquitectura de protocolos OSI.
- La especificación del protocolo Flexray se presenta en el Capítulo 4, así mismo se introduce la metodología SPECS (*Specification and Programming Environment for Communications Software*) utilizada para presentar y documentar de manera formal el trabajo realizado.
- El Capítulo 5 presenta los resultados obtenidos de las pruebas realizadas a la especificicación.
- Se presentan las conclusiones en el Capítulo 6 y por último las referencias bibliográficas utilizadas durante el presente trabajo de tesis. Los apéndices correspondientes a los acrónimos usados y una breve descripción sobre cómo instalar y usar la herramienta Cinderalla SDL.

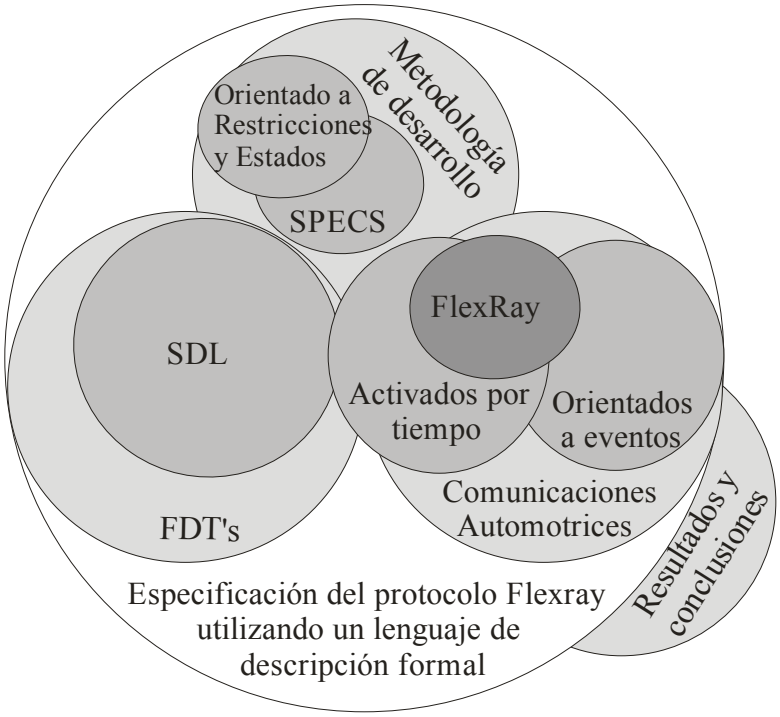


Figura I. 2. Estructura del documento de tesis.

1. Técnicas de descripción formal

Las FDTs (*Formal Description Technique*) surgieron por la necesidad de contar con métodos rigurosos de diseño para el desarrollo de sistemas de computación y de comunicaciones [30, 31]; a partir de la década de los 60's, los equipos y protocolos de dichos sistemas presentaron un incremento en su complejidad, por lo que su diseño y verificación representaba una tarea compleja y de altos costos, tanto en recursos humanos como económicos. Para resolver tal problema se planteó el desarrollo de lenguajes de especificación formal basados en conceptos matemáticos que asegurasen un diseño correcto de equipos, sistemas y protocolos [24].

Las descripciones convencionales normalmente se realizan con lenguajes naturales (español, inglés, francés, etc.) o con diagramas, lo que hace que presenten ambigüedades y sean difíciles de analizar. Generalmente los errores se detectan en la etapa de desarrollo físico del sistema y su rectificación es costosa en tiempo y en dinero, debido a que se tiene que volver a la etapa de diseño para corregir los errores y reiniciar la construcción física. Con las especificaciones formales se evitan ambigüedades ya que se puede simular paso a paso la evolución de la especificación y realizar diferentes pruebas (*test*) de comportamiento [18], con la finalidad de detectar posibles errores u omisiones en la etapa de diseño; llegando a la etapa de desarrollo físico solamente cuando se sabe con certeza que el sistema, equipo o protocolo diseñado es correcto; lo anterior representa un ahorro en costos y tiempos durante la realización total.

Las FDTs surgieron como resultado del trabajo realizado durante dos décadas encaminado al desarrollo de lenguajes de especificación formal que proporcionen métodos rigurosos para la descripción de sistemas basados en computadora [31]. En esencia todos estos lenguajes se fundamentan en una teoría matemática que asegura su precisión y manejabilidad. Las principales áreas de aplicación de las FDTs son el desarrollo de sistemas complejos [6, 24]:

- *Concurrentes*: Sistemas distribuidos, sistemas en tiempo real, diseño hardware, procesado paralelo.
- *De calidad crítica*: Aplicaciones financieras, telecomunicaciones, sistemas operativos.
- *De seguridad crítica*: Defensa, medicina, industria nuclear, equipos militares señalización ferroviaria, telecomunicaciones, aparatos de vuelo en aeronaves.
- *De confidencialidad*: Sistemas de información, prevención de accesos no autorizados.
- *Descripción de normas internacionales*: De amplio uso y que deben ser interpretadas uniformemente (sin ambigüedad) en todo el mundo.

Las principales organizaciones que se interesaron por la elaboración de FDTs, y que incentivaron su aparición y aplicación, fueron el CCITT y la ISO. El CCITT empezó su trabajo para la creación de normas mediante FDTs en el año 1972 y la ISO alrededor del año 1978. Inicialmente la investigación del CCITT estuvo dirigida a aplicaciones en el campo de las telecomunicaciones, se-

ñalización y conmutación; mientras que la ISO se dedicó al estudio del procesamiento de datos. Con el tiempo las líneas de investigación de ambas organizaciones se fueron aproximando dando lugar a la aparición de las FDTs [31]. La ISO normalizó las FDTs ESTELLE [17] y LOTOS [19]; mientras que el CCITT elaboró la norma SDL [4].

Debido a que inicialmente las FDTs estaban orientadas a su aplicación en sistemas con distintos objetivos, es difícil encontrar un determinado lenguaje formal que presente una solución global y universal para todos los diseños. Por tal motivo, en 1985 el CCITT y la ISO decidieron trabajar conjuntamente en la elaboración de normas, aplicación e introducción al mercado industrial de las FDTs. Esta colaboración consistió en ofrecer un conjunto de ejemplos de especificaciones, desde sistemas simples hasta sistemas complejos, realizando las descripciones en ESTELLE, LOTOS y SDL; con la finalidad de que los diseñadores de sistemas pudieran estudiar y comparar las técnicas formales y elegir la que mejor se adecuara a su campo de aplicación.

1.1. Normalización de las FDTs

Las FDTs son una herramienta de uso común para los equipos de desarrollo de sistemas durante su ciclo de vida, para ello deben contar con reglas bien definidas para que los distintos grupos que interactúan en el desarrollo puedan realizar especificaciones con la misma FDT y exista compatibilidad entre ellas. El CCITT y la ISO iniciaron en la década de los 70 el trabajo de normalización de las FDTs, logrando la estandarización de ESTELLE, LOTOS y SDL.

En la década de los 70s, la ISO empezó a trabajar en la producción de normas para la interconexión de sistemas abiertos (OSI, *Open System Interconnection*). Una premisa importante era lograr que los sistemas abiertos pudieran funcionar e interactuar correctamente con cualquier otro sistema abierto de diferente fabricante; lo que planteaba el hecho de que las normas debieran carecer de ambigüedad y ser precisas, sin cabida a diferentes interpretaciones. Se adoptó el término FDT para referirse a los lenguajes y métodos formales que se iban a normalizar. Aunque el término FDT actualmente se utiliza para cualquier lenguaje o método que tenga una base formal, su uso adecuado es para referirse a las técnicas formales normalizadas por la ISO y el CCITT.

La ISO desarrolló dos categorías:

- Métodos formales basados en autómatas de estado finito (ESTELLE).
- Métodos formales basados en métodos algebraicos (LOTOS).

Por otro lado, el CCITT se encarga de realizar normas[3] en el campo de las telecomunicaciones. Con el crecimiento de la complejidad de los sistemas de telecomunicaciones y con la gran influencia que empezó a tener la computación, el CCITT desarrolló técnicas para especificar sistemas computarizados de una forma precisa, estable y sin ambigüedades. En el año de 1976, el CCITT publicó la primera versión de su propia norma sobre FDTs denominada SDL. El CCITT cooperó con la ISO para trabajar en colaboración en el desarrollo de sus respectivas normas sobre FDTs, y la versión más avanzada de SDL [4] se logró al mismo tiempo que ESTELLE y LOTOS eran publicadas como normas. La colaboración entre el CCITT y la ISO se hace patente en que SDL tiene una serie de características comunes con ESTELLE y LOTOS, heredando el concepto de autómata de estado finito empleado por ESTELLE y usando el tipo de datos algebraico de LOTOS.

[3] Denominadas recomendaciones por el CCITT.

1.2. Finalidad de las FDTs

Las FDTs se desarrollaron para asegurar que las especificaciones de los sistemas, equipos y protocolos o normas:

- No presenten ambigüedad, sean claras y concisas.
- Sean completas.
- Sean consistentes con sí mismas o en relación con otras.
- Sean manejables teniendo una fácil legibilidad e interpretación.
- Que las realizaciones se ajusten a ellas.

De esta forma las especificaciones son completas con una única interpretación universal. Con el uso de las FDTs se descubren errores, ambigüedades e inconsistencias en la especificación que se está desarrollando; también ayuda a realizar un estudio estructurado del problema que se quiere resolver, lo que supone una mejora en la etapa de planteamiento de su resolución.

Aunque inicialmente las FDTs se idearon para ser utilizadas en aplicaciones en el campo de las telecomunicaciones, señalización, conmutación, computación y procesado de datos, su uso puede hacerse extensivo a cualquier sistema [6, 23, 24].

1.3. Principales FDTs

Las FDTs especifican el comportamiento de un sistema, como un sistema de transiciones etiquetado (*Labelled Transition System*), donde las diferentes transiciones entre estados se etiquetan con la acción asociada que las provoca. Con respecto a la definición de datos, ESTELLE utiliza los tipos de datos del lenguaje de programación PASCAL [16], mientras que LOTOS y SDL los especifican algebraicamente como tipos de datos abstractos (ADTs, *Abstract Data Types*) [7]. A continuación se describen brevemente dichas FDTs.

1.3.1. ESTELLE

ESTELLE es un lenguaje de especificación formal estandarizado por la ISO bajo la norma ISO/IEC 9074 [17], e ideado para describir sistemas de procesado distribuido o concurrente, en particular aquellos que realizan servicios y protocolos OSI. Se basa en conceptos de comunicación entre máquinas de estado no deterministas (autómatas) aceptadas y utilizadas ampliamente, especificando los sistemas de manera jerárquica y estructurada. Las diferentes máquinas de estado se comunican intercambiando mensajes a través de canales bidireccionales establecidos entre sus puertos de comunicación. Las acciones del sistema especificado se definen utilizando un desarrollo que deriva de la norma de programación PASCAL.

ESTELLE permite especificar y describir el comportamiento de diferentes sistemas que evolucionan en paralelo y de una forma que puede ser síncrona o asíncrona. Las especificaciones pueden tener diferentes niveles de abstracción, desde especificaciones muy abstractas hasta especificaciones orientadas a la realización física que se derivan de las primeras etapas del diseño con la ayuda de herramientas de desarrollo.

1.3.2. LOTOS

LOTOS fue desarrollado basando su comportamiento en CCS (*Calculus of Comunicating System*) [19] y en CSP (*Communicating Sequential Processes*), y la parte de datos en el lenguaje ACT ONE [27] con soporte para tipos de datos abstractos. Por lo tanto, LOTOS tiene una fuerte base matemática y algebraica, que permite especificar sin ambigüedad: los comportamientos secuenciales, elecciones, concurrencias y comportamientos no determinísticos, así como comunicaciones síncronas y asíncronas. Fue normalizado por la ISO y detallado en la norma ISO/IEC 8807 [19].

1.3.3. SDL

SDL fue desarrollado por el CCITT y está detallada en la recomendación Z.100 [4]. Esta organización ha mantenido esta FDT y ha realizado diversas modificaciones desde sus inicios, publicando en 1992 una nueva versión extendida de SDL (SDL-92) [24], la cual se basa en un modelo de máquina de estados finito extendido y en una especificación de datos que permite tipos de datos abstractos (ADT, *Abstract Data Type*)[4].El lenguaje SDL tiene una serie de construcciones para representar diferentes estructuras, comportamientos, interfaces y enlaces de comunicaciones. Este lenguaje tiene un aspecto gráfico y las especificaciones se construyen mediante diagramas de flujo.

Esta FDT fue diseñada para ser utilizada en la especificación de sistemas de telecomunicaciones, incluyendo aspectos de servicios y protocolos. En sus principios se basó en la descripción de los primeros sistemas de conmutación telefónica controlados por computadora.

1.4. Especificación y desarrollo de un sistema mediante FDTs

En la actualidad se ha incrementado el uso de las FDTs en el diseño de sistemas, equipos y normas en una diversidad de aplicaciones, reconociendo los beneficios en el desarrollo de sistemas. El resultado de describir y diseñar un sistema con una FDT se llama especificación. Para un mismo sistema puede haber más de una especificación posible con diferentes niveles de abstracción. Las especificaciones formales sirven como base para extraer realizaciones del sistema, aunque el método de diseño tiene que empezar con especificaciones muy abstractas que oculten detalles de realización y proporcionen una visión general del sistema; mediante sucesivos pasos de refinamiento permite obtener especificaciones menos abstractas y poco a poco se van incluyendo las decisiones de realización [24, 26].

El uso de lenguajes formales de especificación permite realizar análisis y simulaciones de posibles soluciones alternativas a un determinado sistema, pudiendo comprobar cuál de ellas es la más adecuada y efectuar una selección durante la etapa de diseño que supondrá un importante ahorro en tiempo y costos.

La Figura 1.1 muestra un diagrama general para el desarrollo de un sistema mediante una FDT. En el primer paso del diseño formal se hace uso de términos del lenguaje natural para definir un modelo que represente las propiedades significativas del sistema, sobre todo en su parte de comportamiento, y cuya relación sea directa e intuitiva con el sistema. Esto se hace eligiendo nombres para los conceptos formales que se corresponden con los términos en el lenguaje natural, o incluyendo comentarios (como en los lenguajes de programación) en el modelo formal que explique lo que representa.

Una vez definido el modelo formal se realiza una recopilación y análisis de los requerimientos del sistema a especificar, y se pasa a realizar una especificación formal que los incluya. El paso siguiente es realizar pruebas a la especificación obtenida para comprobar su consistencia con respecto a los requerimientos y mediante el análisis de los resultados se efectúa un refinamiento de la especificación obteniendo una nueva especificación mejorada. Esto se repite continuamente hasta que se alcanza el grado de consistencia y complejidad deseada. Una vez realizados todos los pasos de refinamiento y obtenida la especificación formal final con el grado de abstracción deseado, se procede a la realización y validación del sistema [5, 6, 8].

[4] Es el mismo modelo que usa LOTOS con ACT ONE.

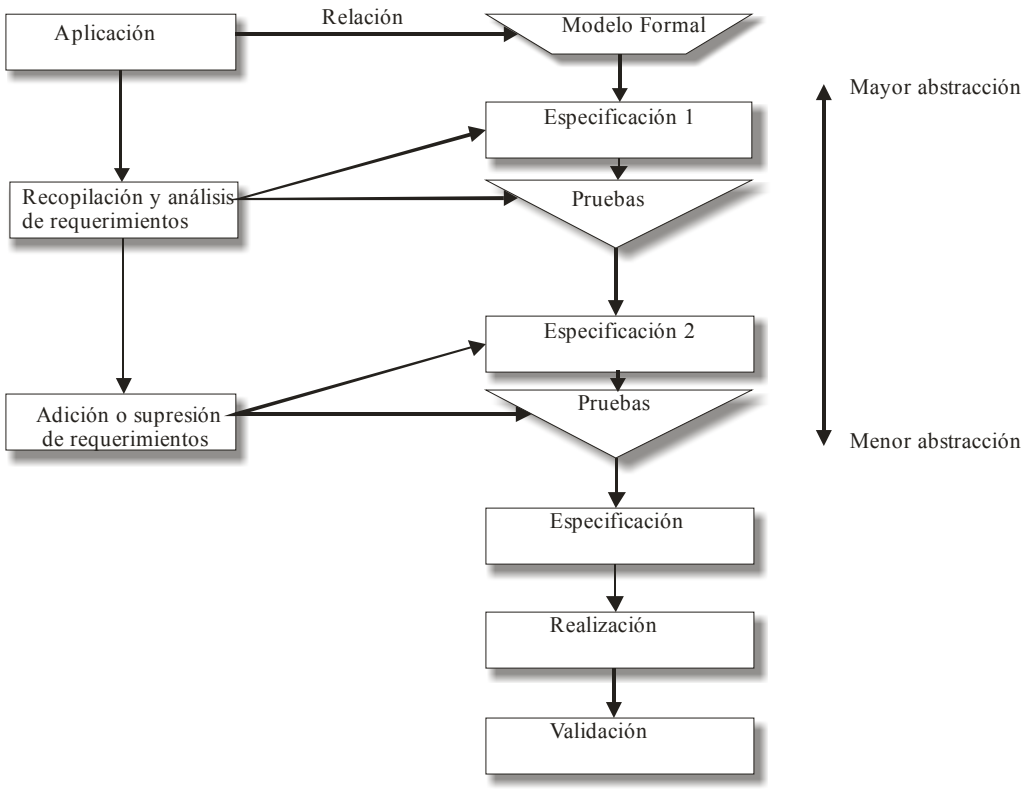


Figura 1.1. Etapas del diseño formal.

Para aplicar el diseño formal de forma correcta y realizar especificaciones formales con sus posteriores etapas de refinamiento, mediante análisis y pruebas de comportamiento, es necesario contar con herramientas de programación (*software*), equipo de cómputo que permitan la creación, análisis, comprobación y refinamiento, y especificaciones formales escritas con algunos de los lenguajes normalizados. Sin estas herramientas de programación, el diseño formal se convertiría en una tarea compleja y no sería eficaz en tiempo y recursos humanos.

2. Lenguaje de descripción formal SDL

El objetivo fundamental de esta tesis es la especificación y descripción de una aplicación de tiempo real mediante una FDT, de entre los principales métodos o lenguajes formales mencionados anteriormente se eligió SDL con base en lo siguiente [25] (metodología de desarrollo, Figura I.1):

- Se trata de una FDT normalizada por un organismo internacional.
- Permite realizar especificaciones de sistemas reactivos[5] como son: sistemas de control para los procesos de fabricación, protocolos de comunicaciones, programas controladores de dispositivos, etc.
- Las herramientas disponibles se encuentran muy desarrolladas, la mayoría de ellas son comerciales e incorporan las actualizaciones más recientes del estándar SDL.
- Su representación gráfica facilita la interpretación de la especificación y su simbología es jerarquizada, lo que permite obtener diferentes niveles de abstracción.
- Soporta notación tanto gráfica (GR) como textual (PR).
- Su estructura orientada a intercambio de señales la hace adecuada para la especificación de protocolos.
- Su capacidad de soportar conceptos orientados a objetos permite escribir sistemas de forma modular permitiendo la reutilización de componentes.

El estándar SDL establece que una especificación describe cuáles son los requerimientos que define el comportamiento global de un sistema, mientras que una descripción indica su comportamiento real; de forma que una especificación ve al sistema como una caja negra y una descripción refleja su estructura interna. Sin embargo, el lenguaje SDL no hace distinción alguna entre especificación y descripción [20], por esta razón en este trabajo de tesis se usa cualquiera de los dos términos para referirse a la especificación del protocolo Flexray.

2.1. Historia del lenguaje SDL

SDL es un lenguaje estándar para especificar y describir sistemas. Fue desarrollado por la CCITT, actualmente ITU-T [URL4], detallado en la recomendación Z.100.

La necesidad de nuevas herramientas para la creación de los cada vez más complejos sistemas de control de programas almacenado (SPC, *Stored Program Control*) hizo que en 1972 se iniciara el desarrollo de nuevos lenguajes, dando origen a SDL.

[5] Sistemas que responden de una forma determinada a los estímulos de su entorno.

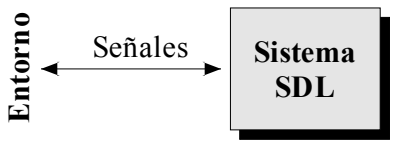


Figura 2.1. Interacción de un sistema SDL con su entorno

Desde su primera versión, publicada en 1976 (libro naranja), SDL ha sufrido cambios graduales en las versiones de 1980 (libro amarillo), 1984 (libro rojo) y 1988 (libro azul), siendo esta última donde contó con una base formal alcanzando una madurez como FDT. En SDL-92 (libro blanco) los cambios fueron notables al incluir conceptos de orientación a objetos, llamadas remotas y procesamiento no determinístico, librerías (*packages*); posteriormente, SDL-96 añade pequeñas mejoras y corrige errores detectados en las versiones anteriores. SDL-2000 permite definir objetos e implementa diagramas UML (*Unified Modeled Languaje*) para representar clases (*type*).

El presente trabajo de tesis se basa en las versiones SDL 92 y SDL-96, resaltando que las diferencias entre éstas no son significativas.

2.2. Estructura de un sistema SDL

SDL ve al mundo dividido en dos partes, el sistema y su entorno. Una especificación SDL es un modelo formal que define las propiedades relevantes de un sistema existente o planificado en el mundo real, todo aquello que no se especifique dentro del sistema se considera como parte de su entorno [5, 24, 26]. La especificación del sistema define cómo reaccionará ante eventos del entorno en que se encuentra, los cuales son comunicados al sistema por medio de señales (*signals*), a esta dependencia del sistema ante su entorno se le conoce como sistema OSI.

Un sistema no existe en el mundo real de forma aislada, sino que necesita interactuar con el exterior. Aquellos eventos que se producen en el entorno y que han de ser considerados por el sistema son enviados a él mediante el uso de señales como se ilustra en la Figura 2.1. Las señales son el único medio de comunicación de un sistema SDL con su entorno. El comportamiento del entorno es generalmente impredecible y aunque su descripción no se incluye en una especificación SDL, es necesario imponer una serie de restricciones que dependerán del sistema bajo análisis. El entorno no puede causar errores dinámicos (envío de señales a procesos no existentes), aún cuando el comportamiento del entorno no estuviera restringido y SDL ignora las acciones no permitidas.

2.2.1. Máquinas de estado finito

Los procesos que forman parte de un sistema SDL se basan en el concepto de teoría de autómatas o máquinas de estado finito (FSM, *Finite State Machine*) [32]. Una FSM consiste en un conjunto finito de estados (Q), uno de los cuales es el estado inicial (q_0), un alfabeto de entrada (A) y una función de transferencia de estado (f), representado mediante la 4-tupla:

$$FSM = (Q, q_0, A, f)$$

La función f proporciona un nuevo estado, que puede ser el actual, para cada combinación de un estado del conjunto Q y de una entrada del conjunto A. La Figura 2.2 muestra el ejemplo de un comprobador de paridad de un bit, donde el conjunto de estados es Q = {par, impar}, el estado inicial es q_0 = par, el alfabeto de entrada A = {0, 1} y la función de transferencia:

$$f(par, 0) \rightarrow par$$

$$f(par, 1) \rightarrow impar$$

$$f(impar, 0) \rightarrow impar$$

$$f(impar, 1) \rightarrow par$$

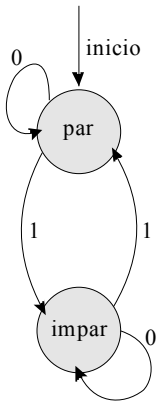


Figura 2.2. Máquina de estados para un comprobador de paridad de un bit.

2.2.2. Máquinas de estado finito extendidas

A medida que el sistema se vuelve más complejo, el número de estados necesarios para representarlo mediante una FSM crece rápidamente, esto se conoce como explosión de estados y se soluciona empleando una máquina de estados finito extendida (EFSM, *Extended FSM*). Así, para realizar un comprobador de paridad de un octeto mediante un FSM se necesitan 256 estados. Por medio de una EFSM, se utiliza una variable para llevar la cuenta de los bits que se han recibido. De este modo, el nuevo conjunto de estados es Q = {par, impar, par_final, impar_final}, el estado inicial q_0 = par, la variable c pertenece el conjunto de variables Z = {c} cuyo valor inicial es cero, definiendo a la EFSM como una 5-tupla:

$$EFSM= (Q, q_0, A, f, Z)$$

La función de transferencia f también se modifica para incluir transiciones condicionales en función de la variable c y acciones sobre ésta. Por ejemplo para el estado inicial y ante una entrada 0 la función de transferencia quedaría como sigue:

$$f(par, 0, c<7/\{c := c + 1\}) \rightarrow par$$

La Figura 2.3 muestra la aplicación de una EFSM para un comprobador de paridad de un octeto, así como su representación en SDL.

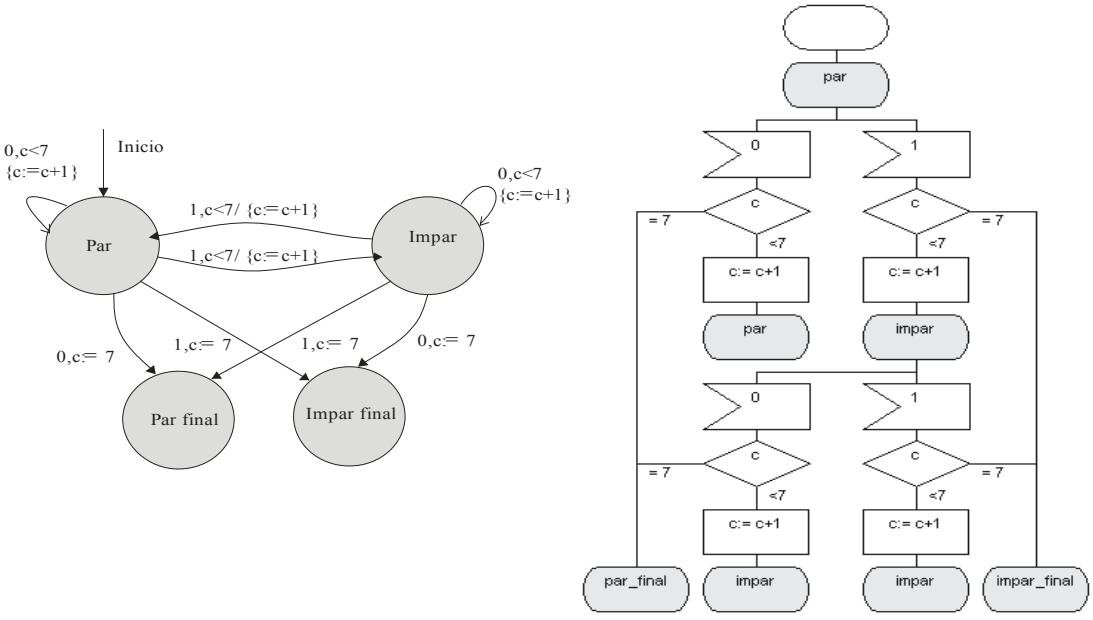


Figura 2.3. CEFSM para un comprobador de paridad de un octeto y su presentación en SDL.

El uso de variables permite modificar el comportamiento de la máquina de estados, cambiando en la función de transferencia el valor 7 por 31 obteniendo como resultado un comprobador de paridad de 32 bits; así mismo, reduce la complejidad de una FSM contribuyendo al estado final, pero no define ningún estado concreto.

Para conocer el estado de una EFSM se completa la 5-tupla que la define con un alfabeto de salida, siguiendo el ejemplo en la Figura 2.3 se tiene que X = {par, impar} (X es el alfabeto de salida) para obtener la 6-tupla:

$$CEFSM = \{Q, q_0, A, f, X, Z\}$$

La información del alfabeto de salida de la CEFSM (*Communicating EFSM*) se proporciona al exterior al alcanzar los estados finales.

2.2.3. Arquitectura de un sistema SDL

En la Figura 2.4 se muestra la arquitectura del diseño de un sistema en SDL, cuyo nivel superior o nivel sistema (*system*) está compuesto por bloques (*blocks*); cada bloque puede contener bloques o procesos pero no ambos; los procesos (*process*) están compuestos por máquinas de estado (CEFSM) comunicándose entre sí mediante señales a través de canales (*channels*) o ruta de señal (*signal route*).

Para la especificación de un sistema se deben cumplir las siguientes reglas [24, 26]:

- Una descripción SDL debe contener sólo una instancia a *system.*
- El sistema (*system*) debe contener al menos un bloque; si sólo contiene un bloque, se puede omitir la instancia a *system*.
- Un bloque puede contener bloques o procesos pero no ambos; si sólo contiene un proceso, se puede omitir la instancia a *block*.
- Un proceso no puede contener procesos o bloques, sin embargo puede contener instancias a servicios (*service*), procedimientos (*procedure*) o bien la especificación de máquinas de estados.

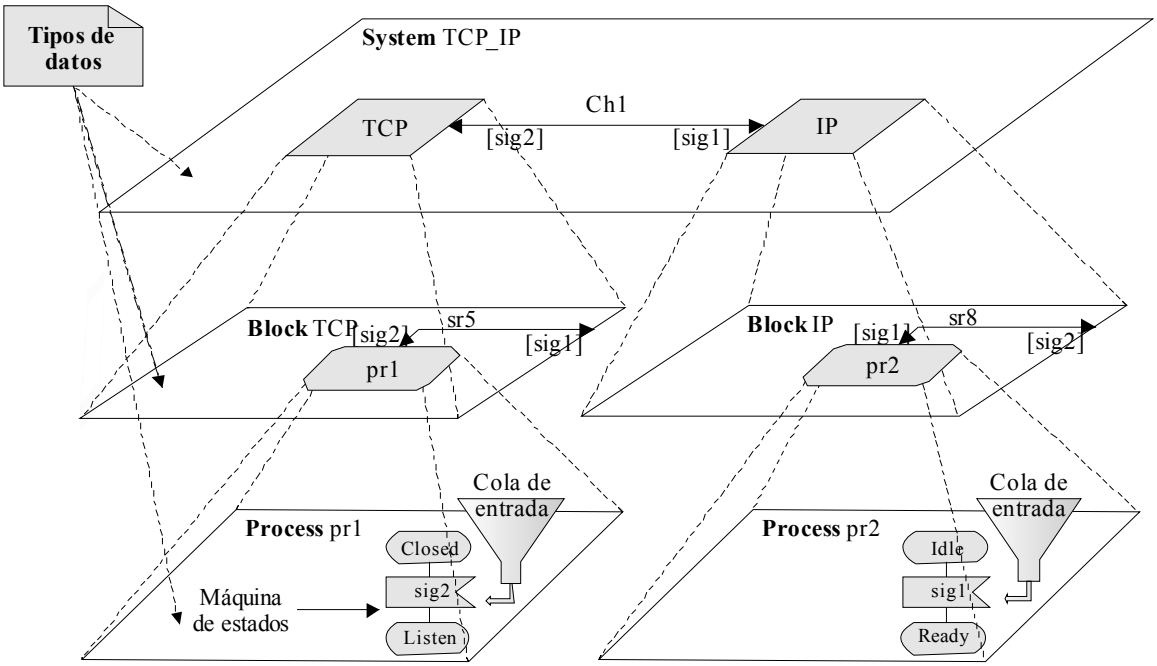


Figura 2.4. Arquitectura de una especificación SDL.

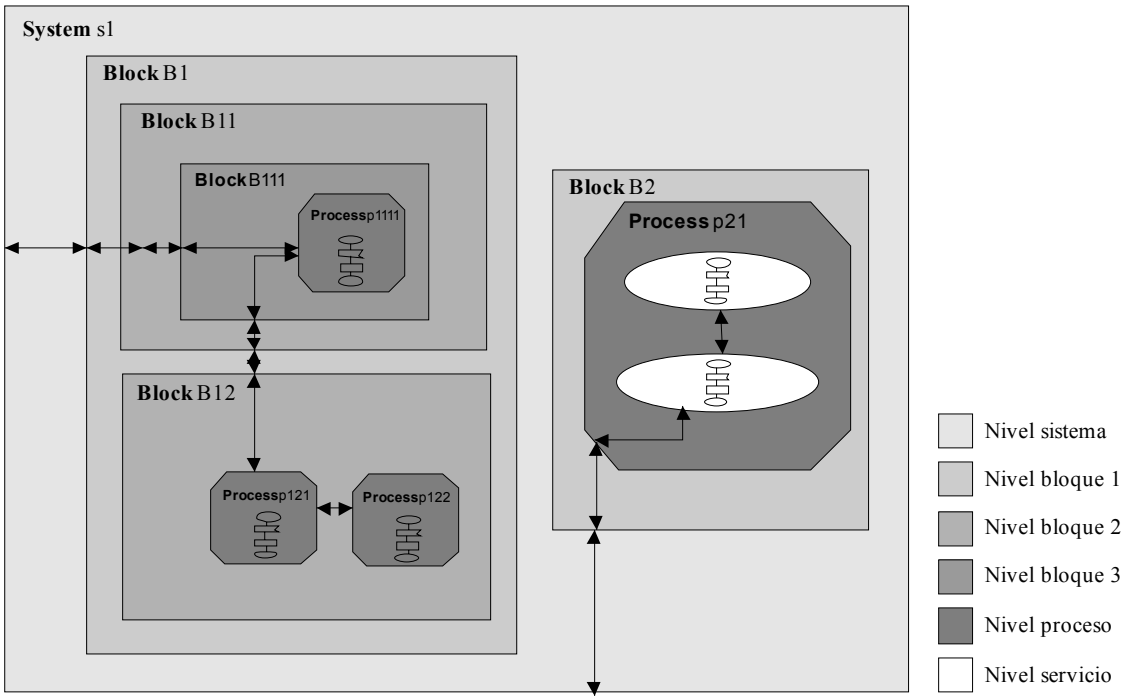


Figura 2.5. Niveles de abstracción de un sistema SDL.

2.2.3.1. Partes de un sistema

Un sistema SDL se descompone en tres partes: datos, descripción estática y descripción dinámica. Los tipos de datos y variables que se declaran en las descripciones estática y dinámica forman la parte de datos de la especificación del sistema.

La descripción estática define al sistema en diferentes niveles de abstracción y establece las vías de comunicación que utilizarán los procesos para comunicarse entre sí y con el entorno; esta descripción se realiza utilizando símbolos SDL como: bloques, canales, procesos, servicios y rutas de señal, señales. Después de jerarquizar el sistema, la parte dinámica define su comportamiento mediante un conjunto de procesos, cuyo comportamiento está regido por una CEFSM.

2.2.3.2. Niveles de abstracción

Se definen como mínimo tres niveles de abstracción: nivel sistema, nivel bloque y nivel proceso (Figura 2.4). El nivel sistema es el más externo y presenta una descomposición en un conjunto de bloques, cada bloque es susceptible de ser dividido en bloques obteniendo en cada división un nivel de bloque o abstracción, como se ilustra en la Figura 2.5. Los bloques del último nivel se componen de procesos, estos últimos representan el nivel de proceso; los procesos pueden dividirse en servicios, en cuyo caso añade a la especificación un nivel de servicio.

2.2.3.3. Reglas de denominación

Las principales reglas de denominación son las entidades y las unidades de ámbito, las cuales se describen a continuación.

2.2.3.3.1. Entidades

Se denomina entidad a un objeto como puede ser un bloque, proceso, etc. Estas entidades se agrupan en clases de forma que los bloques constituyen una clase de entidad, lo mismo aplica para sistemas, canales, ruta de señal, procesos, procedimientos, variables, servicios y temporizadores.

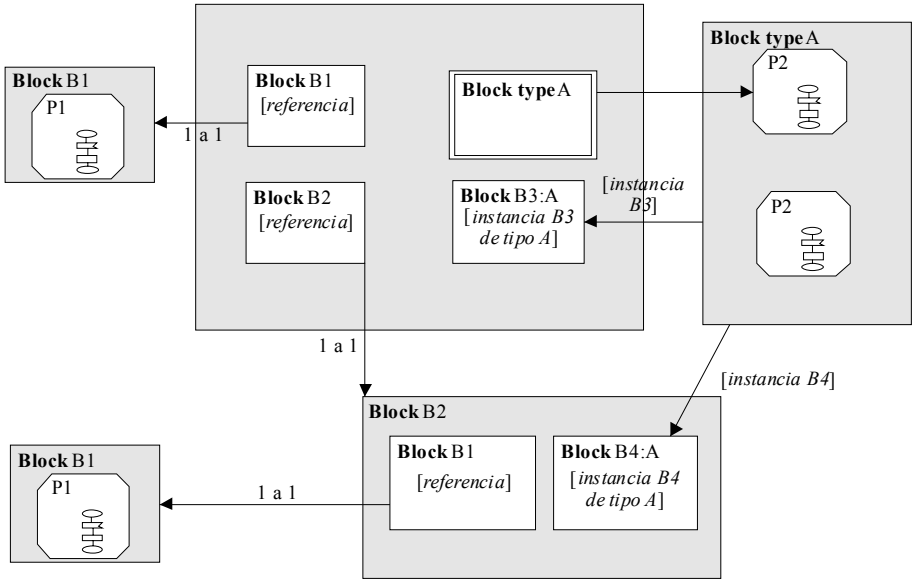


Figura 2.6. Correspondencia entre referencias e instancias de bloque.

2.2.3.3.2. Unidad de ámbito

Algunas entidades son especiales en el sentido que permiten especificar otras entidades dentro de ellas; se pueden especificar procesos y otras entidades dentro de bloques excepto sistemas. Al contexto en el cual se define una entidad se le denomina unidad de ámbito y las unidades de ámbito también se agrupan en clases bajo las siguientes reglas:

- Las entidades que pertenecen a diferentes clases pueden tener el mismo nombre.
- Las entidades pertenecientes a la misma clase pueden tener el mismo nombre si se especifican en diferentes unidades de ámbito.

Estas reglas se ilustran en la Figura 2.6, en donde se definen bloques en diferentes unidades de ámbito.

2.2.3.4. Especificaciones locales y remotas

Los ejemplos presentados en las Figuras 2.4 y 2.5 utilizan un estilo de representación basado en especificaciones locales, en el que cada nivel de abstracción muestra en su interior los detalles del siguiente nivel. Dicha representación impide obtener una visión global del sistema y dificulta la identificación de los diferentes niveles de abstracción al concentrar excesiva información en un sólo diagrama. En SDL, lo anterior se evita utilizando especificaciones remotas[6], las cuales utilizan un símbolo para referirse a:

- Una especificación realizada en alguna otra parte del sistema (símbolo de referencia).
- Una instancia a una especificación realizada en alguna otra parte del sistema.

Un símbolo de referencia identifica con un nombre a una única especificación remota y es el único símbolo que puede hacer referencia a dicha especificación (correspondencia 1 a 1). Sin embargo, una instancia se refiere a una realización concreta de dicha especificación remota y es independiente de otra realización de la misma especificación, aún en la misma unidad de ámbito y tiene correspondencia 1 a muchos.

[6] Formalmente el estílo de representación local no exíste en SDL.

Las especificaciones remotas de las cuales se pueden obtener instancias se denominan tipos (*type*), los cuales se ilustran en la Figura 2.6, en donde los bloques B1, B2 y la subestructura de bloque B1, definida dentro del bloque B2, son especificaciones remotas, mientras que B3 y B4 son instancias del bloque tipo A.

2.3. Datos

El modelo de datos utilizado por SDL se basa en ACT ONE (incluida en la recomendación Z.100) [20, 27], sin embargo la mayoría de aplicaciones no la soportan totalmente debido a su elevada curva de aprendizaje. SDL puede ser utilizado con el lenguaje de descripción de datos ASN.1 (*Abstract Sintaxis Notation One*) [22], recogida en la recomendación Z.105 [21], el cual es soportado totalmente por las herramientas existentes [URL5].

2.3.1. Conceptos generales

Los sistemas SDL manipulan los datos en los procesos durante la transición entre estados. Los conceptos requeridos para comprender las definiciones y manipular los datos son los mismos que utilizan los lenguajes de programación tradicionales, variando en terminología:

- *Valor*: Es el concepto básico utilizado para evaluar expresiones, tomar decisiones, etc. Cada valor pertenece a un tipo de dato específico.
- *Variables*: Contienen valores de un tipo de dato específico y una vez inicializados pueden ser accedidos o modificados. Las variables con tipos de datos definidos por el usuario que deben ser inicializadas, de lo contrario se inician por defecto con un valor igual a *DEFAULT* y no pueden ser accedidas.
- *Expresiones*: Construcciones sintácticas formada por variables, durante la evaluación de la expresión cada variable es sustituida por su valor actual.
- *Literales (literals)*: Son los nombres que denotan valores específicos (‘1’, ‘2’, enteros, ‘A’, ‘B’, caracteres, *HIGH*, *LOW*, niveles lógicos binarios, etc.).
- *Operadores*: Son procedimientos (*procedure*) que al ser referenciados, como resultado producen un valor a partir de una lista de valores (*argumentos*).
- *Sort y tipo de dato*: Un *sort* es un conjunto de valores, mientras que un tipo de dato es una colección de *sorts* definido en un determinado ámbito.
- *Generadores*: Un generador es una construcción que permite definir nuevos tipos de datos parametrizados, es decir, utilizan parámetros. Esto permite crear tipos de datos ligeramente genéricos.
- *Sinónimos*: Permiten utilizar un nombre para referirse a un tipo de valor (constantes).
- *Tipos renombrados (syntype)*: Permiten asignar otro nombre a un tipo de dato ya definido y/o restringir el rango de valores que puede tomar.
- *Estructuras (struct)*: Similares al lenguaje C, permiten definir nuevos tipos de datos más complejos en función a otros ya existentes.
- *Newtype-EndNewtype*: Son las palabras reservadas entre las que se incluyen las sentencias que definen un nuevo tipo de datos.

Para una descripción más detallada del modelo de datos consultar [1, 15, 20, 24, 26, URL5].

2.3.2. Tipos de datos predefinidos

El estándar SDL cuenta con los siguientes tipos de datos predefinidos:

- *Boolean*: Dispone de dos valores, *true* o *false*, y permite realizar operaciones lógicas.
- *Integer*: Este tipo de datos puede tomar valores enteros con signo, el alcance para este tipo de datos es infinito.
- *Real*: Toma valores de punto flotante, alcance infinito.
- *Natural*: Este tipo de dato puede tomar valores enteros positivos, es un tipo derivado de *Integer*, alcance infinito.
- *Character*: Los valores permitidos son todos los pertenecientes al alfabeto internacional CCITT número 5 del código ASCII [1, 4, 15, 20, 24, 25, 26, URL1].
- *CharString*: Derivado del tipo *character* que permite construir cadenas de caracteres.
- *Time y Duration*: Para controlar el funcionamiento de los temporizadores (*timers*).
- *PID*: Define los valores que identifican las instancias de los procesos, el único valor que se puede asignar explícitamente es *NULL*, o de forma indirecta por medio de las variables implícitas de los procesos: *self*, *sender*, *parent* y *offspring*.

2.3.3. Tipos de datos predefinidos ASN.1

El estándar SDL también cuenta con los siguientes tipos de datos ASN.1 predefinidos:

- *Bit*: Parecido al tipo de dato *boolean*, sólo puede tomar dos valores, 0 y 1.
- *BitString*: Permite construir cadenas de bits; a diferencia de la mayoría de los tipos *String* en SDL, el primer elemento de un *BitString* inicia en 0 y no en 1.
- *Octet*: Permite representar valores binarios, octales o hexadecimales de 8 bits, o bien representar un entero entre 0 y 255.
- *OctetString*: Permite construir cadenas de octetos y al igual que *BitString*, el índice inicia en 0.

2.3.4. Generadores predefinidos

Existen tres generadores (*generator*) predefinidos que permiten construir nuevos tipos de datos a partir de los existentes.

- *String*: Se utiliza para definir listas (*list*) de valores de cualquier tipo. Uno de los tipos derivados de este generador es *CharString* y su declaración es String(type TipoElemento, literal EmptyString).
- *Array*: Este generador toma dos argumentos, el primero se usa como índice de búsqueda y el segundo corresponde al tipo de dato que se va almacenar. Los operadores disponibles permiten manipular matrices de una sola dimensión. Su índice puede iniciar con cualquier valor y su declaración es Array(type TipoIndice, type TipoElemento).
- *PowerSet*: Genera tipos de datos que pueden almacenar un conjunto de valores. su formato de declaración es: PowerSet(type TipoElemento).

2.3.5. Declaración de variables

La declaración de variables solamente puede ser llevada a cabo dentro de los procesos en el interior de un símbolo de texto como se ilustra en la Figura 2.7.

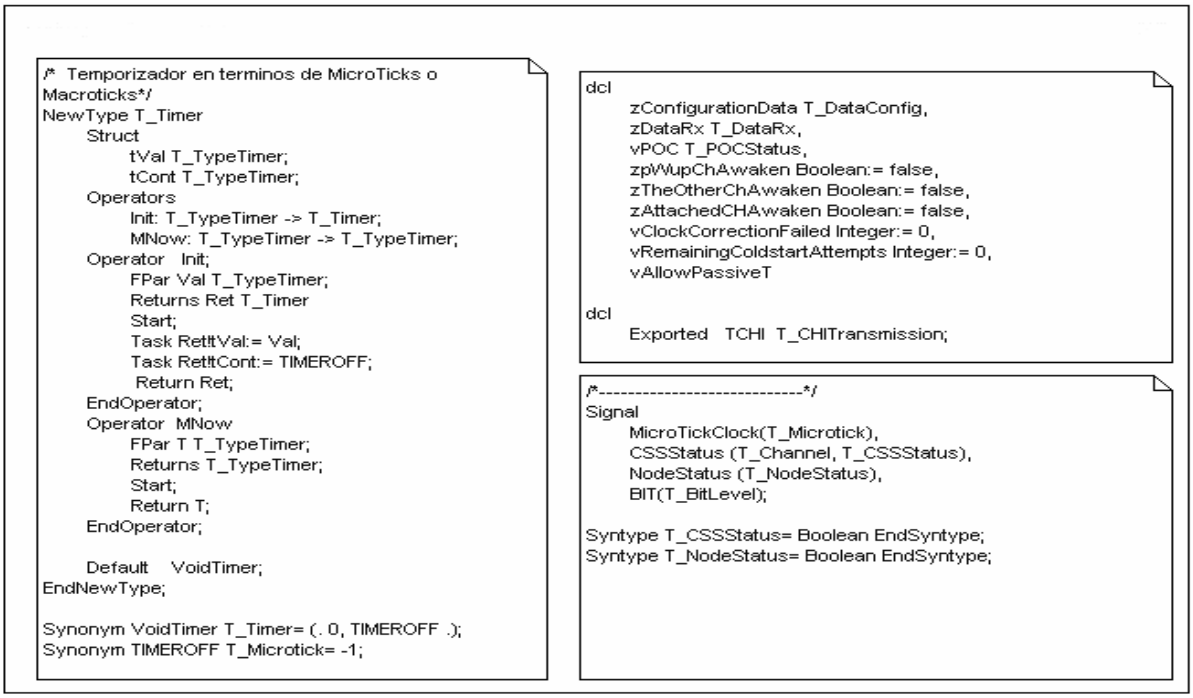


Figura 2.7. Declaración de variables y tipos de datos.

Se usa la palabra reservada *dcl* seguida del nombre de la variable y el tipo de datos al que pertenece. La variable es local al proceso y es accesible sólo en la unidad de ámbito en la que se le declara. El uso de variables remotas permite acceder a las variables que se ubican en diferentes unidades de ámbito incluso en distintos niveles de abstracción. Para ello se hace uso de las construcciones *Import* y *Export* en los procesos receptor y emisor respectivamente, en este orden las señales serán declaradas como *Imported* y *Exported* anteponiéndola a la declaración de la variable. Finalmente en el nivel superior de abstracción deseado se debe realizar la declaración con la construcción *remote*.

2.3.6. Declaración de nuevos tipos de datos

Los tipos de datos se pueden definir en cualquier parte de la especificación del sistema. Como SDL es un lenguaje orientado a objetos, todos los tipos de datos pueden ser utilizados para definir nuevos tipos, los cuales heredan las propiedades de los super tipos y se convierten en tipos especializados. Para realizar declaraciones de nuevos tipos se usa la siguiente estructura (Figura 2.7):

```
Newtype TypeName
        Cuerpo de la declaración
EndNewtype;
```

2.4. Descripción estática de un sistema

Mediante el uso de un conjunto de elementos SDL, la descripción estática descompone un sistema en varias partes atendiendo a sus diferentes funciones. Cada una de estas partes es susceptible de ser dividida en unidades más sencillas hasta alcanzar el nivel de detalle deseado. Las unidades del último nivel contendrán la descripción del comportamiento del sistema.

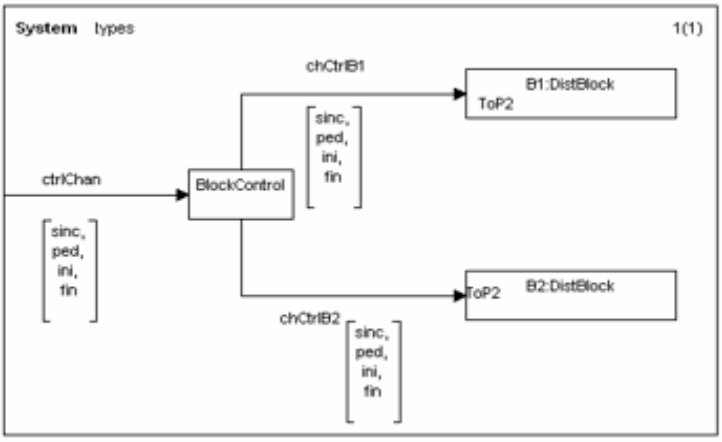


Figura 2.8. Estructura de bloques de un sistema SDL.

2.4.1. Sistema

Un sistema se identifica con el nombre que acompaña a la palabra reservada *system* en la esquina superior izquierda del símbolo que lo representa. Se compone de uno o más bloques que se comunican entre sí y con el entorno mediante señales (Figura 2.8).

2.4.2. Bloques

Los bloques son entidades que permiten dividir la especificación de un sistema en unidades más sencillas, obteniendo diferentes niveles de abstracción. Pueden contener en su interior la especificación de más bloques o procesos como se muestra en las Figuras 2.5 y 2.6.

Un bloque se identifica con el nombre que acompaña a la palabra reservada *block* si hace referencia a una especificación remota o *Block Type* si se refiere a una instancia tipo block. Las instancias se pueden obtener de forma individual como son B1 y B2 de la Figura 2.8 o realizando una generación múltiple de una sola vez, este grupo de instancias es generado de forma estática durante la creación del sistema y por lo tanto no son identificables, es decir, no se puede utilizar un índice para referirse a un bloque dentro del grupo. Si en la instancia al tipo bloque B1 de la Figura 2.8 se declara B1(4):DistBlock, se crearían cuatro instancias de él. Así, aunque el bloque B1 agrupa un conjunto de instancias del tipo bloque DistBlock, el n-ésimo bloque del conjunto no puede ser identificado como B1(n). Esto plantea un problema cuando se pretende enviar una señal a un bloque determinado dentro del conjunto, ya que todos los bloques están conectados al mismo canal.

2.4.3. Procesos

Los procesos contienen la descripción del comportamiento del sistema y se declaran dentro de los bloques; pueden ser definidos de forma remota y utilizados mediante una referencia o se pueden obtener a partir de un tipo proceso (*Process Type*) mediante la declaración de instancias.

En la Figura 2.9 se muestra un tipo bloque DistBlock, el cual contiene instancias a tipo proceso, p11 y p22, así como un proceso declarado de forma remota (proceso ctrl).

A cada proceso o tipo proceso se le asigna un nombre anteponiéndolo a la palabra reservada Process o Type Process. Sin embargo, para identificar procesos se usan cuatro expresiones predefinidas de tipo PID que cada proceso contiene desde el momento en que es creado:

- *Self*: Dirección del proceso mismo.
- *Sender*: Dirección del proceso o instancia del cual proviene la última señal recibida.

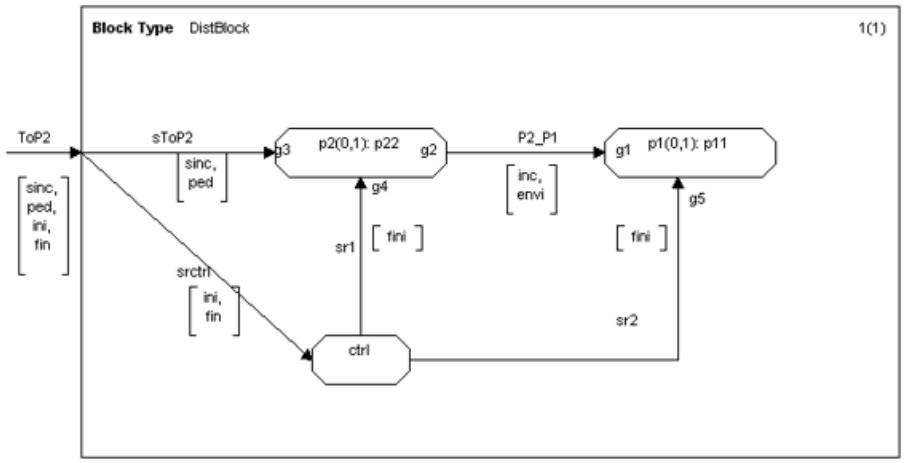


Figura 2.9. Especificación de procesos remotos e instanciados.

- *Offspring*: Dirección del último proceso que fue creado.
- *Parent*: Dirección del proceso que lo creó o proceso padre.

Esto es debido a que los procesos pueden ser creados de forma dinámica, permitiendo la generación de múltiples instancias de un proceso con la misma especificación y el mismo nombre. Es posible especificar el número de instancias que pueden ser creados en tiempo de inicio del sistema y el máximo número de instancias que pueden ser creadas de forma dinámica; en caso de omitir estos valores, por defecto serán 1 e infinito respectivamente. La Figura 2.10 muestra la especificación remota de un tipo proceso.

2.4.4. Canales

Los canales son el medio de intercambio de información entre niveles superiores (bloques) introduciendo un retardo no-determinístico (retardo aleatorio), así mismo es posible configurarlo para que la información fluya sin retardo (determinístico). En cualquier tipo de estos, el orden de las señales siempre se mantiene, es decir, aunque el retardo sufrido por varias señales en un mismo canal pueda ser diferente, una señal no puede adelantar a otra en un mismo canal en un instante de tiempo anterior. Sin embargo, las señales enviadas entre bloques por diferentes canales pueden llegar en un orden arbitrario. El flujo de señales sobre un canal puede ser en un sentido (canal unidireccional) o en ambos (canal bidireccional). En la Figura 2.8 se ilustra el uso de canales unidireccionales, las señales transportadas están especificadas entre corchetes (*SignalList*)

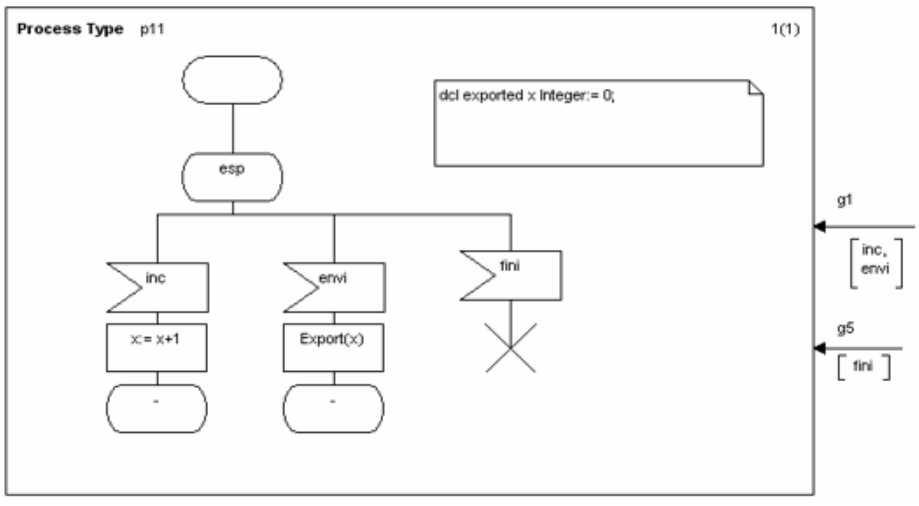


Figura 2.10. Especificación remota tipo procesos.

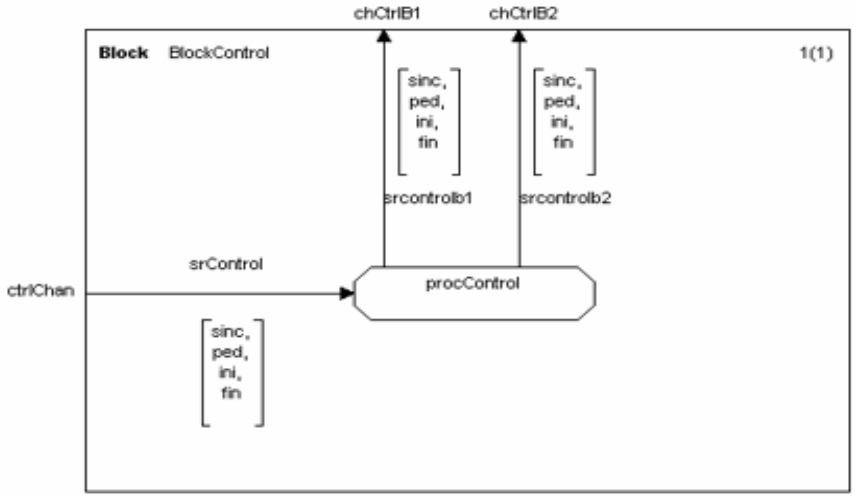


Figura 2.11. Comunicación de procesos, bloques y tipos por medio de compuertas.

2.4.5. Compuertas

Las compuertas (*gate*) permiten establecer los puntos de conexión en la frontera de un bloque si éste está compuesto por más bloques (subestructura de bloque) y con rutas de señal si está compuesto de procesos; ambos casos corresponden a la especificación de tipos como: tipo bloque o tipo proceso. En la Figura 2.9, los procesos ctrl y p2 se conectan al exterior por medio de la compuerta Top2. La Figura 2.11 muestra al proceso procControl conectado a tres compuertas (ctrlChan, chCtrlB1 y chCtrlB2).

2.4.6. Servicios

Los servicios se utilizan para brindar a la especificación un nivel de abstracción adicional cuando la complejidad de los procesos es elevada. Sus funciones deben estar bien definidas y diferenciadas debido a las siguientes restricciones para establecer un nivel de servicio:

- A diferencia de los procesos, los servicios (*service*) que forman parte de una especificación no se ejecutan concurrentemente (no hay paralelismo).
- Comparten la cola de entrada (FIFO, *First In First Out*).
- Cada servicio en un proceso debe tener un conjunto de señales de entrada distinto.
- Para establecer comunicación entre servicios y asegurar que se cumpla la tercera restricción, se debe declarar un conjunto adicional de señales.

La Figura 2.12 muestra una especificación para un nivel de servicios.

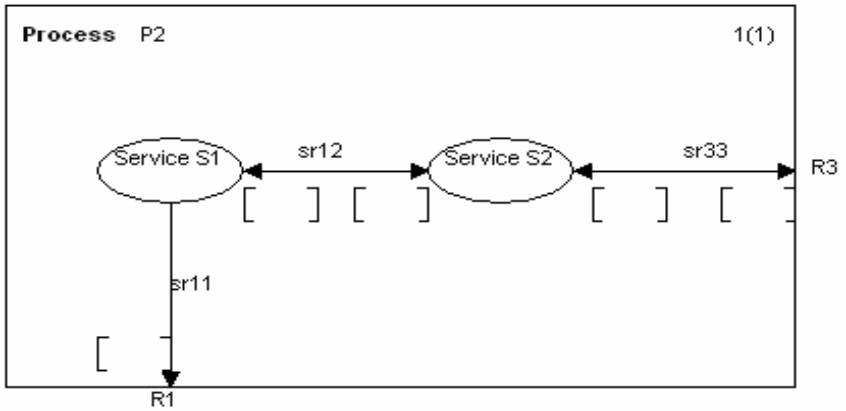


Figura 2.12. Especificación de un nivel de servicio.

2.4.7. Procedimientos

Un procedimiento (*procedure*) es una parte parametrizada de un proceso que dispone de un ámbito independiente para definir nuevos estados, variables, etc. Cuando se llama a un procedimiento se pasa el control de ejecución del proceso y se le devuelve hasta que el procedimiento haya concluido (con su respectivo símbolo de retorno). Así mismo, es posible utilizar los tipos de datos, variables y señales que hayan sido declaradas en los ámbitos superiores al que pertenece.

2.4.8. Rutas de señal

Mediante las rutas de señal (*signal route*) se establece la comunicación a nivel de procesos y/o servicios. Están sujetas a las mismas consideraciones que los canales (*channels*), con la única diferencia que con las rutas de señal siempre se establece una comunicación determinística, es decir que las señales fluyen sin retardo (no consumen tiempo).

2.4.9. Señales

Las señales (*signals)* son la base para establecer comunicación entre procesos, éstas llevan el identificador de proceso (PID, *process identifier*) del proceso emisor y pueden transportar valores (parámetros). En el caso de direccionamiento implícito, adicionalmente llevan el PID del proceso receptor. El esquema de transmisión es asíncrono entre un emisor y un receptor.

2.4.10. Variables y tipos de datos

La declaración de nuevos tipos de datos se puede realizar en cualquier nivel de la especificación del sistema. En los procesos y los servicios es posible declarar variables que serán utilizadas en la definición del comportamiento del sistema. Como la declaración de variables sólo se puede realizar en el ámbito de los procesos, en SDL no está permitido el uso de variables globales como elementos de comunicación, lo que suele ser una fuente importante de errores en la programación tradicional.

2.4.11. Símbolos gráficos estáticos

Cada uno de los elementos descritos posee una definición textual (SDL-PR) y una representación gráfica (SDL-GR). En la Tabla 2.1 se resumen los símbolos gráficos que se utilizan en la descripción de la parte estática de una especificación SDL.

Tabla 2.1. Símbolos SDL para la parte estática.

Símbolo	Descripción	Símbolo	Descripción
Text	Tipos, variables, señales.	System	Sistema
Package	Especificación de librerías	system type	Tipo sistema
Block	Bloques, subestructuras.	block type	Tipo bloque, tipo subestructura
Process	Procesos	process type	Tipo proceso
Service	Servicios	Procedure	Procedimiento, referencia
Nombre [señal1, señal2,..]	Canal sin retardo	Nombre [señal1, señal2,..]	Canal con retardo

2.5. Descripción dinámica de un sistema

Los procesos tienen información que describe cómo interactúa el sistema con su entorno y cómo responde a las situaciones de operación para las que fue diseñado. Esta información está descrita por las máquinas de estado que cada proceso tiene asociado [URL5].

2.5.1. Comunicación entre procesos

Cada instancia de un proceso tiene asociada una cola, que sigue el principio FIFO, en la que se van almacenado las señales recibidas (Figura 2.13). Una señal puede llegar a un proceso en cualquier instante, pero sólo ocasionará un cambio de estado si ésta pertenece al alfabeto de entrada (*SignalSet*), en otro caso la señal será consumida sin que se produzca algún cambio de estado.

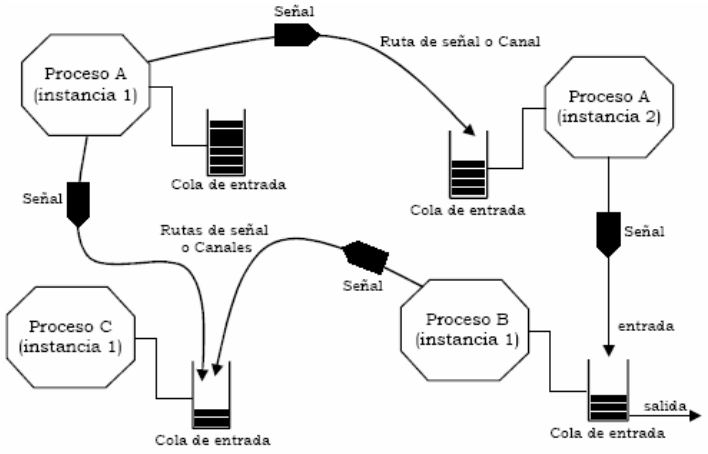


Figura 2.13. Comunicación entre procesos.

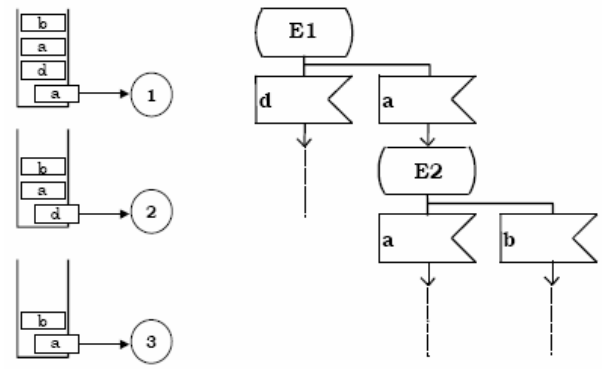


Figura 2.14. Ejemplo de una transición implícita.

2.5.1.1. Transiciones

En cada instante, un proceso se encuentra estable en un determinado estado o cambiando de un estado a otro. Estos cambios entre estados se llaman transiciones y se producen cuando recibe una señal que pertenece al alfabeto de entrada. Es en las transiciones donde realizan las acciones que caracterizan el comportamiento del sistema. Cuando una señal es consumida sin ocasionar cambio de estado alguno, se dice que ocurre una transición implícita (bajo las condiciones descritas). En la Figura 2.14 se presenta un ejemplo que describe dicha situación; la cola de entrada tiene cuatro señales que han llegado en el orden `a`, `d`, `a`, `b`. En el estado `E1`, la señal `a` es consumida y dispara una transición hacia el estado `E2`, en este estado la señal `d` no dispara ninguna transición y es consumida pero dispara una transición implícita. La salida del estado `E2` es llevada a cabo cuando la siguiente señal es consumida.

La comunicación entre procesos se realiza de forma asíncrona, lo que significa que al enviar una señal, un proceso continúa su ejecución sin esperar confirmación de que ha sido recibida por el proceso destino

2.5.1.2. Elementos de las transiciones

Las transiciones están compuestas por un estado inicial, un estado final y un cuerpo de acción. Cada estado en el que se puede encontrar un proceso tiene asignado un nombre que se incluye en los símbolos de estado (*state*). Los símbolos con el mismo nombre, aún cuando estén colocados en diferentes partes del proceso, se refieren al mismo estado. Las transiciones enlazan los estados adecuados para que el comportamiento del proceso evolucione de acuerdo a las especificaciones.

Si ocurren transiciones idénticas de dos o más estados diferentes, es posible agruparlos en un mismo símbolo con los nombres de los estados involucrados. En caso de que todos los estados estén involucrados, se utiliza el signo '*' para especificar excepciones de los estados que no se vayan a involucrar, por ejemplo `*(estado1, estado2, etc.)`. Si se quiere indicar que el estado siguiente es el mismo, puede usarse el signo menos '-' como se ilustra en la Figura 2.15.

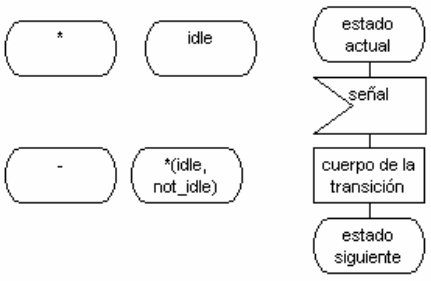


Figura 2.15. Elementos de una transición.

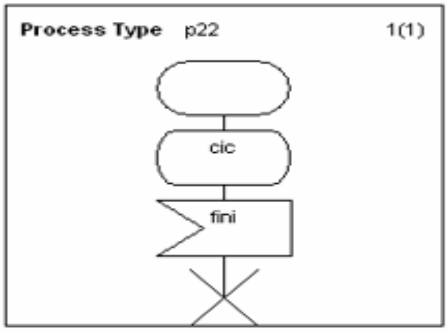


Figura 2.16. Terminación de un proceso.

El cuerpo de la transición está formado por el símbolo de estrada (*input*) para que la transición (explícita) pueda ocurrir, seguida por un conjunto de símbolos en cualquier orden de salida (*output*), decisiones o tareas.

2.5.2. Descripción de procesos

A continuación se presentan las principales características de los símbolos utilizados para realizar la especificación dinámica.

2.5.2.1. Tiempo de vida de un proceso

Cuando la instancia de un proceso ha sido creada, su ejecución comienza en la transición que sigue al símbolo de inicio (*start*), esta transición inicial se utiliza para realizar la tarea de inicialización y a continuación el proceso se sitúa en el primer estado del diagrama. Para finalizar la operación de un proceso, así como la existencia del mismo, el flujo de ejecución lo llevará al símbolo de terminación (*end*) (Figura 2.16).

2.5.2.2. Señales de entrada/salida

Las señales de entrada/salida (E/S) son conocidas como símbolos de control de señales. Un símbolo de estrada (*input*) se ejecuta cuando la señal asociada a dicho símbolo se encuentra presente en la cola de entrada de dicho proceso, procediendo a consumir dicha señal mediante la obtención de la información explícita trasportada por sus parámetros; respecto a la información explícita de la señal se obtiene el identificador del proceso que la envió y es actualizada la variable *sender* automáticamente.

El envío de señales se realiza mediante el símbolo de salida (*output*) en el que se indica el nombre de la señal que se quiere enviar y el valor de sus parámetros. Éstas son transportadas sobre rutas de señal y/o canales hasta alcanzar el proceso destino.

Existen dos formas de direccionar una señal (Figura 2.17):

- *Direccionamiento explícito*: Si se desea direccionar a un proceso específico, se utiliza la palabra reservada `To` en el símbolo de salida, seguida de la dirección (*PID*) de la instancia del proceso que identifica únicamente al proceso a direccionar. La dirección puede ser una de las expresiones predefinidas *self*, *sender*, *offspring*, *parent* o una variable de tipo PID. SDL 92-96 permite direccionar a un proceso mediante su nombre, lo cual no es recomendable si existe más de una instancia con el mismo nombre.
- *Direccionamiento implícito*: Cuando existe sólo un proceso que puede recibir una señal, no hay necesidad de especificar la dirección del mismo, basta con poner el nombre de la señal en el símbolo de salida, al haber sólo una dirección posible para enviar esa señal, la dirección del proceso que puede recibirla se asigna de manera implícita.

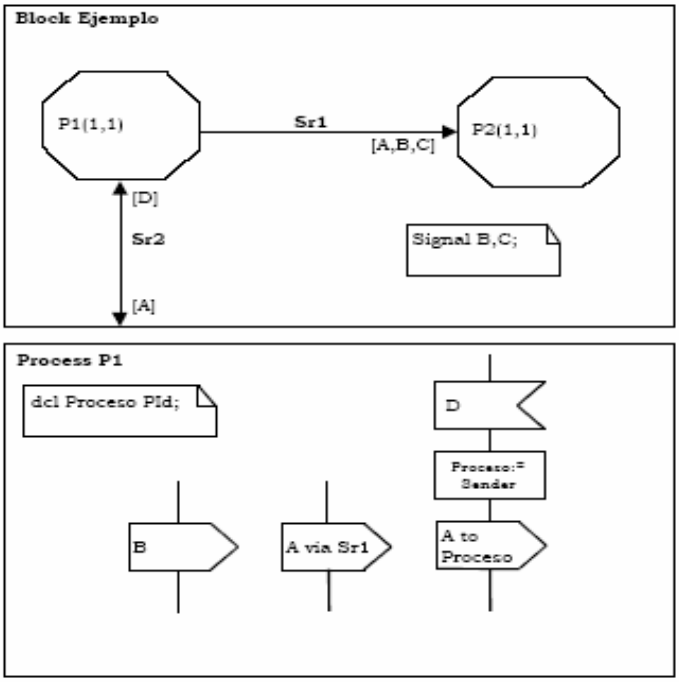


Figura 2.17. Direccionamiento explícito e implícito de una señal.

Si existe una multiplicidad de caminos por la cual la señal puede ser transportada, es decir dos o más procesos conectados a la misma ruta de señal o procesos que están en un ámbito diferente (otro bloque), el envío de la señal se hace con el uso de la palabra reservada Via seguida del nombre de la ruta de señal o compuerta (en caso de tipos). En situaciones complejas es posible el uso de las palabras reservadas Via y To.

2.5.2.3. Tareas

El símbolo de tarea (*Task*) se utiliza para inicializar variables y/o modificarlas, la sentencia de asignación es similar a la utilizada en el lenguaje de programación PASCAL; la sentencia de asignación es ':=' como se muestra en la Figura 2.17 en donde a la variable proceso se le asigna el valor del contenido en Sender: Proceso:= Sender.

El símbolo de tarea no sólo permite modificar variables, sino que también permite llamar a procedimientos que pueden retornar algún valor para asignarla a una variable. Si es necesario expresar más de una tarea en el símbolo, las tareas se separan por comas (este es el caso de la herramienta Cinderella SDL [URL1]).

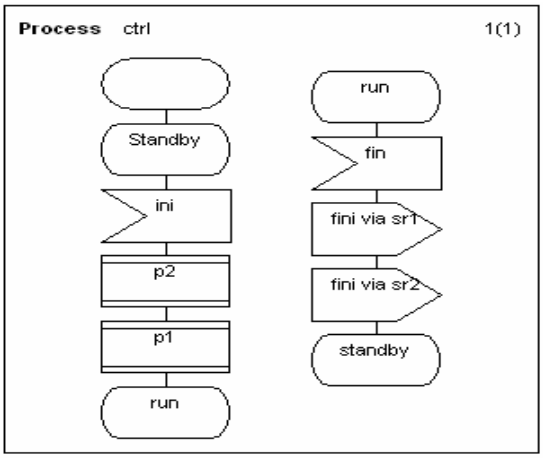


Figura 2.18. Creación dinámica de procesos.

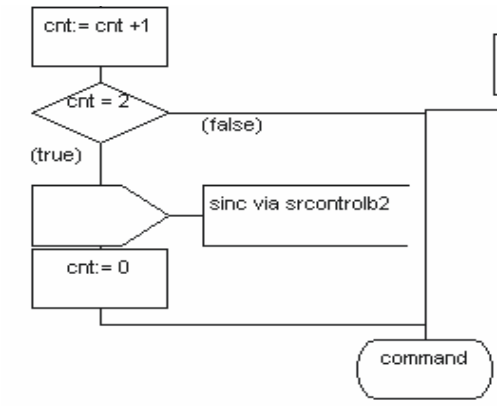


Figura 2.19. Símbolo de decisión dinámica.

2.5.2.4. Creación de procesos

El símbolo de creación de procesos (*Create*) permite crear instancias a procesos de forma dinámica, indicando en el símbolo *Create* el nombre del proceso que se desea instanciar y limitado el máximo número de procesos establecidos en su descripción. La expresión *offspring* contiene la dirección del proceso creado si no se ha excedido el límite establecido, en caso contrario contendrá *NULL* (Figura 2.18).

2.5.2.5. Decisión

El resultado de una expresión construida a partir de variables en el símbolo de decisión (*Decision*) controla o varía el flujo de ejecución del proceso. El número de salidas del símbolo depende de los resultados que se quieran considerar; el resultado de la evaluación de una decisión es una expresión booleana. La Figura 2.19 ilustra este concepto mediante un ejemplo en la que sólo existen dos posibles resultados, falso o verdadero.

La construcción *Select* no dispone de un símbolo gráfico definido, si no que se representa mediante una línea discontinua que agrupa las partes del sistema que se quieren utilizar en una determinada situación.

Esta construcción suele combinarse con el símbolo Tansition Option que permite tomar decisiones como resultado de una expresión, con la diferencia de que Select y Transition Option se realizan de forma estática (parecidas a las instrucciones del precompilador de lenguaje C), permitiendo incluir en el sistema diferentes especificaciones. De esta manera se puede realizar descripciones más genéricas (Figura 2.20).

2.5.2.6. Alteración del orden de las señales en las colas

Cuando llega una señal a la cola del proceso, ésta es consumida inmediatamente; si el estado actual del proceso no contiene asociada una construcción para recibir dicha señal, ésta se perderá o será consumida. Para poder controlar estas situaciones sin que se produzcan transiciones implícitas con pérdida de señales o para alterar el orden de las mismas, SDL dispone de tres operadores:

- *Entrada prioritaria (Priority Input)*: Pone la señal en cuestión al inicio de la cola.
- *Condición de permiso 1 (Enabling Condition)*: Si cumple una condición previa al símbolo de entrada, la señal es consumida y se dispara la transición, en caso contrario no ocurre ninguna de las dos situaciones anteriores.
- *Condición de permiso 2 (Continous Signal)*: Permite iniciar transiciones aún cuando la cola de entrada esté vacía, lo cual se realiza con ayuda de una condición que es evaluada con una expresión booleana; además permite asignar prioridades, mediante la palabra re-

servada *Priority*, si existe más de una condición de permiso y en el caso de que más de una condición se cumpla simultáneamente.

- *Almacenamiento de señales (Save)*: Almacena aquellas señales que se encuentran al inicio de la cola mientras no exista una entrada asociada en el estado actual.

Los símbolos asociados a estas construcciones se presentan en la Tabla 2.2.

2.5.2.7. Llamadas a procedimientos

Un procedimiento puede usar todos los símbolos del nivel superior o de proceso, a excepción de los de inicio y fin, ya que los procedimientos tienen sus propios símbolos para representar estas construcciones. Los procedimientos pueden ser parametrizados: los parámetros por valor se especifican con la palabra reservada In, los parámetros por referencia con In/Out y los valores de retorno con Return.

Para invocar un procedimiento se usa el símbolo de *Procedure Call* y en caso de regresar algún valor, el símbolo de tarea se usa conjuntamente con la palabra reservada Call en su interior, por ejemplo Par:= Call ValorPar(y).

2.5.2.8. Etiquetas

Generalmente es necesario utilizar más de una página para describir un proceso y por ello es útil el uso de etiquetas (*Label*), las cuales permiten expandir el cuerpo de los diagrama de estado si el cuerpo de sus transiciones son muy extensas.

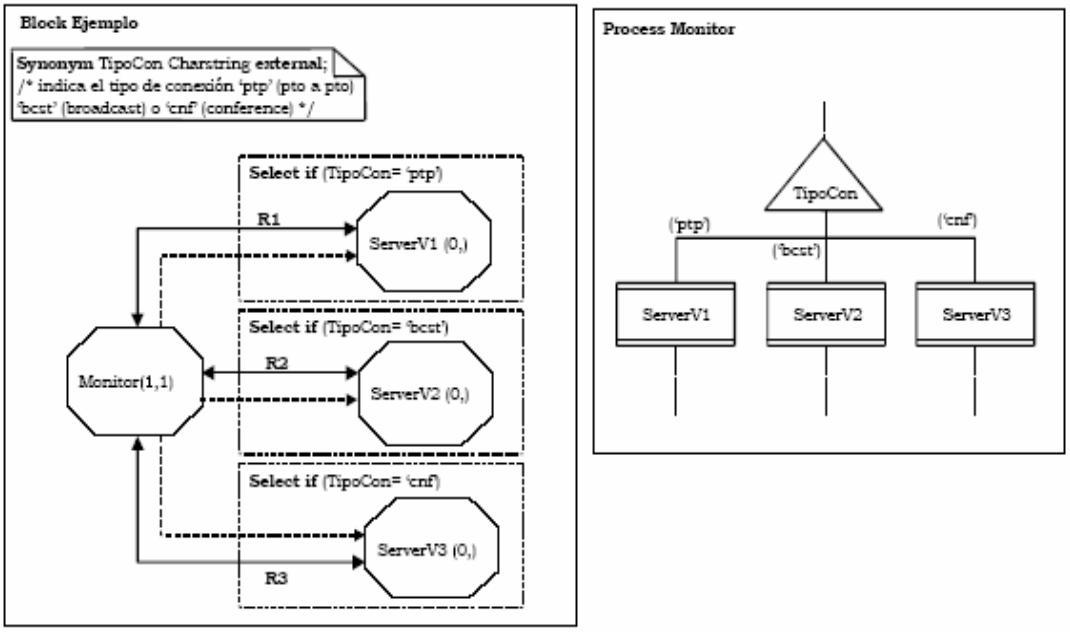
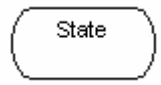
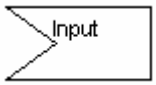
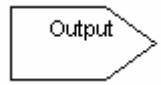
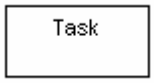
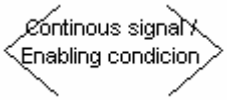
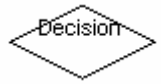
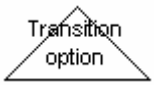
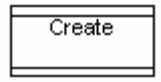
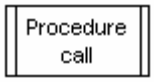
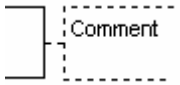


Figura 2.20. Construcción de una decisión estática.

Tabla 2.2. Símbolos SDL para la parte dinámica.

Símbolo	Descripción	Símbolo	Descripción
	Inicio		Terminación
	Estado		Entrada
	Salida		Tarea
	Preservación de señal		Condición de permiso
	Decisión		Transición
	Creación de proceso		Llamada de procedimiento
	Etiqueta		Comentario
	Inicio de procedimiento		Salida de procedimiento

2.5.3. Especificación del tiempo

Los sistemas de telecomunicaciones son generalmente sistemas de tiempo real puesto que deben actuar bajo tiempo limitado. La eficiencia de la representación del tiempo que realiza SDL para la especificación de requerimientos de tiempo real, depende en gran medida de las características del equipo de cómputo a utilizar, principalmente del sistema operativo, velocidad de ejecución del procesador y cantidad de memoria[7].

Una consideración importante es que para las tolerancias de los intervalos de tiempo, SDL no utiliza el segundo como unidad base, sino que ésta está en función del sistema que se esté utilizando; por ello cabe aclarar que SDL no ejecuta las especificaciones formales de sistemas en tiempo real, sino que permite describir y simular el comportamiento de sistemas de tiempo real.

2.5.3.1. Valores del tiempo

Existen dos tipos de datos predefinidos, *Time* y *Duration*, los cuales se utilizan para establecer valores relacionados con el tiempo; el primero se refiere a puntos absolutos en el tiempo, mientras que el segundo almacena intervalos de tiempo.

[7] La herramienta Cinderella SDL recomienda los siguientes requisitos mínimos para su correcta instalación: Sistema Operativo MS Windows 9x/NT/2000/XP, Espacio en disco duro mayor a 60 MB, Velocidad de procesamiento igual o superior a Pentium III 780 MHz, Memoria igual o superior a 128 MB.

Now es una expresión asociada al tipo *Time*, la cual proporciona el tiempo global actual y se usa en la especificación para controlar el paso del tiempo; dado que *Now* devuelve el mismo valor al inicio y final de una transición, se deriva que las transiciones no consumen tiempo.

2.5.3.2. Temporizadores

Por lo general, en los sistemas de telecomunicaciones es de poca utilidad el conocimiento del tiempo global como tal, excepto aquellos casos que lo utilizan para sincronizar acciones. Como estas acciones sincronizadas no se pueden modelar en SDL, la utilidad de los tipos de datos predefinidos *Time* y *Duration*, y la expresión *Now*, potencializan la utilidad de los temporizadores. Un temporizador es un cronómetro que se activa con un tiempo de expiración. La expiración de un temporizador se indica mediante el envío de una señal, con el mismo nombre que el temporizador, a la cola de entrada del proceso que lo activó.

2.5.3.3. Operaciones sobre los temporizadores

La Figura 2.21 resume las operaciones que se pueden realizar sobre los temporizadores:

- *Declarar un temporizador*: Un temporizador se declara, a nivel proceso, utilizando la palabra reservada *Timer* seguida del nombre del temporizador. Se pueden utilizar parámetros para generar instancias del temporizador en cada valor del parámetro; también se puede asignar un valor inicial al temporizador.
- *Desactivar un temporizador*: Para desactivar un temporizador se utiliza la expresión *reset* seguida del nombre del temporizador. Si el temporizador está activo en el momento de recibir esta orden, éste se desactiva; y si ya había expirado, se elimina la señal correspondiente de la cola de entrada. La operación de desactivación se realiza gráficamente en el interior de un símbolo de tarea.
- *Activar un temporizador*: La activación de un temporizador se realiza mediante la expresión *set* seguida del nombre del temporizador y de un valor de tiempo absoluto. Cuando la expresión *Now* alcanza dicho valor, el temporizador expira. Normalmente la activación se realiza de forma relativa al tiempo actual (*Now* + tiempo de activación). La activación de un temporizador que ya ha sido iniciado produce su desactivación automáticamente, lo cual asegura que en la cola de entrada sólo exista una señal por instancia de temporizador. Al igual que en la desactivación, gráficamente se utiliza un símbolo de tarea.

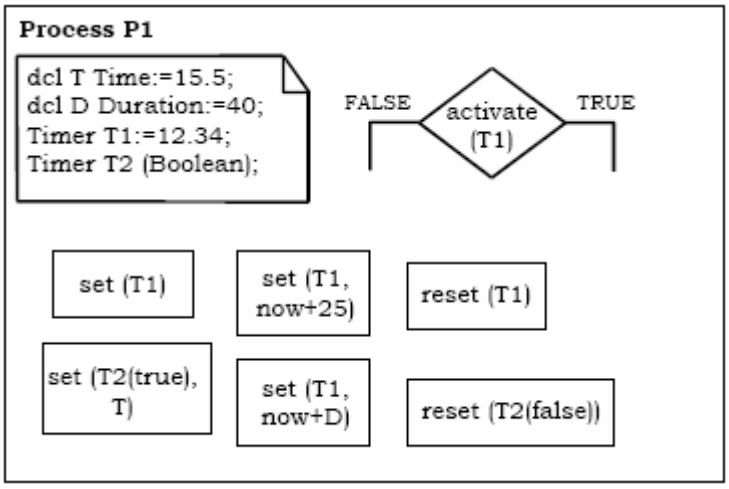


Figura 2.21. Declaración y uso de los temporizadores.

- *Consultar el estado de un temporizador*: Se puede consultar el estado de un temporizador con la expresión *active*, que devuelve una expresión booleana indicando si el temporizador está activo (*true*) o inactivo (*false*).

La Tabla 2.3 muestra el efecto de las operaciones *Set* y *Reset* sobre los temporizadores. Si un temporizador está en estado inactivo y ocurre un evento *Set* (celda 1), el temporizador se activa. Si un evento *Reset* ocurre y el temporizador está activo (celda 4), el temporizador se detiene y pasa al estado inactivo. Una vez que se consume el tiempo del temporizador, se envía una señal, con nombre igual al del temporizador, a la instancia del proceso que la contiene (celda 5).

Si la señal del temporizador está en la cola de entrada del proceso y ocurre un evento *Reset* (celda 7), la señal es removida de la cola y pasa al estado inactivo; mientras que si ocurre un evento *Set* (celda 6), la señal del temporizador es removida de la cola y pasa al estado activo con la duración de tiempo correspondiente.

Tabla 2.3. Funcionamiento de los temporizadores.

Estado	**Eventos**		
	Set	***Reset***	**Tiempo consumido**
Inactivo (conteo desactivado)	1. Activo	2. Inactivo	
Activo (Contando)	3. Desactiva, reinicia el conteo	4. Inactivo	5. Pone el temporizador en la cola
Temporizador en cola	6. Remueve el temporizador de la cola, reinicia el conteo	7. Remueve el temporizador de la cola, inactivo	

3. Protocolo de comunicaciones Flexray

El protocolo de comunicaciones automotrices Flexray está basado en una arquitectura de disparo por tiempo (TTA, *Time Triggered Architecture*) con tiempos de respuesta conocidos. Flexray es un sistema de comunicaciones, síncrona y asíncrona, serial de alta velocidad (hasta 10 Mbps), lo que permite establecer comunicación prácticamente en tiempo real. Flexray es un sistema tolerante a fallos debido a su capacidad de soportar dos canales y el uso de algoritmos de sincronización FTM (*Fault Tolerarant Midpoit Algorithm*). Las diferentes topologías de red soportadas se basan en la conexión punto a punto, lo que permite diseñar sistemas de control automotriz distribuido de tiempo real y alto rendimiento del tipo *x-by-wire* tales como dirección (*steer-by-wire*) y frenado (*brake-by-wire*) [3, 10].

Las tareas realizadas por el protocolo Flexray se basan en un esquema de acceso estático (TDMA, *Time Division Multiple Access*) y/o dinámico (FTDMA, *Flexible TDMA*), y no en base a eventos como lo hace el protocolo CAN [33], esto se lleva a cabo mediante su propio controlador de comunicaciones independiente del MCU (*Microcontroller Unit*).

Flexray no sólo se aplica a la industria automotriz, sino también en la industria aeroespacial y ferroviaria. El núcleo de desarrolladores del consorcio Flexray está compuesto por: BMW AG, DaimlerChrysler AG, Freescale Halbleiter Deutschland GmbH, General Motors Corporation, Philips GmbH, Robert Bosch GmbH y Volkswagen AG [URL3].

3.1. Conceptos básicos

A continuación se definen algunos de los conceptos manejados por el protocolo Flexray:

- *Nodo*: Entidad lógica conectada a una red Flexray, capaz de transmitir y/o recibir tramas de datos.
- *Administrador de bus (BD, Bus Driver)*: Componente electrónico, formado de un trasmisor y un receptor, que conecta al controlador Flexray a un bus de comunicaciones.
- *Cluster*: Sistema de comunicaciones formado de múltiples nodos conectados al menos a un canal.
- *Guardián de bus (BG, Bus Guardian)*: Componente electrónico que protege al bus de comunicaciones de interferencias causadas por trasmisiones que no estén sincronizadas al esquema de comunicaciones del *cluster*.
- *Nodo de arranque en frío (Coldstart Node)*: Nodo capaz de iniciar el proceso de comunicación en el *cluster* mediante el envío de tramas de inicio de carga del *cluster* (*startup frames*).

- *Controlador de comunicaciones (CC, Communication Controller)*: Componente electrónico en un nodo que es responsable de implementar los aspectos del protocolo de un sistema de comunicaciones Flexray.
- *Host*: Parte de una unidad de control electrónico (ECU, *Electronic Control Unit*) donde se ejecuta el software de aplicación, separada del motor o núcleo del protocolo Flexray e interfazada por medio del CHI (*Controller Host Interface*), cabe señalar que el *host* no es parte de la especificación Flexray.

3.2. Arquitectura de nodo

Cada ECU consiste de cinco sub-componentes: host, CC, BG, BD y suministro de energía (*power supply*), como muestra la Figura 3.1. Un nodo puede estar compuesto de un sólo *host* conectado al BG mediante una interfaz SPI; no se requiere un BD si el nodo sólo soporta un canal de comunicaciones.

El presente trabajo de tesis se centra únicamente en el controlador CC, en el que se describen los medios y procesos necesarios para establecer comunicación entre los diferentes nodos de un *cluster*.

3.3. Arquitectura de protocolos Flexray

A continuación se mencionan los niveles OSI que describe el protocolo Flexray [11-14], con base en el estándar IEEE 802.1x [URL6] y adoptado por la ISO bajo los nombres ISO/IEC 8802.1 e ISO/IEC 8802.2. Dichos niveles son la capa física y la capa de enlace de datos, esta última dividida en dos subcapas por la IEEE, control de enlace lógico (LLC, *Logic Link Control*) y control de acceso al medio (MAC, *Medium Access Control*).

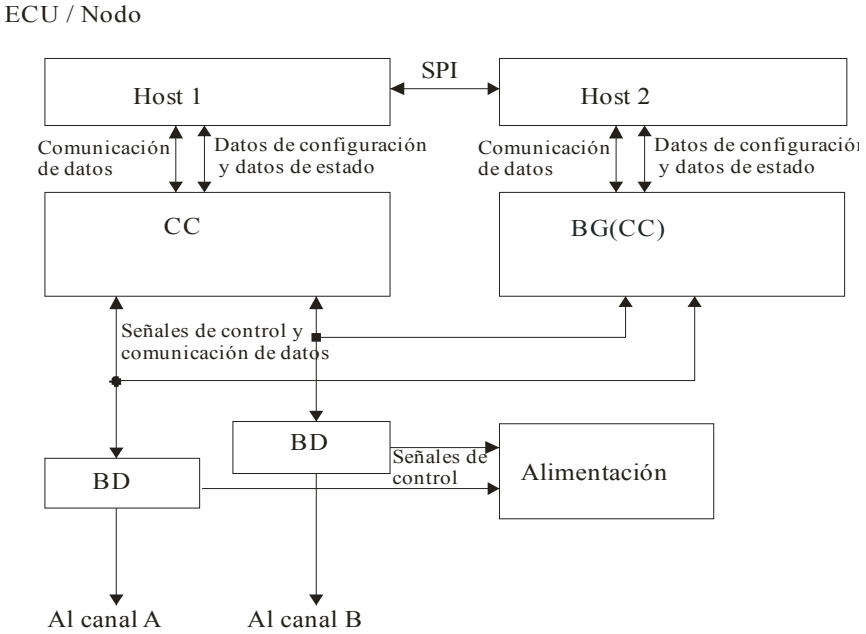


Figura 3.1. Arquitectura de nodo.

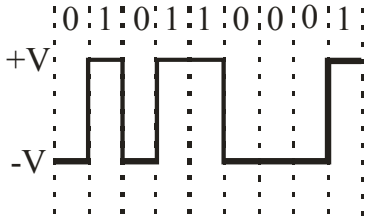


Figura 3.2. Ejemplo de codificación NRZ de un CE.

3.3.1. Capa física

La capa física define la forma en que se transmiten las señales de datos desde el nodo al bus y viceversa, considera los aspectos del medio físico para dicha transmisión haciendo referencia a los niveles de señal, representación, sincronización, tiempo de bit, topología, etc.

El protocolo Flexray especifica el uso de un BG en la capa física para realizar la detección de errores cometidos por el CC en el dominio del tiempo.

El diseño de una red Flexray es independiente de la topología, sin embargo, todos los nodos en el *cluster* deben tener la misma velocidad de transferencia de datos (2.5, 5 ó 10 Mbps), la cual puede variar de un *cluster* a otro.

La especificación del protocolo Flexray describe las características de los componentes que se utilizan en ésta capa con base en el estándar RS 485 [URL7].

3.3.1.1. Representación y sincronización de bits

Un nodo Flexray utiliza el método NRZ (*Non Return To Zero*) para codificar y decodificar la cadena de bits llamada CE (*Communication Element*), como muestra la Figura 3.2. Dado que el protocolo Flexray es independiente de la capa física fundamental, no necesita implementaciones adicionales como es la inserción de bits que realiza CAN para la sincronización de bits [2].

El CC se encarga de la sincronización de bits, la cual se lleva a cabo en cada flanco de caída del BSS (*Byte Start Sequence*) como se muestra en la Figura 3.3.

El nodo Flexray realiza un muestreo por cada bit de duración nominal *gdBit*, que puede ser de 100, 200 ó 400 ns, dependiendo de la velocidad de transferencia de datos. Por lo tanto, se tiene que el tiempo de muestreo nominal es:

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

Se denomina *BitStrobing* al punto de muestreo (*cStrobeOffset = 5*) en el que es tomado el valor del bit. Para eliminar posibles rebotes y evitar tomar el valor de bit antes o después del tiempo nominal, se considera una tolerancia de $3*T_{nom}$ en caso de un adelanto y $4*T_{nom}$ para un retardo, con lo cual se asegura tomar el valor del bit de forma correcta.

3.3.1.2. Enlace físico de comunicación

El enlace físico de comunicación consiste en una interconexión entre nodos, por medio de la cual se transporta la información. Todos los nodos conectados a este enlace (no hay repetidores, acopladores de estrella, pasarelas (*gateway*), etc.) comparten las señales transportadas en él. Un ejemplo de un enlace físico de comunicación puede ser una topología de bus, una conexión punto a punto entre un nodo y un acoplador de estrella. Un canal puede ser construido mediante la combinación de uno o más enlaces físicos de comunicación conectados con acopladores de estrella.

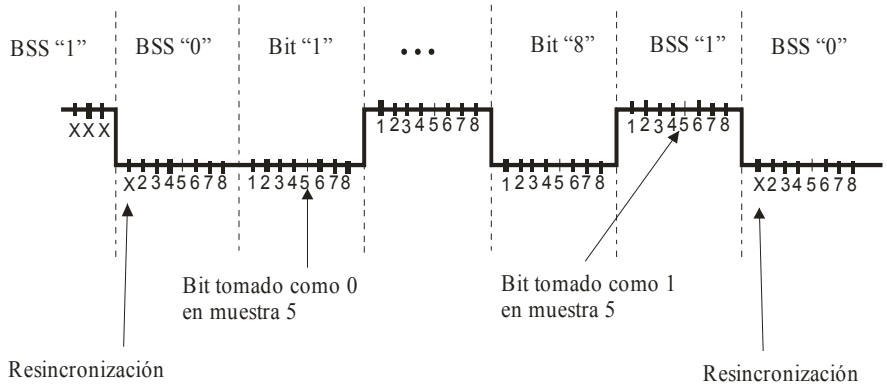


Figura 3.3. Sincronización, muestreo y elección (*BitStrobing*) de bits.

3.3.1.3. Especificaciones eléctricas

El protocolo de comunicaciones Flexray está diseñado para funcionar a una velocidad de transferencia de datos máxima de 10 Mbps, por lo que el tiempo nominal de bit (*gdBit*) mínimo es de 100 ns. El bus deberá soportar las características que a continuación se mencionan.

3.3.1.3.1. Parámetros de cableado

La Tabla 3.1 muestra las características para cableado de los buses Flexray, estos pueden o no estar protegidos contra interferencia.

Tabla 3.1. Características de los cables usados en el bus.

Nombre	Descripción	Mín.	Máx.	Unidades
Z0	Impedancia en modo diferencial a 10MHz	80	110	Ω
T’0	Retardo de línea específico.		10	ns/m
α5MHz	Atenuación del cable a 5MHz		82	dB/Km

3.3.1.3.2. Parámetros de conectores

Los conectores del bus Flexray deben soportar las restricciones de la Tabla 3.2.

Tabla 3.2. Parámetros de conectores para un bus Flexray.

Nombre	Descripción	Mín.	Máx.	Unidad
RDCC	Resistencia de contacto		50	mΩ
ZC	Impedancia del conector	70	200	Ω
LC	Longitud de acoplamiento del conector		150	mm

3.3.1.3.3. Terminador de la interfaz de conexión

El terminador más simple para un acoplador entre el nodo y el bus consiste de un resistor entre las terminales BP (*Bus Plus*) y BM (*Bus Minus*) del administrador del bus (Figura 3.4).

Un terminador complejo que permita mejorar el rendimiento de la conexión del nodo con el bus, cosiste de un arreglo de resistores, capacitares y bobinas de choque [11, 12], de forma que incremente la protección respecto a ESD (*Electrostatic Discharge*) y EMC (*Electro Magnetic Compatibility*).

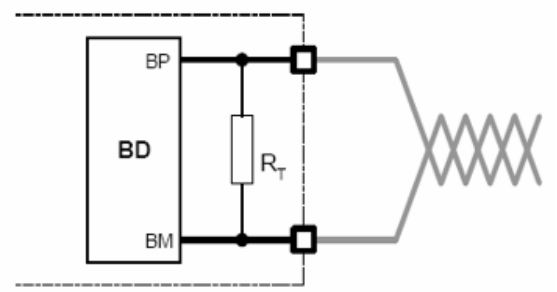


Figura 3.4. Acabado de la interfaz de conexión.

En todas las conexiones punto a punto (nodo-estrella, estella-estrella), las líneas deben terminar con un resistor cuyo valor sea igual a la impedancia nominal del cable. En una conexión de bus, los dos nodos que tengan la máxima distancia eléctrica deberán terminar con un resistor cuyo valor sea igual o ligeramente mayor a la impedancia nominal del cable, el resto con un arreglo resistivo en configuración Y, con los resistores conectados a los cables BP y BM en el orden de 1,300 Ω, el tercer resistor de 10 Ω en serie con un capacitor de 4.7 nF conectado a tierra [11].

3.3.1.3.4. Retardo de propagación

En una secuencia de datos binarios, transmitidos del nodo M al nodo N, el retardo de propagación $dPropagationDelay_{M,N}$ se mide a partir del flanco de caída del primer BSS [14] (Figura 3.5). El retardo de propagación permite sincronizar el esquema de tiempo del nodo con el *cluster* y el BSS permite realizar la sincronización de bit (Inciso 3.3.1.1).

El retardo de propagación depende de la topología en uso y no puede exceder los 2,500 ns.

3.3.1.3.5. Señalización eléctrica

La Figura 3.6 muestra las señales eléctricas sobre las líneas del bus (*BP y BM*) y la Tabla 3.3 lista los niveles mínimos requeridos del voltaje diferencial en un transmisor y un receptor.

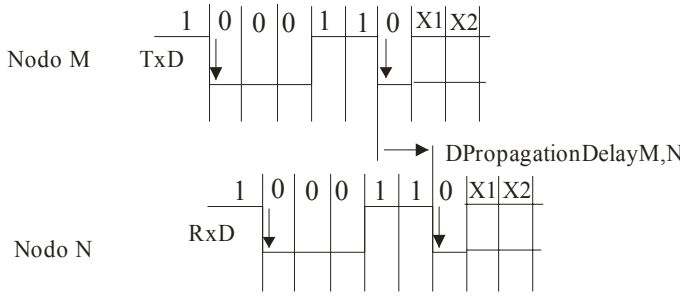


Figura 3.5. Retardo de propagación.

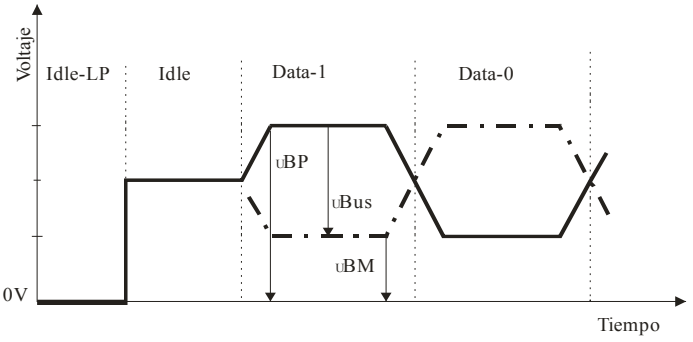


Figura 3.6. Esquema de los niveles de voltaje durante la señalización eléctrica.

Tabla 3.3. Integridad de la señal en el bus.

Nodo	Descripción	Mín.	Unidad
A	Transmisor	± 600	mV
B	Receptor	± 400	mV

El bus Flexray puede tomar cuatro posibles estados: *Idle_LP*, *Idle*, *Data_1* y *Data_0*. Los voltajes de los cables del bus, *BP* y *BM*, se denotan por *uBP* y *uBM*, ambos medidos con respecto a la tierra del sistema. El voltaje diferencial (*uBus*) en el bus se define como: *uBus = uBP – uBM*.

El BD no diferencía entre los estados *Idle_LP (Low Power)* e *Idle*, ya que en ambos casos no existe señal de datos en el bus; cuando entra en el estado *Data_1*, al menos un BD genera una diferencia de potencial positiva entre *BP* y *BM* e inversamente para *Data_0*. La Tabla 3.4 muestra la relación entre el BD y los estados lógicos del bus.

Tabla 3.4. Nivel de señal en el bus en términos de los estados del BD.

Estado del BD	TxEN	BGE[1]	TxD	Resultado de la señal en el bus
Activo	High	X	X	*Idle*
	X	Low	X	*Idle*
	Low	High	Low	*Data_0*
	Low	High	High	*Data_1*
Baja potencia (LP)	X	X	X	*Idle_LP*

1. La señal BGE pertenece a la interfaz entre el BG y el BD, no implementada en este trabajo de tesis.

3.3.1.4. Topologías de red

Un *cluster* Flexray se puede configurar como una topología de red tipo bus, estrella o una combinación híbrida de canal único o canal dual, en está última existen dos canales, A y B, y cada nodo en el *cluster* puede estar conectado a uno o ambos canales. En condiciones libre de fallas, todos los nodos conectados al canal A estarán habilitados para comunicarse entre sí y lo mismo para el canal B. Si un nodo necesita conectarse con más de un *cluster*, debe hacerlo con un controlador diferente (por ejemplo una pasarela), ya que no es válido conectar el canal A de un nodo al canal B en un *cluster* diferente.

Todas las topologías Flexray son de tipo lineal, lo que significa que está libre de anillos cerrados. Un *cluster* puede contener hasta 64 nodos conectados a la red, de los cuales un máximo de 15 nodos serán *coldstart*, es decir que están habilitados para transmitir tramas de sincronización.

3.3.1.4.1. Conexión punto a punto

La configuración punto a punto es la representación más simple de un bus (Figura 3.7) y es el elemento básico para la construcción de buses más complejos.

3.3.1.4.2. Topología de bus

Un bus lineal pasivo es una estructura sin anillos cerrados y sin elementos activos; la Figura 3.8 muestra un bus con tres nodos, en donde el nodo A está conectado a ambos canales (A yB) y los nodos B y C están conectados a un sólo canal, canal A y canal B respectivamente.

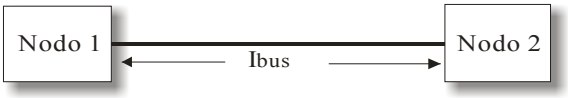


Figura 3.7. Conexión punto a punto.

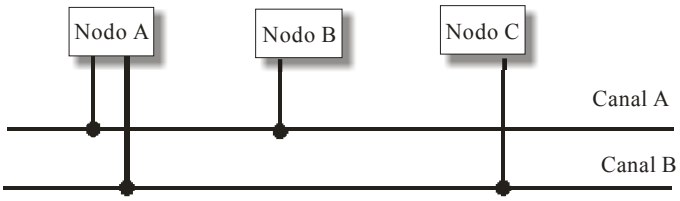


Figura 3.8. Topología de bus.

3.3.1.4.3. Topología de estrella

Una topología Flexray puede ser construida como una topología de estrella múltiple, que a diferencia del bus, soporta conexiones redundantes como muestra la Figura 3.9. Una configuración Flexray de estrella debe tener las siguientes características:

- Cada canal debe estar libre de anillos cerrados.
- No puede haber más de dos acopladores de estrella por cada canal en una red.
- La señal de entrada a un acoplador, es redireccionada al resto de los nodos.

En el ejemplo de la Figura 3.9 se utilizan acopladores de estrella activos con conexiones punto a punto. El uso de configuración de estrella pasiva es un caso particular de una red de bus con un empalme (*stub*) conectado al acoplador pasivo.

3.3.1.4.4. Topología híbrida

Las topologías híbridas pueden estar conformadas por configuraciones de bus, estrella pasiva y activa. Flexray soporta las topologías híbridas tanto como lo permitan los limites para cada topología individual. La Figura 3.10 muestra una topología híbrida con dos acopladores de estrella en cascada y una topología tipo bus.

3.3.1.4.5. Restricciones

La Tabla 3.5 resume los valores límites para cada una de las topologías mencionadas, las recomendaciones deberán tomarse en cuenta cuando se está diseñando la topología, con la finalidad de incrementar su rendimiento y determinar los valores óptimos para las terminaciones de los acopladores de bus; para ello se debe cumplir con lo siguiente:

- Un empalme no deberá conectar a más de dos empalmes.
- Mantener la longitud total del cable tan corta como sea posible, considerar: $\Sigma IStub_i + \Sigma ISD_{i,j} < 24$ m.
- Para evitar cables (*bus*) no acoplados se recomienda conectar las ECUs opcionales a bifurcaciones separadas en una estrella activa.

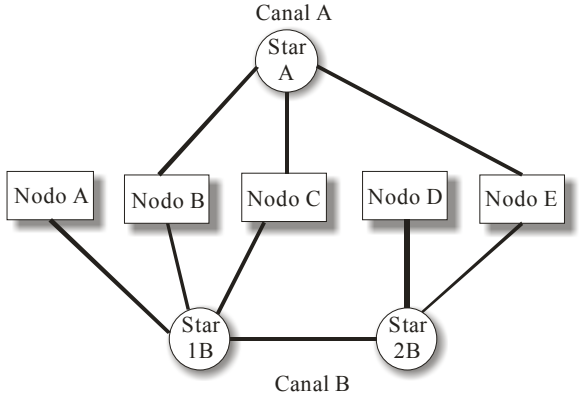


Figura 3.9. Topología de estrella con dos acopladores en cascada.

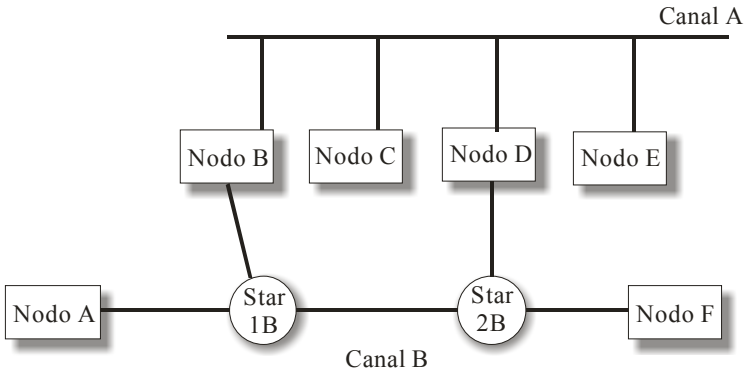


Figura 3.10. Topología híbrida.

Tabla 3.5. Restricciones para el diseño de una topología de red.

Nombre	Descripción	Mín.	Máx.	Unidad
	Conexión punto a punto			
IBus	Distancia de una conexión punto a punto		24	m
	Estructura tipo bus			
IBus	Distancia eléctrica entre dos ECUs		24	m
nStub	Número de empalmes (conectores del ECU al bus)	4	22	
ISDi,j	Distancia entre empalmes		< 24	m
	Red de estrella activa			
IStarStar	Distancia eléctrica entre dos acopladores de estrella activa		24	m
IStarActivaN	Longitud de una bifurcación de un nodo N al acoplador		24	m

3.3.1.5. Colisiones

Flexray está diseñado para realizar la comunicación entre nodos sin colisiones durante operaciones normales; sin embargo, durante la fase de inicio (*startup*) del protocolo pueden ocurrir colisiones en el canal. En la capa física-eléctrica no es posible resolver este conflicto, ni determinar que señal recibirán los nodos.

3.3.1.6. Manejo de errores

En este inciso se detallan los aspectos relacionados al manejo de errores descrito por el protocolo Flexray.

3.3.1.6.1. Errores generados en el entorno

Cuando ocurre un error generado en el entorno de un nodo Flexray, el BD envía una señal al *host* indicando el tipo de error que ha detectado, el cual puede deberse a niveles bajos de voltaje, niveles altos de temperatura, pérdida de conexión con el canal, etc.

3.3.1.6.2. Niveles de voltaje

En caso de que la batería muestre una irregularidad, el BD no permanecerá en modos de operación indefinido, sino que cambiará a un estado dinámico de manejo de energía (Figura 3.11). Cuando durante un periodo de tiempo el nivel de voltaje (V_{bat}) cae por debajo del límite permitido, el BD notifica al *host* considerando los datos de la Tabla 3.6.

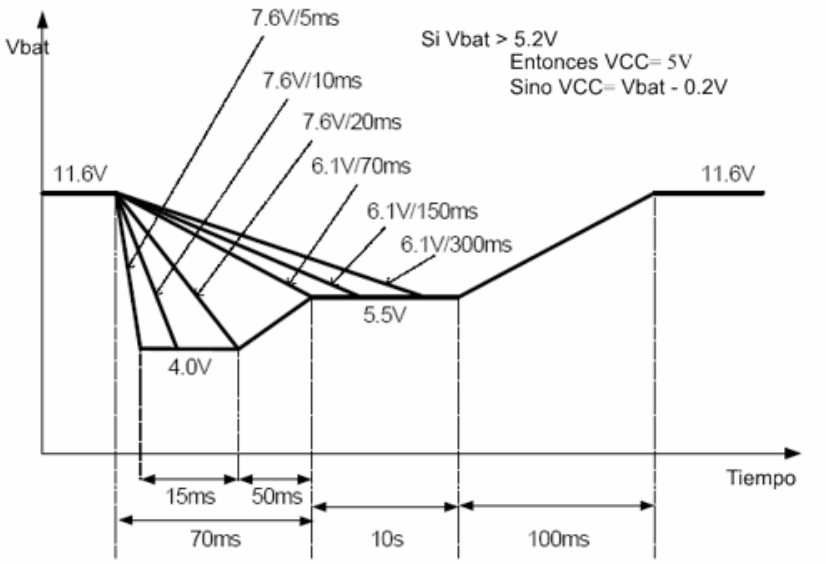


Figura 3.11. Manejo dinámico del voltaje de la batería.

Tabla 3.6. Umbral de tiempo en estado de bajo voltaje.

Nombre	Descripción	Mín.	Máx.	Unidad
dV_{bat}	Tiempo de reacción ante voltaje bajo	-	1000	ms
V_{bat}	Umbral de detección de voltaje bajo	2	5.5	V

3.3.1.6.3. Detección de fallas en el bus

El BD cuenta con los mecanismos necesarios para detectar fallas en el bus, los cuales se notifican al *host* mediante señales.

3.3.1.6.4. Protección contra calentamiento

El BD proporciona los medios para monitorear los niveles de temperatura en el rango de -40 y 125 ºC; en caso de superar el umbral, el BD deshabilita el transmisor con el objetivo de evitar un mayor calentamiento de la unidad. La función de recepción se mantendrá activa tanto como sea posible. La información de sobrecalentamiento se señaliza al *host*.

3.3.2. Capa de enlace

La capa de enlace de datos (DLL, *Data Link Layer*) representa el núcleo del protocolo, establece mecanismos de detección de errores y define el método de acceso al medio mediante su temporización, sincronización, carga e inicio, etc. Esta capa es independiente, en la medida de lo posible, de la topología de red que se utilice.

En este apartado se describen los principios y fundamentos básicos de la DLL, la cual está dividida en dos subcapas, control de enlace lógico (LLC, *Logia Link Control*) y control de acceso al medio (MAC, *Medium Access Control*).

3.3.2.1. Subcapa de control de enlace de datos

La subcapa LLC describe la parte alta de la capa DLL y define las tareas independientes del método de acceso al medio. Los servicios proporcionados son:

- Control de interfaz con el *host*.
- Control de operaciones del protocolo.

3.3.2.1.1. Control de interfaz con el host

El control de interfaz con el *host* (CHI, *Control Host Interface*) administra el flujo de datos y el control entre el *host* y el motor del protocolo Flexray en cada nodo. El CHI realiza las siguientes funciones:

- *Manejo de datos del protocolo*: Controla el intercambio de datos relevantes para la operación del protocolo, como son los de configuración, de control y de estado.
- *Manejo de mensajes de datos*: Controla la memoria temporal (*buffer*) para intercambio de datos, mensajes a transmitir y datos de configuración, de control y de estado.
- *Filtrado de mensajes*: Proporciona un medio para la selección de mensajes con base a su identificador; el identificador de una trama no indica la dirección destino, pero permite determinar a todo receptor activo en la red si el mensaje le es o no útil.
- *Manipulación del vector de red*: El vector de red permite establecer un enlace de comunicación entre dos nodos específicos localizados en el mismo o en diferente *cluster*.

De este modo, el CHI proporciona los medios para una interacción estructurada entre el *host* y los mecanismos del protocolo, incluyendo al control de operaciones del protocolo.

3.3.2.1.2. Control de operaciones del protocolo

La tarea fundamental del control de operaciones del protocolo (POC, *Protocol Operation Control*) consiste en iniciar, coordinar y apagar el núcleo del protocolo, así como iniciar el *cluster* y el sistema, proporcionando detección de errores. El propósito del POC es reaccionar a las órdenes del *host* a través del CHI, y modificar los modos de operación del protocolo en respuesta a las condiciones cambiantes del nodo.

El POC realiza dos tareas primordiales para sincronizar el esquema TDMA:

- *Inicio del cluster (wakeup)*: El objetivo es preparar a los nodos conectados al *cluster* para inicializar al sistema, sólo los nodos *coldstart* pueden hacer esta tarea; si el bus es dual, la trama se envía una a la vez en cada canal con objeto de evitar interferencias en los canales. Dado que el *host* configura un canal del nodo para recibir o transmitir la trama *wakeup* (si recibe el patrón de inicio de *cluster* en otro canal, lo ignora), un nodo diferente inicializa el segundo canal. El nodo da soporte a cualquier número de nodos que intentan iniciar el *cluster* y se asegura de que sólo un nodo transmita el patrón *wakeup*.
- *Inicio del sistema (startup)*: Un sistema de comunicaciones basado en un esquema TDMA requiere de un proceso de sincronización para todos los nodos conectados al *cluster*. Flexray utiliza una estrategia distribuida tolerante a fallos, que consiste en: los canales que conforman al *cluster* deben estar activos antes de iniciar el sistema; sólo los nodos *coldstart* pueden realizar la inicialización del sistema; el cuál comienza con la transmisión del símbolo CAS (*Collision Avoidance Symbol*) y sólo el nodo *coldstart* que transmita el símbolo CAS continuará transmitiendo tramas de sincronización (*startup*); los primeros ocho ciclos son utilizados para realizar la sincronicación de los nodos, primero los nodos *coldstart* y después el resto.

Un CC puede establecer la sincronización de sí mismo al *cluster* de tres posibles maneras:

- *Nodo coldstart principal*: Si el nodo no detectó ningún CE en cualquiera de los canales a los que está conectado, iniciará la transmisión del símbolo CAS en el primer ciclo regular (cuyo valor inicial es cero), a partir del cual el nodo transmite sus tramas *startup*; dado que cada nodo *coldstart* está habilitado para realizar acciones *coldstart*, puede ocurrir que varios nodos transmitan simultáneamente el símbolo CAS, esta situación se resuelve

durante los primeros cuatro ciclos después de la transmisión del CAS, dado que tan pronto reciba un símbolo CAS o una cabecera de trama durante estos cuatro ciclos, regresará al estado de detección. En el cuarto ciclo, el resto de nodos *coldstart* iniciarán la transmisión de sus tramas *startup*.

- *Nodo coldstart de seguimiento*: Si se detecta un CE, tratará de integrarse al nodo *coldstart* transmisor si recibe un par de tramas *startup* válidas en los siguientes dos ciclos que derive el cronograma y corrección del reloj del temporizador en términos del nodo *coldstart* transmisor. Si la corrección del reloj no genera ningún error y continúa recibiendo tramas del mismo nodo, se considerará integrado e iniciará la transmisión de sus tramas *startup*. Si en los siguientes tres ciclos la corrección del reloj no genera error y es visible al menos otro nodo *coldstart*, el nodo entrará en operación, de lo contrario regresará al estado de detección.
- *Integración de nodos no coldstart*: Si se recibe un CE, tratará de integrarse al nodo *coldstart* transmisor en los siguientes dos ciclos. Tratará de encontrar al menos un par de nodos que transmitan tramas *startup* que ajuste con su propio cronograma, si resulta errónea, abortará y realizará un nuevo intento. Después de recibir pares de trama *startup* validas durante dos ciclos consecutivos dobles (cuatro ciclos), todos los nodos del *cluster* entrarán en operación.

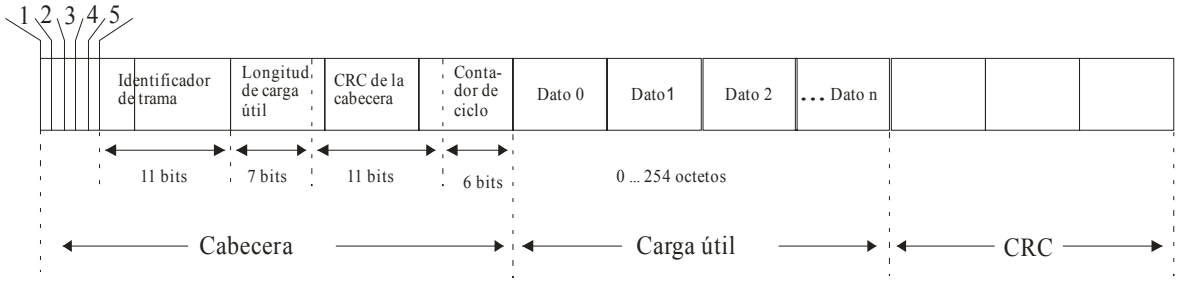
3.3.2.1.3. Condición y manejo de errores

El POC tiene dos mecanismos básicos para responder a errores. Para errores significativos el POC detendrá toda operación de manera inmediata, mientras que para errores de los que se puede recuperar en un periodo de tiempo limitado, usará un modelo de degradación de errores. Los errores que generarán un alto total son:

- Condiciones de error detectadas por un producto específico (BIST, *Build In Self Test*).
- Condiciones de error detectadas por el *host* que resultan en una orden de alto total.
- Condiciones de error fatal detectados por el POC en uno de los mecanismos del núcleo del protocolo.

El modelo de degradación de errores de tres niveles está diseñado para reaccionar a ciertas condiciones detectadas por el mecanismo de sincronización y que no requieren de acción inmediata debido al mecanismo de sincronización tolerante a fallos. Esto hace posible evitar detener toda operación mientras se corrige el error. El modelo se define como:

- *Operación normal activa*: Condiciones libres de errores o al menos dentro de límites tolerables, tal que el nodo está habilitado para trasmitir y recibir mensajes (dado que está sincronizado al *cluster* no hace interferencia con otros nodos).
- *Operación normal pasiva*: La sincronización con el *cluster* se ha degradado, se deshabilita la transmisión de mensajes debido a que es posible que haya colisiones con trasmisiones de otros nodos. La recepción de tramas se mantiene habilitada, ya que es posible recobrar la sincronización y volver al modo de operación activa.
- *Alto total (Halt)*: Si los errores persisten o son graves, se detendrá toda operación del nodo y dado que no es posible recuperarse de estos errores, el POC preparará los mecanismos para reinicializar el nodo. Estas condiciones serán notificadas al *host* que actuará según el protocolo.



1. Bit de reserva.
2. Indicador de preámbulo en la carga útil.
3. Indicador de trama nula.
4. Indicador de trama síncrona.
5. Indicador de trama startup.

Figura 3.12. Formato de la trama Flexray.

3.3.2.2. Control de acceso al medio

Esta sección describe la parte baja de la capa DLL y presenta una breve descripción de los mecanismos del núcleo del protocolo Flexray como son el formato y validación de la trama, el método de acceso al medio y el mecanismo de sincronización.

3.3.2.2.1. Estructura de la trama Flexray

La Figura 3.12 muestra el formato de la trama Flexray, la cual consiste de tres segmentos, cabecera, carga útil y CRC.

El segmento de cabecera está compuesto por cinco octetos divididos en:

- *Bit de reserva*: Destinado a futuras aplicaciones, su valor siempre es un cero lógico y es ignorado por el protocolo.
- *Indicador de preámbulo en la carga útil*: Indica si está presente un vector de red (*opcional*) en la carga útil de la trama; si es transmitida en el segmento estático, o en el segmento dinámico; y la presencia de identificador de mensaje al inicio de la carga útil.
- *Indicador de trama nula*: Indica que la trama transmitida es nula.
- *Indicador de trama síncrona*: Bit utilizado por los mecanismos de sincronización; si tiene un valor uno lógico, la trama será utilizada para la sincronización del reloj.
- *Indicador de trama de inicio (startup)*: Indica si es una trama de inicio de sistema, las cuales sólo pueden ser enviads por los nodos *coldstart* y por lo tanto estos son los únicos que pueden modificar esta bandera.
- *Identificador de trama (Frame ID)*: El identificador de trama consta de 11 bits, define la ranura (*slot*) en la cual se transmite la trama, la cual no puede ser usada más de una vez en un ciclo de comunicación y tiene un rango de 1 a 2047 ranuras; la trama con un ID igual a cero no es válida o es una trama errónea.
- *Longitud de carga útil*: Campo de 7 bits utilizado para indicar la longitud del segmento de la carga útil, el cual se codifica dividiendo el número de octetos de datos entre dos. Los valores aquí almacenados van de 0 a 127 palabras dobles.
- *CRC de la cabecera*: Contiene el código de redundancia cíclica (CRC, *Cyclic Redundancy Check*) calculado sobre los campos: indicador de trama síncrona, indicador de trama de inicio, identificador de trama y longitud de carga útil.

- *Contador de ciclo*: Campo de 6 bits que indica el valor del contador del ciclo desde el punto de vista del nodo transmisor al tiempo en que la trama fue transmitida.

El segmento de carga útil puede contener de 0 a 254 octetos de datos (0 a 127 palabras dobles). Las tramas transmitidas en el segmento dinámico y los primeros dos octetos pueden ser usados como campo para el identificador de mensajes (ID). Para las tramas transmitidas en el segmento estático, los primeros 12 octetos pueden ser usados como vector de red, ambos casos se indican con un uno lógico en el bit de preámbulo.

El segmento de CRC es un campo de 24 bits, utiliza una distancia Hamming para longitudes de hasta 248 octetos del segmento de datos es de 6 y una distancia Hamming de 4 para longitudes mayores a 248 octetos. El nodo emplea un vector de inicialización diferente para cada canal.

3.3.2.3. Subcapa MAC

En Flexray, la subcapa MAC es un ciclo de comunicaciones periódico. En un ciclo de comunicación existen dos esquemas de modos de acceso al medio, TDMA y FTDMA, este último consiste en un esquema de miniranuras (*minislots*).

3.3.2.3.1. Ciclo de comunicación

El ciclo de comunicación es el elemento fundamental del esquema del acceso al medio y está definido por la jerarquía de temporización mostrada en la Figura 3.13.

La capa más alta define el ciclo de comunicación, el cual contiene los segmentos estático y dinámico, la ventana de símbolo y el NIT (*Network Idle Time*). Dentro del segmento estático se utiliza un esquema TDMA para el arbitraje de las transmisiones; en el segmento dinámico se utiliza FTDM con base en la manipulación de miniranuras; y en la ventana de símbolo se pueden transmitir símbolos y el NIT, que es un periodo libre de transmisiones.

La capa de arbitraje es el núcleo del control de acceso al medio, el cual durante el segmento estático está formado por intervalos de tiempo consecutivos llamado ranuras estáticas. El segmento dinámico se compone por intervalos de tiempo denominados miniranuras.

La capa de arbitraje se construye en términos de *macroticks* (formado por un número específico de *microticks*). Los instantes específicos de tiempo en que el nodo realiza una acción específica en sincronización con su tiempo local base, son llamados puntos de acción (AP, *Action Point*), los cuales son instantes específicos en que la transmisión inicia y en el caso de segmento dinámico el fin de la transmisión.

La capa de *microtick* determina la unidad mínima de tiempo utilizada en el esquema del control de acceso al medio.

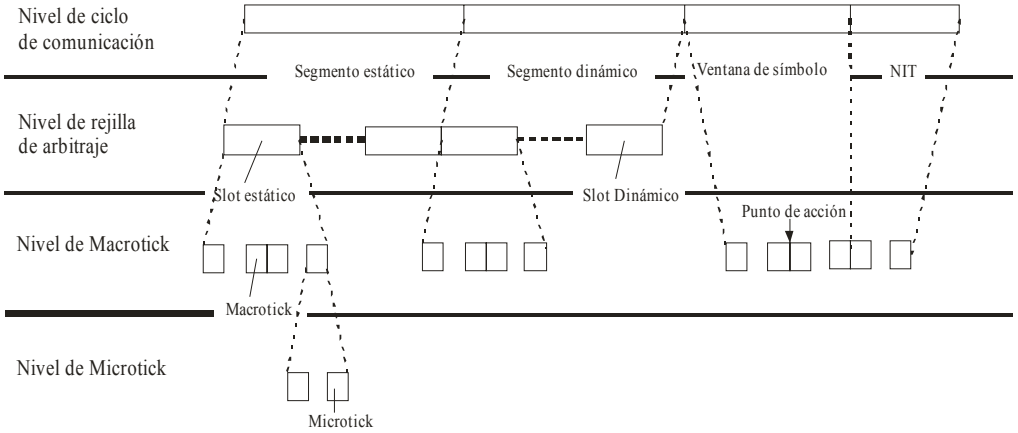


Figura 3.13. Elementos de un ciclo de comunicación.

El arbitraje sobre los segmentos estático y dinámico también se basa en el identificador de trama único asignado a cada ranura de cada nodo en el *cluster* para cada canal, un esquema de conteo que produce ranuras de transmisión numeradas. El identificador de la trama determina la ranura y el segmento en el que será transmitida.

3.3.2.3.2. Segmento estático

El segmento estático utiliza el esquema de acceso TDMA con un criterio estático para coordinar la transmisión de mensajes y dado que todas las ranuras son configuradas estáticamente, para tener la misma longitud y duración independientemente de la cantidad de datos a transmitir, se consideran las siguientes restricciones al transmitir durante el segmento estático:

- Las tramas de sincronización se deben transmitir en todos los canales a los que esté conectado el nodo.
- Las tramas que no se utilizan para la sincronización, pueden ser transmitidas en cualquier canal o ambos si es necesario.
- Sólo un nodo transmite un ID específico asociado a una ranura.
- Si el *cluster* es configurado en modo de ranura única, todos los nodos no síncronos designan a una trama como de ranura única, es decir, el nodo transmite en una sola ranura.

3.3.2.3.3. Segmento dinámico

El esquema usado para el arbitraje de transmisiones se basa en un esquema de miniranuras o TDMA dinámico de duración fija. La duración y longitud de las ranuras (dinámicas) pueden ser diferentes dependiendo de la cantidad de datos a transmitir en el segmento de carga útil. A diferencia del segmento estático, el acceso al medio en los dos canales no ocurre de manera simultánea.

Si ocurre una transmisión de datos, la ranura dinámica puede estar constituida por varias miniranuras, de lo contrario la ranura será de longitud igual a la miniranura. La transmisión de tramas inicia con el AP en la primera miniranura correspondiente a la ranura dinámica, el fin de la transmisión también es reconocida con el AP de una miniranura. Esto se lleva a cabo con la transmisión del símbolo DTS (*Dynamic Trailing Sequence*).

El arbitraje de este segmento asegura que todos los nodos libres de errores conozcan implícitamente la ranura dinámica en la que inicia la transmisión, así como la minuranura en la que ésta reinicia, de forma que sean capaces de relacionar el contador de ranuras del nodo transmisor con el ID contenido en la trama.

3.3.2.3.4. Ventana de símbolo

Durante el segmento de ventana de símbolo (*Symbol Window*) sólo se envía un símbolo (*symbol*). El objetivo de este espacio es resincronizar los nodos del *cluster* que se hayan desincronizado o verificar el correcto funcionamiento del *cluster* mediante la transmisión de un símbolo de prueba de acceso (MTS, *Media Access Test Symbol*) dentro de la ventana de símbolo.

El arbitraje sobre diferentes transmisores del símbolo no es parte de la especificación y debe ser llevado a cabo por un protocolo de capa superior.

3.3.2.3.5. Tiempo libre de red

El tiempo libre de red (NIT, *Network Idle Time*) sirve como fase durante la cual los nodos calculan y aplican el factor de corrección del reloj. Así mismo, su duración puede extenderse ya que contiene los *macroticks* que no se han usado en los segmentos estático y dinámico, y en la ventana de símbolo. Cuando este segmento está activo significa que no existe tráfico de CEs en la red.

3.3.2.3.6. Sincronización

En un sistema de comunicaciones distribuido cada nodo tiene su propio reloj. Sin embargo el tiempo en el reloj de cada nodo, aún sincronizados al inicializar, puede variar debido a fluctuaciones de temperatura, voltaje y tolerancias de la fuente de temporización. En el caso de los sistemas TTA, se asume que cada nodo en el *cluster* tiene una visión global del tiempo y la utiliza para sincronizarse, con lo cual se asegura que la diferencia entre los tiempos de reloj de cada nodo se encuentre dentro de los límites de tolerancia establecidos y calibrados. A la diferencia máxima permitida se le conoce como precisión.

El protocolo Flexray utiliza un mecanismo de sincronización distribuido en el que cada nodo se sincroniza a sí mismo con el *cluster* mediante las tramas de sincronización de otros nodos.

3.3.2.3.7. Representación del tiempo

La representación del tiempo en un nodo Flexray se basa en ciclos (hasta 64), *macroticks* y *microticks*, como muestran las Figura 3.13 y 3.14, cada ciclo se compone de un número entero de *macroticks* y éste a su vez de *microticks*:

- *Microticks*: Es la unidad de tiempo derivada directamente del oscilador del CC y pueden tener diferente duración de un nodo a otro.
- *Macrotick*: Dentro de un margen de tolerancia, la duración de un *macrotick* es idéntica en todos los nodos en el *cluster*. El número de *microticks* entre un *macrotick* y otro puede ser diferente aún en el mismo nodo.
- *Ciclo*: El número de *macroticks* por ciclo es el mismo en todos los nodos del *cluster*. En cualquier tiempo dado, todos los nodos deben tener el mismo número de ciclo.

3.3.2.3.8. Proceso de sincronización

La principal tarea del mecanismo de sincronización es asegurar que la diferencia de tiempo entre los nodos del *cluster* esté dentro de los límites de precisión. Existen métodos de sincronización que usan corrección de fase o frecuencia, Flexray usa una combinación de ambos. La sincronización del reloj se compone de dos partes concurrentes:

- La generación de *macroticks*, control del ciclo, control del contador de *macroticks* y aplicar la corrección de los valores de fase y frecuencia.
- La sincronización se ejecuta al inicio del ciclo, medición de fase y registro de los valores de desviación y calcular la corrección de fase y frecuencia.

El cálculo de la corrección de fase se ejecuta cada ciclo, pero la corrección se realiza al final de cada ciclo impar durante el segmento correspondiente al NIT. El cálculo y la aplicación de la corrección de frecuencia se llevan a cabo durante el segmento estático en un ciclo impar y se basa en los valores medidos en cada ciclo par e impar.

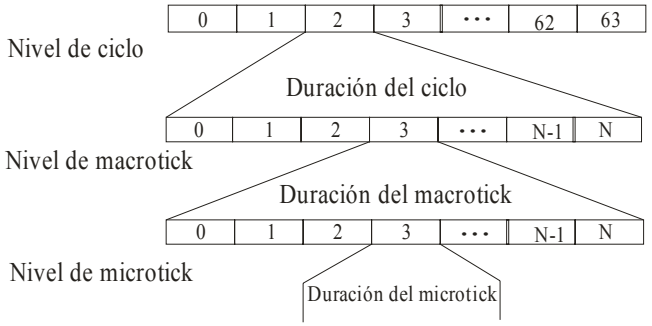


Figura 3.14. Representación del tiempo.

3.3.2.3.9. Medición del tiempo

Cada nodo mide y registra, para cada canal, la diferencia (en *microticks*) entre el tiempo esperado de llegada de la trama y el tiempo real en que llega, esta tarea se repite en todas las tramas de sincronización transmitidas durante el segmento estático. Se almacenan los valores de la desviación y el estado lógico, que permite validar la trama como válida o de inicio, esto se realiza para cada canal (A y B) y para cada ciclo, par e impar.

El tiempo esperado de llegada de la trama es el AP de cada ranura estática. Durante la recepción de la trama, la unidad de decodificación toma el punto de referencia secundario en el tiempo al momento en que éste es detectado (el flanco de caída del primer BSS); usa este valor para calcular el punto de referencia primario en el tiempo mediante la resta de un valor de *offset* configurable (depende de la longitud de las líneas de transmisión, acopladores, retardos de propagación, etc.); la medición de la fase termina al final del segmento estático.

3.3.2.3.10. Cálculo del término de corrección

La técnica usada para calcular el factor de corrección (en *microticks*) es un algoritmo de punto medio tolerante a fallos (FTM), el cual funciona de la siguiente manera [9]:

- El algoritmo determina el valor del parámetro *k* con base a los valores medidos de desviación (Tabla 3.7). *k* representa el número de valores que serán descartados como muestra la Figura 3.15.
- Se descartan los *k* valores más pequeños y más grandes.
- Se promedian los *k+1* valores más grandes y más pequeños para obtener el punto medio. El valor resultante representa la desviación de los nodos del tiempo global y sirve como factor de corrección.

Tabla 3.7. Criterio para determinar el parámetro *k* del algoritmo FMT.

Número de valores	Parámetro k
1 - 2	0
3 - 7	1
> 7	2

Para el caso en que la trama de sincronización sea transmitida en ambos canales, el algoritmo determina el valor de desviación más pequeño de los dos y será utilizado por el algoritmo. Si el resultado está dentro de los límites de precisión, el nodo estará sincronizado al *cluster*.

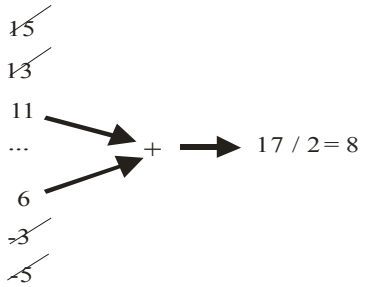


Figura 3.15. Algoritmo para corrección del reloj (*k=2*).

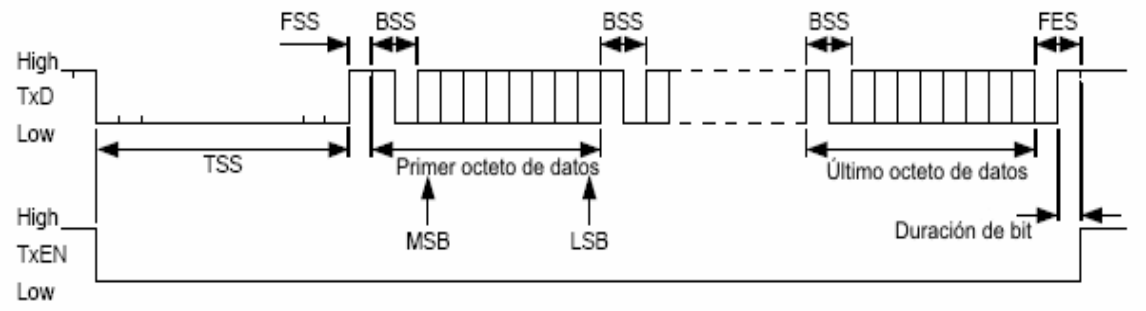


Figura 3.16. Ensamblado de la trama en el segmento estático.

3.3.2.4. Transmisión de tramas

3.3.2.4.1. Transmisión de mensajes

Con objeto de transmitir tramas de datos en los segmentos estático y dinámico, el nodo ensambla una cadena de bits con los elementos que se describen a continuación.

La trama de mensajes está formada por los siguientes patrones más los bits de datos:

- *Secuencia de inicio de transmisión (TSS, Transmisión Start Sequence)*: Secuencia de bits para inicializar correctamente la conexión a través de la red.
- *Secuencia de inicio de trama (FSS, Frame Start Sequence)*: Se utiliza para compensar los posibles errores de cuantización en el primer octeto de inicio de la TSS.
- *Secuencia de inicio de octeto (BSS, Byte Start Sequence)*: Para la sincronización de bits, así como para realizar la sincronización del *cluster*.
- *Secuencia de fin de trama (FES, Frame End Sequence)*: Para marcar el final de la secuencia de octetos transmitidos, se inserta inmediatamente después del último octeto de datos transmitido.
- *Secuencia de rastreo dinámico (DTS, Dinamic Trailing Sequence)*: Se utiliza únicamente en el segmento dinámico e indica el punto exacto en el tiempo en que ocurre el AP de la miniranura transmisora, también prevé detecciones prematuras del NIT por los nodos receptores. Cuando se transmite en el segmento dinámico, el nodo transmite un DTS inmediatamente después de que se transmite el FES.

Las Figuras 3.16 y 3.17 muestran la secuencia de bits ensambladas en el segmento estático y dinámico respectivamente. Como se puede observar, la diferencia entre ambas es el símbolo DTS, cuyo propósito es sincronizar dichas transmisiones del segmento dinámico.

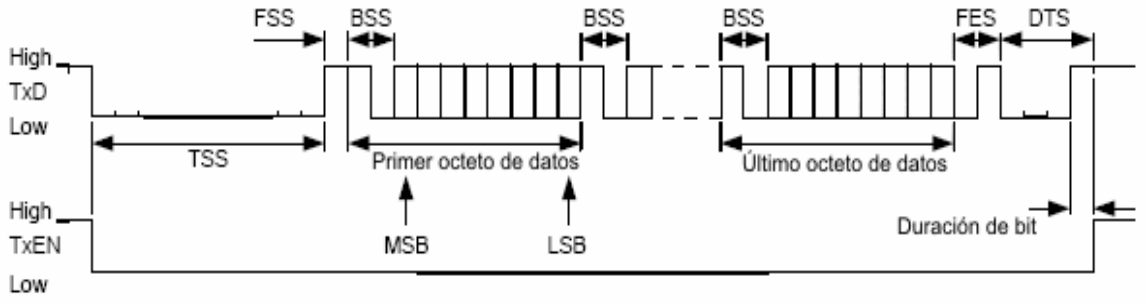


Figura 3.17. Ensamblado de la trama en segmento dinámico.

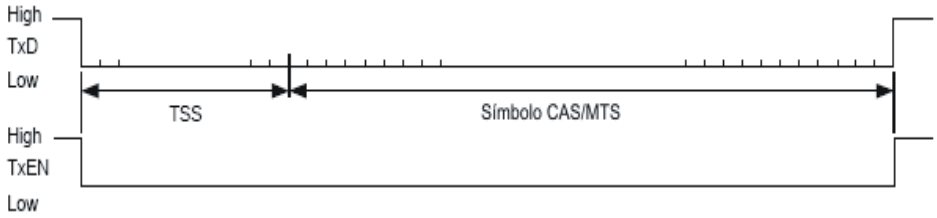


Figura 3.18. Representación de los símbolos CAS y MTS.

3.3.2.4.2. Transmisión de símbolos

Flexray define tres patrones especiales de tramas (*símbolos*), las cuales se utilizan al inicio del *cluster* y del sistema (si el *host* lo solicita, también en la ventana de símbolo) y se representan por dos patrones de bits:

- *Patrón 1*: Símbolo de evasión de de colisiones (CAS, *Collision Avoidance Symbol*) y símbolo de prueba de acceso al medio (MTS, *Media Access Test Symbol*). El nodo ensambla los símbolos CAS y MTS de la misma forma; los nodos receptores distinguen entre estos símbolos basándose en su estado, a excepción del proceso de codificación que no hace distinción entre estos símbolos (Figura 3.18).
- *Patrón 2*: Símbolo de inicio (WUS, *Wakeup Symbol*). El símbolo WUS tiene como función inicializar el *cluster* mediante el patrón de la Figura 3.19, puede apreciarse que está compuesto por una parte alta y otra baja, el símbolo completo se forma de repetir al menos dos veces el símbolo WUS.

3.3.2.4.3. Condiciones de error

Cuando se detectan errores de decodificación, el nodo toma al primer bit erróneo como si fuera el último del actual CE, es decir, detiene la decodificación, informa del error y espera a que detecte el final de la transmisión para reiniciar la decodificación.

3.3.2.4.4. Validación de tramas

Es el principal subnivel entre la decodificación de tramas y símbolos, y el CHI. Se encarga de verificar que la temporización de las tramas y de los símbolos sea correcta con respecto al esquema TDMA; y aplica rigurosas pruebas sintácticas y semánticas a las tramas recibidas. Los errores sintácticos ocurren bajo una de las siguientes condiciones:

- Un nodo inicia una transmisión fuera de su ranura asociada.
- Ocurre un error de decodificación.
- Una trama es decodificada durante los segmento de ventana de símbolo o NIT.
- Un símbolo es decodificado fuera del segmento de ventana de símbolo.
- Dos o más símbolo son decodificados en el segmento de ventana de símbolo.

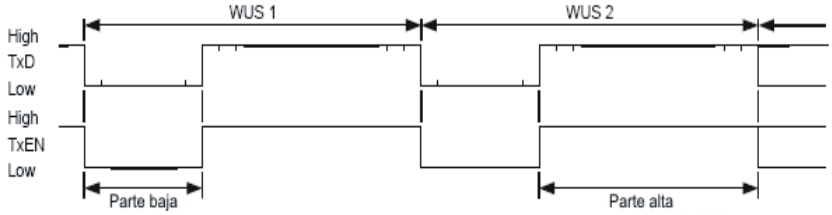


Figura 3.19. Representación del símbolo WUS.

- Se recibe una trama dentro de la ranura después de recibir una trama correcta.

Los errores semánticos ocurren bajo las siguientes condiciones:

- En el segmento estático, la longitud de la cabecera contenida en el encabezado de la trama recibida no tiene relación con la longitud registrada en este segmento con respecto al parámetro *gPayloadLengthStatic*.
- En el segmento estático, el bit indicador de inicio de trama, contenido en la cabecera de la trama recibida, está puesto a uno mientras que el indicador de trama de sincronización es puesta a cero.
- En el segmento estático o dinámico, el identificador de la trama recibida no se relaciona con el contador de ranuras del nodo o es cero en el segmento dinámico.
- En el segmento estático o dinámico, el valor del conteo del ciclo decodificado no tiene relación con el contador del ciclo del nodo.
- En el segmento dinámico, el indicador de trama, de sincronización o de inicio, decodificado está puesto a uno.
- En el segmento dinámico, el indicador de trama nula está puesto a cero.

Es posible decodificar tramas con las siguientes características:

- Se recibe una trama válida, ocurre un error de sintaxis y no existe error en su contenido (error).
- Se recibe una trama válida sin error de sintaxis pero con error en su contenido (error).
- Una trama semánticamente correcta es válida, si no presenta errores de sintaxis ni en su contenido (sin error).

3.4. Guardian del bus

La función del BG es supervisar el correcto funcionamiento del controlador Flexray en el esquema TDMA. De forma opcional, un sistema Flexray puede incluir esta subcapa de supervisión en un sistema Flexray y/o una capa de redundancia en el controlador Flexray que supervise al BG [13].

Si el CC está sincronizado con el *cluster*, el BG habilita al BD para realizar la transmisión; y en caso de que se pierda la sincronización con el *cluster*, notifica al *host* del error y deshabilita al BD para transmitir.

3.5. Capa de aplicación

La capa de aplicación no es parte de la especificación de Flexray, sin embargo es posible desarrollarla mediante el uso de protocolos de capa superior; la principal función de esta capa es permitir la comunicación con otros protocolos como pueden ser CAN, MOST, LIN, etc. Un ejemplo de capa de aplicación es AUTOSAR (*AUTomotive Open System Architecture*), iniciativa en desarrollo para establecer enlaces de comunicación entre diferentes protocolos aplicados a la industria automotriz [28]. AUTOSAR hace uso de las capas de red y de transporte.

Dado que Flexray es independiente de la capa de aplicación, es posible el uso de cualquier CPU (*host*) para empotrar la capa de aplicación, el único requisito es que los parámetros de configuración del *cluster* (prefijo g + nombre parámetro) sean los mismos en cada nodo, ya que cada nodo o controlador Flexray tiene control absoluto sobre el bus. Los parámetros correspondientes al nodo en sí (prefijo p + nombre parámetro) pueden diferir entre nodos conectados al mismo *cluster*. Estos parámetros son establecidos en tiempo de diseño del sistema.

Flexray aún no es un estándar, por ello la mayoría de los sistemas desarrollados por distintas empresas no son compatibles, sin embargo se recomienda el uso de la herramienta de diseño Flex-Config en el desarrollo de sistemas Flexray de alta complejidad.

3.5.1. Herramienta de desarrollo FlexConfig

FlexConfig es una herramienta universal para el desarrollo y configuración de redes Flexray. FlexConfig permite establecee los parámetros necesarios mediante una aplicación de MS Windows; la mayoría de parámetros son verificados con respecto a restricciones específicas; además permite a los usuarios generar todos los parámetros de configuración para cada nodo del *cluster* permitiendo seleccionar un controlador diferente para cada nodo [URL8].

La Figura 3.20 muestra la interfaz gráfica de usuario (GUI, *Graphic User Interface*) de la herramienta FlexConfig, cuya ventana principal se divide en el explorador (vista jerárquica) y los datos de configuración del *cluster* a configurar.

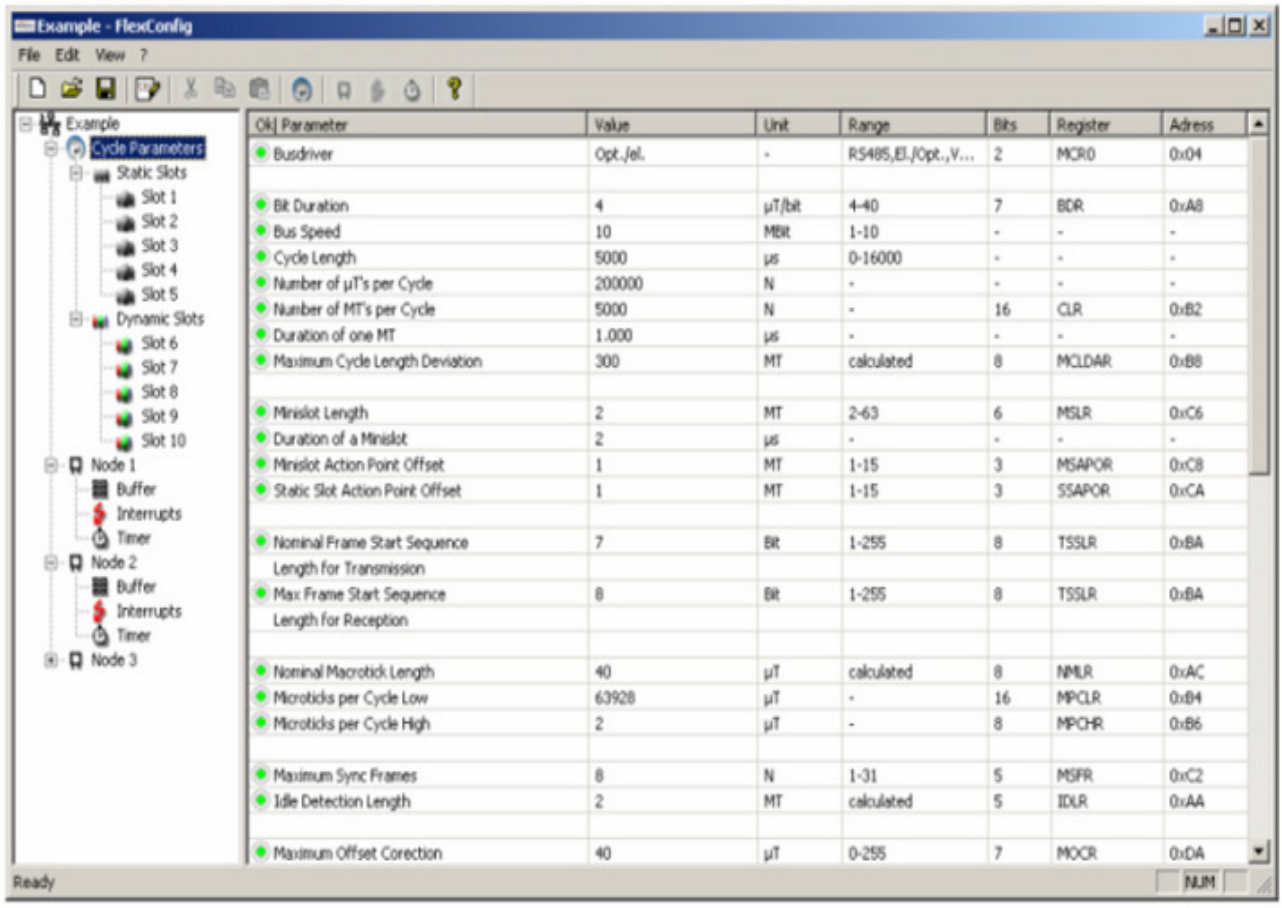


Figura 3.20. Interfaz gráfica de usuario de la herramienta FlexConfig.

4. Especificación del protocolo Flexray con la FDT SDL

El presente capítulo detalla el desarrollo de la especificación del protocolo Flexray mediante el uso de la FDT SDL. Se definen los requerimientos y el alcance del sistema, se realiza una breve descripción de los objetos definidos en la especificación y se presenta la especificación de datos, estática y dinámica del sistema final. De acuerdo con la metodología de desarrollo (Figura I.1) se realizó una clasificación de la información a partir de la especificación del protocolo (no documentado en este trabajo de tesis) con objeto de obtener los requisitos de usuario, los requisitos de la especificación formal se han cubierto en el Capítulo 2.

4.1. Requerimientos y acotamiento del sistema

Con base en las características del sistema de comunicaciones Flexray (Capítulo 3) se definen las siguientes restricciones respecto a su especificación:

- Este trabajo se centrará en la especificación del núcleo del protocolo (CC).
- El sistema estará compuesto de un sólo *cluster*.
- La especificación no incluirá la capa de supervisor (BG y *host 2* de la Figura 3.1).
- La especificación del administrador de bus (BD) será mínima.
- Debido a que la capa de aplicación (*host 1*) no es parte de la especificación del protocolo, sólo se realizará una especificación mínima para poder realizar la ejecución de la especificación del protocolo.
- La CHI, cuya función consiste en servir como un medio de enlace entre el *host* y el CC del protocolo, implementará la especificación mínima del *host*.
- No se especificará el soporte de la administración del vector de red, el servicio para filtrado de identificador de mensajes y el servicio del vector de interrupciones.
- Se realizará una especificación mínima para la memoria temporal de recepción y de transmisión de mensajes (datos).
- Especificación mínima de la administración de ranuras para la transmisión de mensajes.

Debido a que la especificación de un sistema en SDL no se ejecuta en tiempo real y su ejecución puede llevar demasiado tiempo, se plantean las siguientes restricciones de configuración:

- El *cluster* estará compuesto por un sólo canal (canal A, puede estar compuesto por dos canales, canal A y canal B) y por lo tanto no se requiere un BD, con lo cual se reduce la carga de operaciones durante la ejecución de la especificación.

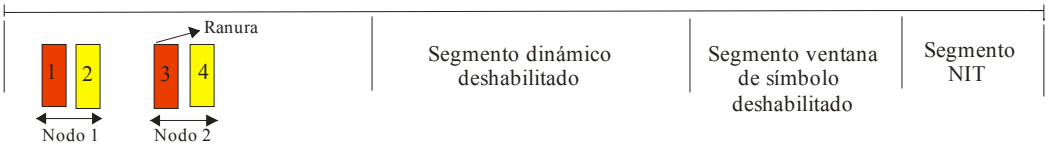


Figura 4.1. Distribución de ranuras para cada nodo.

- El *cluster* estará formado por dos nodos, ya que es el mínimo de nodos necesarios para establecer un enlace de comunicación Flexray.
- La transmisión de datos sólo se realizará durante el segmento estático.
- El segmento dinámico estará deshabilitado.
- No se realizará la transmisión del *Symbol* (durante el segmento ventana de símbolo), debido a que requiere de un proceso de arbitraje realizado por un protocolo de capa superior y éste no es parte de la especificación del protocolo.
- Cada nodo contendrá sólo dos ranuras (Figura 4.1). Es posible configurar cada nodo para soportar hasta 16 ranuras (esta es la única restricción que afecta al cálculo de los parámetros de configuración).

El número total de ranuras soportadas en el segmento estático es de 1023 en *clusters* con el máximo de nodos soportados. La distribución de ranuras en cada nodo pueden ser consecutivos: n1_1, n2_2, n3_3…n16_16; o pueden estar distribuidas en orden diferente: n1_1, n2_9, n3_14…n16_32 (el primer índice indica el número de ranura asociado al nodo, mientras que el segundo índice se refiere a la posición de la ranura en el segmento).

4.2. Objetos de la especificación

A continuación se realiza una breve introducción de los objetos identificados para el desarrollo de la especificación del sistema; es decir, los bloques, sub-bloques, así como los procesos que constituyen la especificación estática del protocolo Flexray. Los nombres asociados a cada objeto describen la función que estos desempeñan y se presentan en orden descendente: arquitectura del sistema, arquitectura de bloque y nivel de procesos, definiendo el diseño arquitectural de la metodología de diseño (Figura I.1). La Figura 4.2 muestra los niveles jerárquicos en un diagrama de árbol.

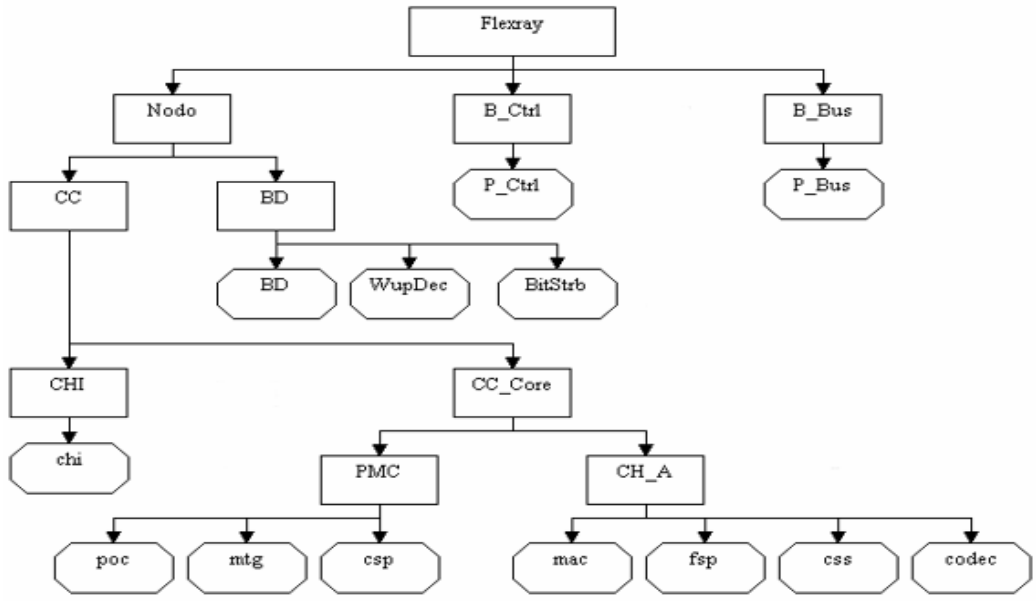


Figura 4.2. Jerarquía de los objetos de la especificación.

El sistema estará constituido por los siguientes elementos que representan los objetos SDL:

- Una unidad que realiza la función del medio físico, representando una topología de bus.
- Una unidad de control autónoma (sin interfaz con el usuario) que se encarga de realizar la configuración básica para cada nodo del sistema.
- Un conjunto de estaciones (2 nodos), cada nodo estará formado por dos objetos que representan las dos capas del protocolo, capa física y capa de enlace de datos.
- *BD*: Realiza las funciones de enlace con el medio físico.
- *CC*: Representa la unidad de control de comunicación del protocolo y realiza las funciones de las subcapas LLC y MAC de la capa de enlace DLL; estará formado por dos objetos:
  - *CHI*: Realiza la interfaz con el *host* que contiene la aplicación.
  - *CC_Core*: Representa el núcleo del protocolo y está formado por dos objetos:
    - *PMC*: Se encarga de las operaciones del protocolo, temporización y sincronización.
    - *CH_A*: Realiza la gestión del control de acceso al medio para el canal A; para sistemas duales se requiere de una unidad adicional para el canal B.

La Tabla 4.1 resume los objetos SDL de los procesos, su unidad de ámbito y la descripción de las funciones que realizan.

Tabla 4.1. Objetos SDL del sistema.

Objetos SDL					Descripción
Unidad de ámbito o bloque				Proceso	
BUS				Bus_P	Realiza las funciones del medio físico
B_Ctrl				P_Ctrl	Realiza la configuración básica el sistema
Nodo	CC	BD		bd	Interfaz con el medio físico
				wupdec	Decodificación del patrón WUP
				bitstrb	Filtrado de bit y punto de muestra de bit
		CHI		chi	Interfaz entre el *host* y el CC
		CC_Core	PMC	poc	Control de operaciones del protocolo
				mtg	Generador de *microticks* y *macroticks*
				csp	Proceso de sincronización del reloj
			CH_A	mac	Control de acceso al medio
				fsp	Procesamiento de la trama y de símbolos
				css	Proceso de sincronización durante la fase de inicio del sistema
				codec	Codificación y decodificación de mensajes y símbolos

4.3. Especificación de datos

Los tipos de dato usados en la especificación del protocolo Flexray han sido declarados dentro de la librería (*package*) FlexrayDefsLib, declarada y enlazada al nivel de entorno del sistema, lo que permite que sean accesibles en todos los niveles de abstracción. Los parámetros globales, locales y constantes del protocolo han sido declarados en las librerías FlexrayClusterParameter, FlexrayNodeParameter y FlexrayProtocolConstant respectivamente (Apartado 4.4.2 y Subcapítulo 4.6).

4.3.1. Tipos de datos para el control del protocolo

Los tipos de datos de control permiten gestionar y supervisar el correcto funcionamiento de la especificación del protocolo. La clasificación de los tipos de datos de control se describe para cada proceso de la especificación.

4.3.1.1. Tipos de datos relacionados con el proceso poc

Los tipos de datos relacionados al proceso poc son (Figura 4.3):

- *T_POCState*: Es un tipo de dato para monitorear el estado del poc y reaccionar según sus condiciones. Sus valores son: CONFIG, DEFAULT_CONFIG, HALT, NORMAL_ACTIVE, NORMAL_PASSIVE, READY, STARTUP, WAKEUP y están en función del resultado de los procesos de inicio del *cluster* y del sistema, así mismo indica el estado en que se encuentra el modelo de degradación de errores.
- *T_WakeupStatus*: Tipo de dato que permite monitorear el estado del proceso de inicialización del *cluster*, los valores que puede tomar son: UNDEFINED, RECEIVED_HEADER, RECEIVED_WUP, COLLISION_HEADER, COLLISION_WUP, COLLISION_UNKNOWN y TRANSMITTED.
- *T_StartupState*: Tipo de dato que permite monitorear el estado del proceso de inicio del sistema, los valores que puede tomar depende del tipo de nodo; es decir, estos pueden ser distintos para el nodo *coldstart* principal, nodos *coldstart* de seguimiento y nodos no *coldstart*, estos valores son: UNDEFINED, COLDSTART_LISTEN, INTEGRATION_COLDSTART_CHECK, COLDSTART_JOIN, COLDSTART_COLLISION_RESOLUTION, COLDSTART_CONSISTENCY_CHECK, INTEGRATION_LISTEN, INITIALIZE_SCHEDULE, INTEGRATION_CONSISTENCY_CHECK y COLDSTART_GAP.
- *T_ErrorMode*: Tipo de dato que define el modelo de degradación de errores definido en el proceso poc, el cual depende del estado de los procesos que conforman al nodo, sus valores son: ACTIVEE, PASSIVE y COMM_HALT.
- *T_SlotMode*: Tipo de dato que indica el estado de las ranuras del nodo, el cual puede estar en modo de ranura única (lo que tiene efecto sólo en nodos *coldstart*) o bien tener habilitadas todas las ranuras asociadas al nodo. Sus valores asociados son: SINGLE, ALL_PENDING y ALL_T.
- *T_POCStatus*: Estructura de datos para registrar el estado global del poc con objeto de mantener informado y actualizado al chi.
- *T_Channel*: Tipo de dato que permite hacer referencia a un canal específico asociado al poc; también es usado como índice de los arreglos que registran información asociados a cada canal. Los valores que puede tomar son: A y B.
- *T_ChannelBoolArray*: Arreglo que almacena un estado lógico para cada canal, indicando si éstos están disponibles o existe un enlace de comunicación a través de ellos. Sus valores son establecidos por el poc y son usados durante la fase de inicio del cluster y del sistema.

4.3.1.2. Tipos de datos relacionados al proceso csp, css y mtg

La Figura 4.4 muestra la especificación formal en SDL de los tipos de datos para establecer el modo de operación y realizar la sincronización, los cuales se describen a continuación.

- *T_CspMode*: Tipo de dato que permite definir el modo de operación del csp, el cual puede tomar tres valores dependiendo del estado del nodo, STANDBY, NOSYNC y SYNC.

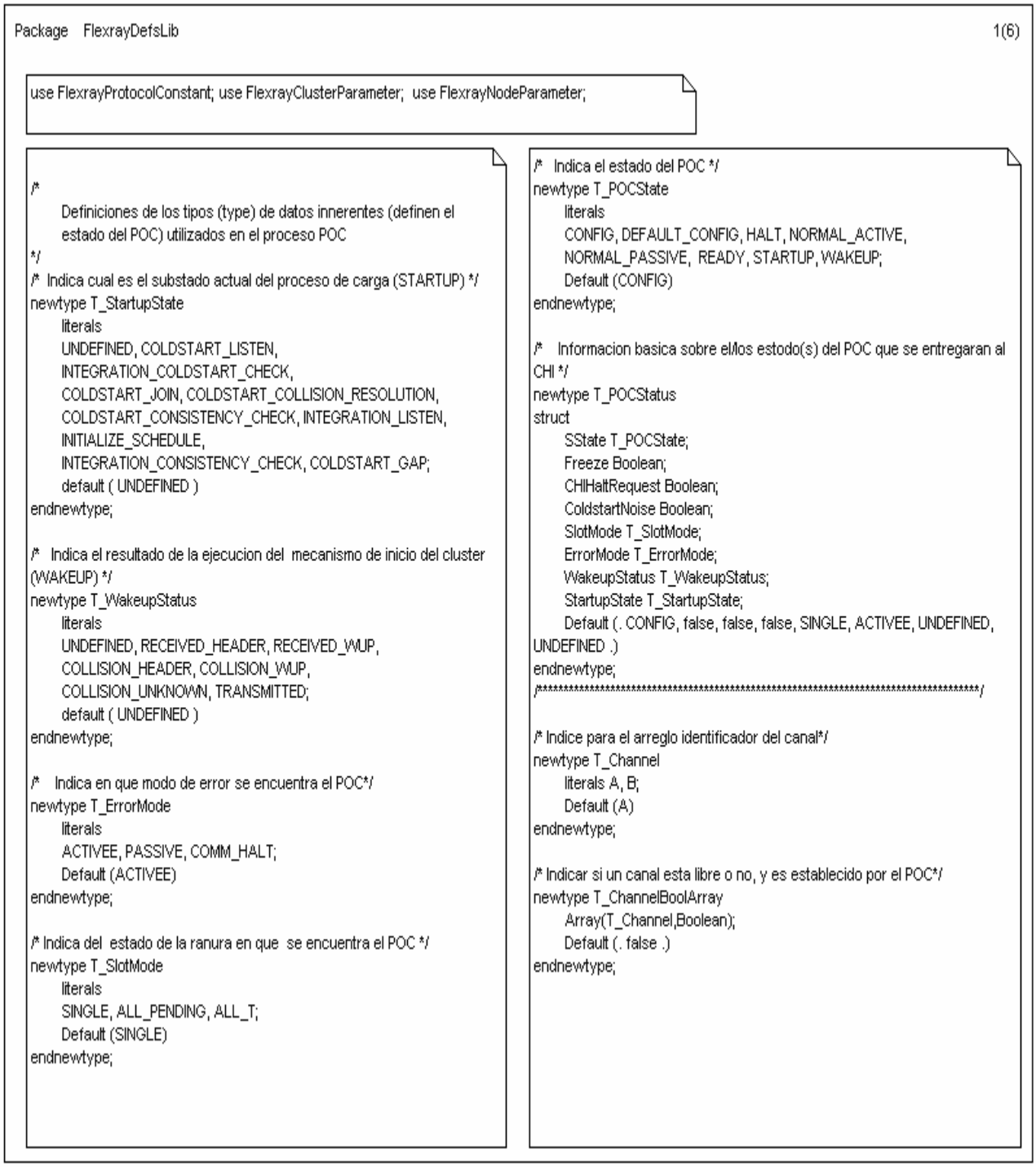


Figura 4.3. Tipos de datos asociados al poc.

4.3.1.2.1. Tipos de datos relacionados con la temporizacion y la sincronización

Los tipos de datos relacionados con la temporización y la sincronización son:

- *T_SyncCalcResult*: Tipo de dato que puede tomar los valores: WITHIN_BOUNDS, EXCEEDS_BOUNDS y MISSING_TERM, lo que depende del resultado del cálculo y ajuste del factor de corrección del reloj (fase y frecuencia).
- *T_FrameIDTable*: Arreglo para registrar los identificadores de las tramas de sincronización recibidas.

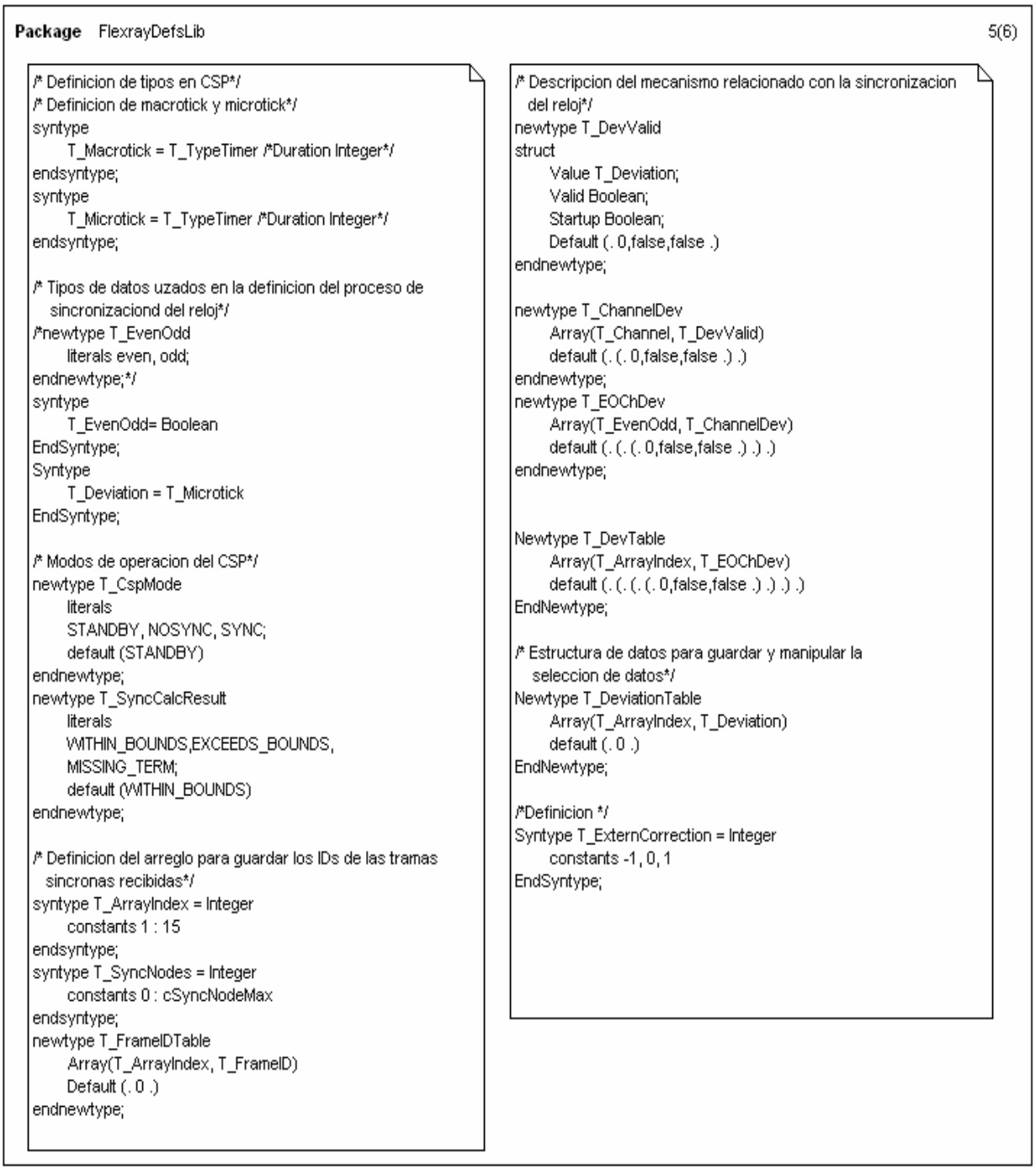


Figura 4.4. Tipos asociados a los procesos csp, css y mtg.

- *T_Macrotick*: Es un tipo de dato redefinido a partir del tipo T_TypeTimer, y es usado para representar el flujo del tiempo en términos de *macroticks*.
- *T_Microtick*: Redefinido del mismo modo que T_Macrotick, pero en *microticks*.
- *T_Deviation*: Tipo de dato que permite registrar los valores de desviación (retraso o adelanto) de la trama de sincronización en estructuras que en breve serán descritas.
- *T_DeviationTable*: Es un arreglo unidimensional en el que serán guardados los valores de desvío de la fase y frecuencia del reloj.

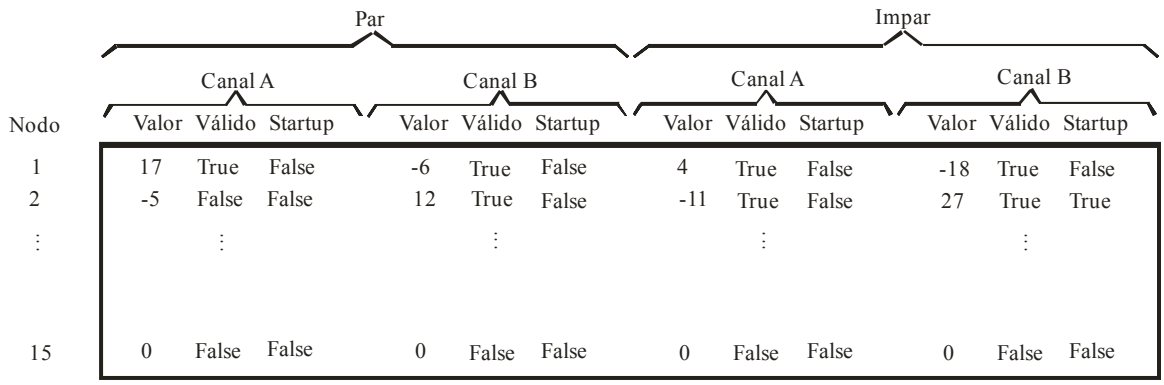


Figura 4.5. Estructura lógica del registro del estado de las tramas de sincronización.

- *T_EvenOdd*: Tipo de dato cuya variable es el índice para registrar los valores de desvío para cada ciclo de comunicación par e impar.

Las siguientes estructuras de datos y arreglos permiten almacenar los valores de desviación calculados, el procesamiento sobre los valores que pueden almacenar se realizan al final de cada ciclo de comunicación, el factor de corrección frecuencia para sincronizar el reloj de cada nodo se realiza cada dos ciclos en el ciclo impar. Estas definiciones permiten saber si la trama recibida es válida, si es un a trama de sincronización (*startup*) y su valor de desvío. La Figura 4.5 muestra su estructura lógica:

- *T_DevValid*: Es una estructura de datos que toma un conjunto de valores usados para el cálculo del factor corrección en el proceso de sincronización
- *T_ChannelDev*: Arreglo cuyo índices son los identificadores para cada canal, cada elemento del arreglo almacena un conjunto de valores en función a la estructura T_DevValid.
- *T_EOChDev*: Es un arreglo que almacena un conjunto de valores del tipo T_ChannelDev para cada ciclo de comunicación par e impar.
- *T_DevTable*: Arreglo en que la información de desvío de las tramas de sincronización recibidas sobre cada canal durante un ciclo de comunicación es almacenada para cada nodo *coldstart* configurado en el *cluster*.

4.3.1.3. Tipos de datos relacionados a los procesos codec, bitstrb y wupdec

A continuación se describen los tipos de datos que permiten establecer los modos de operación apropiados para establecer un enlace de comunicación según la etapa en que se encuentra el nodo. La Figura 4.6 muestra la definición formal en SDL de los tipos de datos asociados al codec, y a los procesos bitstrb y wupdec, estos últimos son controlados por el proceso codec (Apartado 4.3.2).

- *T_CodecMode*: Este tipo de dato puede tomar 4 valores: STANDBY y READY, la primera es configuración por defecto, mientras que la segunda prepara al proceso para iniciar operaciones; los otros dos valores son: WAKEUP y NORMAL, el primero de estos dos valores indica que el nodo esta en al fase de inicio, mientras que el segundo, indica que el nodo opera en condiciones de sincronización.
- *T_Type*: Tipo de dato que permite diferenciar el tipo de trama que se va a transmitir, estática, dinámica, símbolo o patrón WUP; las cuales son descritas por: STATFRAME, DYNFRAME, SYMBOL, WUP respectivamente.
- *T_TransmitFrame*: Estructura de datos que contiene los datos del mensaje a transmitir, se forma por la cabecera y los datos de carga útil.

- *T_ReceiveFrame*: Estructura de datos similar a T_TransmitFrame. T_ReceiveFrame es usada en la recepción de las tramas de mensajes, contiene un par de campos adicionales, uno es para indicar el canal en el que la trama ha sido recibida y otro para indicar el punto de referencia de tiempo o TRP (*Time Reference Point*) primario.
- *T_StrbMode*: Tipo de dato que permite establecer los modos de operación del proceso bitstrb, los valores asociados a este tipo son: STANDBY y GO.
- *T_WupDecMode*: Similar a T_StrbMode, pero asociado al proceso wupdec.
- *T_MicrotickTime*: Es una redefinición del tipo T_Microtick.

Package FlexrayDefsLib 2(6)

```
/*Definidos en el proceso CODEC*/
/*Modos de operacion del CODEC establecido por el POC*/
newtype T_CodecMode
      literals STANDBY, WAKEUP, NORMAL, READY;
      Default (STANDBY)
endnewtype;

/*Describe los niveles de TxD y TxEN, lo cuale es la interfaz
  entre el CODEC y el BD*/
Synonym HIGH Bit= 1;
Synonym LOW Bit= 0 ;
Syntype T_BitLevel= Bit EndSyntype;

/* Define la cadena de bits que sera transmitida por el BD */
Synonym ByteNum Integer= 8 + gPayloadLengthStatic * 2 ;
Synonym BitNum Integer= gdTSSTransmitter + cdFSS +
                        (cdBSS + 8)* ByteNum;
Newtype T_BitStreamArray
      Array (Integer, T_BitLevel);
      Default (. LOW .)
endNewtype;

/*Define el tipo de transmision del CE*/
newtype T_Type
      literals STATFRAME, DYNFRAME, SYMBOL, WUP;
      Default (WUP)
endnewtype;

/*Las siguientes definiciones son uzadas por la macro
  que realiza la decodificacion de la trama de bytes*/
newtype T_ByteArray
      Array (Integer, T_BitLevel);
      Default (. LOW .)
endnewtype;

newtype T_ByteStreamArray
      Array (Integer, T_ByteArray);
      Default (. (. LOW .) .)
endnewtype;
```

```
syntype
      T_CRCCheckPassed = Boolean
endsyntype;

syntype
      T_MicrotickTime = T_Microtick
endsyntype;

newtype T_ReceiveFrame
struct
      PrimaryTRP T_MicrotickTime;
      ChannelT T_Channel;
      Header T_Header;
      Payload T_Payload;
      Default (. 0, A, (. 0,0,0,0,0,0,0,0,0 .), (. 0 .) .)
endnewtype;

/*Datos a trasmitir por el CODEC recibidos del MAC*/
newtype T_TransmitFrame
struct
      Header T_Header;
      Payload T_Payload;
      Default (. (. 0,0,0,0,0,0,0,0,0 .), (. 0 .) .)
endnewtype;

/* Modos de operacion del sub proceso BITSTRB*/
newtype T_StrbMode
      literals STANDBY, GO;
      Default (STANDBY)
endnewtype;

/* Modos de operacion del sub proceso WOPDEC*/
newtype T_WupDecMode
      literals STANDBY, GO;
      Default (STANDBY)
endnewtype;
```

Figura 4.6. Tipos de datos de control asociados a los procesos codec, bitstrb y wupdec.

4.3.1.4. Tipos de datos relacionados al proceso mac

Los tipos de datos relacionados al proceso mac permiten administrar el método de acceso, así como las ranuras de transmisión y los mensajes (Figura 4.7):

- *T_MacMode*: Establece los modos de operación del proceso mac, los valores que puede tomar son: STANDBY, NOCE, STARTUPFRAMECAS, STARTUPFRAME, SINGLESLOT y ALL_T.
- *T_SlotCounter*: Tipo de dato para control de las ranuras de transmisión de mensajes.
- *T_Assignment*: Los valores asociados a este tipo, determinan si la ranura actual esta habilitada para la transmisión de mensajes, sus valores son: UNASSIGNED y ASSIGNED.
- *T_Message*: Tipo de dato cuya variable asociada contendrá los datos del mensaje a transmitir si la ranura está habilitada.

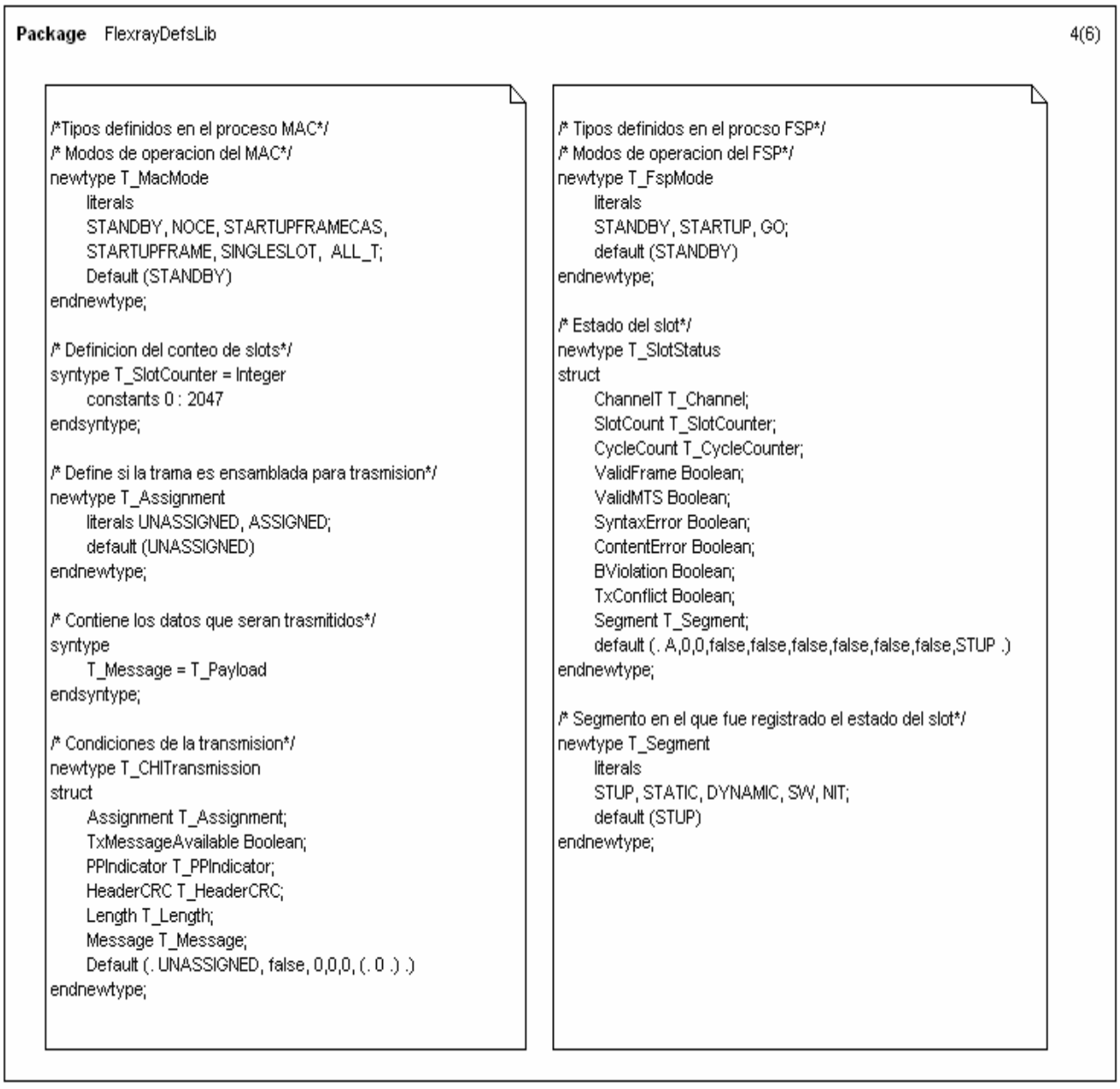


Figura 4.7. Tipos de datos de control asociados a los procesos mac y fsp.

- *T_CHITransmission*: Es una estructura de datos que contiene información sobre el tipo de mensaje a transmitir así como la ranura en la que se realizará, esta información es mantenida por el chi.

4.3.1.5. Tipos de datos relacionado con el proceso fsp

Los tipos de datos relacionado con el proceso fsp son (Figura 4.7):

- *T_FspMode*: Establece los modos de operación del proceso fsp, los valores asociados a este tipo son: STANDBY, STARTUP y GO.
- *T_SlotStatus*: Es una estructura en la que se registra el estado de la ranura en la que es decodificado un mensaje. Esta información es usada por el fsp para realizar el procesamiento sobre tramas o símbolos y es mantenida en el chi.
- *T_Segment*: Tipo de dato que permite indicar el segmento y tipo de trama en el momento en que es registrado el estado de la ranura en una variable asociada al tipo T_SlotStatus. Los valores que puede tomar este tipo de dato son: STUP, STATIC, DYNAMIC, SW y NIT.

4.3.2. Tipos de datos para el procesamiento de tramas

Esto tipos de datos permiten manipular la información contenida en las tramas, se dividen en tipos de datos para procesamiento de alto nivel (Figura 4.8) y tipos de datos para procesamiento de bajo nivel (Figura 4.6). Cabe señalar que la especificación del protocolo no establece esta separación formalmente.

4.3.2.1. Tipos de datos para procesamiento de alto nivel

Los siguientes tipos de datos son considerados para procesamiento de alto nivel, ya que permiten modelar cada segmento del formato de la trama Flexray (Figura 3.12):

- *T_Reserved*: Bit de reserva para usos futuros del protocolo.
- *T_PPIndicator*: Bit indicador de preámbulo en la carga útil, indica si está presente el vector de red o ID de mensaje, esto depende del segmento en que se transmite la trama, estático o dinámico respectivamente.
- *T_NFIndicator*: Bit indicador de trama nula, indica si el segmento de carga útil contiene datos usables o no.
- *T_SyFIndicator*: Bit que indica si la trama es de sincronización o no.
- *T_SuFIndicator*: Bit indicador de tramas para inicio del sistema (*startup*), este indicador solo será puesto en alto en tramas de sincronización, en caso contrario será puesto en bajo.
- *T_FrameID*: El identificador de trama, define la ranura en la que será transmitido una trama, se usará un identificador de trama en cada canal en un ciclo de comunicación; así, cada trama transmitida en un *cluster* tendrá asignado un identificador de trama a sí mismo.
- *T_Length*: Este campo es usado para indicar la longitud del segmento de carga útil, en tramas transmitidas en el segmento estático la longitud de T_Length será la misma y el valor de su magnitud puesta al máximo números de octetos configurado en el *cluster*, dado por gPayloadLengthStatic (Subcapítulo 4.6). La longitud de este segmento puede variar de una trama a otra en el segmento dinámico.
- *T_HeaderCRC*: El CRC de cabecera contiene el código de redundancia cíclica calculado.

- *T_CycleCounter*: El contador de ciclos indica el valor del ciclo actual de los nodos transmisores al tiempo en que se lleva a cabo la transmisión de una trama.
- *T_Header*: Estructura de datos que engloba los tipos de datos anteriores, con la finalidad de defininir formalmente el segmento de cabecera.
- *T_Payload*: Es un arreglo que describe el segmento de datos de la trama Flexray configurable para soportar de 0 hasta 254 octetos de datos.
- *T_FrameCRC*: Representa el segmento del CRC de la trama completa calculado sobre todos los campos mencionados.

Package FlexrayDefsLib 3(6)

```
/* Estructuras para modelar el formato de la trama FlexRay */
/*Bit reservado para futuros usos del protocolo*/
syntype  T_Reserved = Integer
       constants 0 : 1
endsyntype;

/* Bit indicador de preambulo en la carga util*/
syntype T_PPIndicator = Integer
      constants 0 : 1
endsyntype;

/*Bit Indicador de carga nula*/
syntype T_NFIndicator = Integer
      constants 0 : 1
endsyntype;

/* Bir indicador de trama sincrona*/
syntype T_SyFIndicator = Integer
      constants 0 : 1
endsyntype;

/* Bit indicador de trama de carga (startup)*/
syntype T_SuFIndicator = Integer
      constants 0 : 1
endsyntype;

/* Frame ID, 11 Bits, define el slot en el que la trama sera trasmitida*/
syntype T_FrameID = Integer
      constants 0 : 2047
endsyntype;

/* Payload length, 7 Bits, Indica el tamaño del segmento de carga util */
syntype T_Length = Integer
      constants 0 : cPayloadLengthMax
endsyntype;

/* Header CRC, 11 Bits*/
syntype T_HeaderCRC = Integer
      constants 0 : 2047
endsyntype;
```

```
/* Structura para definir el encabezado de la trama HEADER*/
newtype T_Header
struct
      Reserved T_Reserved;
      PPIndicator T_PPIndicator;
      NFIndicator T_NFIndicator;
      SyFIndicator T_SyFIndicator;
      SuFIndicator T_SuFIndicator;
      FrameID T_FrameID;
      Length T_Length;
      HeaderCRC T_HeaderCRC;
      CycleCount T_CycleCounter;
      default(. 0,0,0,0,0,0,0,0,0 .)
endnewtype;

/* FlexRay payload segment*/
newtype T_Payload
      Array(T_Length, Integer)
      Default (. 0 .)
endnewtype;

/* CRC trailer segment*/
syntype T_FrameCRC = Integer
      constants 0 : 16777215  /* 0x000000 : 0xFFFFFF */
endsyntype;

/* Cycle counter, 6 Bits*/
Syntype T_CycleCounter = Integer
      constants 0 : 63
endsyntype;
```

Figura 4.8. Tipos de datos de alto nivel para el procesamiento de tramas.

4.3.2.2. Tipos de datos para procesamiento de bajo nivel

Los tipos de datos que a continuación se describen son considerados para procesamiento de bajo nivel, ya que permiten manipular la transmisión, recepción y realizar otras operaciones sobre tramas en formato binario.

- *T_BitLevel*: Tipo de dato que permite definir los niveles de bit que serán transmitidos y que sirve de base para las siguientes definiciones de tipos.
- *T_BitStreamArray*: Define la cadena de bits asociada a una trama y que será transmitida por el nodo.
- *T_ByteArray*: Es un arreglo que define un octeto de datos binarios, este tipo de dato es usado durante la decodificación de una trama.
- *T_ByteStremArray*: Es un arreglo que representa los octetos recibidos durante la recepción de una trama.

4.3.3. Tipos de datos adicionales

Los tipos de datos adicionales no son parte de la especificación del protocolo de comunicaciones Flexray, más bien son introducidos con objeto de construir una especificación ejecutable del protocolo Flexray (Figura 4.9):

- *T_TypeTimer*: Tipo de dato para el control del flujo del tiempo en términos de *macroticks* y *microticks*.
- *T_BDMode*: Tipo de dato para habilitar o deshabilitar la detección del CHIRP (*Channel Idle Recognition Point*) en el proceso bitstrb. Los valores asociados a este tipo de dato son WUP_DEC y NORMAL.
- *T_Timer*: Tipo de dato que define un temporizador en términos de *macroticks*.
- *T_DataConfig*: Es una estructura de datos usada para enviar al nodo los parámetros básicos de configuración, las ranuras asociadas y los datos para la transmisión de mensajes.
- *T_DataRx*: Estructura de datos usada por los nodos para informar a la unidad de control P_Ctrl al final de cada ciclo de comunicación, los datos que has sido recibidos, la ranura en la que se recibieron y el identificador del proceso que lo envía.
- *Int*: Este tipo de dato hereda todas las propiedades del super tipo Integer, es introducido con objeto de crear un procedimiento que determine si un número dado es par o impar.

4.4. Especificación estática

Los objetos identificados para el desarrollo de la especificación del protocolo Flexray son usados para construir la especificación estática del sistema (Tabla 4.1). Como se muestra en la Figura 4.2, la especificación estática está dividida en varios niveles de abstracción, lo que define la arquitectura del sistema mediante la distribución de los objetos que la componen y las señales del flujo de información entre ellos.

Es importante mencionar que la arquitectura de la especificación estática del protocolo Flexray no está descrita en la especificación del protocolo, sólo indica los procesos que la conforman.

4.4.1. Especificación de canales, rutas de señal y compuertas

La especificación de los objetos SDL se realiza utilizando un estilo de representación remota basada en tipos (*type*), a excepción de los bloques B_Bus y B_Ctrl en los que se usa el estilo de especificación remota. Dado que los tipos (bloque o proceso) son especificaciones genéricas, no tienen

conexión alguna con canales o rutas de señal específicas, es necesario un medio para al intercambio de señales; esto se lleva a cabo mediante la especificación de compuertas, las cuales pueden ser unidireccionales o bidireccionales estableciendo en cualquiera de los dos casos las señales que participarán en el intercambio.

A partir de la versión SDL-96 es posible omitir los nombres de los canales o rutas de señal así como el identificador del alfabeto de entrada que tengan asociadas si esto no causa ambigüedad. En el presente trabajo de tesis se hace uso extenso de esta propiedad, para facilitar tanto la escritura como la lectura de la especificación final.

Para obtener este grado se simplicidad, en la especificación estática se consideran los siguientes aspectos, válidos para los tipos bloque (*block type*) y proceso (*process type*):

- Se puede aplicar la propiedad de omisión a las entidades (instancias) que participan en el intercambio de señales dentro de la misma unidad de ámbito.
- Si una instancia necesita comunicarse con su entorno, sólo se omite el nombre del canal o ruta de señal, la especificación del alfabeto de entrada es obligado.

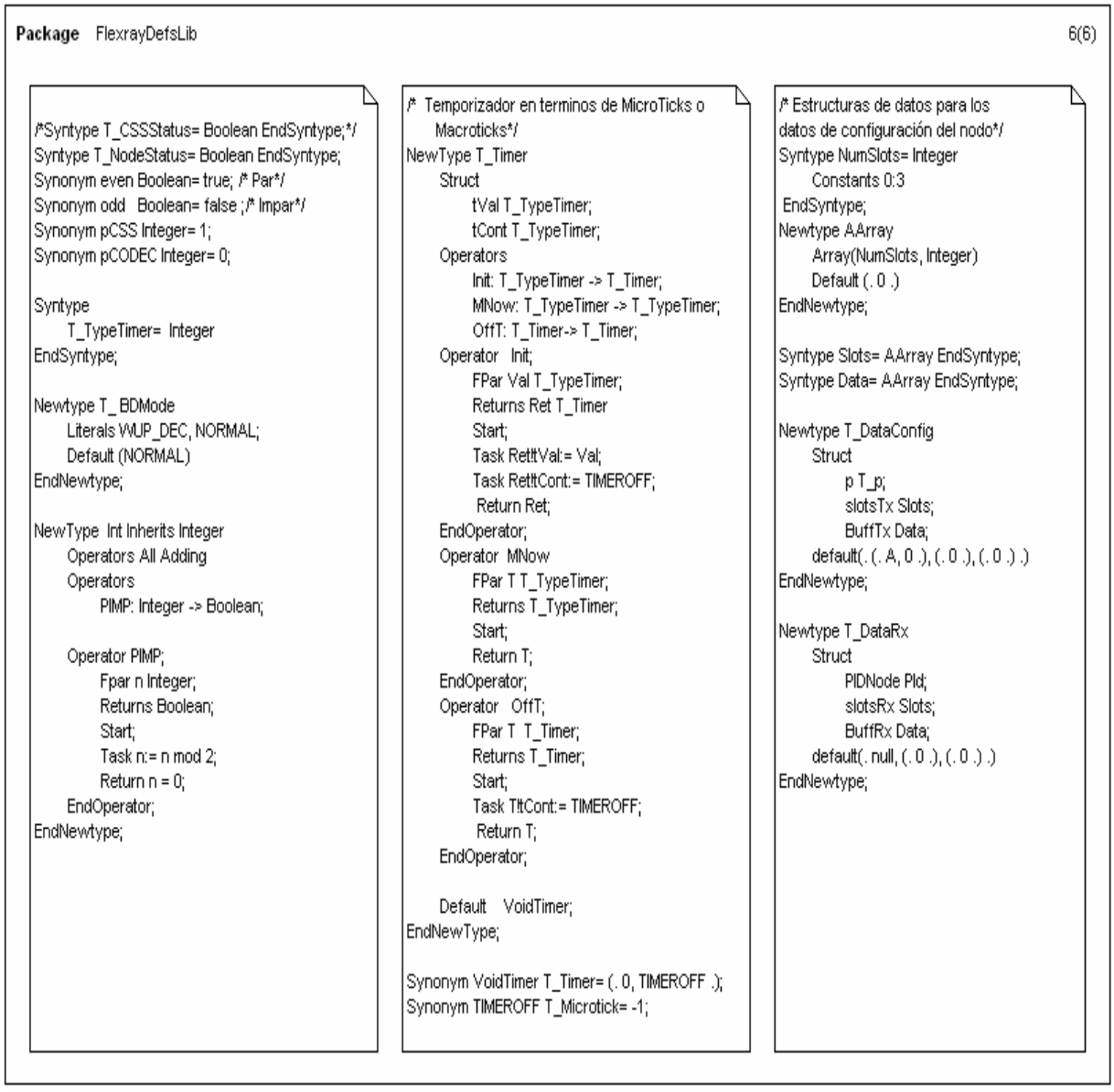


Figura 4.9. Tipos de datos adicionales.

- Si una instancia interactúa con otras entidades descritas con un estilo de especificación remota en la misma unidad de ámbito, es posible omitir el alfabeto de entrada; el nombre de la ruta o vía es obligada.
- Para especificaciones remotas que interactúan entre sí; para este trabajo de tesis se eligió especificar explícitamente tanto el nombre como el alfabeto de entrada (es posible omitir ambas o una de ella).

De este modo toda posibilidad de ambigüedad es evitada (tomando en cuenta que prácticamente todo el sistema está desarrollado con un estilo de especificación remota basada en tipos). El conjunto de reglas detalladas para estas construcciones se puede consultar en [1, 15, 24 y 26].

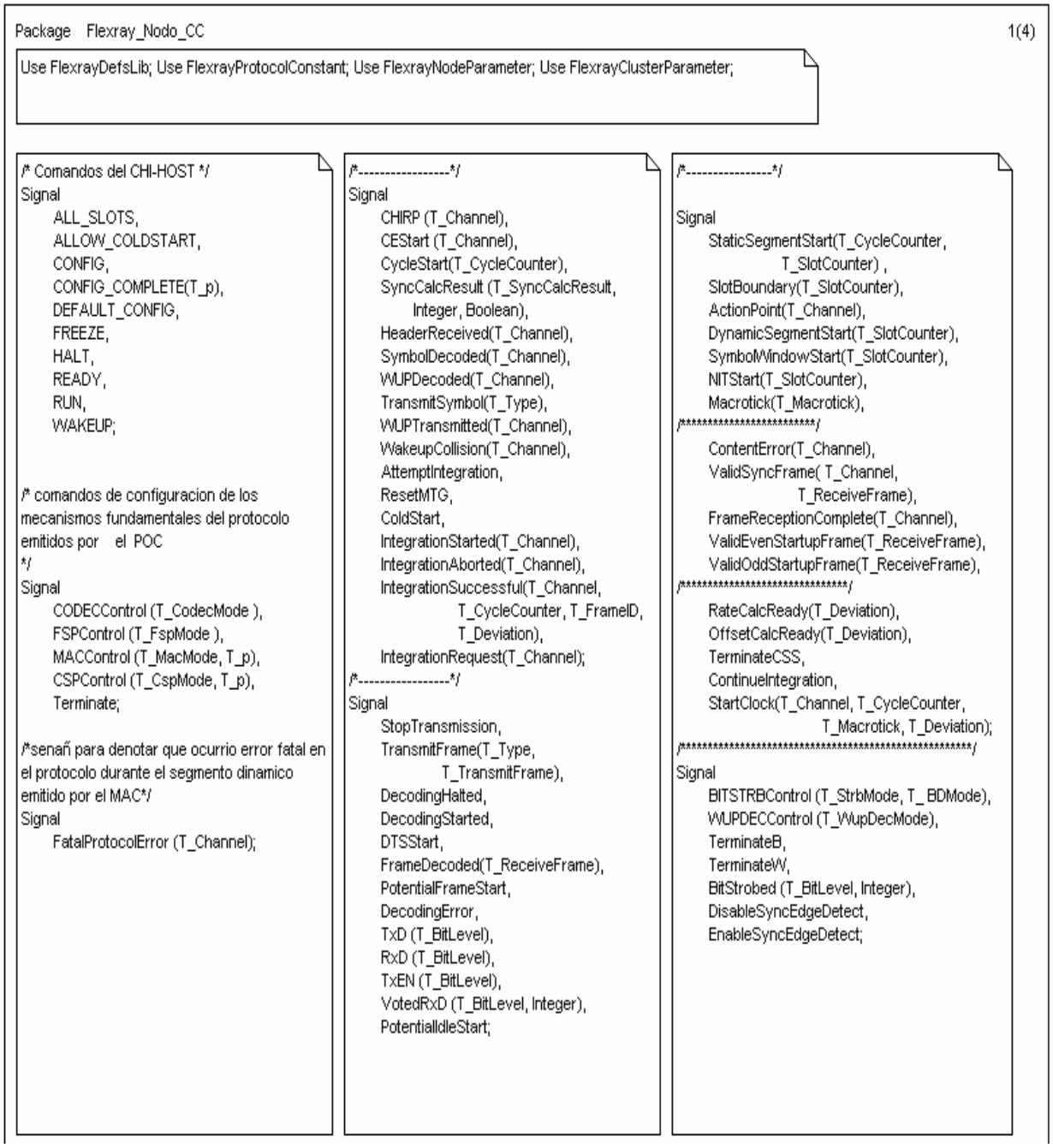


Figura 4.10. Especificación de señales.

4.4.1.1. Especificación de señales

Las señales son la base para establecer la comunicación entre las distintas entidades del sistema, son transportadas a través de las rutas especificadas, canales o rutas de señal. La Figura 4.10 muestra las señales descritas por el protocolo mientras que la Figura 4.11 muestra las señales que no son parte de la especificación y que han sido introducidas con objeto de construir una especificación ejecutable para la especificación del protocolo. Algunas de estas señales son parametrizadas, es decir, transportan información necesaria para procesamiento o control del protocolo.

Package Flexray_Nodo_CC 4(4)

```
/*---------------------------*/
Signal
      uTickSignal (T_Microtick),
      IntReq,
      NodeStatus (T_NodeStatus),
      BIT(T_BitLevel, Pld),
      BD_Pid (Pld, T_Channel),
      ChaSig (T_Channel);

/* Señales y tipos usados para establecer comunicación entre el CHI
y el host*/
Signal
      ConfigRequest(Pld),
      DataRxReport(T_DataRx),
      DataConfig (T_DataConfig),
      BusReady,
      StartupFrame (Pld),
/* Señales para actuaizar el estado del poc al CHI*/
      POCStatusSignal (T_POCStatus),
      ClockCorrectionFailedSignal (Integer),
      RemColdstartAttemptsSignal (Integer),
      ColdstartAbortSignal,
      AllowPassiveToActiveSignal (Integer),
/* Señales para actualizar el contador de ciclos y slots en el cHI*/
      SlotCounterSignal (T_SlotCounter),
      pLatestTxviolationSignal,
      LastDynTxSlotSignal (T_SlotCounter),
      TCHISignal (T_CHITransmission),
/* Seales para el control de ciclos*/
      CycleCounterSignal (T_CycleCounter),
      MacrotickSignal (T_Macrotick),
/* Señales para actualizar las tramas recibidas y el estado del slot en
que fué recibida FDP-CHI */
      SS_RF_Signal (T_SlotStatus, T_ReceiveFrame),
      SlotBoundaryViolationSignal,
/* Señales para indicar los valores de configuración así como los
valores recibidos por cada nodo*/
      Config_Signal (Pld, T_DataConfig),
      Rx_Signal (T_DataRx);

SignalList MacToChi=
       SlotCounterSignal,
      pLatestTxviolationSignal,
      LastDynTxSlotSignal;
```

Figura 4.11. Especificación de señales (continuación).

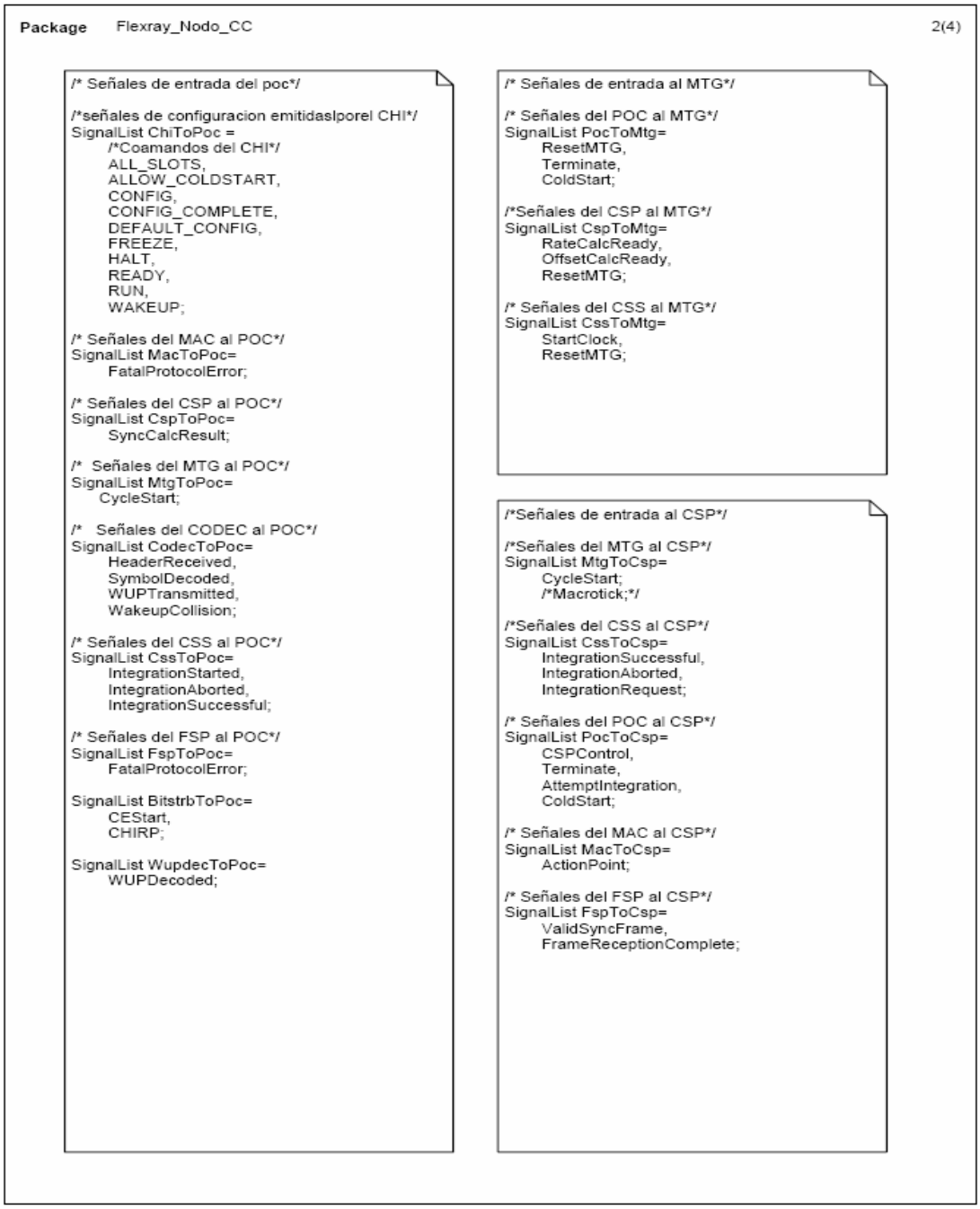


Figura 4.12. Declaración del alfabeto de entrada como listas de señales.

La especificación de señales se realiza de dos formas posibles:

- La especificación de señales que son parte del alfabeto de entrada se realiza directamente entre corchetes asociados a la ruta o vía de comunicación: [HeaderReceived, SymbolDecoded, WUPTransmitted, WakeupCollision].
- Definiendo el alfabeto de entrada como una lista de señales de la forma: SignalList CodecToPoc= HeaderReceived, SymbolDecoded, WUPTransmitted, WakeupCollision; y la especificación de señales se reduce a escribir el nombre que identifica a dicho alfabeto de entrada entre corchetes de la forma: [(CodecToPoc)].

El uso de *SignalList* en la especificación facilita el desarrollo y mantenimiento de especificaciones escritas en SDL, tal que es posible establecer y determinar las dependencias entre el origen y

destino de las mismas de manera sencilla y natural. Las Figuras 4.12 y 4.13 muestran las señales agrupadas por medio de identificadores de tipo *SignalList* los cuales forman parte del alfabeto de entrada de procesos específicos.

El significado de las señales se realiza a detalle en la especificación dinámica del protocolo, debido a que en la especificación estática se hace un uso excesivo de la propiedad de omisión.

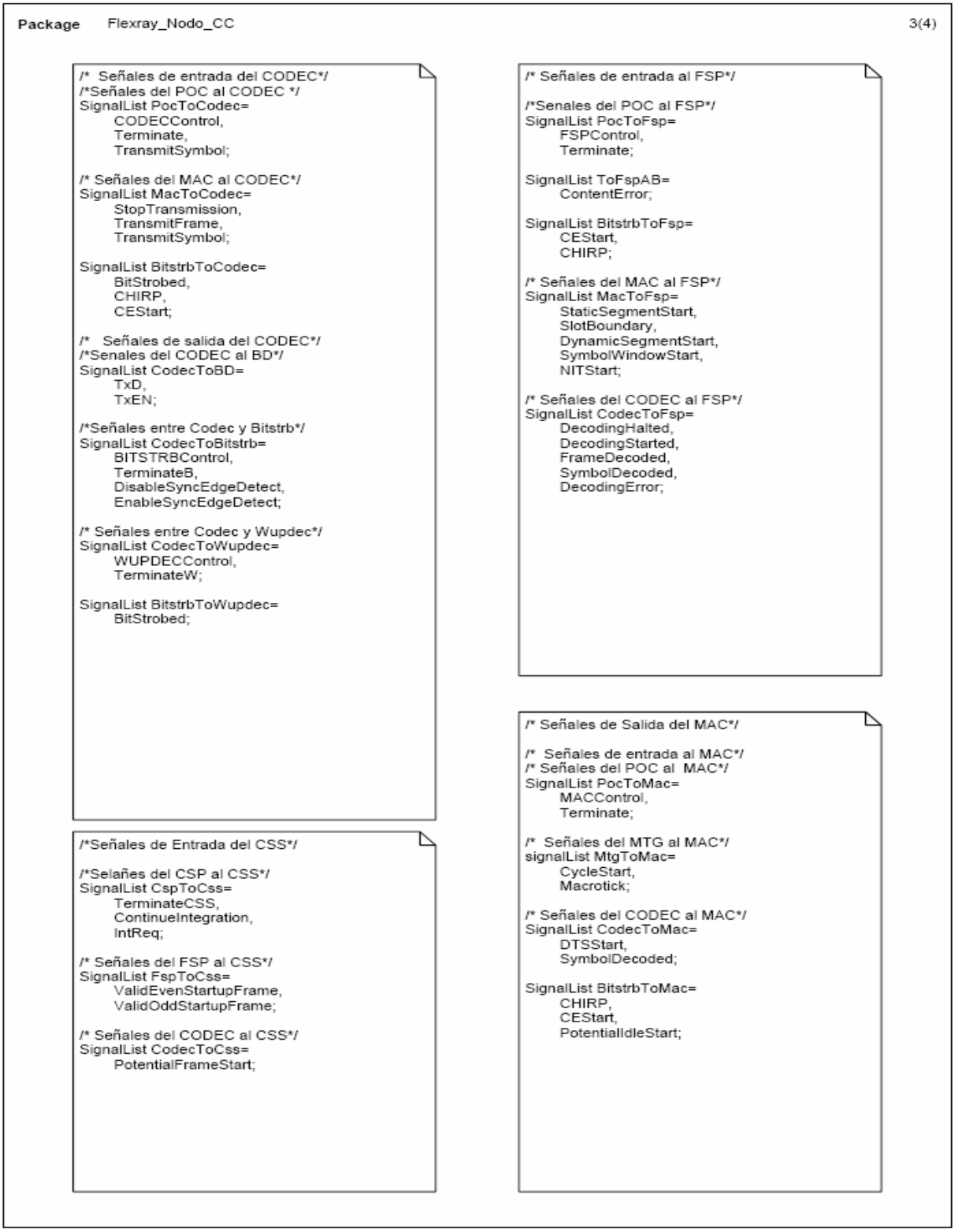


Figura 4.13. Declaración del alfabeto de entrada como listas de señales (continuación).

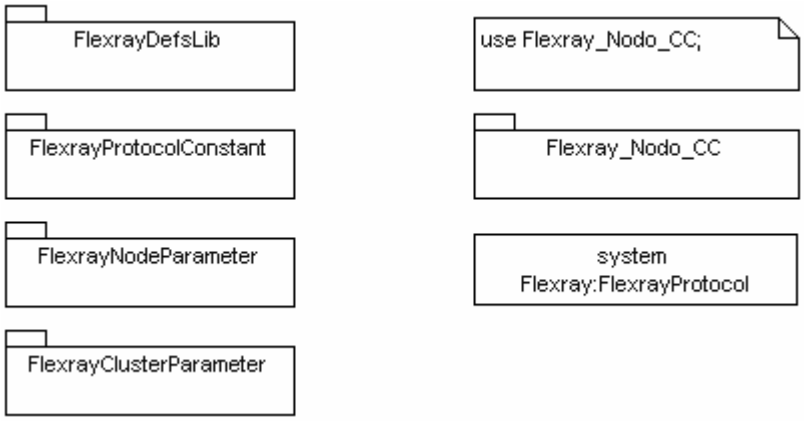


Figura 4.14. Librerías de la especifiación del protocolo Flexray.

4.4.2. Sistema Flexray

Los tipos de datos y constantes son declarados en librerías (Figura 4.14) mientras que los tipos para cada objeto SDL del sistema son declarados en la librería Flexray_Nodo_CC (Figura 4.15).

El sistema consiste de una instancia del tipo FlexrayProtocol, el cual representa el nivel de abstracción más externo. La declaración del sistema Flexray:FlexrayProtocol está formado por los bloques Bus, B_Ctrl y el conjunto de instancias del tipo bloque NODO_T (el número de nodos ya ha sido justificado al inicio en el Subcapítulo 4.1). La Figura 4.16 muestra la relación así como la arquitectura de los bloques a nivel sistema.

4.4.2.1. Bloque BUS

Representa el medio físico para la transmisión de datos, modelando una topología de bus (basada en la conexión punto a punto) mediante el cual se establece un enlace físico de comunicación entre los nodos conectados a una red Flexray.

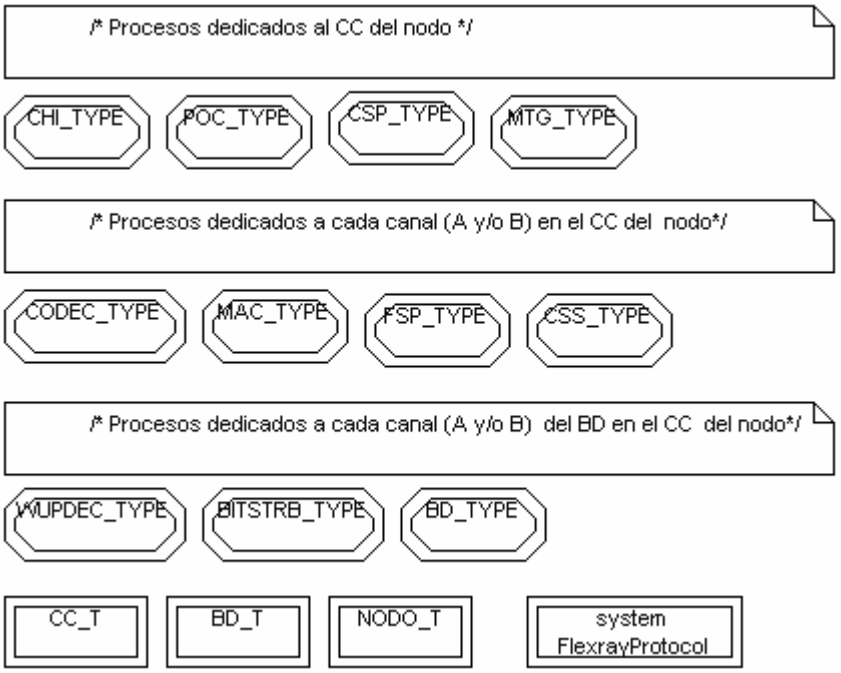


Figura 4.15. Definición de los objetos SDL.

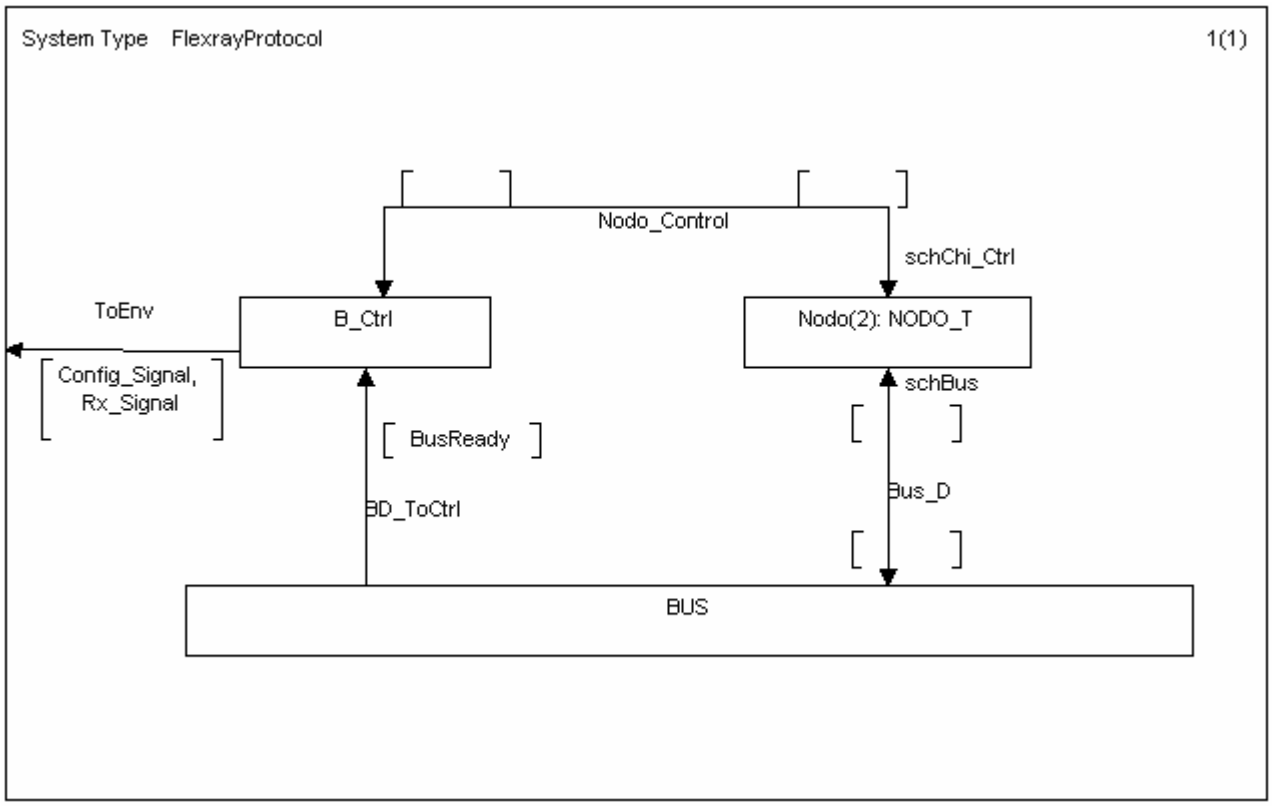


Figura 4.16. Especificación del sistema Flexray.

El bloque BUS se compone de un único proceso llamado Bus_P conectado a los canales Bus-ToCtrl y Bus_D a través de las rutas de señal r1 y r2 respectivamente (Figura 4.17). Cuando se establece un enlace de comunicación por medio de la señal BIT, el BUS distribuye la señal del nodo transmisor a todos los demás nodos que estén conectados a la red, para ello es necesario que conozca la dirección de los procesos receptores; tal información es transportada sobre la señal BD_Pid.

Una vez que el BUS conoce la dirección de todos los procesos destinatarios, se comunica con el bloque B_Ctrl por medio de la señal BusReady para indicarle cuando los nodos del *cluster* están habilitados para establecer un enlace de comunicación entre ellos.

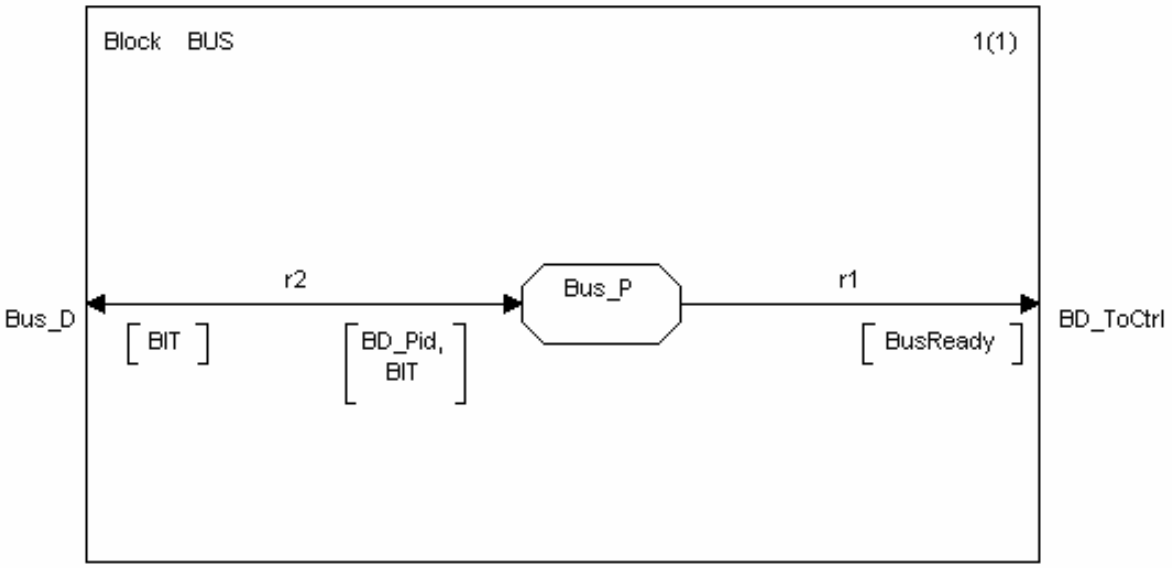


Figura 4.17. Diagrama SDL del bloque BUS.

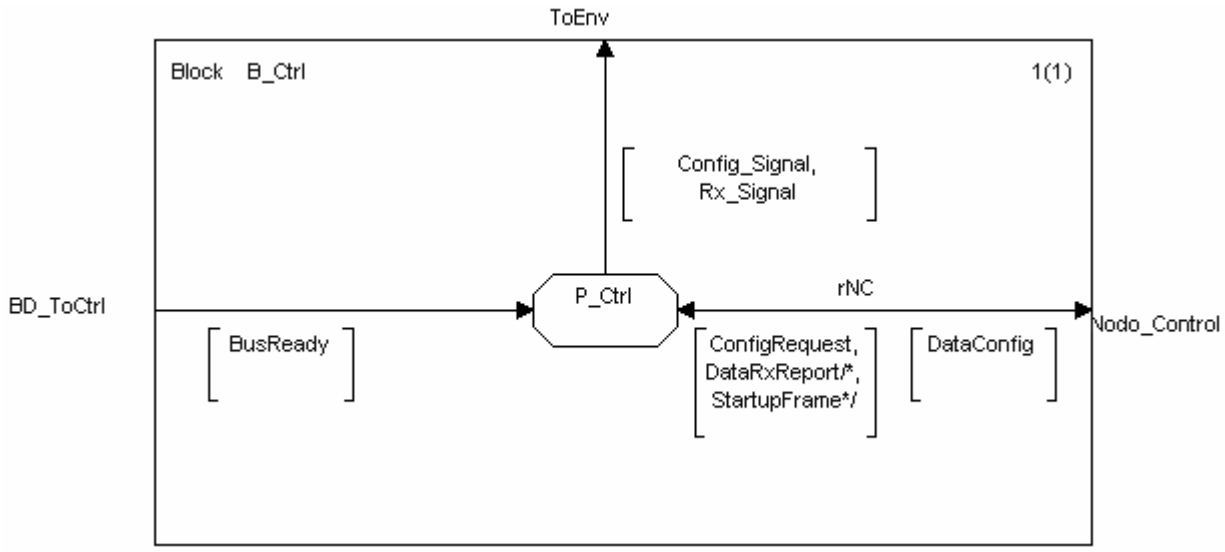


Figura 4.18. Diagrama SDL del bloque B_Ctrl.

4.4.2.2. Bloque B_Ctrl

El bloque B_Ctrl está formado por un solo proceso: P_Ctrl, el cual se conecta a los canales BD_ToCtrl, ToEnv, y el Nodo_Control (Figura 4.18). Su objetivo es establecer los parámetros básicos de configuración para cada nodo, así como monitorizar las tramas recibidas por cada nodo una vez que operan bajo condiciones de sincronización.

Para realizar estas tareas, espera hasta que la señal BusReady haya sido recibida; después de ello, brinda atención a las peticiones de los nodos del *cluster* para que los configure cuando recibe la señal ConfigRequest, que adicionalmente transporta el identificador del proceso que lo solicita con lo que cada nodo es identificado por el B_Ctrl y atiende tales peticiones con la señal DataConfig. Cuando el sistema opera bajo condiciones de sincronización, el B_Ctrl recibe la señal DataRxReport con los datos recibidos por los nodos receptores.

Finalmente, se envían al entorno del sistema los datos de configuración y la información que recibe el B_Ctrl de cada nodo, para su posterior análisis.

4.4.2.3. Instancias del tipo bloque NODO_T

Dado que SDL no permite la creación de bloques de forma dinámica, durante la creación del sistema se realiza una generación múltiple del número de instancias (Subcapítulo 4.1) del tipo bloque NODO_T de forma estática. El inicio de operaciones de cada nodo del sistema comienza con la recepción de la señal DataConfig con los parámetros de configuración que ésta transporta. Cada nodo se conecta con los canales Nodo_Control y Bus_D a través de las compuertas schChi_Ctrl y scHBus respectivamente.

4.4.3. Tipo bloque NODO_T

El tipo bloque NODO_T define la arquitectura de un nodo o ECU y se conforma por un par de instancias de tipo bloque, una para el controlador de comunicaciones (CC:CC_T) y otra para el administrador de bus del canal A (BD_A:BD_T)[8]; también se hace una instancia al bloque idA, cuyo propósito es informar al BD_A que está asociado al canal A.

[8] El *host* para la aplicación no se desarrolla en este proyecto de tesis.

Como se aprecia en la Figura 4.19, el CC y BD_A interactúan entre sí por medio del intercambio de señales a través de canales, cuya especificación es implícita (se han omitido los nombres y listas de señal asociada a cada canal de la especificación), sin embargo, las puertas de enlace han sido explícitamente establecidas.

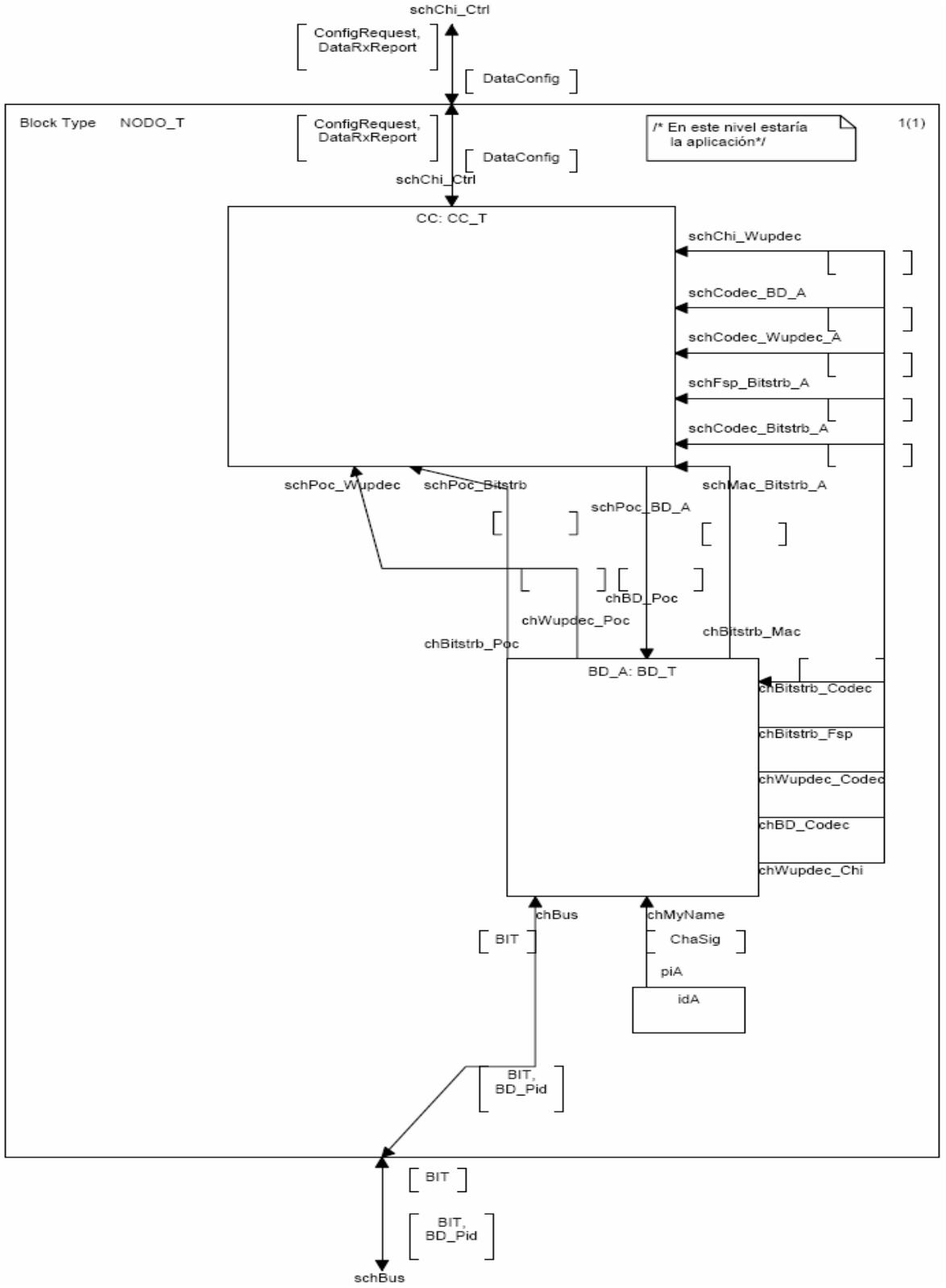


Figura 4.19. Diagrama SDL de la arquitectura de un nodo.

4.4.3.1. Tipo bloque CC_T

La Figura 4.20 describe la arquitectura SDL del controlador de comunicaciones, el cual se compone de las instancias de tipo bloque: CHI:CHI_T y CC_Core:CC_Core_T, adicionalmente muestra los símbolos de los tipo bloques para los objetos CHI_T, CC_Core_T, PMC_T y CH_T. El bloque CHI, cuyo objetivo es servir como un medio de enlace entre el *host* y el controlador de comunicaciones, no realiza esta tarea como tal, más bien realiza las funciones del *host*, como son: establecer los modos de operación del nodo, administración de ranuras, establecer los parámetros básicos de configuración y supervisar el estado del nodo.

El núcleo del protocolo o CC_Core ejecuta las operaciones necesarias para establecer un enlace de comunicación permanente (independientemente de que haya o no mensajes para transmitir), estas tareas son: control del protocolo, temporización, sincronización, codificación, decodificación y procesamiento de tramas.

4.4.3.1.1. Tipo bloque CHI_T

El tipo bloque CHI_T se compone de una sola instancia del tipo proceso CHI_TYPE (Figura 4.21). Las señales de entrada y salida, asociadas a su respectiva compuerta, permiten conocer el estado del nodo y alterar su operación fundamental como respuesta a un evento en el entorno.

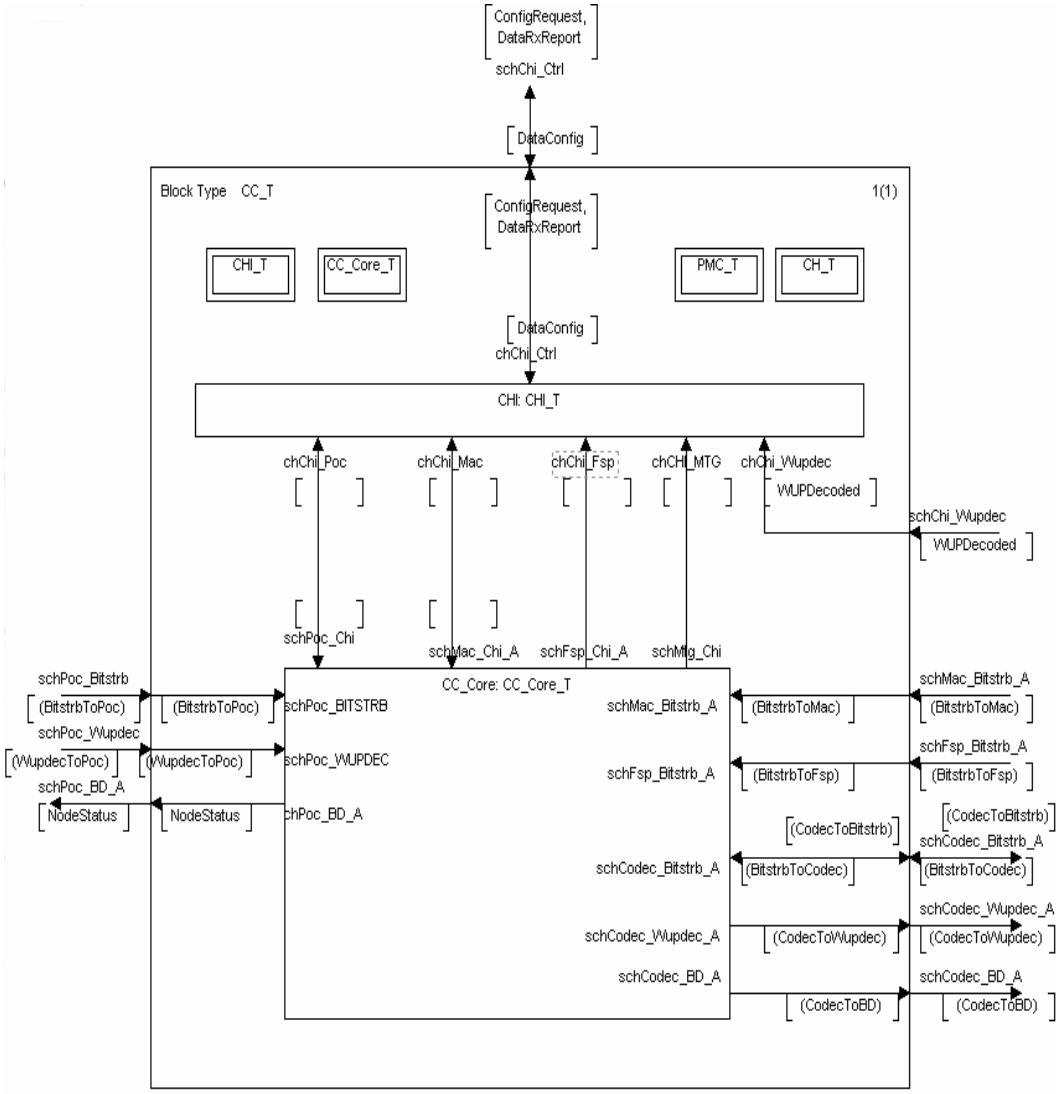


Figura 4.20. Diagrama SDL del controlador de comunicaciones.

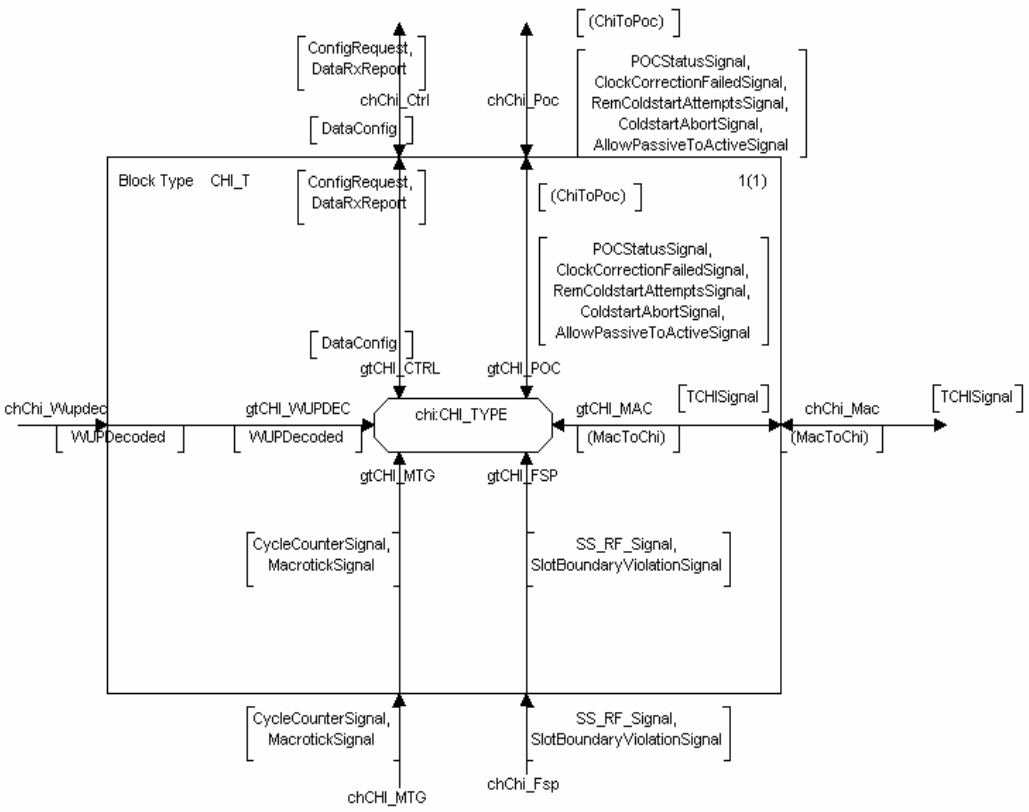


Figura 4.21. Diagrama SDL del tipo bloque CHI_T.

4.4.3.1.2. Tipo bloque CC_Core_T

El funcionamiento primario del protocolo Flexray está contenido en cuatro mecanismos fundamentales: codificación y decodificación, control de acceso al medio, procesamiento de tramas y símbolos, y sincronización del reloj.

Adicionalmente, el proceso chi debe proporcionar los mecanismos para que el *host* interactúe de manera estructurada con estos mecanismos y con el control de operaciones del protocolo. La Figura 4.22 muestra el diagrama SDL del tipo bloque CC_Core_T, el cual representa el núcleo del protocolo. Dado que Flexray define ciertos procesos dedicados a cada canal para incrementar la funcionalidad de dichos procesos y en caso de que la transmisión se realice hasta en dos canales como máximo, se optó por hacer la separación de ellos en diferentes bloques; este objeto está conformado por el siguiente conjunto de instancias:

- *PMC*: Realiza las tareas de la subcapa LLC y se define como PMC:PMC_T. El tipo bloque PMC_T está conformado por tres instancias, poc:POC_TYPE, mtg:MTG_TYPE y csp:CSP_TYPE (Figura 4.20). Además del control del protocolo realizado por el proceso poc, realiza la temporización y sincronización llevado a cabo por los procesos mtg y csp respectivamente.
- *CH_A*: Realiza las tareas dedicadas a cada canal, las cuales son descritas por la subcapa MAC y se define como CH_A:CH_T. Este objeto está conformado por el siguiente conjunto de instancias: mac:MAC_TYPE, fsp:FSP_TYPE, css:CSS_TYPE y codec:CODEC_TYPE (Figura 4.21). Las operaciones que realiza son: preparar la sincronización durante la fase de inicio (*startup*) llevado a cabo por el proceso css, codificación, decodificación y procesamiento de tramas por los procesos codec y fsp respectivamente, el control de acceso al medio es ejecutado por el proceso mac.

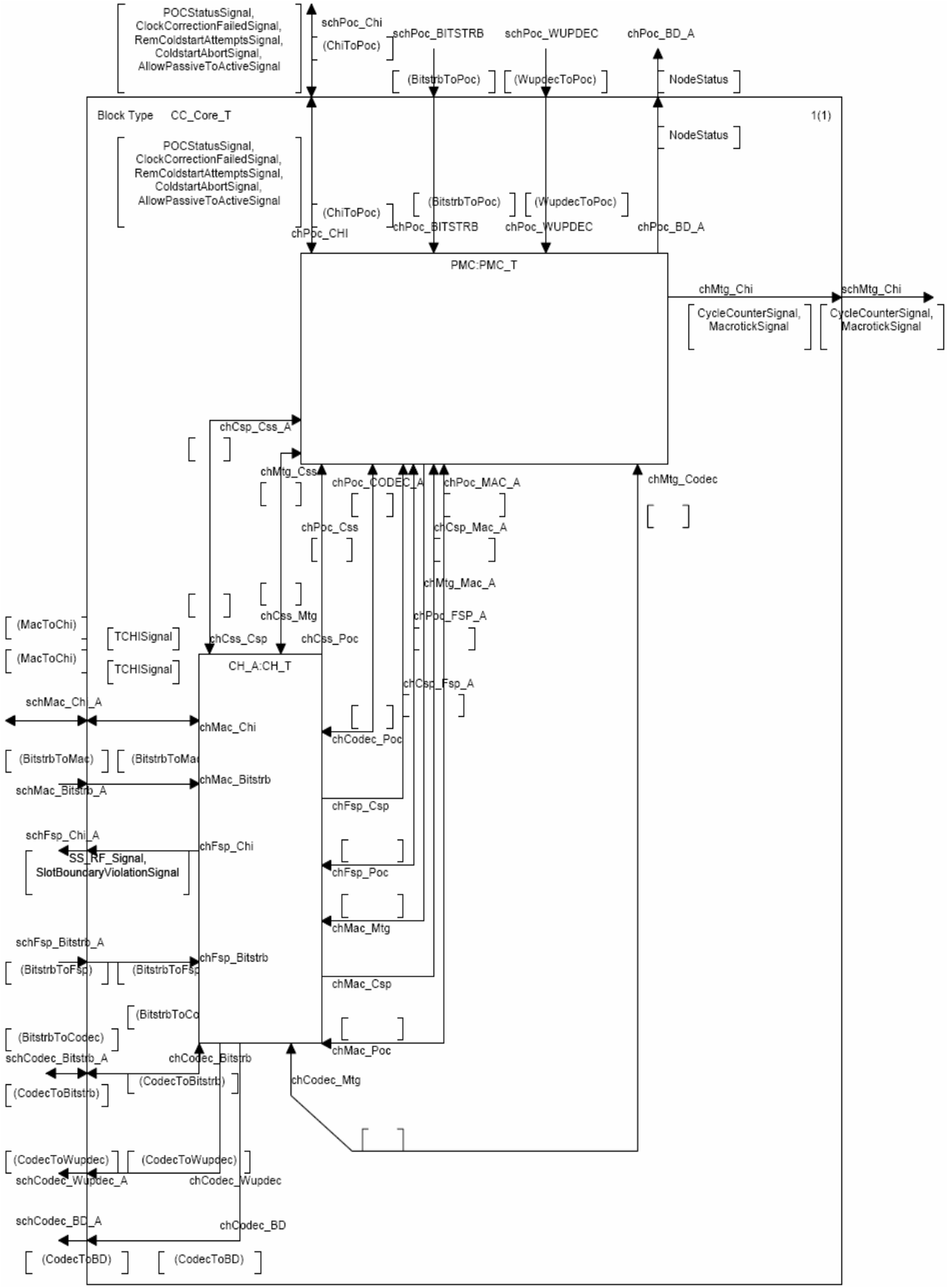


Figura 4.22. Diagrama SDL del tipo bloque CC_Core_T.

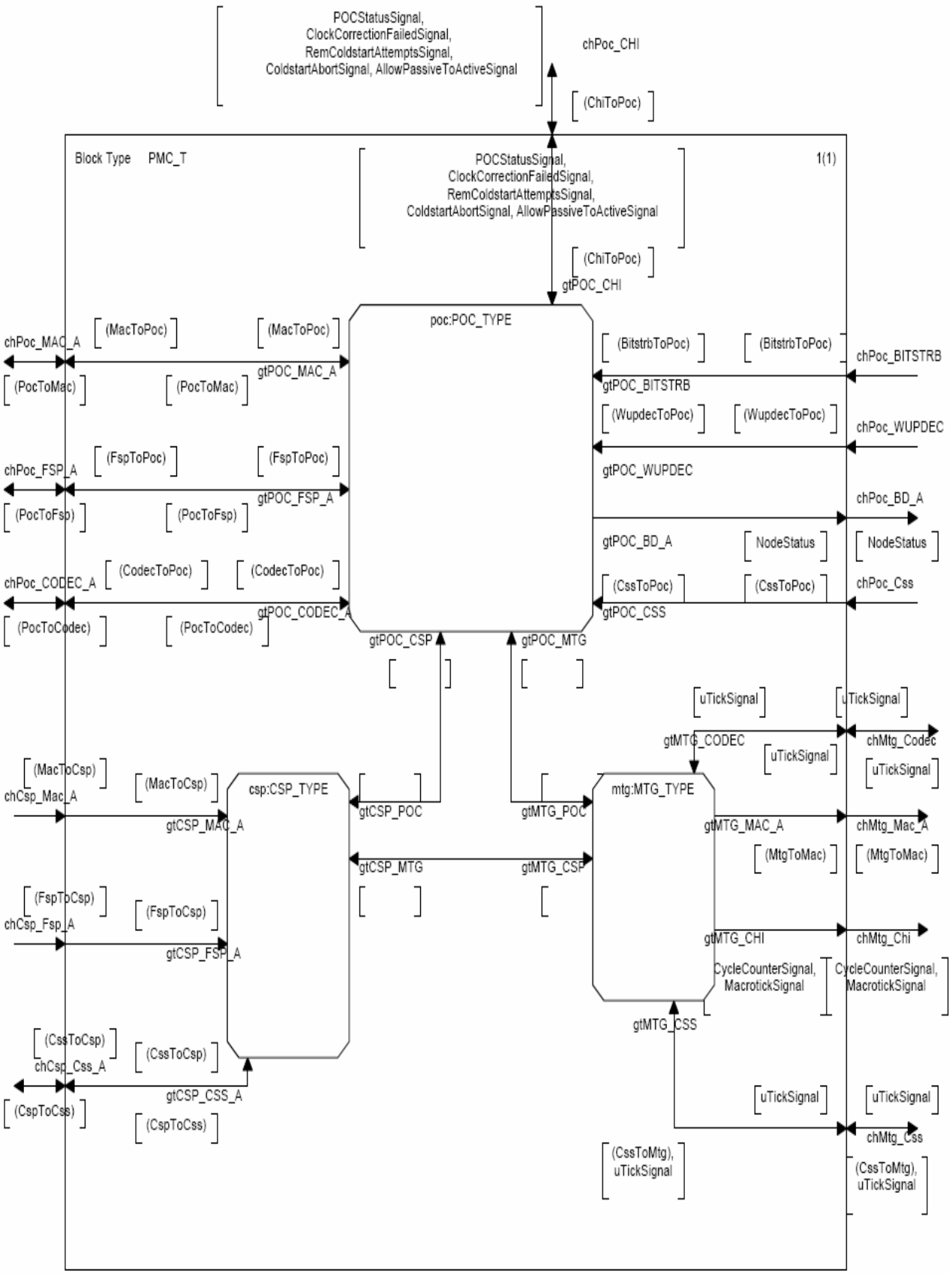


Figura 4.23. Diagrama SDL del tipo bloque PMC_T.

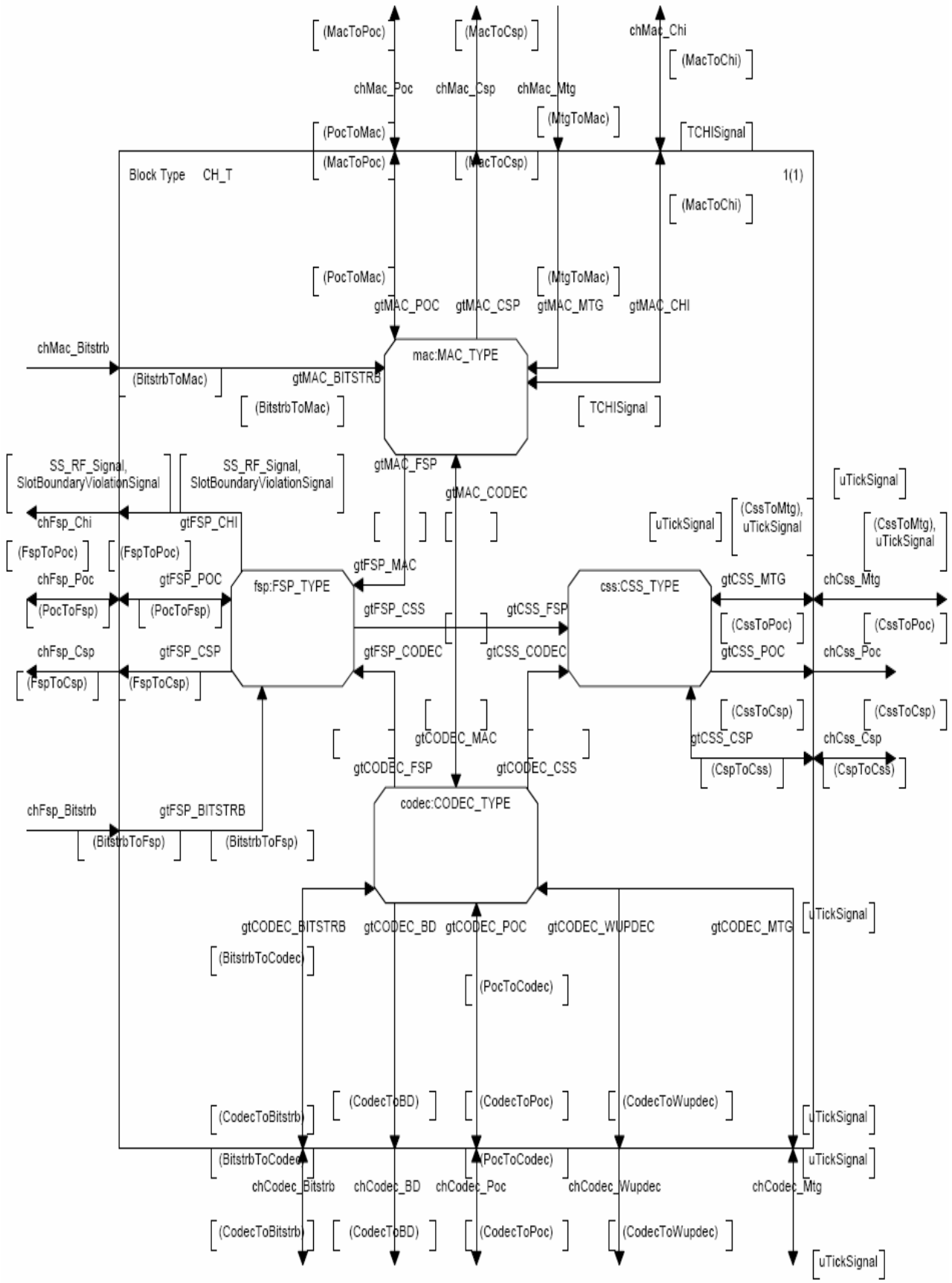


Figura 4.24. Diagrama SDL del tipo bloque CH_T.

4.4.3.2. Tipo bloque BD_T

El administrador del bus, denotado por el objeto BD_T de la Figura 4.25, representa la capa de enlace entre el controlador de comunicaciones del nodo con la capa física, y está conformado por el siguiente conjunto de instancias:

- *bd*: Se encarga de monitorizar las líneas de transmisión; tan pronto detecte actividad en el bus abandona el modo inactivo para ejecutar la transmisión o recepción y muestreo de CEs de manera constante hasta que se le indica que no hay actividad en el bus.
- *bitstrb*: Ejecuta el filtrado de las señales recibidas para determinar el valor del bit en el punto de muestreo previamente establecido, así como el mecanismo que sincroniza el temporizador para el muestreo de bit. Adicionalmente detecta el inicio y final de cada trama.
- *wupdec*: Realiza la detección de patrón WUP durante la fase de inicio del *cluster*.

Este conjunto de instancias son declaradas como bd:BD_TYPE, bitstrb:BITSTRB_TYPE y wupdec:WUPDEC_TYPE.

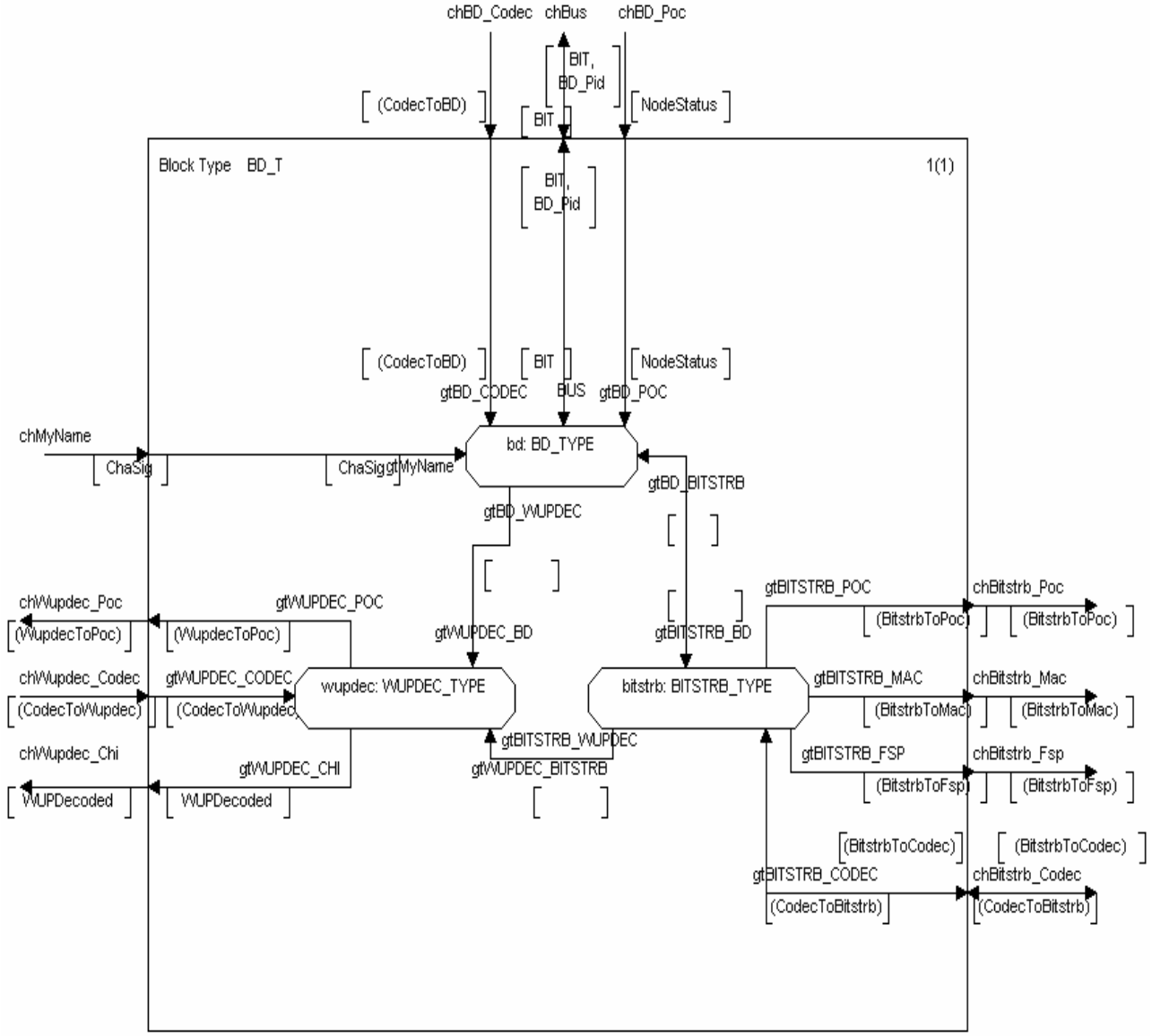


Figura 4.25. Diagrama SDL del tipo bloque BD_T.

4.5. Especificación dinámica

La especificación dinámica describe el comportamiento del sistema y está contenida dentro de los procesos que lo conforman. Como se menciona en el Apartado 4.4.1, los objetos SDL de los procesos de la especificación dinámica se han declarado usando un estilo de representación remota, basada en tipos y un estilo de especificación remota; sin embargo, a lo largo de este subcapítulo, por simplicidad serán referidos como "proceso" omitiendo la palabra "tipo" antepuesta (independientemente del estilo de representación usado). Dado que la funcionalidad de los procesos ya ha sido definida en la especificación estática, a continuación se detalla el funcionamiento interno basado en máquinas de estado.

Debido a que la especificación de las CEFSMs de los procesos es muy extensa, a excepción de los procesos P_Ctrl, Bus_P, la descripción de cada proceso se realiza de la siguiente forma:

- *Se usará la primera página de la especificación formal de los procesos*: Esto es debido a que la primera página de cada proceso incluye las definiciones de las puertas de enlace, listas de señales y declaración de variables.
- *Los procesos serán descritos con diagramas de visión de estados*: Son diagramas de comportamiento en SDL en los que se eliminan todos los símbolos excepto los de los estados y las señales de entrada (también puede contener los símbolo de las señales de salida). Simplifica la representación del interior de los procesos, ocultando información sobre las tareas realizadas en las transiciones, tal que se pueda visualizar la evolución de procesos complejos en un solo diagrama; lo que es válido para el diseño y formalización de la especificación de sistemas y soportado por la metodología SPECS[9], la cual no se utiliza con formalidad en el presente proyecto de tesis.

Para una revisión exhaustiva de la especificación formal del protocolo Flexray, se entregan los siguientes documentos conjuntamente con este trabajo de tesis:

- *Especificación formal en formato *.cbf*: Formato de la especificación generada por la herramienta CinderellaSDL.
- *Especificación formal en formato *.pdf*: En caso que el lector no disponga de la herramienta CiderellaSDL, pueda acceder a la especificación formal.
- *Diagrama MSC en formato *.pdf*: Los diagramas MSCs (*Message Sequence Chart*) representan el resultado de la prueba (ejecución) de la simulación como una secuencia de mensajes ordenada en el tiempo, los cuales se han usado durante la evolución del proyecto y se abordan en el Capítulo 5 para documentar la evaluación de resultados.

4.5.1. Proceso Bus_P

Una vez que el símbolo de inicio es ejecutado, el proceso entra al estado wait_for_First_BD_id en espera de que llegue la primera señal BD_Pid con el identificador del proceso receptor del BD; después de recibir el primer identificador, el proceso cambia al estado wait_for_second_BD_id en espera del segundo identificador. Cuando ambos identificadores del BD de cada nodo han sido recibidos, se envía la señal BusReady al proceso P_Ctrl indicando que todos los nodos del cluster están habilitados para realizar un enlace de comunicación. Cuando el proceso cambia al estado point_to_point en espera de un CE y ocurre esto último, el CE es distribuido a un nodo destinto al nodo transmisor.

[9] La metodología SPECTS está constituida de tres actividades conocidas como CR&F: clasificación, estructurar un documento informal de un sistema; rigurosidad, generar información que permita producir una especificación formal; y formalización: producir una especificación formal de un sistema [25].

Para sistemas con muchos nodos, lo óptimo es generar una tabla que contenga los identificadores de los BD de lo nodos receptores; dado que en el presente trabajo de tesis se ha diseñado una red para dos nodos, es suficiente la descripción mostrada en la Figura 4.26.

4.5.2. Proceso P_Ctrl

El proceso P_Ctrl se muestra en la Figura 4.27; tras ejecutar el símbolo de inicio permanece en el estado wait_for _bus_ready retrazando cualquier petición de configuración hasta que sea recibida la señal BusReady y cambia al estado wait_for_first_config_request en el que dará atención al primer nodo que lo requiere cuando recibe la señal ConfigRequest, tras establecer los parámetros básicos los envía al nodo en cuestión por medio de la señal DataConfig, cuando alcanza el estado wait_for_second_config_request da atención a la petición de configuración del segundo nodo de manera similar al primer caso. En ambos casos, los valores con que son configurados cada nodo son enviados al entorno del sistema con la señal Config_Signal. Cuando el proceso alcanza el estado wait_for_RxDatas, recibirá los mensajes que cada nodo haya decodificado y serán enviados a su entorno con auxilio de las señales DataRxReport y Rx_Signal respectivamente.

Similar al caso anterior, este proceso se ha descrito exclusivamente para dos nodos, sin embargo, debido a que estos parámetros no son configurables estáticamente (definidos como constantes) como la mayoría de ellos y dado que cada nodo de un sistema real es configurado en tiempo de diseño, es deseable que cada nodo pueda comunicarse con una PC para realizar la configuración; del mismo modo, los datos a transmitir son tomados del exterior en tiempo real.

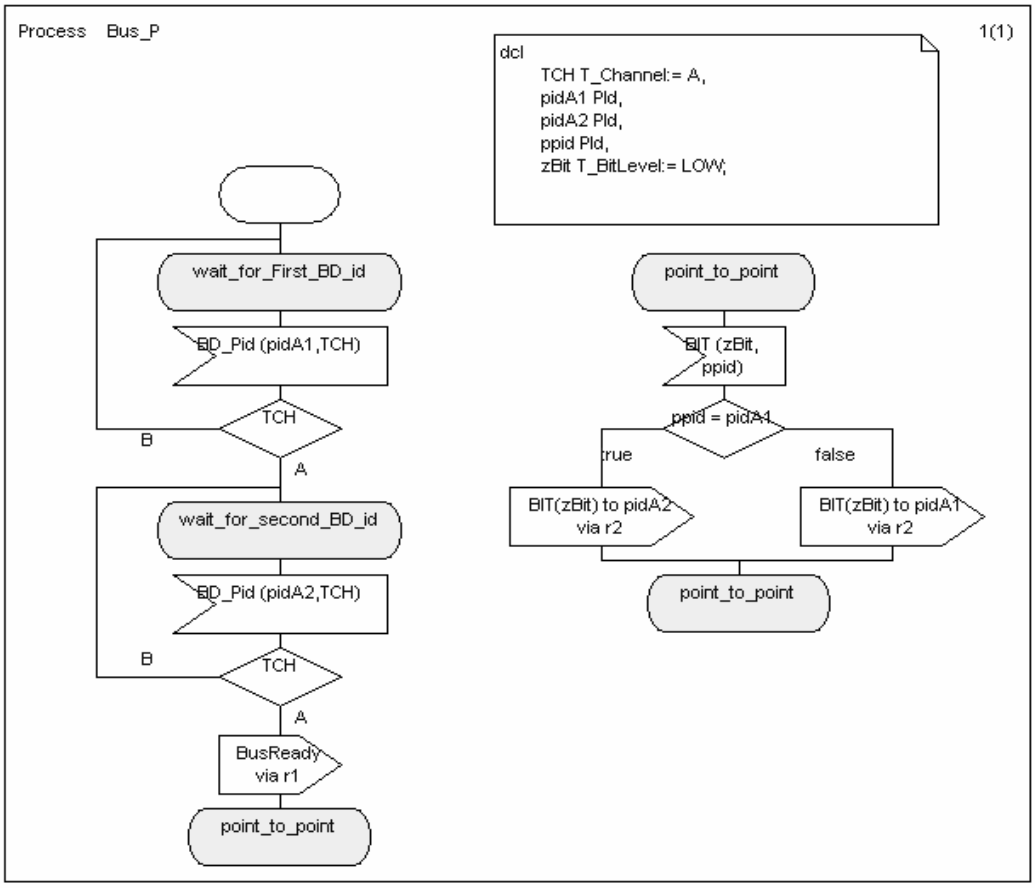


Figura 4.26. Descripción del proceso Bus_P.

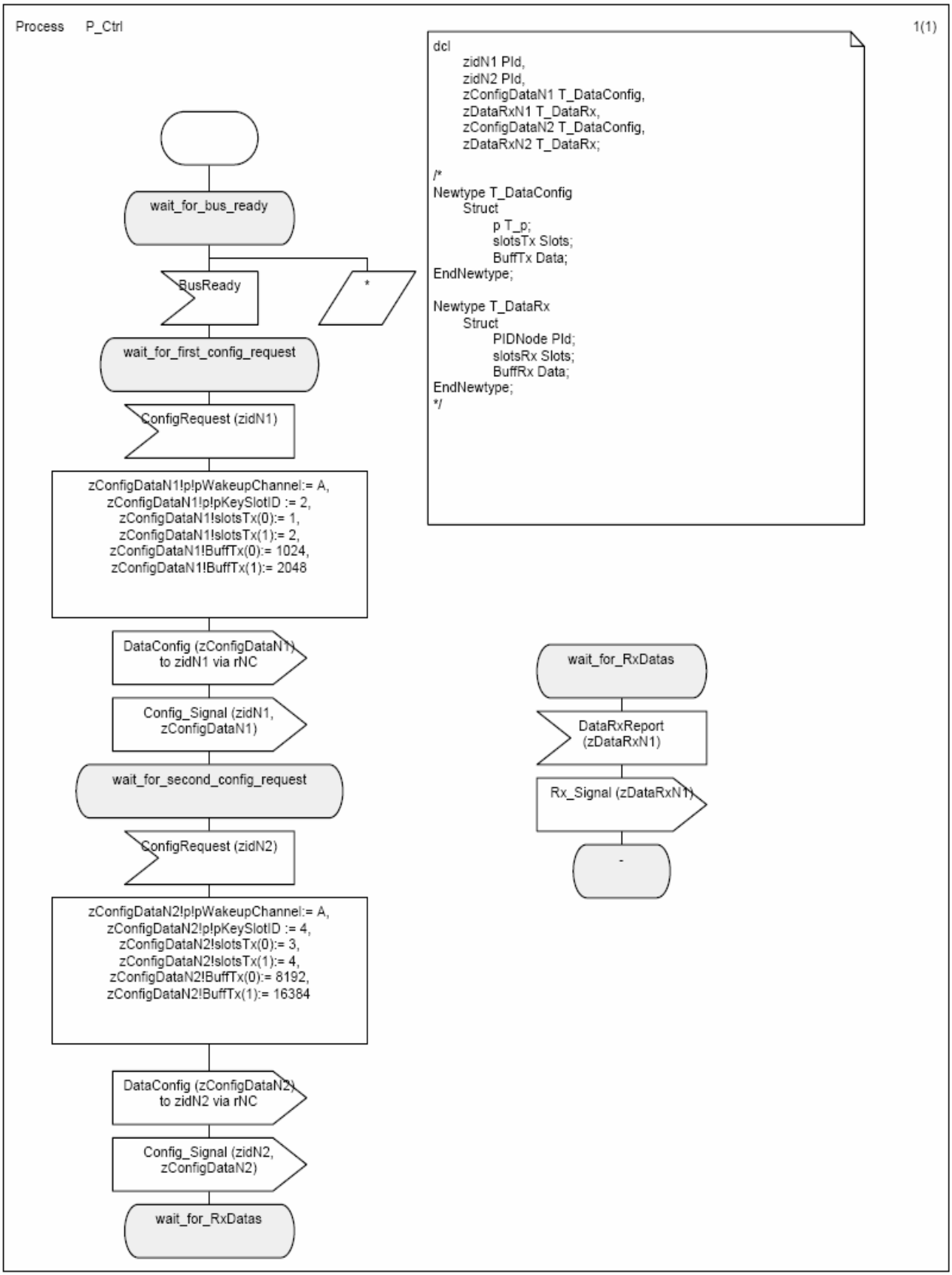


Figura 4.27. Descripción del proceso P_Ctrl.

4.5.3. Proceso chi

El proceso chi se muestra en las Figuras 4.28 y 4.29; el cual inicia con la transmisión de la señal ConfigRequest a través de la compuerta gtCHI_CTRL solicitando los parámetros de configuración para ese nodo y entra al estado Wait_For_Data_Configuration, estado en el que permanece hasta que llega la señal DataConfig con los parámetros solicitados e inmediatamente ordena al proceso poc para que termine la configuración del nodo. Cuando el proceso recibe la señal POCStatusSignal en el estado Wait_For_Config_Complete_Confirmation ordena al proceso poc a realizar una de las siguientes tareas según el estado de la variable zAttachedCHAwaken:

- *False*: Indica que los canales a los que se encuentra conectado el nodo no han sido inicializados (no ha ocurrido ninguna transmisión de CEs) por lo que envía la orden WAKEUP por la compuerta gtCHI_POC indicando al poc que dé comienzo a la fase de inicio del *cluster*; si el resultado de esta operación es COLLISION_UNKNOWN, ordena que repita la operación y en cualquier otro caso emitirá una orden al poc para que ejecute la fase de inicio del sistema.
- *True*: Significa que el *cluster* está activo y por tanto envía la orden RUN por la compuerta gtCHI_POC indicando al poc que ejecute la fase de inicio del sistema

Cuando el proceso chi alcanza el estado POC_Error_Degration_Model_Run, cada nodo en el *cluster* ya se ha sincronizado y preparado para realizar la transmisión y recepción de tramas de manera continua. Una vez que el proceso chi está consciente de que el proceso poc está ejecutando el modelo de degradación de errores, puede recibir la señal SS_RF_Signal con el mensaje de la trama recibida y el estado de la ranura en que se ha recibido tal trama; información que es enviada conjuntamente con el nodo receptor al proceso P_Ctrl a través de la señal DataRxReport vía gtCHI_CTRL.

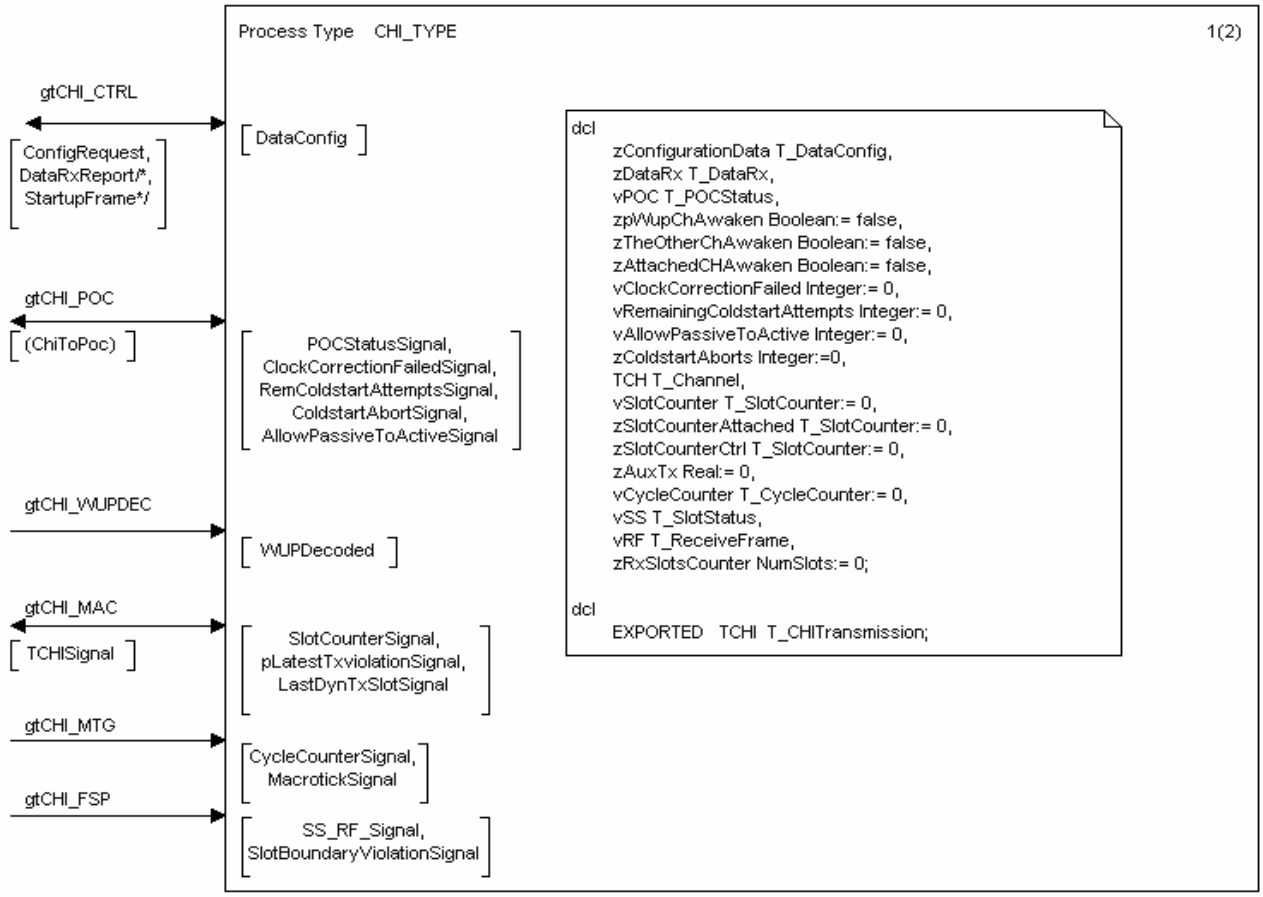


Figura 4.28. Descripción del proceso CHI_TYPE.

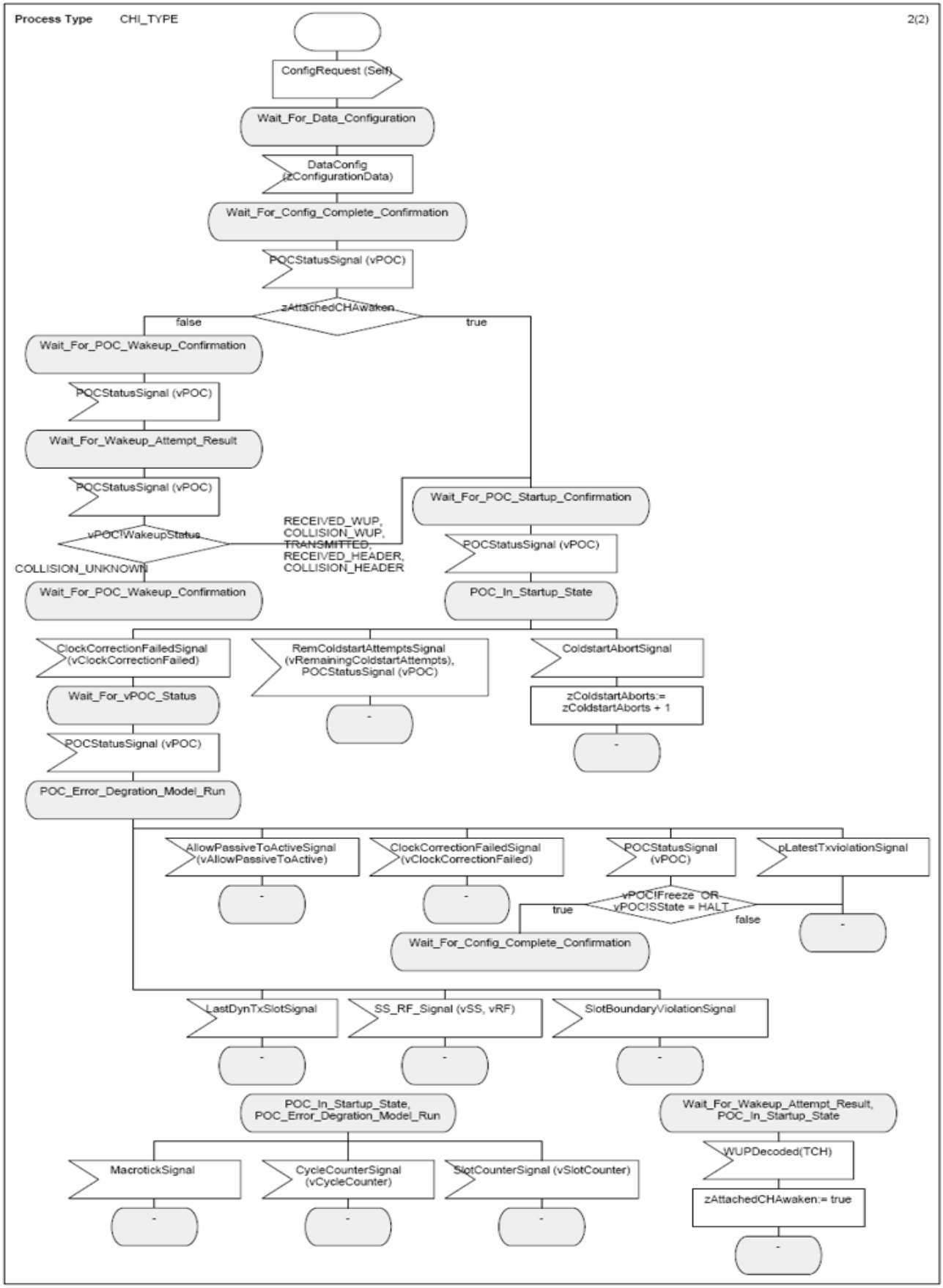


Figura 4.29. Descripción del proceso CHI_TYPE (continuación).

Cuando se recibe la señal POCStatusSignal y el proceso cambia al estado Wait_For_Config_-Complete_Confirmation, significa que el nodo ha perdido la sincronía con el *cluster* y por tanto debe reiniciar la configuración del nodo enviando las órdenes apropiadas al proceso poc. La administración de ranuras y mensajes es ejecutada en cualquiera de los estados POC_In_Startup_State o POC_Error_-Degration_Model_Run con la entrada de las señales CycleCounterSignal o SlotCounterSignal, cuya descripción operacional depende del modo definido para las ranuras denotado por la variable vPOC!SlotMode:

- *SINGLE*: Indica que el nodo sólo realizará transmisiones en una sola ranura.
- *ALL_T*: En este modo, todos los nodos son habilitados para realizar transmisión de tramas en todas las ranuras asignadas; se habilita con la orden ALL_SLOTS una vez que los nodos han alcanzado el estado POC_Error_Degration_Model_Run en el proceso chi.

Adicionalmente, si el proceso chi se encuentra en los estados Wait_For_Wakeup_Attempt_-Result o POC_In_Startup_State tras la entrada de la señal WUPDecoded, habilita el inicio del sistema con la orden ALLOW_COLDSTART; cuando los nodos conectados a un sólo canal entran a esta fase (*startup*), automáticamente estarán habilitados; los nodos duales pueden entrar en esta fase pero sólo la iniciarán cuando reciban la orden ALLOW_COLDSTART después de que sus dos canales asociados hayan sido inicializados correctamente, es decir, que el proceso chi haya recibido la señal WUPDecoded de cada canal.

La Tabla 4.2 lista las órdenes enviadas por el proceso chi al poc para establecer los modos fundamentales de operación del nodo. Formalmente estas órdenes son emitidas por el *host* que contiene la aplicación; dado que en la especificación del protocolo son tratados como si fueran emitidas por el proceso chi, en este documento de tesis se maneja de la misma forma.

Tabla 4.2. Órdenes que emite el proceso chi.

Orden	Descripción	Procesamiento
CONFIG	Prepara al poc para iniciar la configuración	Inmediato
CONFIG_COMPLETE	Indica al poc que termine la configuración	Inmediato
DEFAULT_CONFIG	Indica el inicio de configuración básico	Inmediato
ALL_SLOTS	Habilita la transmisión en todos las ranuras del nodo	Al final de ciclo
ALLOW_COLDSTART	Habilita el comienzo de la fase de inicio del sistema	Inmediato
FREEZE	Indica un alto en todas las operaciones	Inmediato
HALT	Indica un alto en todas las operaciones	Al final de ciclo
READY	Habilita al poc para iniciar operaciones	Inmediato
RUN	Habilita el inicio sistema	Inmediato
WAKEUP	Inicio del *cluster*	Inmediato

Es importante notar que todos los nodos *coldstart* están habilitados para iniciar el sistema y de este modo convertirse en el nodo principal o maestro (el concepto de nodo maestro y esclavo no está definido en el protocolo). Esta problemática se resuelve de manera natural, siguiendo el concepto de distribución de ranuras, en donde cada nodo tiene un identificador para cada ranura asignada y cuando el nodo *coldstart* habilite la primera ranura configurada para transmitir tramas de sincronización, éste puede convertirse en el nodo principal.

4.5.4. Proceso poc

En este apartado se describe la forma de establecer los modos de operación de los mecanismos fundamentales del protocolo, los cuales necesitan realizar el proceso de sincronización en el inicio del cluster, inicio del sistema y reintegración (Incisos 4.5.4.1-2).

Las Figuras 4.30 y 4.31 muestran la descripción del proceso poc mediante diagramas de visión de estados. Una vez ejecutado el símbolo de inicio, el proceso poc establece la configuración por defecto y entra al estado default_config en espera de la orden CONFIG y entrar al estado config; en este estado prepara a los mecanismos del protocolo y al recibir la orden CONFIG_COMPLETE pasa al estado ready e inicia sus operaciones de acuerdo a las siguientes órdenes que recibe:

- *WAKEUP*: Llama al procedimiento WAKEUP para iniciar el *cluster* y regresa al mismo estado.
- *RUN*: Prepara a los mecanismos del protocolo para iniciar el sistema y llama al procedimiento STARTUP para coordinarlo. Si esta tarea es realizada satisfactoriamente, el proceso cambia al estado normal_active; pero si se excede el número de intentos configurado, volverá al estado ready tras recibir la orden READY y reiniciará esta operación.
- *CONFIG*: Esta orden habilita al proceso chi para cambiar la configuración del nodo (si es requerido) forzando al proceso poc a reentrar al estado config.

Una vez que el nodo se ha sincronizado, el proceso poc inicia el modelo de degradación de errores, conformado por los siguientes estados:

- *normal_active*: Cuando se alcanza este estado, el nodo está habilitado para el envío y la recepción de tramas, además de que evalúa el estado de la sincronización del nodo al final de cada ciclo de comunicación; si existe un factor de perdida de sincronización durante gMaxWithoutClockCorrectionPassive ciclos par-impar, el modelo se degradará cambiando al estado normal_passive.
- *normal_passive*: Su funcionamiento es similar al estado normal_active, con la diferencia de que en este estado se deshabilita la transmisión. Si el nodo se mantiene resincronizado durante los siguientes pAllowPassiveToActive -1 ciclos par-impar, regresará al estado normal_active o de lo contrario se mantendrá en el mismo estado.
- *halt*: Si el nodo no puede mantener la sincronización con el *cluster* durante gMaxWithoutClockCorrectionFatal ciclos, el proceso cambiará al estado halt, estado en el que sólo podrá reiniciar operaciones tras la recepción de la orden DEFAULT_CONFIG.

La evaluación del estado de la sincronización del nodo sólo se realiza en los estados normal_active y normal_passive tras recibir la señal SyncCalcResult; adicionalmente, el proceso poc establece los modos de operación de los mecanismos fundamentales del protocolo a través de las señales CODECControl (T_CodecMode), FSPControl (T_FspMode), MACControl (T_MacMode, T_p) y CSPControl (T_CspMode, T_p) en función del estado del nodo; estas señales transportan valores que indican el modo en que cada mecanismo debe operar. La Figura 4.31 muestra un conjunto de señales definidos de la siguiente manera:

- *Comandos asíncronos*: Pueden ser recibidos en cualquier instante, pero su procesamiento se realiza al final del ciclo.
- *Soporte*: Habilitan o aceleran el desarrollo de las fases de inicio del *cluster* y sistema.
- *Órdenes de establecimiento*: Cuando una de estas señales es recibida, se atribuye un error en alguno de los mecanismos del protocolo.

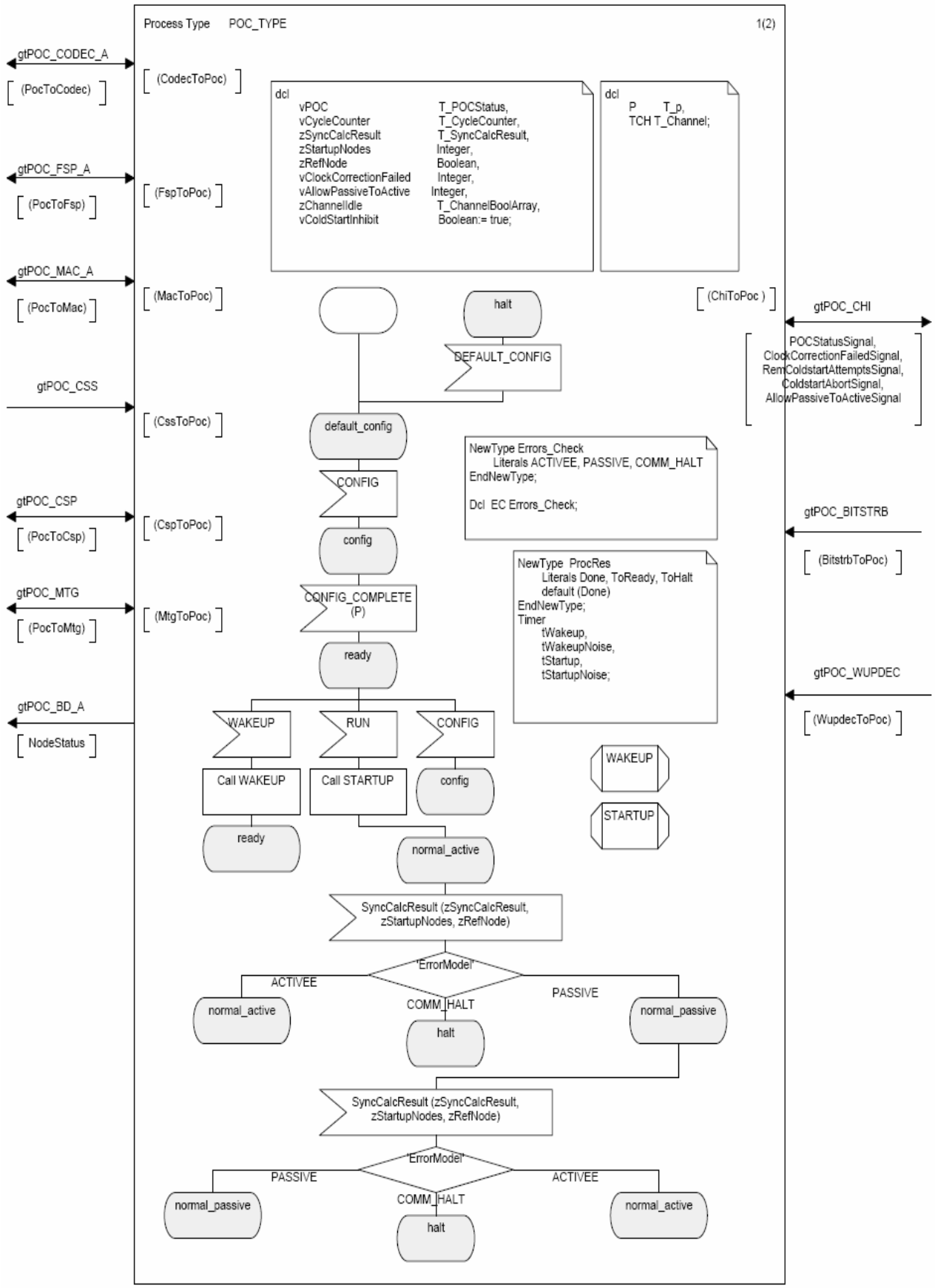


Figura 4.30. Descripción del proceso POC_TYPE.

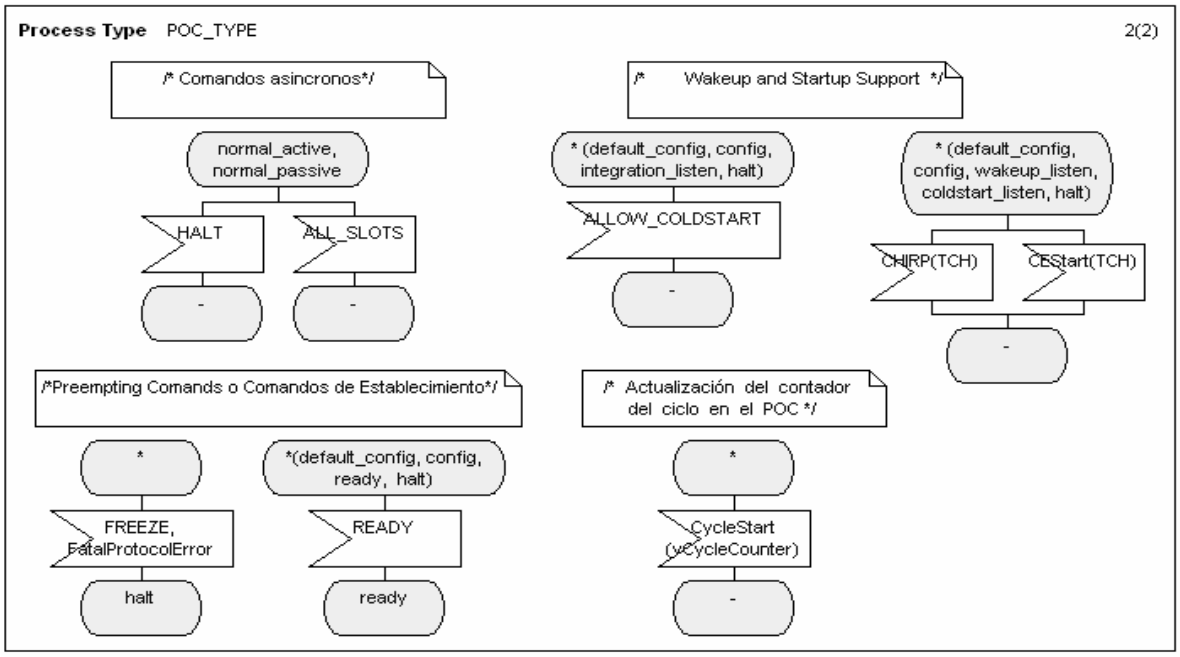


Figura 4.31. Descripción del proceso POC_TYPE (continuación).

4.5.4.1. Procedimiento de inicio del cluster

El procedimiento de inicio del *cluster* establece los modos de operación apropiados de los procesos codec, fsp, mac y csp e inicializa los temporizadores tWakeup, para habilitar en el nodo un inicio rápido del *cluster* en ausencia de ruido, y tWakeupNoise, para dar soporte en presencia de ruido. Una vez que alcanza el estado wakeup_listen, monitoriza al proceso codec y al BD; si se consume alguno de los temporizadores, envía la señal TransmitSymbol(WUP) al codec para que inicie la transmisión del patrón WUP y cambia al estado wakeup_send, estado en el que espera la confirmación de esta transmisión mediante la señal WUPTransmitted(TCH); en caso de existir alguna colisión pasa al estado wakeup_detect (Figura 4.32).

Cualquier resultado causará que el proceso poc entre en la fase de inicio del sistema, a excepción de COLLISION_UNKNOWN que lo obliga a realizar un nuevo intento.

4.5.4.2. Procedimiento de inicio del sistema

Durante la fase de inicio de sistema, los nodos de una red Flexray no pasan por los mismos estados, éstos pueden ser (Figuras 33 y 34):

- Después de que un nodo *coldstart* alcanza el estado coldstartt_listen, permanece monitorizando los canales a los que está conectado; si no se detecta algún CE, ordena el inicio del sistema y cambia al estado coldstart_collision_resolution en el que permanece los siguientes cuatro ciclos; cuando alcanza el estado coldstart_consistency_check espera recibir al menos un par de tramas válidas en los dos ciclos siguientes y entrará en operación. Si el nodo detecta que aún no hay nodos *coldstart* conectados al *cluster*, entra al estado coldstart_gap en el que se interrumpe toda actividad y espera al siguiente siclo para reiniciar un nuevo intento *coldstart*, regresa al estado coldstart_collision_resolution. Cualquier otra condición causa una interrupción o un nuevo intento de inicio de sistema.
- Una vez que un nodo *coldstart* alcanza el estado coldstartt_listen permanece monitorizando los canales a los que está conectado; si se detecta algún CE, intentará integrarse al nodo *coldstart* transmisor. El nodo puede alcanzar el estado initialize_schedule por dos vías:

a partir del estado coldstart_listen o integration_listen tras la recepción de la señal IntegrationStarted (TCH). Una vez que el nodo haya recibido un par de tramas válidas en los siguientes dos ciclos habrá generado su propio esquema de temporización y sincronización adoptando el esquema del nodo principal; el css enviará la señal IntegrationSuccessful (TCH) al proceso poc y entra al estado integration_coldstart_check para iniciar la transmisión de tramas de sincronización; si continúa recibiendo tramas válidas durante los siguientes dos ciclos cambiará al estado coldstart_join en el que entrará en operación si continúa recibiendo tramas válidas en los siguientes tres ciclos y la sincronización no genera error alguno. En caso contrario abortará y reiniciará un nuevo intento.

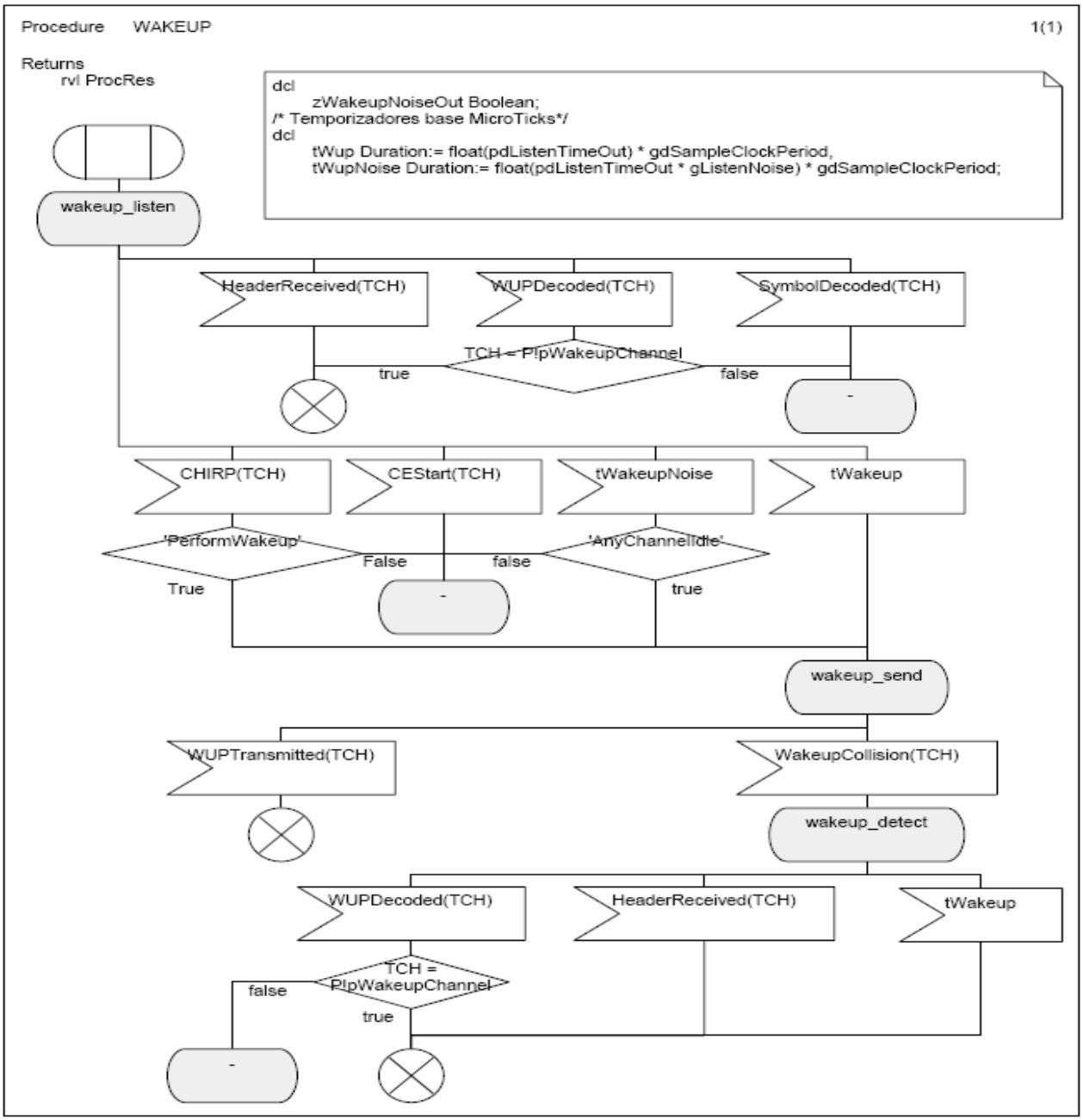


Figura 4.32. Descripción del procedimiento WAKEUP.

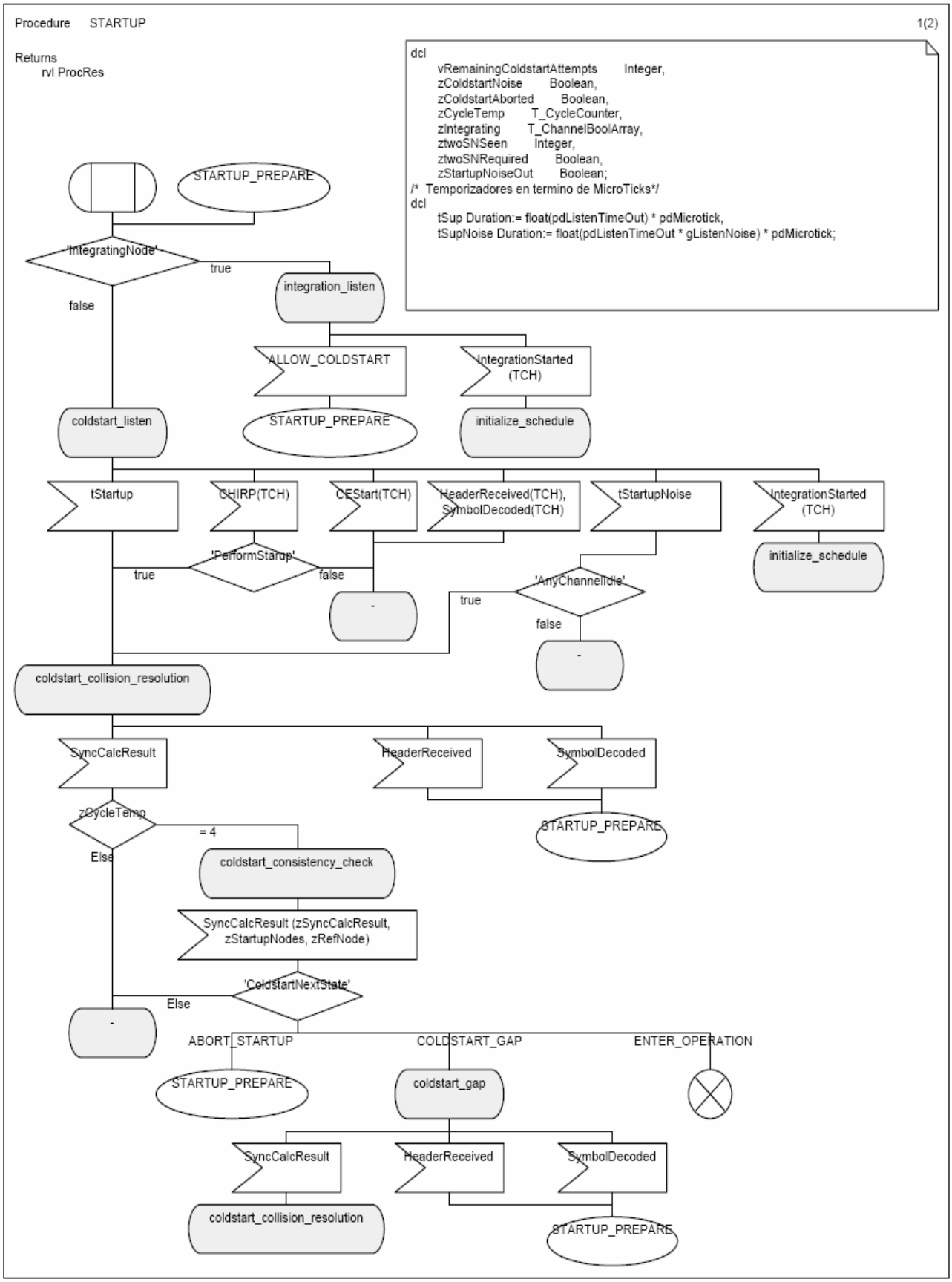


Figura 4.33. Descripción del procedimiento STARTUP.

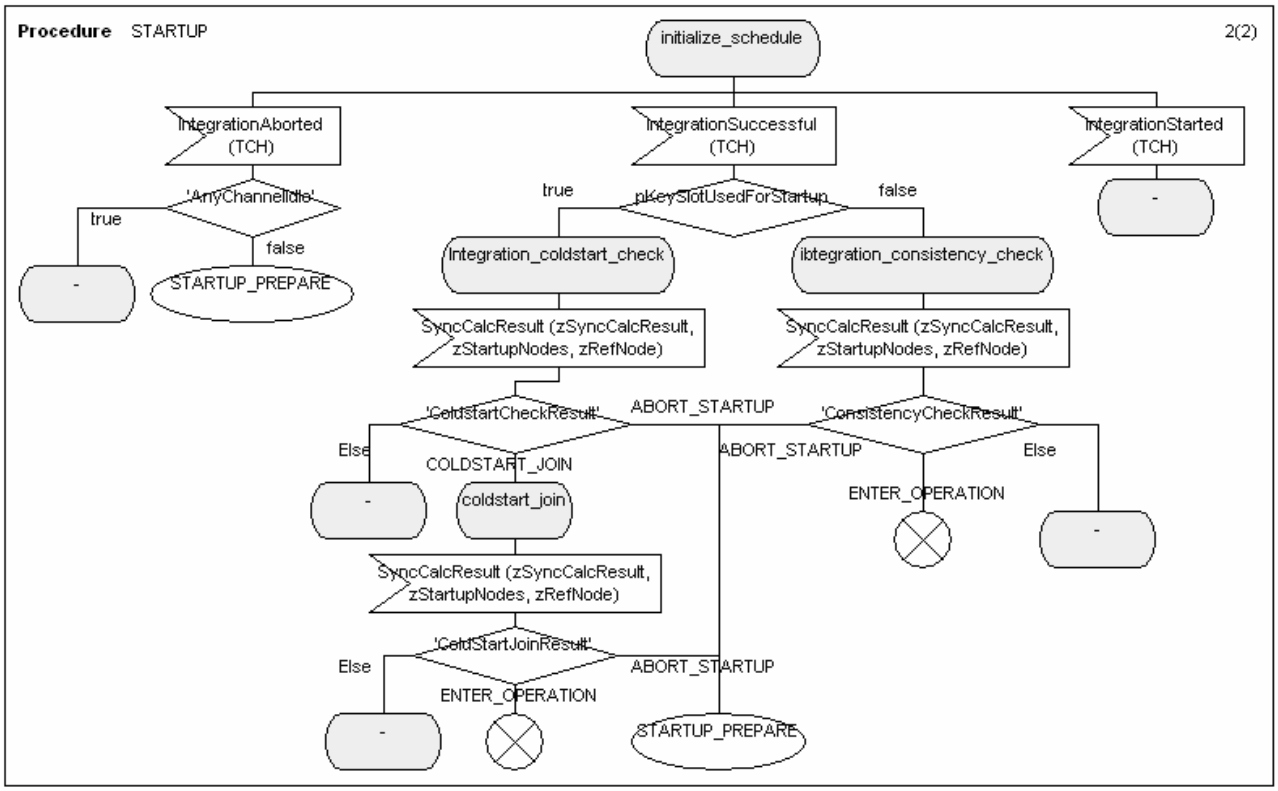


Figura 4.34. Descripción del procedimiento STARTUP (continuación).

4.5.5. Proceso mtg

Una vez calculados los factores de corrección de fase y de frecuencia, éstos se utilizan para modificar el temporizador local y mantener sincronizado el cronograma de tiempo global; en este caso se ajusta el número de *microticks* por *macrotick*. Existen dos formas de iniciar el proceso de generación de *macroticks*:

- Por medio del proceso poc, si el nodo es *coldstart* principal.
- Si es nodo *coldstart* de seguimiento o nodo no *coldstart*, intentará integrarse.

Cuando se crea el proceso mostrado en la Figura 4.35, entra al estado wait_for_start en espera de que el nodo inicie operaciones, tras la recepción de la señal ColdStart emitida por el proceso poc, el proceso mtg cambia al estado wait_for_microtick; la señal StartClock emitida por el proceso css puede ser recibida en cualquiera de los dos estados mencionados, lo que ocasiona un reinicio en la temporización. Una vez que se alcanza el estado wait_for_microtick se pueden recibir las siguientes señales:

- *tMicrotick*: Cuando se consume el tiempo asignado a un *microtick*, el temporizador envía una señal del mismo nombre; cuando se alcanza el número de *microticks* por *macrotick*, se emite la señal macrotick al proceso mac y actualiza su contador; si este último llega a cero, el mtg envía la señal CycleStart a los procesos poc, mac y csp.
- *OffsetCalcReady*: Esta señal es emitida por el proceso csp con el factor de corrección de fase actualizado.
- *RateCalcReady*: Señal emitida por el proceso csp con el factor de corrección de frecuencia actualizado.
- *ResetMTG*: La recepción de esta señal ocasiona que la temporización se detenga, enviando al proceso al estado wait_for_start.

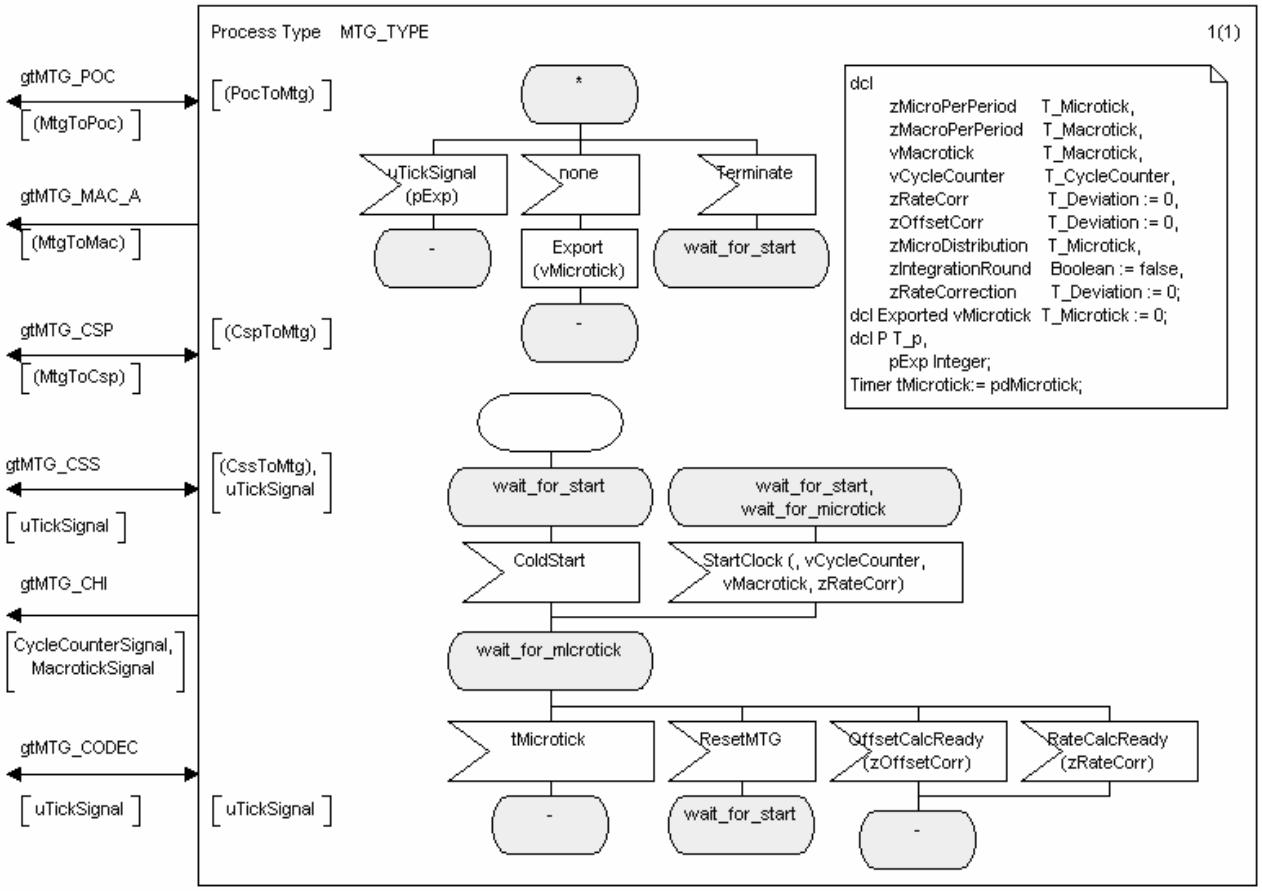


Figura 4.35. Descripción del proceso MTG_TYPE.

Concurrentemente con el proceso mtg, el csp toma nuevos valores de medición y los utiliza para calcular nuevos factores de corrección, estos nuevos valores son aplicados por el proceso mtg:

- La nueva corrección de fase al final del periodo de corrección de fase en el ciclo de comunicación impar.
- La nueva corrección de frecuencia al inicio del ciclo en un ciclo de comunicación par.

4.5.6. Proceso csp

El procedimiento de inicio del sistema da comienzo tras la recepción de la señal AttemptIntegration, emitida por el poc cuando el proceso csp se encuentra en el estado wait_for_startup y pasa al estado run para ejecutar el control de integración. Si no se detecta un CE en el canal, el proceso poc emite la señal ColdStart forzando al nodo a tomar el rol de nodo principal del *cluster*, lo que genera las siguientes acciones:

- El proceso de sincronización del reloj al inicio del sistema o css del sistema será llevado al estado standby con la emisión de la señal TerminateCSS.
- El control de integración del proceso csp será abortado llevándolo al estado wait_for_cycle_start.
- Simultáneamente el proceso mtg inicia la temporización del nodo.

Si se detecta un CE o el nodo no es *coldstart*, el nodo intenta integrarse al cronograma de tiempo del *cluster* adoptando la frecuencia y el número de ciclo del nodo transmisor; para realizar esto, el proceso csp obliga al proceso css a abandonar el estado standby preparándolo para la sincronización durante la fase inicial. Cuando se recibe una trama de sincronización válida, el nodo es

habilitado para precalcular los valores iniciales del contador del ciclo y de *macroticks*, después el nodo espera la correspondiente trama de sincronización válida impar. Al detectar una segunda trama potencial, el control de integración activa el temporizador tMicroInitialOffset, cuando éste expira el proceso mtg es inicializado con los valores precalculados.

Una vez que es decodificada la segunda trama de sincronización válida (*impar*), se noticifa al proceso csp de que la integración ha sido completada y tras la recepción de la señal IntegrationSuccessful alcanza el estado wait_for_cycle_start; la temporización ha sido previamente inicializada, por lo tanto, cuando el proceso mtg genera el inicio de un nuevo ciclo notifica al proceso csp con la señal CycleStart llevándolo al estado wait_for_sync_frame en el que ejecuta la medición del valor de desvío de cada trama de sincronización recibida, lo que deriva:

- El proceso csp toma el valor actual del contador de *microticks* tras recibir la señal ActionPoint indicando el inicio de transmisión o recepción de una trama en la ranura activa.
- Tras la recepción de la señal ValidSyncFrame se ejecuta la medición restando el valor previamente tomado al TRP primario.
- Cuando se recibe la señal FrameReceptionComplete emitida por el proceso fsp, lleva a cabo el cálculo de los factores de corrección, los cuales son notificados al proceso mtg con la señal OffsetCalcReady y RateCalcReady. Se notifica el resultado del proceso de sincronización al proceso poc por medio de la señal SyncCalcResult.

El diagrama de visión de estados del proceso de sincronización se muestra en las Figuras 4.36 y 4.37.

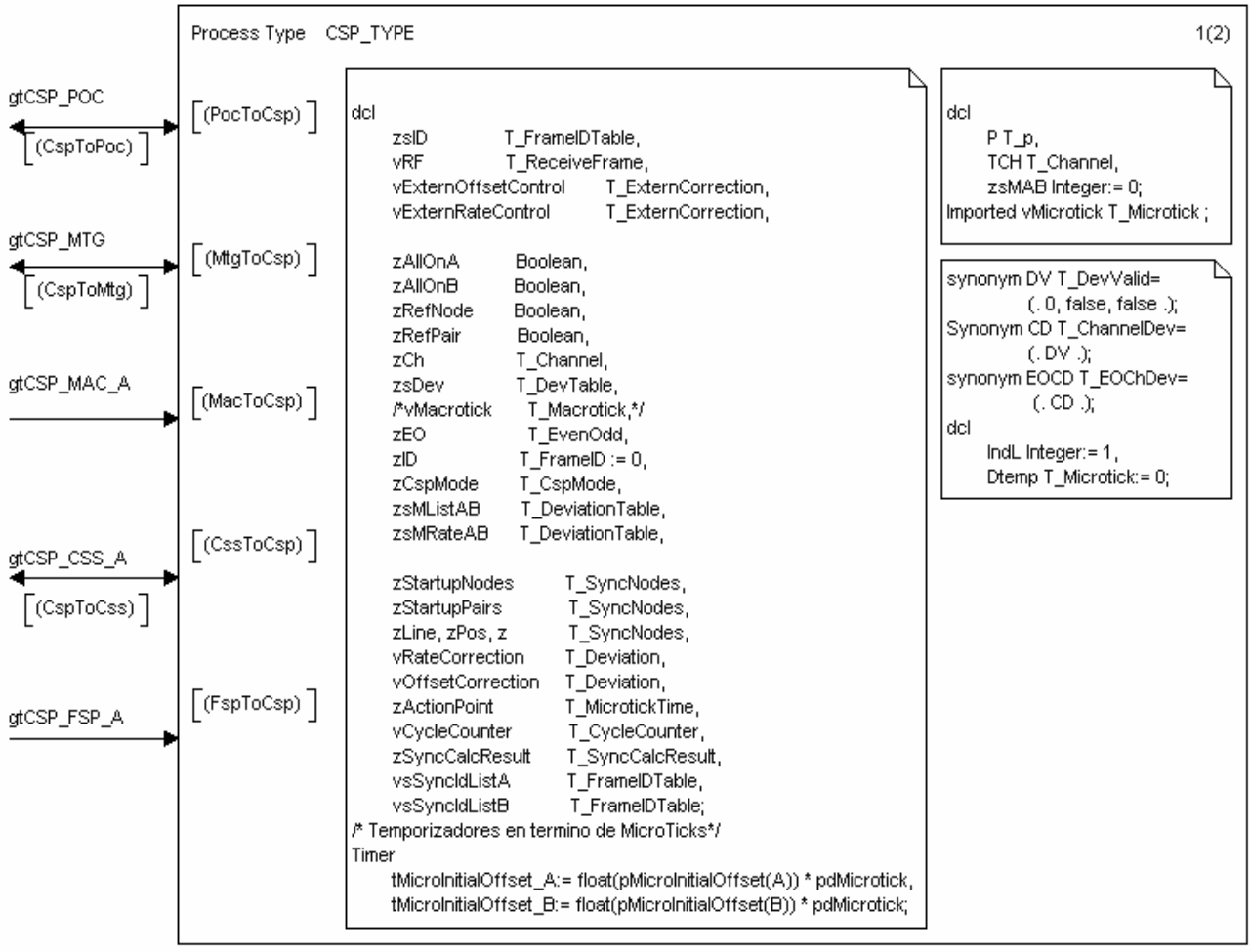


Figura 4.36. Descripción del proceso CSP_TYPE.

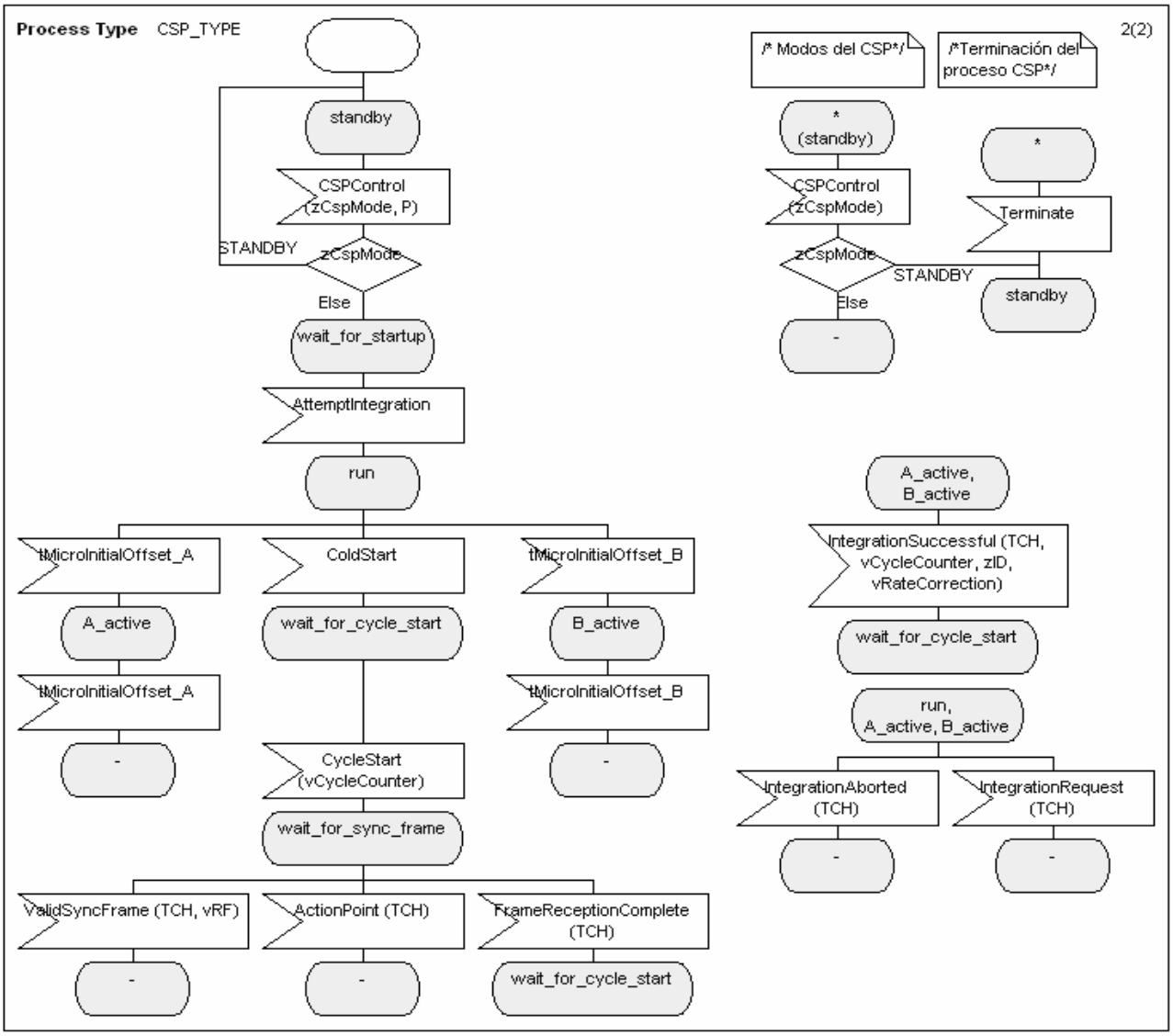


Figura 4.37. Descripción del proceso CSP_TYPE (continuación).

4.5.7. Proceso mac

El control de operaciones del protocolo establece los modos de operación del control de acceso al medio con la señal MACControl; si el valor del parámetro transportado es distinto a STANDBY, puede alcanzar los estados wait_for_cas_action_point o wait_for_cycle_start dependiendo del modo de operación con que se configure al mac:

- En el modo STANDBY se suspende el acceso al medio.
- En el modo NOCE se ejecuta el control de acceso al medio, pero no está habilitado para realizar transmisión de tramas o símbolo.
- En el modo STARTUPFRAMECAS se restringen las transmisiones a una trama nula de sincronización por ciclo en cada canal, configurado si el nodo es *coldstart*. Adicionalmente el nodo envia el símbolo inicial CAS antes del primer ciclo de comunicación.
- En el modo STARTUPFRAME se restringen las transmisiones a una trama nula de sincronización por ciclo en cada canal configurado, si el nodo es *coldstart*.

- En el modo SINGLESLOT se restringen las transmisiones a una trama nula de sincronización por ciclo en cada canal configurado si el nodo es *coldstart*; o da soporte a la transmisión de tramas en una ranura específica por ciclo si el nodo es configurado para soportar el modo de ranura única.
- En el modo ALL_T se envían tramas y símbolos de acuerdo con la distribución de ranuras asociadas a cada nodo.

Cuando se alcanza el estado wait_for_cycle_start espera que el peoceso mtg envíe la señal CycleStart para administrar el acceso al medio en un nuevo ciclo. Tras la recepción de la señal CycleStart, se notifica al proceso fsp para que inicie el segmento estático con la señal StaticSegmentStart y entra el estado wait_for_the_action_point en espera que el temporizador tActionPoint sea consumido, indicando el inicio de la ranura actual (Figura 4.38). Si la ranura es asignada para transmisión, el proceso mac envía la señal TransmitFrame al codec con la información para ensamblar la trama. Cuando se consume el temporizador tSlotBounday, si el número de ranuras del cluster gNumberOfStaticSlots es superado por el contador de ranuras, el control regresa al estado wait_for_the_cycle_start notificando al proceso fsp que inicia el segmento NIT con la señal NITStart.

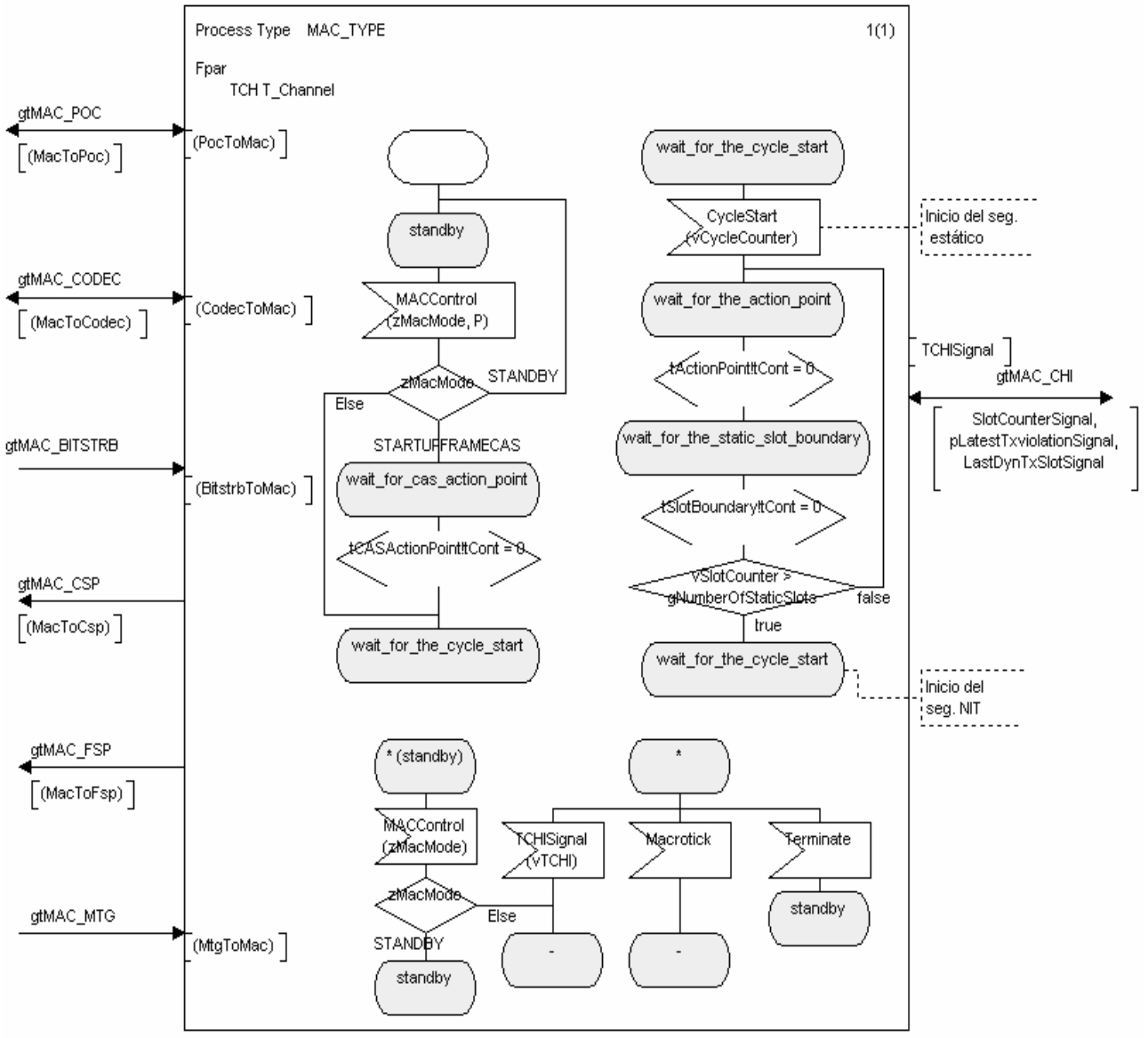


Figura 4.38. Descripción del proceso MAC_TYPE.

4.5.8. Proceso fsp

El diagrama de visión de estados del proceso de validación y procesamiento de tramas y símbolo se muestra en las Figuras 4.39a y 4.40. Tan pronto como el nodo es habilitado para iniciar operaciones, el proceso fsp es llevado el estado wait_for_CE_start en el que la recepción de las siguientes señales causará un cambio de estado:

- Cuando el proceso mac detecta que la ranura actual está asignada al nodo, inicia la transmisión de la trama por medio del proceso codec, el cual envía la señal DecodingHalted al proceso fsp ocasionando un cambio al estado wait_for_transmission_end en el que espera ser notificado del fin de la transmisión de la trama.
- Si un CE se detecta, tras recibir la señal CEStart cambiará al estado decoding_in_progress.

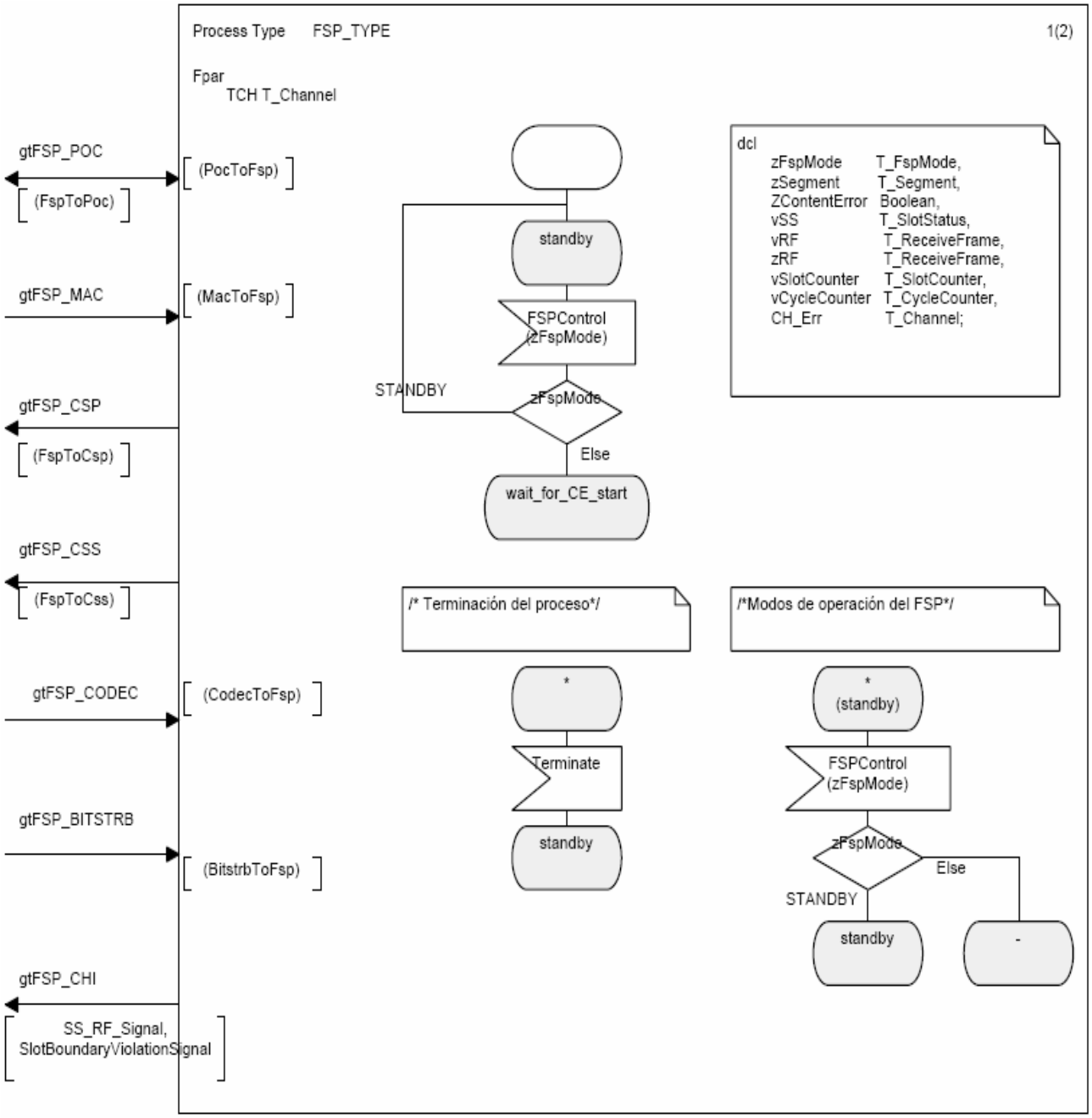


Figura 4.39. Descripción del proceso FSP_TYPE.

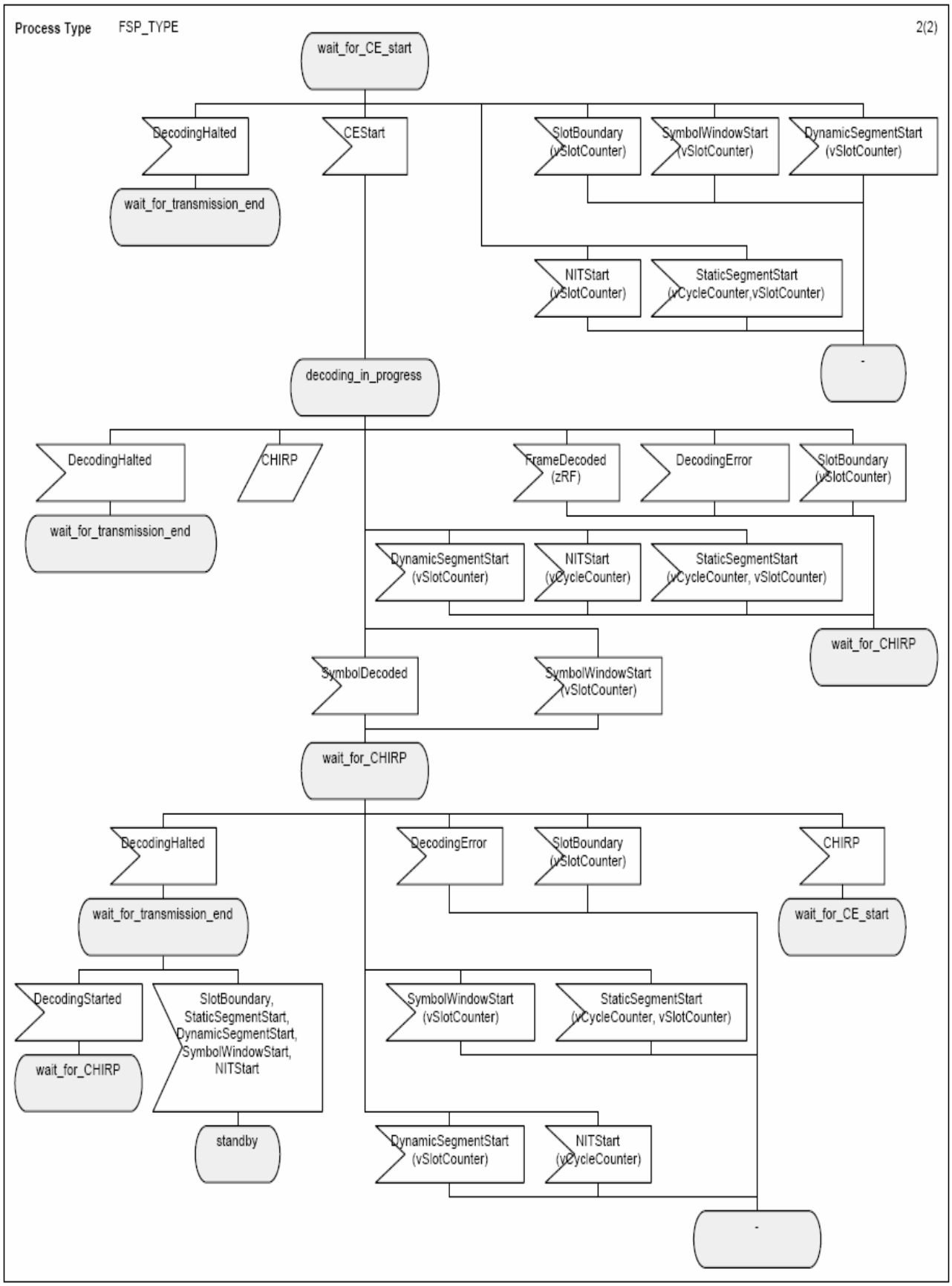


Figura 4.40. Descripción del proceso FSP_TYPE (continuación).

Cuando se alcanza el estado decoding_in_progress, cualquiera de las siguientes situaciones causa una transición al estado wait_for_CHIRP:

- La decodificación de una trama notificada con la señal FrameDecoded es validada apropiadamente en el segmento estático o dinámico, es decir, verifica que sean válidas. Si el nodo se encuentra en la fase de inicio del sistema y si la trama de sincronización es válida, lo notifica al proceso csp con la señal ValidEvenStartupFrame o ValidOddStartupFrame.
- La detección de un error notificada con la señal DecodingError.
- Si se decodifica un símbolo, se valida que haya ocurrido en el segmento ventana de símbolo.

Si el nodo inicia la transmisión de un CE, tras la recepción de la señal DecodingHalted, se activa la bandera de error y pasa al proceso wait_for_transmission_end. Además, dado que durante el estado decoding_in_progress no es posible detectar inactividad en el canal, se introduce el símbolo de preservación de señal (*save*) para evitar que la señal CHIRP sea leída prematuramente (este error ocurre por la velocidad de procesamiento del PC en uso).

Cuando se alcanza el estado wait_for_chirp, tras la recepción de la señal CHIRP, el proceso fsp es llevado al estado wait_for_CE_start. Si se alcanza el estado wait_for_transmission_end, la transmisión de una trama está en ejecución, por tanto permanece en este estado hasta que el proceso codec notifique el final de la transmisión con la señal DecodingStarted.

Se pueden recibir las señales SlotBoundary, StaticSegmentStart, DynamicSegmentStart, SymbolWindowStart y NITStart en todos los estados mencionados a excepción de estado standby, lo que deriva en la evaluación de la trama recibida al final de la ranura activa según el estado en que se encuentre el proceso fsp:

- No hay error de violación de la ventana de la ranura si el estado es wait_for_CE_start.
- Si el estado actual del proceso fsp es wait_for_transmission_end causa un error fatal del protocolo y es llevado al estado standby.
- Cualquier otro estado causa un error de violación de la ventana de la ranura.

4.5.9. Proceso codec

Debido a lo extenso de este proceso, éste se ilustra mediante un diagrama de visión de estados reducido poniendo mayor énfasis en el algoritmo para el cálculo del CRC y el ensamblado de la trama. Flexray define dos segmentos de CRC en el formato de la trama, uno contenido en el segmento de de cabecera (*Header CRC*) y el segmento de CRC de la trama (*trailer segment*). La funcionalidad del proceso codec depende de que ambos CRCs sean verificados mientras se procesa la recepción de la trama o verificar cada CRC después de la recepción del segmento de cabecera o segmento de carga útil (como se realiza en esta tesis). Adicionalmente el codec agrega el CRC de la trama cuando ésta se transmite. Formalmente el *host* debe calcular y llenar el campo del CRC de cabecera, debido a que no se dispone de operadores en SDL para operaciones binarias sobre números enteros, es necesario implementar la conversión de enteros a binarios para realizar el cálculo del CRC, tarea que introduce un significativo retraso en la ejecución de la especificación, razón por la que se decidió implementarla en el proceso codec.

El algoritmo que se muestra en la Figura 4.41 da comienzo con la inicialización del vector o registro de desplazamiento con un valor apropiado (depende del CRC a calcular) CRCStream:= CRCInit; se determina si el polinomio CRCPol debe ser aplicado por medio de la operación XOR entre el bit más significativo del registro de desplazamiento CRCStream y el siguiente bit de datos de la trama; el vector CRCStream se desplaza a la izquierda por un bit. Si el valor de la variable NextCRCBit representa un estado lógico alto (*HIGH*) el polinomio CRCStream:= CRCStream XOR CRCPol es

aplicado. El algoritmo termina cuando se ha calculado el CRC sobre todos los bits contenidos en la variable BS, BS representa el formato de la trama en estado binario en orden de red[10].

Una vez que el proceso codec entra en operación, la recepción de la señal TransmitFrame en cualquier estado distinto a standby o ready provoca que el nodo realice la transmisión de la trama. Para ello llama al procedimiento PrepBitStream (Figuras 4.42 y 4.43), antes de realizar el ensamblado de la trama, el codec convierte los campos de la trama Flexray a formato binario con la llamada a la función TFToBitStream. Ambos CRCs son calculados con la función TxCRCCalc, el CRC de cabecera es calculado con los siguientes parámetros:

- Longitud del registro de 11 bits representado con la constante cHCRCSize.
- El vector es inicializado con el valor 0x1A para ambos canales, representado por la constante cHCRCInit.
- El polinomio cHCRCPolynomial es representado por el valor hexadecimal 0x385.

Los parámetros para calcular el CRC de la trama son:

- Longitud del registro de 24 bits representado por cCRCSize.
- Inicialización del vector 0xFEDCBA para el canal A y 0xABCDEF para el canal B, ambos representados por las constantes cCRCInitA y cCRCInitB respectivamente.
- Su representación polinomial en hexadecimal 0x5D6DCB denotado por la constante cCRCPolynomial.

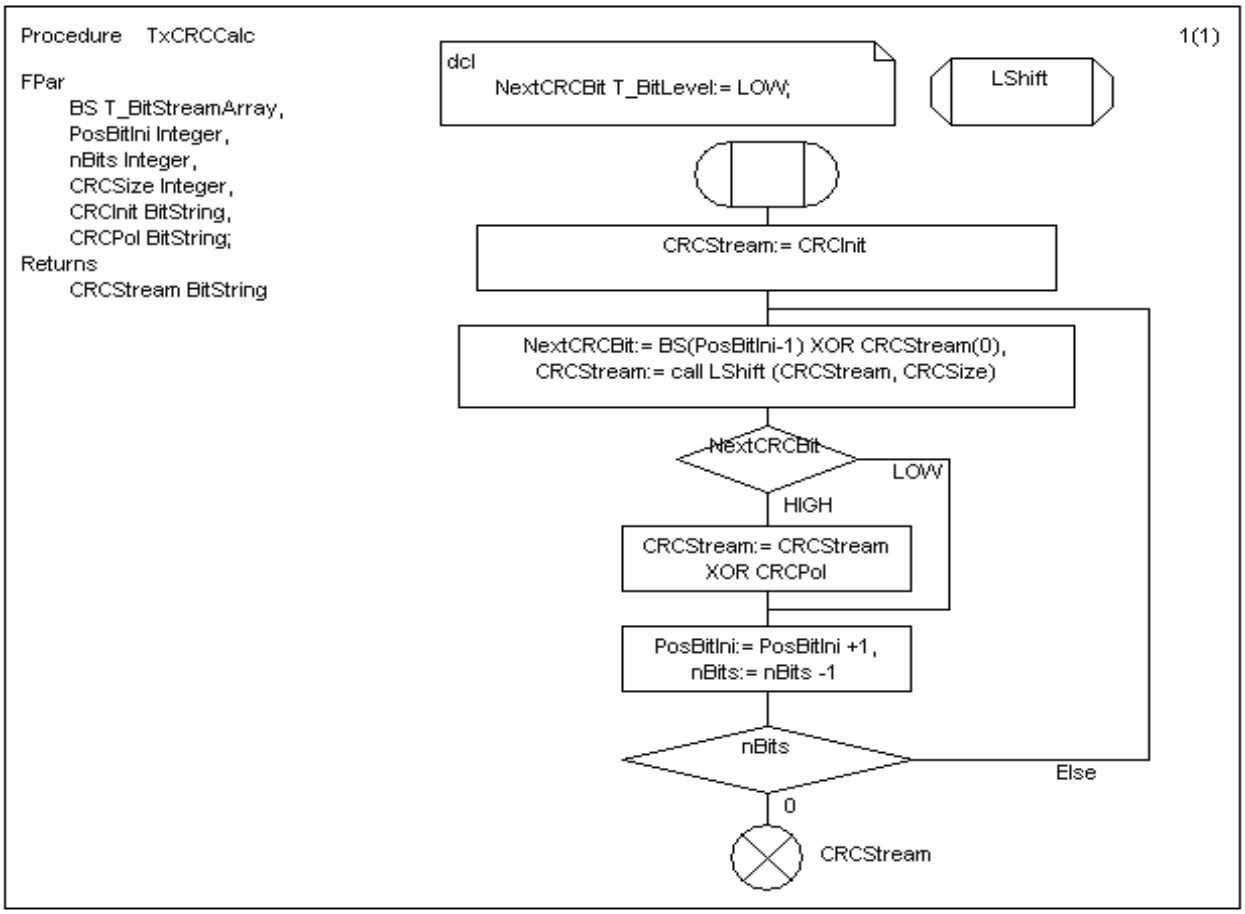


Figura 4.41. Algoritmo para el cálculo del CRC.

[10] Flexray define el término "orden de red" como la transmisión de cada campo de la trama iniciando con el bit más significativo.

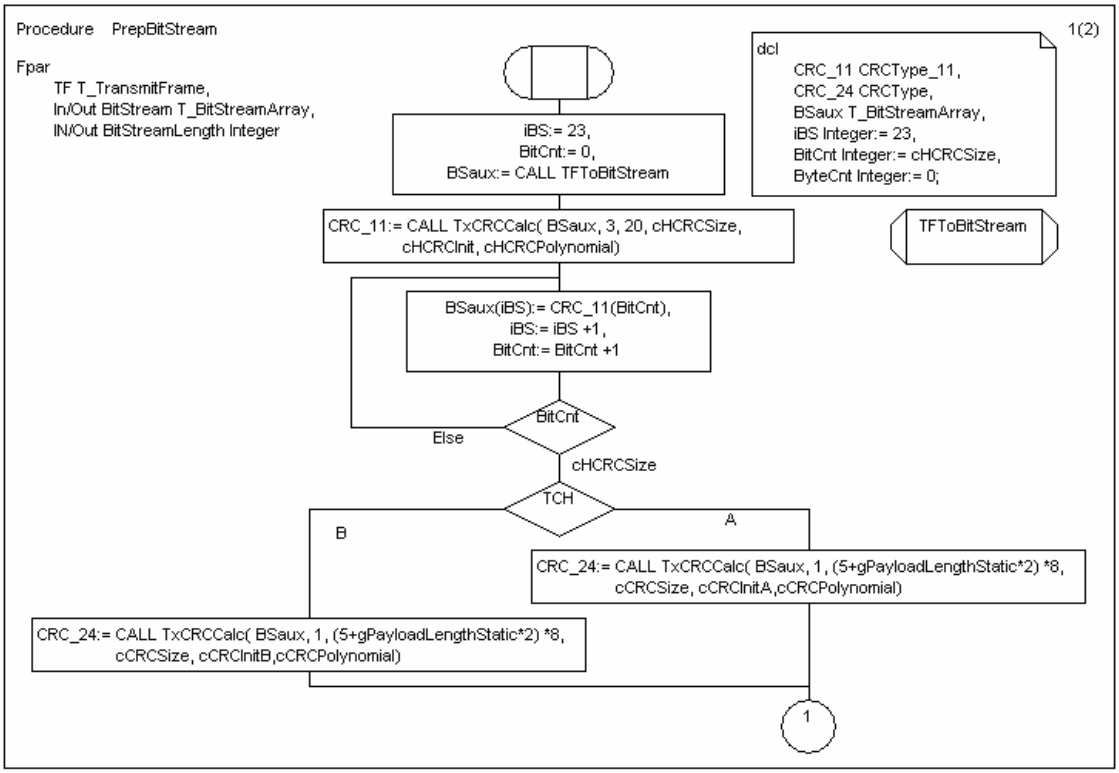


Figura 4.42. Ensamblado de la trama Flexray.

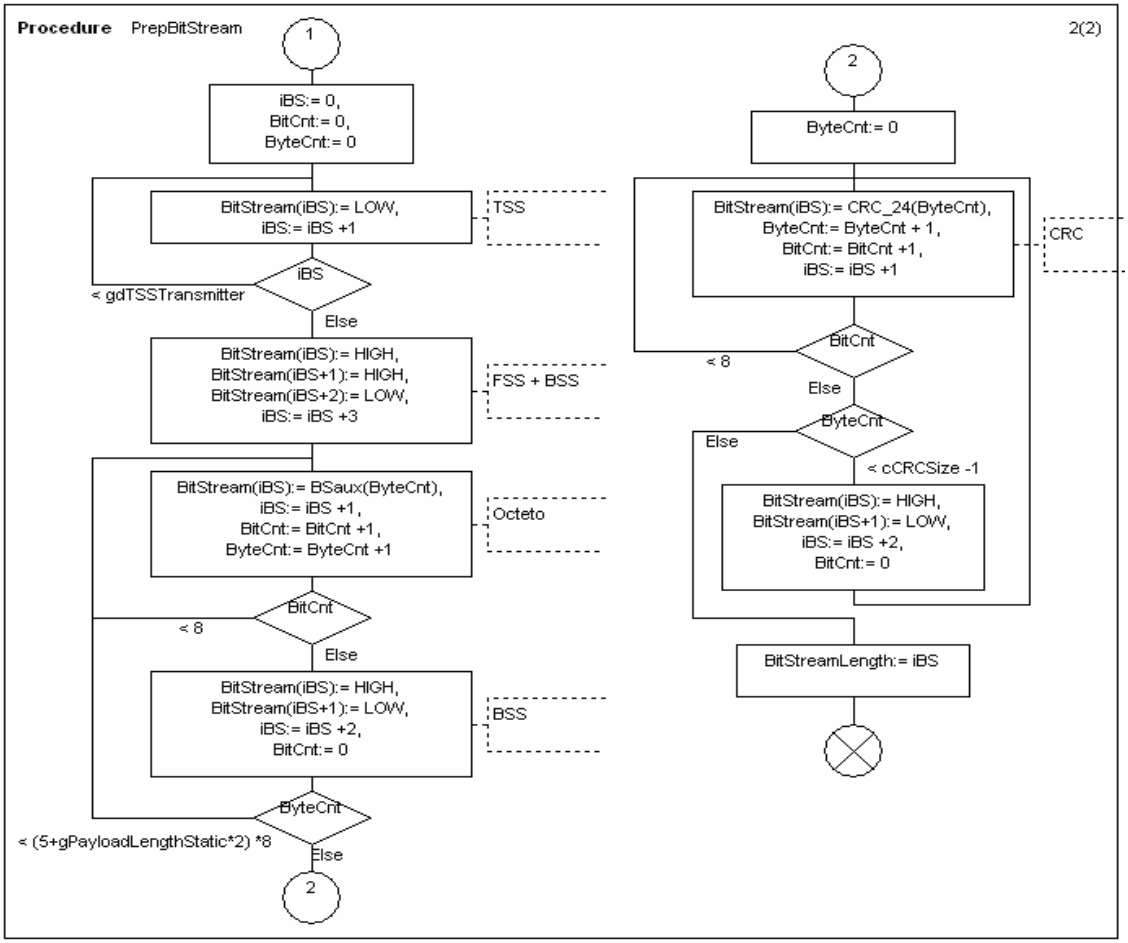


Figura 4.43. Ensamblado de la trama Flexray.

Cuando ambos CRCs han sido calculados, se procede al ensamblado de la trama como se muestra en la Figura 4.43; el procedimiento inicia con la inserción de los símbolos TSS, FSS y BSS. Cada octeto de la trama Flexray es introducido uno a uno después de cada BSS. Por último, agrega el CRC de la trama, previamente calculado octeto por octeto. El número total de bits a transmitir es actualizado en la variable BitStreamLength, mientras que la variable BitStream contiene todos los bits que serán transmitidos.

Si el elemento a transmitir es un símbolo, tras recibir la señal TransmitSymbol, el proceso codec determina si el símbolo es el CAS, en cuyo caso prepara una cadena de bits y se transmiten como si se trata de cualquier trama; si el símbolo es el patrón WUP, se procesa de diferente manera, ya que el codec debe ser capaz de determinar si hay posibles colisiones durante la transmisión (durante el inicio del *cluster* exite esta posibilidad).

Cuando el nodo entra en operación, el proceso pasa al estado Wait_For_CEStart en espera que el BD detecte actividad en el bus, si es así, es notificado con la señal CEStart con la que dá inicio al proceso de decodificación de tramas o símbolos. Si el elemento a decodificar es una trama, el proceso codec verifica que el CRC de cabecera sea correcto después de decodificarlo; cuando termina de decodificar la trama completa, verifica el CRC de trama; para ello hace uso de las funciones HEADER_CRC_CHECK y FRAME_CRC_CHECK (Figuras 4.44 y 4.45). Estas funciones hacen uso de RxCRCCalc para calcular el CRC de las tramas recibidas. La definición de una función diferente para calcular el CRC de las tramas que han sido recibidas se debe a que las tramas en formato binario son manipuladas con diferentes estructuras; sin embargo el algoritmo tiene la misma estructura. Finalmente, si la trama es recibida sin errores, convierte los datos en estado binario al tipo de dato con que se representa la trama en alto nivel.

La Figura 4.46 muestra el diagrama de estados reducido del proceso codec.

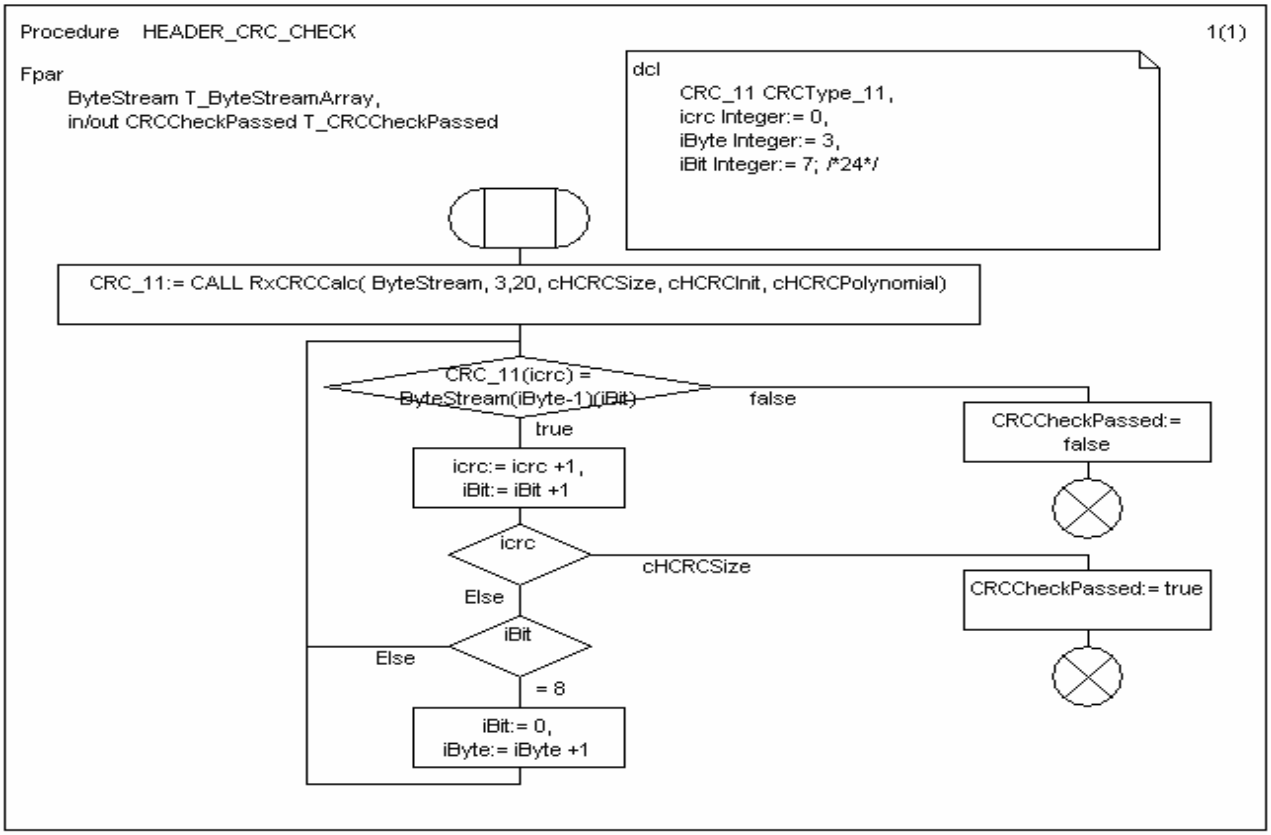


Figura 4.44. Verificación del CRC de cabecera.

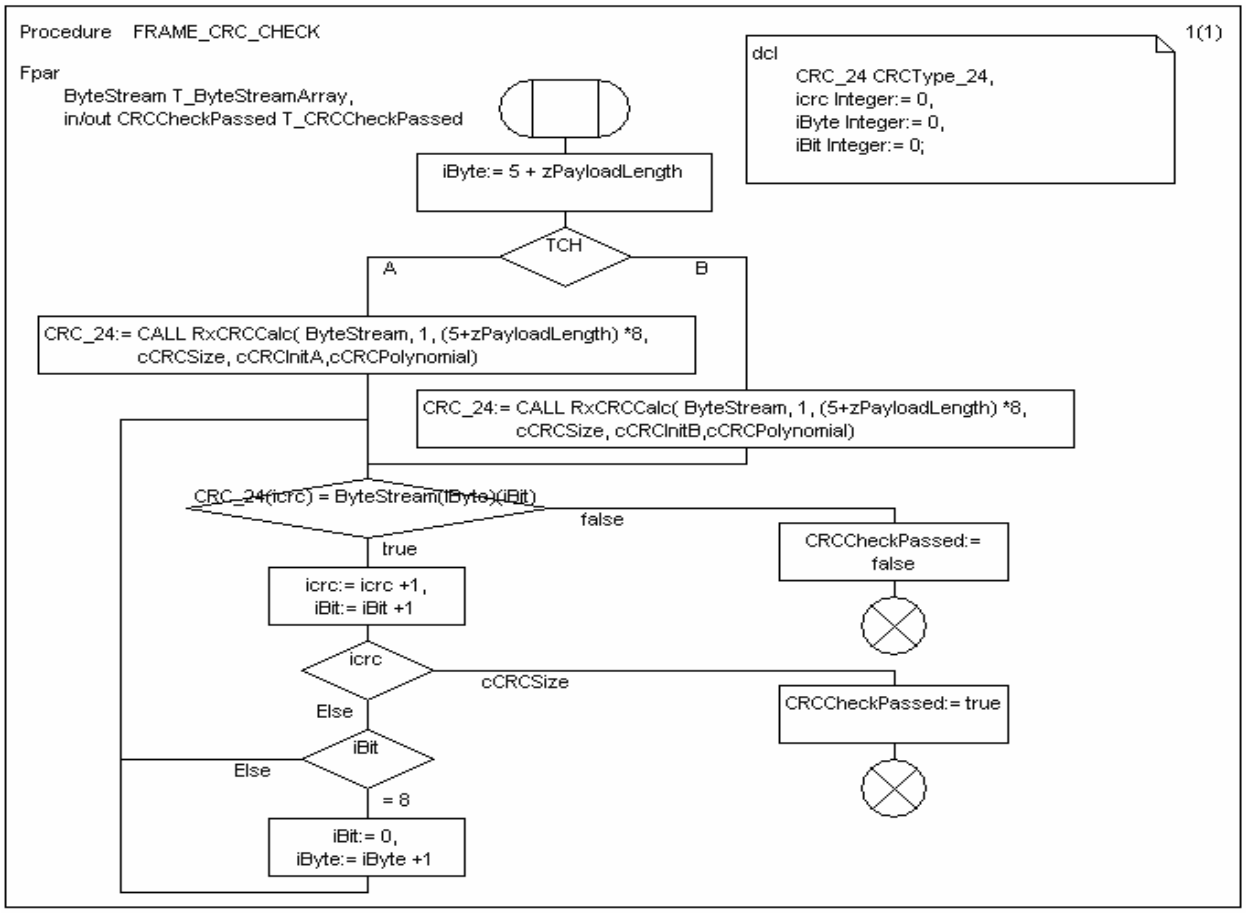


Figura 4.45. Verificación del CRC de la trama.

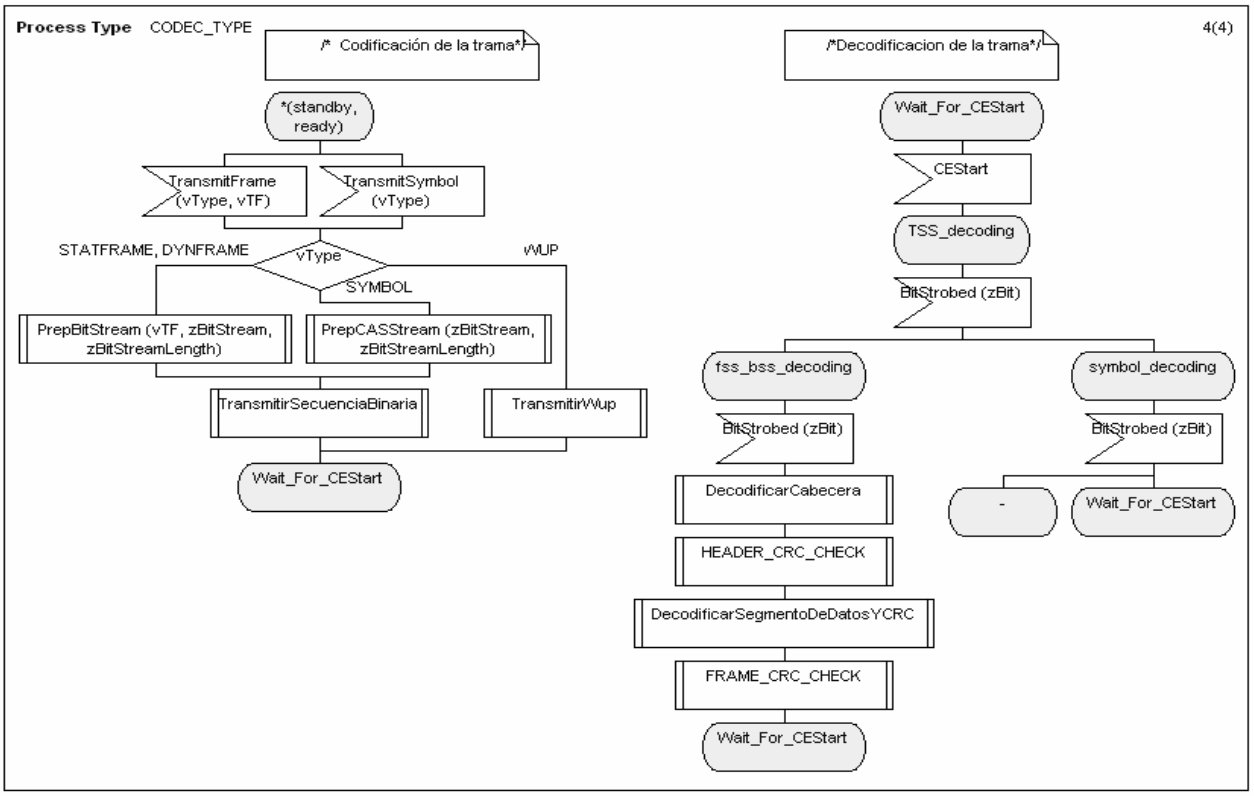


Figura 4.46. Diagrama de estados reducido del proceso codec.

4.6. Parámetros de configuración

Antes de definir los parámetros de configuración y determinar sus valores, la Tabla 4.3 presenta las constantes definidas en la especificación del protocolo Flexray.

Tabla 4.3. Constantes definidas para la especificación del protocolo Flexray.

Parámetro	Descripción	Valor
cCASActionPointOffset	Valor para el temporizador del AP del símbolo CAS	1 MT
cChannelIdleDelimiter	Duración para el detector del canal libre	11 gdBit
cClockDeviationMax	Máxima desviación de frecuencia del reloj	0.0015
cCRCInitA	Vector de inicialización para el cálculo del CRC de la trama en el canal A	0xFEDCBA
cCRCInitB	Vector de inicialización para el cálculo del CRC de la trama en el canal B	0xABCDEF
cCRCPolynomial	Polinomio para el CRC de la trama	0x5D6DCB
cCRCSize	Tamaño del registro para el cálculo del CRC de la trama	24 Bits
cdBSS	Duración del BSS	2 gdBit
cdCAS	Duración de la parte baja lógica de los símbolo CAS y MTS	30 gdBit
cdCASRxLowMin	Duración mínima para la ventana aceptable del CAS	29 gdBit
cdFES	Duración del FES	2 gdBit
cdFSS	Duración del FSS	1 gdBit
cdWakeupMaxCollision	Número de bits continuos en 0 lógico durante la fase *idle* de la transmisión del símbolo WUS que ocasionará que nodos transmisores detecten una colisión con este símbolo	5 gdBit
cdWakeupSymbolTxLow	Duración de la parte baja del símbolo WUS	6 µs
cdWakeupSymbolTxIdle	Duración de la parte *idle* entre dos partes bajas del patrón WUS	18 µs
cHCRCInit	Vector de inicialización para el cálculo del CRC de la cabecera	0x01A
cHCRCPolynomial	Polinomio para el cálculo de l CRC de la cabecera	0x385
cHCRCSize	Tamaño del registro del CRC de cabecera	11 Bits
cSamplesPerBit	Número de muestras tomadas para la detección del valor de bit	8
cStrobeOffset	Punto del Muestreo donde el valor del bit es tomado	5

Las ecuaciones que definen los parámetros para el segmento dinámico y para la ventana de símbolo no se detallan debido a que están deshabilitadas y no tienen una ventana de tiempo asignado, están puestas a cero. Las Tablas 4.3-7 muestran los parámetros que tienen influencia directa con la operación del protocolo (incluyendo los referentes al segmento dinámico y ventana de símbolo).

En este trabajo, estos parámetros están definidos para una velocidad de transferencia de datos de 10 MBps, una duración de cada ciclo de comunicación de 292 µs, cuatro muestras por *microtick* con una duración de 0.050 µs por cada *microtick*.

Las unidades de los parámetros son microsegundos (µs), *microticks* (µT), *macroticks* (MT) y duración nominal del tiempo de bit (gdBit).

4.6.1. Parámetros globales

Los parámetros globales definen las características del *cluster* y cada nodo debe conocerlos. La Tabla 4.4 muestra los parámetros globales establecidos en tiempo de diseño. Estos parámetros dependen de la aplicación que se esté desarrollando. Para los primeros 4 parámetros de la Tabla 4.4 es recomendable establecer los valores más altos si la aplicación es extensible a más nodos.

Tabla 4.4. Requerimientos del sistema para los parámetros del cluster.

Parámetro	Descripción	Rango	Valor
gColdStartAttempts	Máximo número de veces que un nodo en el cluster puede intentar iniciarlo	2 - 31	2
gListenNoise	Multiplicador del parámetro *pdListenTimeout* en presencia de ruido	2 - 16	2
gMaxWithoutClockCorrectionFatal	Define el número de ciclos par-impar consecutivos con pérdidas en el factor de corrección del reloj que ocasionará un alto total del protocolo	(gMaxWithoutClockCorrection Pasive) – 15	2
gMaxWithoutClockCorrectionPssive	Define el número de ciclos par-impar consecutivos con pérdidas en el factor de corrección del reloj que ocasionará una transición de PocActive a PocPassive en el modelo de degradación de errores del proceso POC	1 – 15	2
gNumberOfStaticSlots	Número de ranuras en el segmento estático	2 - 1023	4
gPayloadLengthStatic	Longitud de la carga útil en el segmento estático	0 – 127 palabras dobles	2
gSyncNodeMax	Máximo número de nodos que pueden enviar tramas de sincronización	2 - 15	2
gdSampleClockPeriod	Duración del periodo de muestreo	0.0125, 0.025, 0.05 [μs]	0.0125 μs

La Tabla 4.5 muestra los valores obtenidos con la aplicación de las restricciones y ecuaciones referentes a los parámetros globales de acuerdo a los requerimientos[11].

4.6.2. Parámetros locales

Los parámetros locales definen los parámetros específicos de cada nodo en el mismo *cluster*, los cuales pueden variar de un nodo a otro; sin embargo, éstos deben calcularse para la misma velocidad de transferencia y con la misma duración de *macrotick*. En el presente trabajo de tesis, los parámetros locales son los mismos para ambos nodos (debido a que tienen las mismas características), a excepción de las ranuras asociadas a cada nodo. Los parámetros de configuración iniciales se muestran en la Tabla 4.6.

En la Tabla 4.7 se resumen los valores obtenidos al aplicar las ecuaciones y restricciones definidas para los parámetros locales[12].

[11] Las ecuaciones y restricciones utilizadas para calcular el valor de los parámetros globales son las que el protocolo Flexray define en [14].

[12] Las ecuaciones y restricciones utilizadas para calcular el valor de los parámetros locales son las que el protocolo Flexray define en [14].

Tabla 4.5. Configuración de los parámetros globales.

Parámetro	Rango	Valor
gdActionPointOffset	1 - 63 [MT]	9 MT
gdCASRxLowMax	67 - 99 [MT]	81 MT
gdDynamicSlotIdlePhase	0 – 2 mini ranuras	0
gdMinislot	2 - 63 [MT]	0
gdMinislotActionPointOffset	1 - 31 [MT]	0
gdStaticSlot	4 - 661 [MT]	26 MT
gdSymbolWindow	0 - 142 [MT]	0
gdTSSTransmitter	3 - 15 [gdBit]	6 gdBit
gdWakeupSymbolRxIdle	14 - 59 [gdBit]	59 gdBit
gdWakeupSymbolRxLow	11 - 59 [gdBit]	55 gdBit
gdWakeupSymbolRxWindow	76 - 301 [gdBit]	301 gdBit
gdWakeupSymbolTxIdle	45 - 180 [gdBit]	180 gdBit
gdWakeupSymbolTxLow	15 - 60 [gdBit]	60 gdBit
gMacroPerCycle	10 - 16000 [MT]	146 MT
gNumberOfMinislots	0 - 7986	0
gOffsetCorrectionStart	9 - 15999 [MT]	142 MT

Tabla 4.6. Parámetros de configuración local establecidos en tiempo de diseño.

Parámetro	Descripción	Rango	Valor
pAllowHaltDueToClock	Bandera que controla la transición al estado PocHalt debido a errores de sincronización	-	False
pAllowPassiveToActive	Número de ciclos par-impar consecutivos que deben tener un término de corrección del reloj válido para habilitar al CC cambiar del estado PocPassive al estado PocActive	0 – 31 ciclos par-impar	2
pChannels	Canales a los cuales el nodo está conectado	A, B, A&B	A
pKeySlotID	Identificador de la ranura usada para transmitir tramas de sincronización	1 - 1023	2, 4
pKeySlotUsedForStartup	Bandera para indicar si la ranura llave es usada para transmitir tramas de sincronizaión	-	True
pKeySlotUsedForSync	Bandera para indicar si la ranura llave es usada para transmitir tramas de sincronizaión	-	True
pSingleSlotEnabled	Bandera para indicar si el nodo entrará en modo de ranura única	-	True
pWakeupChannel	Canal usado por el nodo para transmitir el patrón WUS	A, B	A
pWakeupPattern	Número de veces que el símbolo WUS será combinado para formar el patrón WUS	2 - 63	2
pSamplePerMicrotick	Número de muestras por *microtick*	1, 2, 4	4

Tabla 4.7. Configuración de los parámetros locales.

Parámetro	Rango	Valor
pdAcceptedStartupRange	0 – 1875 [µT]	163 µT
pClusterDriftDamping	10 – 20 [µT]	5 µT
pDecodingCorrection	14 – 143 [µT]	18 µT
pDelayCompensation[A]	0 – 200 [µT]	200 µT
pdListenTimeout	1284 – 1283846 [µT]	11716 µT
pdMaxDrift	2 – 1923 [µT]	18 µT
pExternOffsetCorrection	0 – 7 [µT]	0 µT
pExternRateCorrection	0 – 7 [µT]	0 µT
pLatestTx	0 – 7980 Mini Ranuras	0
pMacroInitialOffset[A]	2 – 68 [MT]	15 MT
pMicroInitialOffset[A]	0 – 239 [µT]	22 µT
pMicroPerCycle	640 – 640000 [µT]	5840 µT
pOffsetCorrectionOut	13 – 15567 [µT]	161 µT
pRateCorrectionOut	2 – 1923 [µT]	18 µT
pdMicrotick	0.0125, 0.025, 0.050, 0.10 [µs]	0.050 µs

5. Resultados

Los Subcapítulos 4.4-5 asi como el presente capítulo definen la etapa de diseño detallado de la metodología de diseño de la Figura I.1, ya que en éstos se presenta la especificación formal con SDL así como la simulación, depuración y refinamiento de la especificaión.

Una vez que se obtiene la especificación formal de un sistema, norma o protocolo, ésta debe pasar por tres etapas, simulación, validación y pruebas de comportamiento (*test*); de manera formal dichas etapas deben llevarse a cabo simultáneamente al desarrollo de la especificación con ayuda de herramientas de software específicas [6, 25]; en este caso en particular no se dispone de herramientas adicionales, por lo que las etapas mencionadas se realizaron en forma secuencial al desarrollo de la especificación empleando las propiedades de edición, análisis (sintáctico y semántico), simulación integral y generación de MSCs que proporciona la herramienta Cinderella SDL.

Las etapas de simulación y de validación se aplican a la especificación formal para corregir errores de diseño y comprobar que satisfaga los requerimientos de comportamiento. La etapa de pruebas de comportamiento tiene como finalidad verificar que una realización particular de la especificación reaccione ante los eventos de manera adecuada y se obtenga un grado elevado de confiabilidad del sistema. Debido a que el desarrollo de esta última etapa es una tarea compleja, que exige el dominio de lenguajes como TTCN-3 (*Testing and Test Control Notation*, tercera versión) [25] y el uso de herramientas de validación y de verificación no disponibles, no se realizó la etapa de pruebas de comportamiento en el presente trabajo de tesis de manera formal.

5.1. Etapas de simulación y validación

La especificación en SDL del protocolo Flexray se realizó mediante la construcción de un sistema compuesto por un conjunto de bloques, los cuales contienen procesos que describen su funcionamiento. Para realizar la etapa de simulación, o prueba general del sistema (prueba de caja negra), se realizó la configuración inicial de cada nodo, incluyendo los valores a transmitir (Figura 4.27) y como resultado se obtuvo la siguiente información:

- Los valores con que se ha configurado cada nodo.
- La recepción de las tramas de inicio del sistema y el ID del proceso receptor, este último identifica a cada nodo.
- Una vez que se logra la sincronización del *cluster*, los nodos transmiten los valores asignados en la siguiente ranura disponible, previamente configurada, para realizar su transmisión. Si por alguna razón no existen datos disponibles para transmisión, el nodo transmite una trama nula; todo ello se refleja en los resultados de la simulación.

Durante la etapa de simulación de la especificación del protocolo Flexray, se llevó a cabo la etapa de validación, mediante la supervisión constante del explorador de Cinderella SDL (Figura 5.1), el cual visualiza el estado en que se encuentra la CEFSM asociada al proceso que se esté valorando y los valores de las variables declaradas en dicho proceso. Durante la etapa de simulación se analizó la reacción del proceso ante la recepción de las señales que constituyen su alfabeto de entrada.

Cinderella SDL es capaz de realizar el proceso de depuración al brindar la posibilidad de establecer puntos de ruptura donde se quiere detener la ejecución, habilitar la ejecución paso por paso o seguir la ejecución de llamadas a procedimientos. De forma que la herramienta permite hacer un seguimiento, en modo gráfico, del símbolo SDL que se está procesando en todo momento; sin embargo, analizar el funcionamiento de un conjunto de procesos en el interior de un bloque o de un proceso con muchos estados, dificulta el seguimiento de la ejecución (como la mayoría de los procesos de la especificación del protocolo).

Los errores detectados durante la etapa de simulación consisten principalmente en: tramas ensambladas erróneamente, códigos de CRC calculados erróneamente, transiciones implícitas, interpretación errónea de algún aspecto del protocolo, etc. Se detectaron errores en el funcionamiento de la especificación, cuyo origen no fue posible determinar con el depurador o explorador de Cinderella SDL, como los originados por el funcionamiento intrínseco de la norma SDL y de la especificación misma, para ello se recurrió al generador de MSCs; en algunos casos, se parametrizaron algunas señales para poder determinar el origen de los errores. La Figura 5.2 muestra el aspecto de un conjunto de señales dentro del esquema MSC (para generar el MSC la simulación se puso en pausa y llevar a cabo el análisis del MSC generado), el cual permite observar la evolución de la simulación como una secuencia de eventos ordenadas en el tiempo.

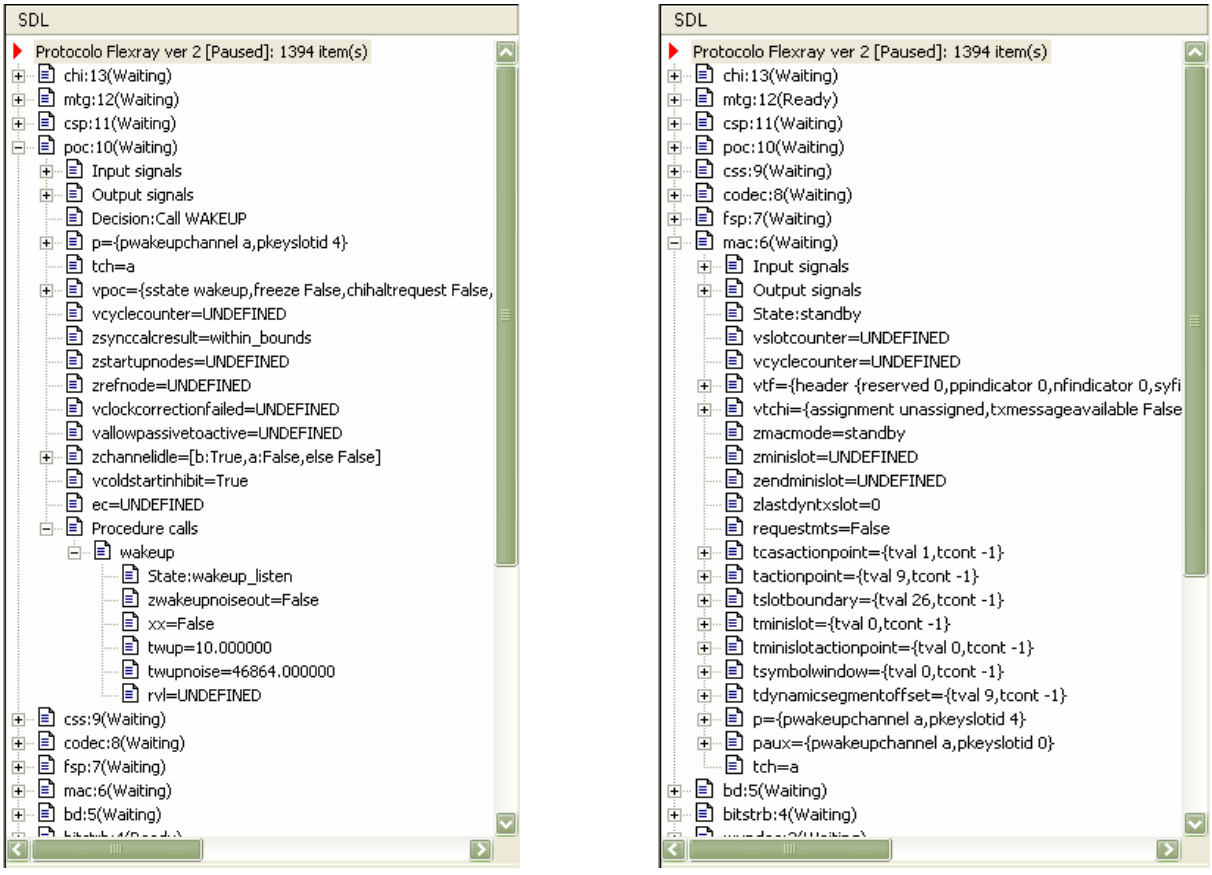


Figura 5.1. Exploración de procesos durante la etapa de simulación

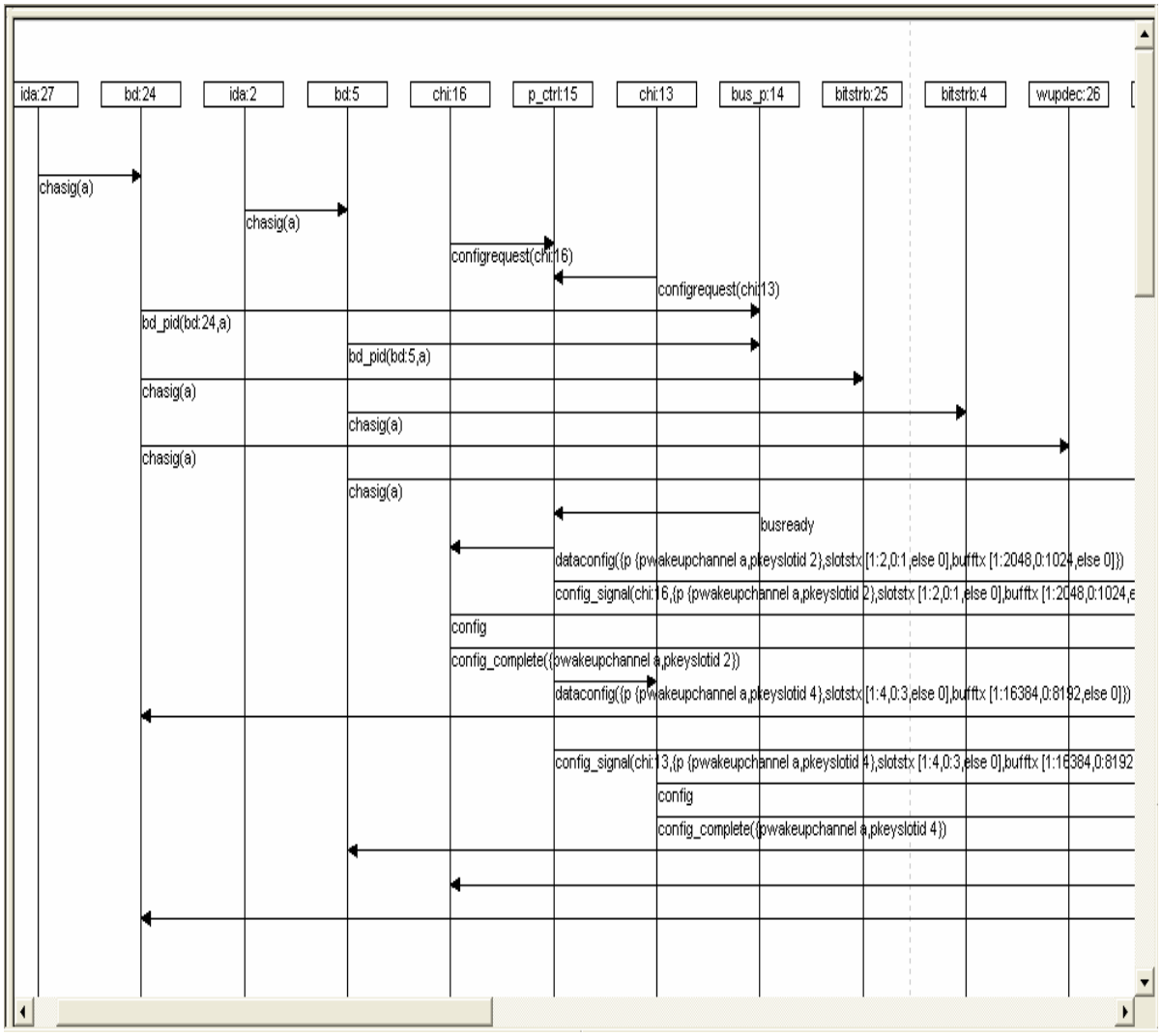


Figura 5.2. Ejemplo de los casos de usos o MSCs generados con Cinderella SDL.

Parte de la complejidad de un protocolo de comunicaciones reside en la gestión de sus temporizadores, por lo que en la validación del sistema es fundamental comprobar el correcto funcionamiento de los mismos. Durante este proceso se detectaron errores en el comportamiento de la especificación; una vez corregidos los errores, se dio inicio a un nuevo ciclo de simulación (Figura 1.1, etapas del diseño formal, Subcapítulo 1.4). En cada caso, la validación se consideró completa después de verificar un conjunto de esquemas MSC representando varios casos de uso, y tras una exploración de estados en la que no se detectaron errores.

Es importante señalar que tanto la verificación y la exploración fueron llevados a cabo de forma manual, y que los errores más complejos de solucionar se originaron debido a las características de rendimiento de la PC en que se ejecutó la simulación de la especificación; para solucionar lo anterior se optimizaron algunos procedimientos de cálculo, procesos y el uso de tiempo libre de red entre cada ranura. Para la representación del tiempo base (Apartado 2.5.3) o unidad mínima de tiempo se utilizó la unidad, esto es: gdSampleClockPeriod Duration= 1 representando el valor 0.0125 µs de la que se derivan dos temporizadores básicos: gdBit Duration= float (cSamplesPerBit) * gdSampleClockPeriod que define la duración nominal de bit con cSamplesPerBit= 8, la duración del temporizador de *microticks* definido como: pdMicrotick Duration= float (pSamplesPerMicrotick) * gdSampleClock-

Period con pSamplesPerMicrotick= 4 el cuál sirve como fuente de temporización de los temporizadores de capas superiores.

Adicionalmente para un mejor rendimiento de la herramienta durante la simulación (Apartado 2.5.3), se llevaron a cabo los siguientes ajustes al equipo de cómputo:

- Se redujo la carga de operaciones y servicios del sistema operativo en uso.
- La velocidad de procesamiento se incremento hasta sus límites tolerables.
- Se incremento significativamente la RAM (*Random Access Memory*) del equipo.

En consecuencia a lo anterior, las características de rendimiento del equipo de cómputo usado durante la simulación de la especificación es:

- Sistema operativo MS Windows XP.
- Espacio en disco duro superior a 10 GB.
- Procesador Semprom AMD a 2090 MHz.
- Memoria RAM 1.46 GB.

5.2. Análisis de los resultados

La etapa de validación de la especificación del protocolo Flexray se dividió en dos partes:

- Verificación de la detección de errores, mediante el cálculo del CRC.
- Verificación completa del sistema, omitiendo el cálculo del CRC.

Dicha división fue necesaria debido a que la herramienta Cinderella SDL no genera archivos de simulación, sino que almacena toda la información en memoria y conforme avanza la simulación, el procedimiento para el cálculo y verificación del CRC requiere más tiempo de procesamiento ya que existe menos memoria disponible. Lo anterior ocasiona el siguiente conjunto de errores o mal funcionamiento del protocolo:

- Imposibilita la sincronización en la fase inicial de la carga del sistema dentro de los primeros cuatro ciclos de comunicación.
- No se realiza la sincronización del *cluster* durante los siguientes cuatro ciclos de comunicación.
- Transmisión y/o recepción de tramas fuera del segmento que le corresponde.
- Colisión de tramas.
- Generación de un error fatal del protocolo debido a la pérdida total del cronograma local de temporización y sincronización.
- Por lo anterior, se eliminó el procedimiento de muestreo de bit debido a que incrementa el consumo de memoria a razón de 8 a 1 por cada bit, de manera que no afecta la capa de enlace entre el nodo y la capa física, y por lo tanto no afecta el funcionamiento del protocolo.

5.2.1. Verificación y pruebas sobre el CRC

La especificación para realizar pruebas sobre el cálculo de CRC y verificar su correcto funcionamiento está constituido por: los procesos codec, bitstrb y bd (para cada nodo), y el medio de transmisión de bits descrito por el proceso bus; sin embargo, es necesario señalar que han sido ligeramente modificados para poder realizar las pruebas exclusivas sobre el CRC, manteniendo los estados y la descripción de cada proceso según la especificación del protocolo. Dichas modificaciones consisten en:

- Redireccionamiento de algunas señales como son: DecodingHalted, HeaderReceived, PotentialIdleStart, etc. La dirección de las señales entre los procesos codec, bitstrb y bd se mantienen inalteradas.
- El proceso codec se modificó para que una vez ejecutado el símbolo de inicio cambie al estado Wait_For_CEStart después de forzar el modo de operación normal; a partir de este estado el proceso puede ejecutar la transmisión o recepción de tramas si las señales que causan una transición son: TransmitFrame o CEStart respectivamente.
- El procedimiento PrepBitStream, encargado de ensamblar la trama, es modificado ligeramente al finalizar, con objeto de introducir errores en la trama binaria que ha sido preparada para transmisión, de este modo el codec del nodo receptor puede determinar si existe error alguno, además de notificar si hubo o no error en la trama.
- Se introdujo un pequeño conjunto de señales para notificar que no existió ningún error, estas señales son HeaderCRCOK y FrameCRCOk, mientras que en caso de existir algún error se emplean las señales ErrorHeaderCRC o ErrorFrameCRC.

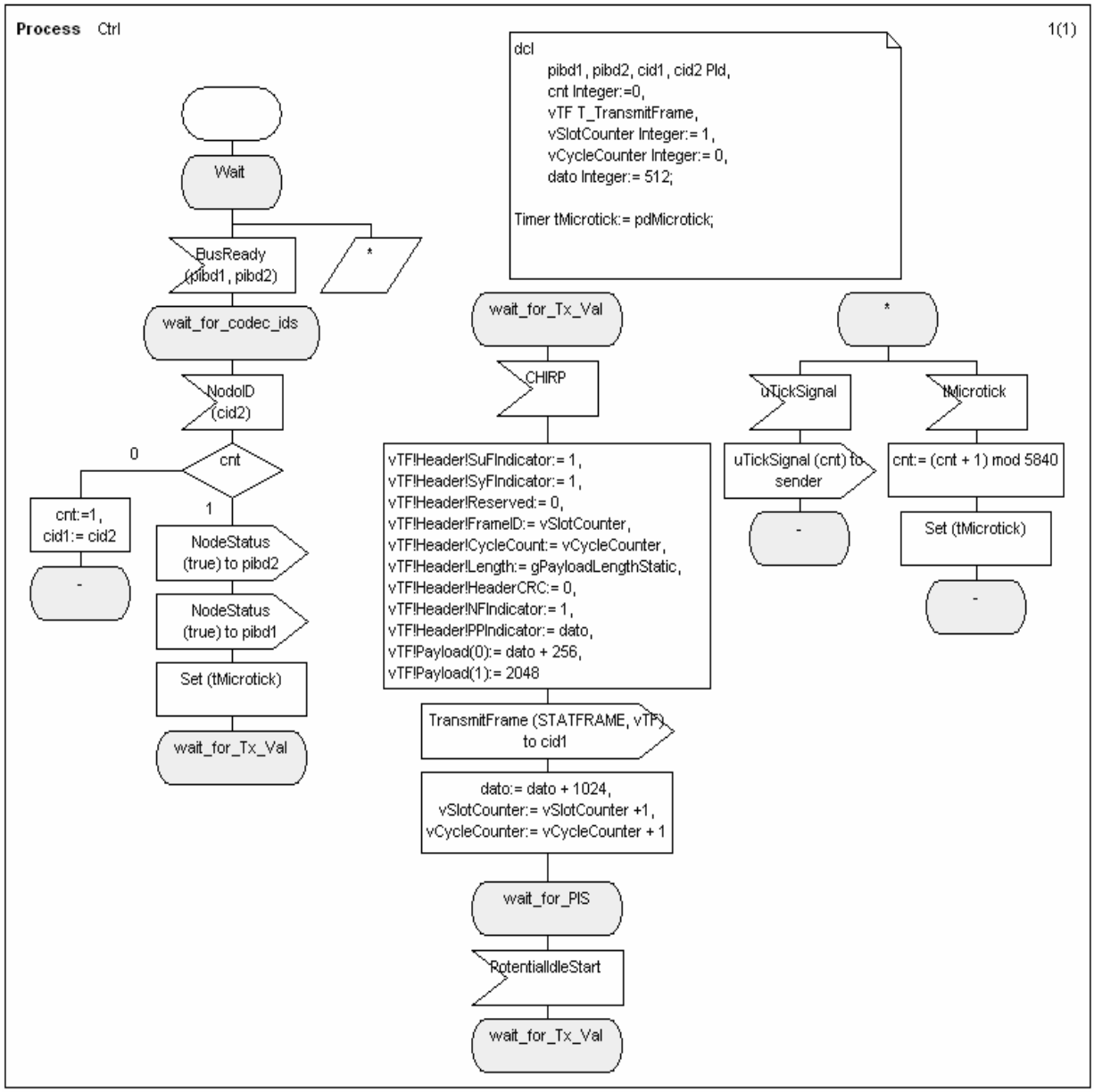


Figura 5.3. Proceso administrador de mensajes para la evaluación y pruebas sobre CRC.

El proceso Ctrl tiene como función administrar la transmisión de mensajes (no confundir con el proceso P_Ctrl del sistema completo) y para la evaluación del procedimiento del CRC utiliza la señal NodoID para diferenciar a los procesos codec de cada nodo que referenciará como transmisor como receptor (Figura 5.3). Una vez que alcanza el estado wait_for_Tx_Val, el proceso espera hasta recibir la señal CHIRP de cualquiera de los nodos, mientras reconoce que no existe actividad en el *bus* y puede iniciar la transmisión de un mensaje con el nodo asociado al identificador de proceso almacenado en la variable cid1. Actualiza los valores de la trama en la variable vTF y los envía al codec correspondiente por medio de la señal TransmitFrame, incrementa el contador de ciclos y de ranuras en uno, vCycleCounter y vSlotCounter respectivamente.

La variable dato se incrementa después de ser actualizada en vTF en el campo correspondiente a la primer palabra doble, mientras que la segunda siempre se le asigna el valor de 2048. Tras ejecutar la transición, el proceso entra al estado wait_for_PIS, cuando se recibe la señal PotentialIdleStart indica que el nodo receptor está decodificando una trama y el control es llevado al estado wait_for_Tx_Val para iniciar un nuevo ciclo de evaluación del CRC.

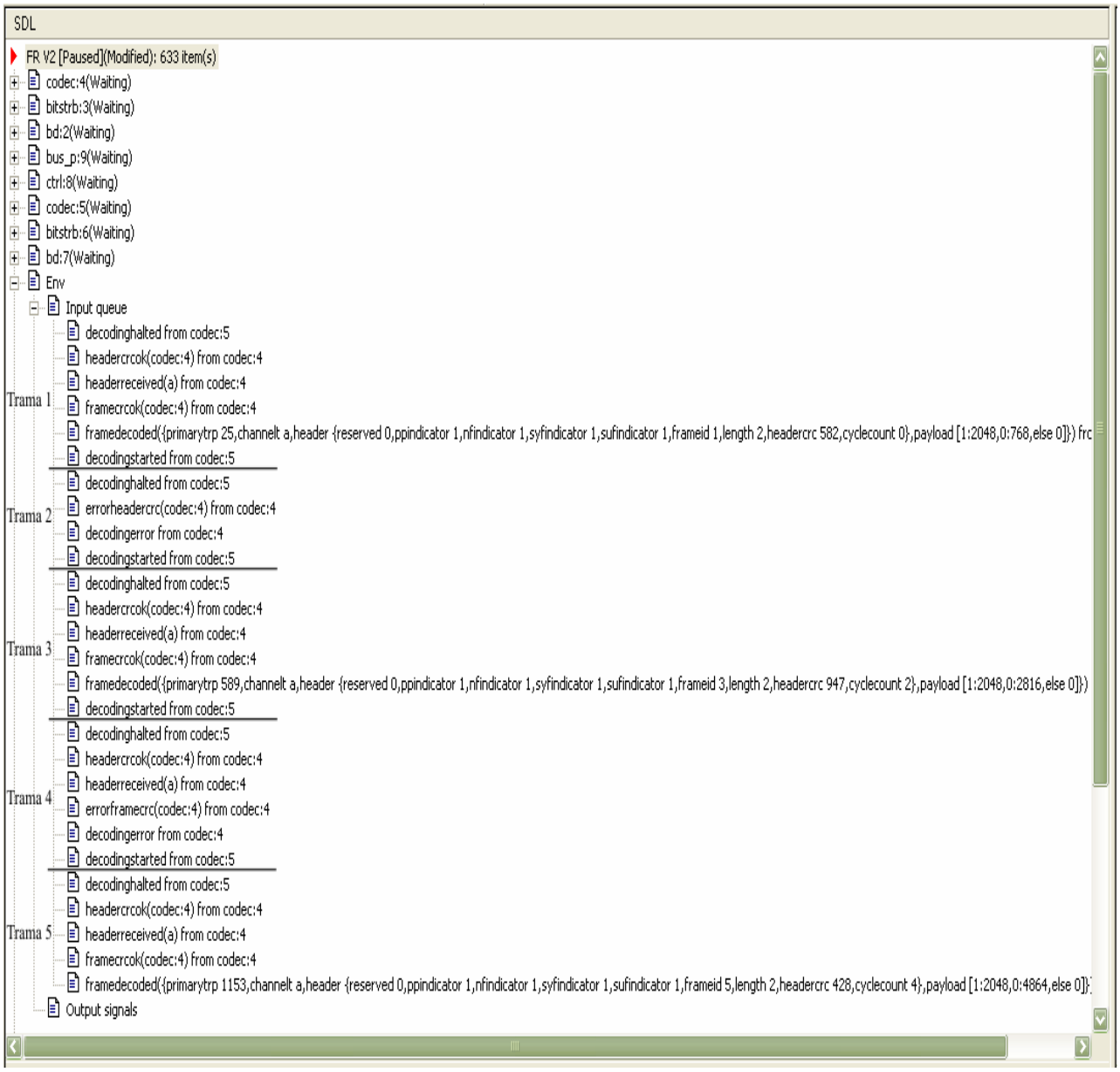


Figura 5.4. Pruebas de verificación y detección de errores con el CRC.

La Figura 5.4 muestra el conjunto de señales como resultado de la simulación durante las pruebas sobre el CRC. Estas señales son emitidas por el proceso codec de cada nodo, de cada transmisor y receptor. La Figura 5.4 muestra que se usaron cinco tramas de datos para evaluar el funcionamiento del CRC, cada trama denotada por Trama 1,…, Trama 5. El enfoque con que se desarrolló la prueba sobre el CRC es conocido como prueba de caja negra, por lo tanto no es posible visualizar el proceso llevado a cabo, más bien visualiza los resultados esperados como un conjunto de señales. Analizando este conjunto de señales se observa lo siguiente:

- Cada inicio y final de trama se identifican con las señales DecodingHalted y DecodingStarted, independientemente de que se hayan detectado errores en la trama con la evaluación del CRC. Ambas señales son emitidas por el proceso transmisor del mensaje cuyo pid es 5.
- Las señales en cada trama se envían por el proceso receptor cuyo pid es 4. Tres de las cinco tramas fueron recibidas correctamente lo cual se indica con la señal FrameDecoded y las señales que le anteceden a tal trama. Las tramas 2 y 4 fueron decodificadas con error, la primera causó un error en el encabezado y fue señalizado con la señal ErrorHeaderCRC, el segundo caso se detectó un error en la parte alta del formato de la trama y se notificó con la señal ErrorFrameCRC. En ambos casos la última señal emitida por el receptor es DecodingError.

Los resultados obtenidos en esta primera parte de la evaluación son consistentes con la especificación, ya que según la descripción del procedimiento PrepBitStream, sólo se alteraron dos bits de aquellas tramas cuyo campo CycleCount contenía el valor uno o tres:

- *Tramas con el contador de ciclos igual a uno*: Los bits alterados son el 11 y el 21, los cuales son parte de la cabecera de la trama.
- *Tramas con el contador de ciclos igual a tres*: Los bits alterados son el 61 y el 71 cuya ubicación es superior a la longitud del encabezado de la trama y son correspondientes al segmento de carga útil.

Por lo tanto se concluye que la detección de errores con auxilio de la especificación del CRC es funcional, ya que el contador de ciclos de las tramas recibidas son 0, 2 y 4. Adicionalmente se emuló la transmisión y recepción de mensajes usados por la especificación completa del protocolo; es decir, se utilizó un medio físico para transmitir, un nodo transmisor y uno receptor simulados.

Los archivos correspondientes a la especificación para llevar a cabo las pruebas sobre el CRC y su correspondiente MSC generado se pueden consultar en la documentación que se anexa en formato digital a este documento de tesis.

5.2.2. Simulación del sistema omitiendo el CRC

En este apartado se analizan los resultados obtenidos de la simulación de la especificación, los cuales representan el funcionamiento del sistema en conjunto. La Figura 5.5 muestra la señal que transporta los parámetros con que cada nodo se configuró de manera dinámica mediante la señal Config_Signal, esto se debe a que dichos parámetros siempre serán distintos de un nodo a otro. El conjunto de señales rx_StartupFrame son emitidas por cada nodo durante la fase de sincronización, cuando inician operación bajo condiciones de sincronía las señales rx_Signal transportan los siguientes parámetros:

- El ciclo de comunicación.
- ID del nodo receptor del mensaje.
- Las ranuras en las que fueron decodificadas y
- Los mensajes contenidos en cada trama decodificada.

La Figura 5.6 muestra el comportamiento de los nodos durante la fase de inicio del sistema y cuando alcanzan el modelo de degradación de errores. Las siguietes consideraciones deben tomarse en cuenta para entender los resultados mostrados las Figuras 5.5 y 5.6:

- Cada nodo en el *cluster* es referenciado con el identificador de su procesos CHI asociado, los cuales son N13 y N16 (nodo de seguimiento y nodo principal respectivamente).
- La Figura 5.5 presenta las tramas de sincronización decodificadas por el nodo receptor, mientras que la Figura 5.6 bosqueja las mismas tramas pero desde la perspectiva del nodo transmisor.

El comportamiento general durante la sincronización se puede describir fundamentalmente en dos etapas:

- Una vez inicializado el *cluster* (con el envío del patrón WUP), el poc de cada nodo entra al modo de inicio de sistema; el nodo N16, cuyos temporizadores para la ventana de detección expiran, establece los mecanismos para que asuma el rol de nodo principal e inicie la fase de sincronización con la transmisión del símbolo CAS (en varias de las pruebas realizadas ambos nodos transmitieron este símbolo simultáneamente). Como se muestra en la Figura 5.6, únicamente el nodo N16 se mantiene los siguientes cuatro ciclos transmitiendo tramas de sincronización (con las señales rxStartupFrame decodificadas por el nodo N13 en la Figura 5.5).
- El nodo N13 intenta acoplarse al nodo transmisor en los primeros dos ciclos generando su propio cronograma de temporización a partir del nodo N16.y verifica que la integración con el mismo sea consistente durante los siguientes dos ciclos; es decir, verifica que las tramas recibidas estén dentro del cronograma de temporización.

Figura 5.5. Resultados obtenidos de la simulación de la especificación.

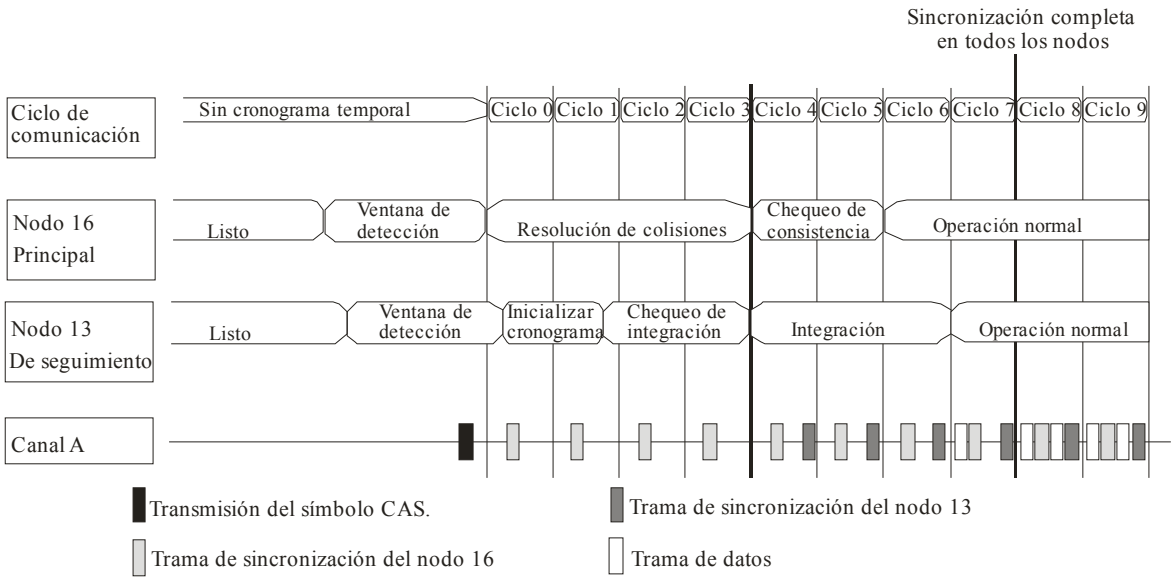


Figura 5.6. Comportamiento de los nodos bajo condiciones libre de errores.

Cuando los nodos del *cluster* alcanzan la segunda etapa, denominada consistencia e integración, ambos nodos son habilitados para transmitir tramas de sincronización, lo cual se refleja en la Figura 5.5 donde ahora el nodo N16 detecta las tramas enviadas por el nodo N13:

- Durante los subsecuentes dos ciclos, el nodo principal N16 verifica que las tramas recibidas sean consistentes con su cronograma; en el sexto ciclo se activa el modelo de degradación de errores y el nodo entra al estado operación normal activa.
- Si el nodo de seguimiento N13 se mantiene los siguientes tres ciclos recibiendo las tramas de sicronización se considera integrado al *cluster* y activa el modelo de degradación de errores en el séptimo ciclo para alcanzar el estado operación normal activa.

Cuando cualquiera de los nodos en el *cluster* entra en el modo de operación normal, la trasmisión de mensajes se activa un ciclo después; es decir, para los nodos N16 y N13 dan inicio a la transmisión de mensajes en los ciclos 7 y 8 respectivamente.

Una vez que se alcanza el modelo de degradación de errores, los nodos son habilitados para realizar transmisión de mensajes, de este modo las tramas de sincronización ahora pueden transportar mensajes de datos; una característica adicional del protocolo es el soporte para tramas nulas, es decir, cuando un nodo no tiene información disponible para transmitir puede ensamblar una trama nula lo que significa que no contiene datos en el segmento de carga útil utilizables. Esto se refleja en la Figura 5.5 y se detalla en la Tabla 5.1 en la que se identifican que tramas son nulas y cuales no, las tramas de sincronización así como el valor de los mensajes que transportan, el ciclo al que corresponden y los nodos transmisor y receptor.

Tabla 5.1. Descripción de los resultados obtenidos durante operación normal activa.

Ciclo	ID Tx	Sinc	Valor	Ranura	Nula
6	N16	No	x	1	Si
	N16	Si	x	2	Si
	x	x	x	x	x
	x	x	x	x	x
7	N16	No	1130	1	No
	N16	Si	2154	2	No
	N13	No	x	3	Si
	N13	Si	x	4	Si
8	N16	No	1183	1	No
	N16	Si	2207	2	No
	N13	No	8298	3	No
	N13	Si	16490	4	No
9	N16	No	x	1	Si
	N16	Si	x	2	Si
	N13	No	x	3	Si
	N13	Si	x	4	Si
10	N16	No	1289	1	No
	N16	Si	2313	2	No
	N13	No	8404	3	No
	N13	Si	16596	4	No
11	N16	No	x	1	Si
	N16	Si	x	2	Si
	N13	No	x	3	Si
	N13	Si	x	4	Si
12	N16	No	1395	1	No
	N16	Si	2419	2	No
	N13	No	8510	3	No
	N13	Si	16702	4	No

6. Conclusiones y trabajos futuros

En el presente trabajo de tesis se ha realizado una investigación en dos áreas muy importantes en el desarrollo tecnológico e industrial: buses de campo y FDTs; específicamente, el protocolo de comunicaciones automotrices e industriales Flexray y la FDT SDL. La culminación de la investigación realizada y aterrizada en la obtención de una especificación ejecutable del protocolo Flexray como caso de estudio, (objetivo central de la presente tesis) deriva el siguiente conjunto de aportaciones:

- Se realizó un breve estudio de las principales técnicas de descripción formal resaltando las características más significativas de cada una (Capítulo 1).
- Se realizó un exhaustivo estudio de la norma SDL detallada en la recomendación z.100, lo cual fue básico para lograr un mejor entendimiento del protocolo Flexray.
- Se estudió a detalle el protocolo Flexray, el cual es un tipo de protocolo estandarizado independientemente de las principales organizaciones: ISO, ITU-T, IEEE, y es mantenido por el consorcio FlexRay™; fue importante dominar el lenguaje SDL, dado que el aspecto funcional del protocolo Flexray viene descrito en [14] con diagramas SDL y para poder establecer las dependencias de las señales entre emisor(es) y receptor(es), además de complementar la especificación en SDL y lograr que sea ejecutable para su posterior simulación, validación y prueba. Otro aspecto importante fue el estudio de los parámetros de configuración para llegar al diseño correcto de una red (*cluster*) Flexray con dos nodos tipo *coldstart*.
- Fue necesario estudiar profundamente el lenguaje ASN.1, detallado en la recomendación z.105, durante la fase de pruebas, ya que el procedimiento para calcular los CRCs tomaba demasiado tiempo en su ejecución y con la manipulación correcta de esos tipos de datos se pudo reducir en gran medida el tiempo para calcular los CRCs.
- Se ha obtenido una especificación formal SDL que describe la capa DLL o capa de enlace de datos del protocolo Flexray sin ambigüedades incluyendo una detallada descripción de la capa física y una reducida descripción de la capa de aplicación.

La valoración final sobre la utilidad de la aplicación de las FDTs en el diseño de sistemas de comunicaciones y protocolos es positiva desde el punto de vista de la especificación, ya que permite especificar y describir a detalle el funcionamiento de sistemas de acuerdo a los requerimientos previamente establecidos, así como su ejecución, simulación, validación y pruebas.

Tras el trabajo realizado, se proponen las siguientes líneas de investigación:

- Complementar la especificación del protocolo Flexray con la inclusión de la especificación de la capa de aplicación, así como una especificación completa de la interfaz CHI.

Habilitar el segmento dinámico del ciclo de comunicación Flexray, lo que implica realizar un estudio de la asignación de prioridades en sistemas con arquitectura TTA.

- Incluir la capa de supervisor y de redundancia en el CC del ECU o nodo.
- Realizar una valoración de la verificación usando herramientas de software con objeto de obtener una mayor rigurosidad.
- Implementar el protocolo Flexray en FPGAs y en microcontroladores para diseñar una red que contenga al menos dos nodos *coldstart* y al menos uno no *coldstart*.

Bibliografía

[1] Anderson, O., Faergemand, O., Moller-Pedersen, B., Reed, R., and Smith, J.R.W., Systems Engineering Using SDL-92, North-Holland, 1994.

[2] Bosch, CAN Specification Version 2.0, Robert Bosch GmbH, September 1991.

[3] Burns, A., & Wellings, A., A Structured Design for Real-time Systems, Real-time and Distributed Systems Research Group, Department of computer Science, University of York, Heslington, York, YO1 5DD, UK.

[4] CCITT, Specification and Description Language, Recommendation CCITT Z.100, 1988.

[5] Daniel, A., & Williams, W., System Analysis and Modeling: 4th International SDL and MSC Workshop, Springer: 1, edition April 2005.

[6] Díaz, R., Reutilización de Requisitos Funcionales de Sistemas Distribuidos Utilizando Técnicas de Descripción Formal, Universidad de Vigo, Tesis doctoral, Febrero de 2002.

[7] Ehrig, H., & Mahr, B., Fundamentals of Algebraic Specification 1, Springer-Verlag, 1985.

[8] Ernest, P., Focus On SDL (The Premier Press Game Development Series), Course Technology PTR; 1 edition

[9] F., W., Von, Henke, Formal Analysis of Fault-Tolerant Algorithms in the Time-Triggered Architecture, Universität ULM, July 2003.

[10] Fernández, J., Tiempo Real en Redes de Comunicaciones, Depto de Ingeniería de Sistemas y Automática, Universidad Málaga.

[11] FlexRay Electrical Physical Layer Application Notes, V2.1 Rev.B, Nov. 2006.

[12] FlexRay Electrical Physical Layer Specification, V2.1 Rev.B, Nov. 2006.

[13] FlexRay Preliminary Node-Local Bus Guardian Specification, V2.0.9, Dec. 2005.

[14] FlexRay Protocol Specification, V2.1 Rev.A, Dec. 2005.

[15] Getting Started With Cinderella SDL, Cinderella i/s, 1998.

[16] ISO/IEC 7185, Programming Language – PASCAL, 1982.

[17] ISO/IEC 9074, Information processing systems, Open System Interconnection, ESTELLE: A formal description technique based on an Extended Transition Model, 1989.

[18] ISO/IEC 9646, Information processing systems, Open System Interconnection, Conformance Testing Methodology and Framework, 1991.

[19] ISO/IEC 9907, Open System Interconnection, LOTOS: A formal description technique based on the Temporal Ordering of the Observational Behavior, 1989.

[20] ITU-T, Recommendation Z.100, Specification and Description Language SDL, 1993.

[21] ITU-T, Recommendation Z.105, Use of SDL with ASN.1, 1995.

[22] ITU-T, Recommendations X.280 and X.680-683, Abstract Syntax Notation One (ASN.1), 1994.

[23] J.A., de la Puente, Diseño de sistemas de tiempo real, Universidad Politécnica de Madrid, Septiembre de 2001.

[24] Ellsberger, J., European Telecommunications Standards Institute, University of Lübeck, Lübeck, Prentice Hall, 1997

[25] Nogueira, J., METODOLOGIAS DE ESPECIFICACION FORMAL APLICADAS A MODELOS NORMALIZADOS DE PROTOCOLOS DE COMUNICACIÓN INDUSTRIALES, E.T.S.I. Telecomunicación, Universidad de Vigo, Tesis Doctoral, Octubre 2000.

[26] Laurent, D., Visual Design Executable Models, 2001

[27] Meer, J., Roth R & Vuong S., "Introduction to algebraic specifications based on the language ACT ONE", Computer Networks and ISDN Sytems, 23, North-Holland, pp. 363-392, 1992.

[28] Mr. Pleil, M., Mr. Busse, M., EASIS - Electronic Architecture and System Engineering for Integrated Safety Systems - Data exchange concepts for gateways, Nov. 2006.

[29] Simon, D., An Embedded Software Primer, Addison Wesley, 1999.

[30] Turner, K.J., (Editor), "An engineering approach to formal methods". Proceedings of the Protocol Specification, Testing and Verification XIII, pp. 357-380, 1993.

[31] Turner, K.J., (Editor), Using Formal Description Techniques: An Introduction to Estelle, Lotos and Sdl, John Wiley & Sons. 1993.

[32] Ullman, H., "Introduction to Automata Theory, Languages and Computation", Addison Wesley, 1979.

[33] Wearn M., Real Time Communication, Evaluation of protocols for automotive system, Institute of Technology in Stockholm (KTH, Kungliga Tekniska Högskolan), Master of science Thesis, 2003.

Sitios de internet

[URL1] http://www.cinderella.dk/, "Página Web de la empresa Cinderella Aps", Octubre 2006.

[URL2] http://preso.wanadoo.fr/doldi/sdl/, "Página personal del autor Laurent Doldi" Noviembre 2006.

[URL3] http://www.flexray.com/, "Página Web de Flexray Consortium creadora del protocolo Flexray", Julio 2006.

[URL4] http://www.itu.int/ITU-T/, "Página Web de ITU, sector telecomunicaciones ITU-T", Febrero 2007.

[URL5] http://www.sdl-forum.org/, "foro de SDL", Marzo 2007.

[URL6] http://www.ieee802.org/1/pages/802.1x.html, "Página Web de IEEE para publicación de documentos", Octubre 2007.

[URL7] http://www.maxim-ic.com/appnotes.cfm/appnote_number/736/, "Pagina web de Maxim Integrated Products, Dallas Semiconductor, proveedor de C.I. y componentes discretos", Abril 2007.

[URL8] http://www.tzm.de, "página Web de Transfer Zentrum Mikroelektronik", Octubre 2007.

[URL9] http://webs.uvigo.es/servicios/biblioteca/arquivo/TESIS_02_05.html, Junio 2008.

[URL10] http://www.coit.es/index.php?op=actos_premios_180, Junio 2008.

[URL11] http://spic.kaist.ac.kr/~selab/html/Study/Lab%20Seminar/, Junio 2008.

A. Acrónimos

μT	Microtick
AP	Action Point
BD	Bus Driver
BIST	Built-In Self Test
BITSTRB	Bit Strobing Process
BSS	Byte Start Sequence
CAS	Collision Avoidance Symbol
CC	Communication Controller
CE	Communication Element
CHI	Controller Host Interface
CHIRP	Channel Idle Recognition Point
CODEC	Coding and Decoding Process
CRC	Cyclic Redundancy Code
CSP	Clock Synchronization Process
CSS	Clock Synchronization Startup Process
DTS	Dynamic Trailing Sequence
ECU	Electronic Control Unit, Same as node
EMC	Electromagnetic Compatibility
ERRN	Error Not signal
FES	Frame End Sequence
FSP	Frame and Symbol Processing
FSS	Frame Start Sequence
FTDMA	Flexible Time Division Multiple Access (media access method)
FTM	Fault Tolerant Midpoint

ID	Identifier
INH	Inhibit signal
MAC	Media Access Control Process
MT	Macrotick
MTG	Macrotick Generation Process
MTS	Media Access Test Symbol
NIT	Network Idle Time
NM	Network Management
POC	Protocol Operation Control
RxD	Receive data signal from bus driver
RxEN	Receive data enable signal from bus driver
SDL	Specification and Description Language
SPI	Serial Peripheral Interface
STBN	Standby Not signal
SuF	Startup Frame
SW	Symbol Window
SyF	Sync Frame
TDMA	Time Division Multiple Access
TRP	Time Reference Point
TSS	Transmission Start Sequence
TxD	Transmit Data signal from CC
TxEN	Transmit Data Enable Not signal from CC
WUP	Wakeup Pattern
WUS	Wakeup Symbol
WUPDEC	Wakeup Pattern Decoding Process

B. Guía e inicialización con Cinderella SDL.

En este apéndice se presenta una breve introducción sobre cómo instalar la herramienta de implementación Cinderella SDL, así como una guía sobre su uso, presentando un ejemplo práctico, que muestra cómo editar, simular y verificar un sistema SDL, En capítulos anteriores se han mencionado las principales características y los requerimientos mínimos que Cinderella SDL necesita para su correcta instalación.

En Los Laboratorios Avanzados de Electrónica, se cuenta con equipo de cómputo con:

- Procesador de doble núcleo.
- Memoria RAM superior a 1 GB.

Lo que asegura soporte para especificaciones muy robustas, independientemente del sistema operativo en uso; sin embargo, dado que SDL fue diseñado para especificar y/o describir sistemas de comunicaciones, sean éstos de tiempo real o no, es necesario una correcta gestión de los temporizadores.

Cinderella SDL es una herramienta de implementación del estándar SDL, la cual se puede descargar de [URL1]. Una vez instalado, la aplicación se puede ejecutar inmediatamente como versión de prueba durante 15 días, tras su expiración es necesario registrar la instalación para obtener la licencia y de este modo cambiar el estado de la instalación a modo permanente [15].

A continuación se describe paso a paso cómo hacer una especificación SDL muy simple, este ejemplo es útil para ilustrar algunos conceptos básicos de SDL y de Cinderella SDL.

- *Paso 1*: Abrir Cinderella SDL y en el menú File dar clic en New.
- *Paso 2*: Abrir el menú Vew y posicionar el cursor en el elemento Toolbars, aparecerá un nuevo submenú, activar todos aquellos elementos del submenú que no lo estén. Un elemento activo aparece con "√"al lado del texto. Si todas las opciones del submenú están activados, continuar con el paso 3.

En este momento, todos los iconos en la ventana de Cinderella SDL son visibles, esto es importante para evitar confusiones cuando en uno de los pasos en esta guía se refiera a uno de ellos. La parte a la izquierda de la ventana es el explorador, mientras que la parte de la derecha es el área de trazado de los símbolos SDL de la especificación.

Con los pasos sucesivos aquí descritos, se creará un sistema SDL compuesto de un único proceso, el cual continuamente actualiza un par de variables, así mismo, muestra el uso de los temporizadores.

- *Paso 3*: Dar clic en el icono del símbolo Process en la barra de símbolos, localizada a la derecha en la ventana de la herramienta Cinderella SDL. Para identificarlo, posicionar el

cursor sobre un icono en la barra de símbolos, esto mostrará el tipo de símbolo al que hace referencia.

- *Paso 4*: Mover el cursor al área de trazado y dar clic. El símbolo del proceso se insertará en el área de trazado de la ventana.
- *Paso 5*: Mientras el símbolo esté seleccionado, introducir el texto loop, sino está activado, dar clic al botón izquierdo del ratón sobre el símbolo Process para seleccionarlo.
- *Paso 6*: Dar clic al botón derecho del ratón sobre el símbolo del proceso para mostrar un submenú.
- *Paso 7*: Seleccionar New Diagram del menú activo. Se creará un diagrama del proceso loop, la ventana de la aplicación muestra el contenido del proceso y la barra de símbolos se actualizará, mostrando los iconos de los símbolos permitidos en los procesos. El explorador muestra un mensaje de error, esto se debe a que el analizador se ejecuta mientras se realiza la especificación.
- *Paso 8*: Con el botón izquierdo del ratón, dar un clic sobre el icono Text en la barra de símbolos.
- *Paso 9*: Mover el cursor del ratón al área de trazado y dar clic al botón izquierdo del ratón, aparecerá el símbolo Text, el cual puede ser redimensionado.
- *Paso 10*: Introducir el texto: dcl counter real:= 0, uTick Integer:= 0; y Timer Tick:= 0.1 como se muestra en la Figura B.1. De esta manera se declaran las variables y temporizadores correspondientes a esta especificación.
- *Paso 11*: Insertar el símbolo Start en la parte superior del área de trazado, con ello se indica el inicio del proceso.
- *Paso 12*: Mientras el símbolo de inicio esté seleccionado, dar doble clic al botón izquierdo del ratón sobre el símbolo Task. El símbolo Task se conectará automáticamente al símbolo Start.
- *Paso 13*: Teclear Set (uTick), Para iniciar el temporizador.
- *Paso 14*: Mientras el símbolo de tarea esté seleccionado, dar doble clic al icono State, para definir un estado. mientras el símbolo esté seleccionado, introducir el texto loop.
- *Paso 15*: Seleccionar e introducir el símbolo Input para indicar la recepción de una señal, la del temporizador. Teclear Tick dentro del símbolo de entrada.
- *Paso 16*: Para conectar dos símbolos, también se puede hacer dando clic en el icono Connect Symbols en la barra de símbolos, posteriormente dar clic en el símbolo del estado loop, desplazar el cursor del ratón al símbolo Input. Y dar clic sobre él.
- *Paso 17*: Mientras el símbolo de entrada esté seleccionado, en la barra de símbolos dar doble clic al símbolo de tarea e introducir el texto Set (Tick) para iniciar de nuevo el temporizador Tick.
- *Paso 18*: Nuevamente, mientras el símbolo de tarea esté seleccionado, dar doble clic al icono de tarea y teclear uTick:= uTick + 1.
- *Paso 19*: Mientras el símbolo de tarea se encuentre seleccionado, en la barra de símbolos dar doble clic al icono State y teclear el signo menos "-", para indicar que el siguiente estado es el mismo; es decir, la transición llevará al proceso al estado loop nuevamente.
- *Paso 20*: Seleccionar el símbolo del estado loop y dar doble clic al símbolo de entrada Input.
- *Paso 21*: Teclear none para generar una transición implícita.

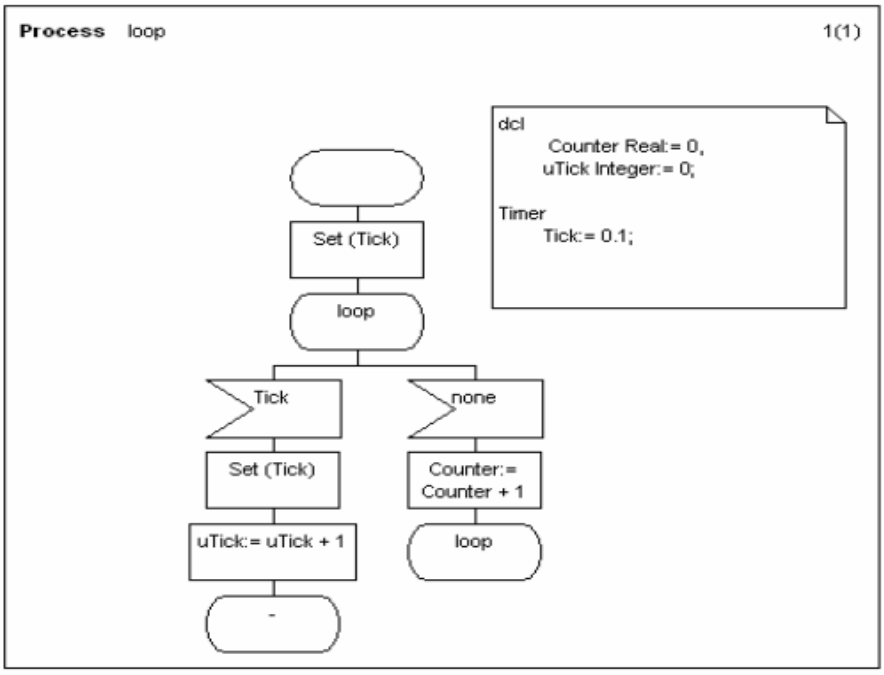


Figura B.1. Especificación final para la máquina de estados.

- *Paso22*: introducir un símbolo de tarea en la especificación y teclear el texto Counter:= Counter + 1.
- *Paso 23*: Mientras el símbolo de tarea esté seleccionado, dar doble clic al icono de estado State en la barra de símbolos y teclear el texto loop para especificar que el estado siguiente es el mismo.
- *Paso 24*: Dar clic sobre un área libre del área de trazado. El proceso será analizado nuevamente. El explorador no debe mostrar error alguno. La Figura B.1 Muestra como debe quedar finalmente la especificación.
- *Paso 25*: Iniciar la simulación dando clic en el icono Run en la barra de estado localizada en la parte inferior de la ventana de la herramienta Cinderella SDL. El explorador de la especificación muestra dos instancias; una corresponde al proceso loop y la otra al entorno del proceso.
- *Paso 26*: Para mostrar el contenido de estas instancias, dar clic a "+" asociado a cada instancia en el explorador. Con ello se visualizará el contenido del proceso o instancia seleccionada. El proceso loop muestra que la variable Counter y uTick son continuamente actualizadas.
- *Paso 27*: Para visualizar la evolución de la simulación con el diagrama de secuencia de mensajes, dar clic al icono View MSC en la barra de herramientas. Para volver a ver la especificación SDL dar clic en el icono View SD*L*.

Como se puede apreciar en la simulación, el explorador no es actualizado cada que una tarea es interpretada, así que no se aprecia cada incremento de la variable. Esto se debe a razones de configuración y rendimiento. Mientras que la Figura B.1 muestra cómo debe quedar la especificación final, la Figura B.2 muestra el resultado de la simulación en el explorador y el flujo de señales correspondiente al temporizador.

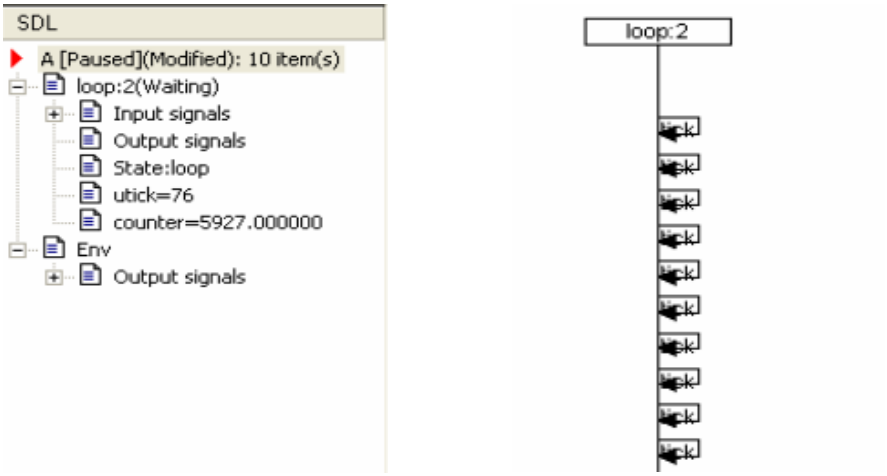


Figura B.2. Resultados de la simulación.