UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"ESPECIFICACIÓN DEL PROTOCOLO DNP3 UTILIZANDO UN LENGUAJE DE DESCRIPCIÓN FORMAL"

TESIS

PARA OBTENER EL TÍTULO DE
INGENIERO EN ELECTRÓNICA

PRESENTA

DIEGO DONIZETTI PÉREZ ORTIZ

DIRECTOR DE TESIS

ING. HERIBERTO I. HERNÁNDEZ MARTÍNEZ

HUAJUAPAN DE LEÓN, OAX.; JULIO DE 2011

Tesis presentada el 12 de Julio de 2011 ante los siguientes sinodales:

M.C. José Antonio Moreno Espinosa
M.C. Maribel Tello Bello
Dr. Alejandro Ernesto Ramírez González

Director de tesis:

Ing. Heriberto Ildefonso Hernández Martínez

Dedicatoria

Con mucho amor y cariño:

A mi madre Martha Ortiz Mendoza

A mi padre Edgar Rubén Pérez Ozuna

A mi hermano José Edgar Pérez Ortiz

A mi flaquita Nallely Hernández López

Diego

Agradecimientos

Agradezco a Heriberto I. Hernández Martínez, por permitirme trabajar bajo su dirección, por brindarme su experiencia, conocimientos, amistad y confianza.

A mis profesores por el conocimiento que me brindaron durante mi vida estudiantil.

A mis padres, aunque no existe forma de agradecer una vida de sacrificio, esfuerzo y dedicación, quiero que sientan que el objetivo logrado no es sólo mío, también es de ustedes porque la fuerza que me ayudó a conseguirlo fue su apoyo, cariño, comprensión y amistad incondicional.

A mi flaquita, por ser mi razón y todas mis razones, por luchar siempre a mi lado brindándome tu amor y tus consejos.

A mis sinodales, José Antonio, Alejandro y Maribel.

A mis amigos, Jorge (Lobo), Gabriela (Enana), Filogonio (Filo), Miguel (Chucky).

A mis compañeros de la carrera, Edilberto, Yova, David, Matías, Miguel, Gildardo y todos lo que se me escapan de la lista.

Diego

Índice

Índice de figuras

Índice de tablas

Glosario de términos

AL	Capa de Aplicación.
CRC	Código de redundancia cíclica.
DNP	Protocolo de red distribuida.
DLL	Capa de Enlace de Datos.
EM	Estación Maestra.
EPRI	Instituto de Investigación de Energía Eléctrica.
ER	Estación Remota.
FDL	Lenguaje de Descripción Formal.
FDT	Técnica de Descripción Formal.
Fragmento	Tipo de datos de la Capa de Aplicación.
HMAC	Código de autentificación de mensaje bajado en tablas *Hash*.
IED	Dispositivos Electrónicos Inteligentes.
IEC	Comité Internacional de Electrotécnicos.
Punto	El tipo de dato de entrada/salida (analógico, digital o contadores).
RTU	Unidades Terminales Remotas.
SCADA	Control supervisor y adquisición de datos.
SDL	Lenguaje de Descripción y Especificación.
Segmento	Tipo de datos de la Función de Transporte.
TF	Función de Transporte.
Trama	Tipo de datos de la Capa de Enlace de Datos.
UCA	Arquitectura de comunicaciones de servicios.

1. Introducción

El protocolo de red distribuida (DNP, *Distributed Network Protocol*) fue desarrollado por la firma WESTRONIC, ahora GE-Harris[1], a inicios de la década de los 90's con la finalidad de obtener un protocolo libre y orientado a soluciones dentro de la industria eléctrica [2]. Los protocolos de comunicaciones existentes eran propietarios y no eran compatibles con productos de diferentes fabricantes, lo cual motivó que la firma WESTRONIC diseñara su propio protocolo de comunicaciones considerando las principales características de los ya existentes.

Los principales protocolos de comunicaciones que sirvieron como referencia al diseño de DNP fueron [2]:

- *IEC 60870-5*: Protocolo de comunicaciones desarrollado por la Comisión Internacional Electrotécnica (IEC**,** *International Electrotechnical Commission*).
- *UCA 2.0*: Arquitectura de comunicaciones de servicios (UCA, *Utility Communications Architecture*) desarrollada por el Instituto de Investigación de Energía Eléctrica (EPRI, *Electrical Power Recearch Institute*)[2] [URL6].

El protocolo de comunicaciones DNP toma las capas, de enlace de datos y de aplicación, de IEC 60870-5 y considera una arquitectura flexible, sin ser totalmente diseñada en capas, como lo indica UCA 2.0. Además se consideraron las siguientes mejoras:

- Reducción del ancho de banda ocupado por el protocolo.
- Uso de la velocidad de transferencia de datos de 1,200 bps (bits por segundo).
- Compatibilidad con redes tipo control supervisor y adquisición de datos (SCADA, *Supervisory Control And Data Acquisition*)**.**

Dichas mejoras se lograron con la reducción de la cantidad de capas en la arquitectura de protocolos respecto al modelo de referencia OSI y considerando una Función de Transporte (subcapítulo 2.2) como una subcapa de la Capa de Aplicación (subcapítulo 2.1) (Figura 1.1). También se definió una trama (*frame*) de tipo 3 (FT3) y se añadió el campo para la comprobación de redundancia cíclica (CRC**,** *Cyclic Redundancy Check*).

---

[1] GE-Harris es una empresa dedicada al diseño y construcción de terminales remotas y de sistemas integrados [URL7].

[2] La versión UCA 1.0 era un protocolo propietario que no consideraba el desarrollo para sistemas SCADA.

Modelo OSI	DNP3
Capa de Aplicación	Capa de Aplicación
Capa de Presentación	-
Capa de Sesión	-
Capa de Transporte	Función de Transporte
Capa de Red	-
Capa de Enlace de Datos	Capa de Enlace de Datos
Capa Física	Capa Física (medio)

Figura 1.1. Arquitectura de protocolos DNP3 respecto al modelo de referencia OSI.

Con base en el estudio de los protocolos existentes, los desarrolladores de DNP3 incorporaron los siguientes servicios [2]:

- *Difusión (Broadcasting)*: Permite el envío de un mensaje a todos los dispositivos activos en una red.
- *Seleccionar antes de ejecutar (Select-Before-Operate-Or Not)*: Permite mayor fiabilidad mediante la utilización de algoritmos de codificación (HMAC[3]).
- *Datos con registro temporal (Time-Stamped-Data)*: Permite registrar la secuencia histórica de eventos.
- *Exactitud en el tiempo de sincronización (Accurate Time Synchronization)*: Permite contabilizar el tiempo de retardo para poder sincronizar la conexión.
- *Señalización de calidad (Quality Flags)*: Permite validar los datos que se reciben.
- *Formatos múltiples de datos (Multiple Data Formats)*: Es la habilidad para representar distintos formatos de datos, por ejemplo: 16-bit, 32-bit, bandera, punto flotante, BCD, paquete, etcétera.
- *Búsqueda de grupos (Scan Groups)*: Es la habilidad para definir y solicitar un conjunto de datos relacionados mediante una única solicitud.
- *Separación de capas (Layer Separation)*: Permite interactuar con funciones SCADA.
- *Reporte por excepción (Report-by-Exception)*: Es la habilidad de generar reportes sólo sí las mediciones cambian.
- *Indicadores internos (Internal Indications)*: Permite responder a las banderas internas del sistema.

Las principales características del protocolo DNP3 son:

- *Direccionamiento (Addressing)*: Se utilizan 2 Bytes para identificar la dirección origen y 2 Bytes para identificar la dirección destino, con lo cual se tiene una capacidad de direccionamiento de hasta 65,000 dispositivos en un enlace simple.
- *Mecanismo de confiabilidad mediante CRC múltiple (Reliability Mechanism)*: Implementa un CRC de 16 bits de longitud por cada bloque de 16 bytes[4] de los 255 bytes de datos que conforman la trama.
- *Tipo de trama (Frame Format)*: Soporta transmisión asíncrona de FT3.
- *Reconocimiento (Acknowledgement)*: Utiliza 10 Bytes para indicar el éxito de la comunicación y el establecimiento del enlace.
- *Procedimiento (Procedures)*: Sólo permite establecer procedimientos balanceados.

[3] Código de autentificación de mensaje basado en tablas hash (HMAC, *Hash Message Authentication Code*).

[4] A este proceso se conoce como CRC múltiple (*Multiple CRC*) debido a que se aplica un CRC a cada bloque que conforma una trama.

- *Flexibilidad*: Soporta las topologías Maestro/Esclavo, red de igual a igual (*peer-to-peer*) y aplicaciones de red.

Actualmente, las principales aplicaciones del protocolo DNP3 dentro de la industria eléctrica e hidráulica son: Automatización, Telemetría, Redes tipo SCADA, Monitoreo en tiempo real, Sistemas de alarmas, Protecciones eléctricas y Comunicación entre terminales remotas (RTU, *Remote Terminal Unit*) y dispositivos electrónicos inteligentes (IED, *Intelligent Electronic Devices*) [URL3].

Por otro lado, las técnicas de descripción formal (FDT, *Formal Description Techniques*) surgieron en los años 60's debido a la necesidad de contar con métodos más eficientes para el desarrollo de sistemas de cómputo y de comunicaciones. Las FDTs están basadas en conceptos matemáticos lo que asegura el diseño correcto de sistemas y protocolos [5, 11,12].

Los tipos de sistemas en lo que se emplean las FDTs son [5, 11,12]:

- *Concurrentes*: Sistemas distribuidos, sistemas en tiempo real, hardware y procesamiento en paralelo.
- *De calidad crítica*: Finanzas, telecomunicaciones y sistemas operativos.
- *De seguridad crítica*: Sistemas de defensa, medicina, industria nuclear, equipos militares, señalización ferroviaria, aparatos de vuelo y telecomunicaciones.
- *De confidencialidad*: Sistemas de información, prevención de accesos no autorizados.
- *De descripción de normas internacionales*: De amplio uso y deben de ser interpretadas por todo el mundo.

Las principales FDTs son STELLE, LOTOS y SDL [5, 11, 14].

Al resultado de describir y diseñar un sistema con una FDT se le llama especificación. Para un mismo sistema puede haber más de una especificación posible con diferentes niveles de abstracción. Las especificaciones sirven como base para extraer la realización 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 tener especificaciones menos abstractas y poco a poco se van incluyendo las decisiones de implementación.

SDL (*Specification and Description Language*) es una normativa para especificar sistemas de tiempo real desarrollada por la CCITT, actualmente ITU-T (*International Telecommunications Union*), y está detallada en la recomendación Z.100 [6, 7].

Inicialmente, el estándar fue desarrollado para realizar especificaciones en el área de las telecomunicaciones, incluyendo aspectos de servicios y protocolos; actualmente se utiliza en la industria para modelar sistemas de tiempo real tales como sistemas empotrados, aeronáutica, automotriz, sistemas distribuidos, entre otros. Permitiendo así modelar el funcionamiento de sistemas sin considerar cómo se llevará a cabo la realización física.

1.1. Justificación

En el Instituto de Electrónica y Mecatrónica (IEM) de la Universidad Tecnológica de la Mixteca (UTM) se cuenta con la línea de investigación de modelado y especificación formal de protocolos de comunicaciones industriales, cuyos principales resultados son [1, 5].

Considerando la importancia del protocolo de comunicaciones DNP3 en el área industrial eléctrica se considera necesario su estudio, con la finalidad de conocer a fondo su funcionamiento

y obtener una especificación formal que ayude a evitar ambigüedades tanto en su uso como en su implementación física.

1.2. Objetivos

1.2.1. Objetivo general

Con base en lo anterior, el objetivo general del presente trabajo de tesis es realizar la especificación formal del protocolo DNP3 utilizando un lenguaje de descripción formal SDL.

1.2.2. Objetivos específicos

Para cumplir con el objetivo general, se plantean los siguientes objetivos específicos:

- Estudiar el protocolo de comunicaciones DNP3.
- Aprender el funcionamiento del SDL.
- Obtener una especificación formal del protocolo DNP3.

1.3. Diseño de la investigación

Para realizar una especificación formal de un protocolo de comunicaciones con una FDT es necesario planificar el desarrollo de la especificación en una serie de etapas. La metodología de desarrollo a usar será la que se muestra en la Figura 1.2 [5].

Como herramienta de desarrollo software se eligió la herramienta Cinderella SDL, herramienta comercial que incorpora las modificaciones más recientes al estándar (SDSL-92 y SDL-95) y que permite crear sistemas SDL de forma amigable, proporcionando las siguientes características SDL [11, URL1]:

- 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.

1.4. Estructura del documento de tesis

El presente documento de tesis está estructurado en cuatro capítulos:

El capítulo 1 presenta una introducción al trabajo de tesis.

El capítulo 2 describe la arquitectura de protocolos DNP3.

El capítulo 3 presenta los resultados obtenidos durante el desarrollo de la especificación formal del protocolo DNP3 utilizando la herramienta Cinderella SDL.

El capítulo 4 plantea las conclusiones y trabajos futuros.

Por último, se presentan las referencias bibliográficas y, en formato digital, el Anexo A incluye la especificación completa del protocolo DNP3.

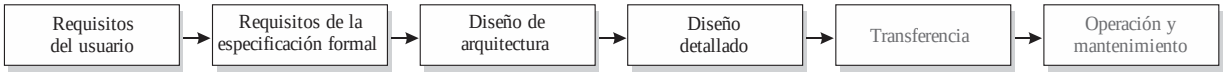


Figura 1.2. Metodología de desarrollo para especificar el protocolo de comunicaciones DNP3.

2. Arquitectura de protocolos DNP3

El protocolo de comunicaciones DNP3 define dos capas (Capa de Aplicación y Capa de Enlace de Datos) y una subcapa (Función de Transporte), como muestra la Figura 2.1 [13].

En la Figura 2.2 se muestra la arquitectura de protocolos DNP3 en una interconexión de una Estación Maestra (EM) con una Estación Remota (ER). A continuación se describe la secuencia de eventos para enviar una orden de la EM a la ER:

- En la EM se recibe la señalización de un determinado Punto (apartado 2.1.1) a través de la Capa de Usuario y se inicia un proceso de solicitud en la Capa de Aplicación.
- La Capa de Aplicación fragmenta la información recibida y pasa los fragmentos obtenidos a la Función de Transporte.
- La Función de Transporte toma los fragmentos y los segmenta para pasarlos a la Capa de Enlace de Datos.
- La Capa de Enlace de Datos obtiene los segmentos, genera la trama DNP3 y la envía a través del medio físico existente.
- La Capa de Enlace de Datos de la ER recibe la trama del medio físico, elimina la cabecera de la trama obtenida y pasa la información a la Función de Transporte.
- La Función de Transporte une los segmentos y pasa a la Capa de Aplicación.
- La Capa de Aplicación desfragmenta la información recibida e indica la solicitud a la Capa de Usuario de la ER correspondiente.
- Para dar respuesta a la solicitud de la EM, la ER realiza el proceso inverso y espera por la confirmación del envío de los datos.

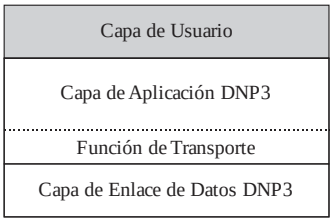


Figura 2.1. Capas del protocolo DNP3.

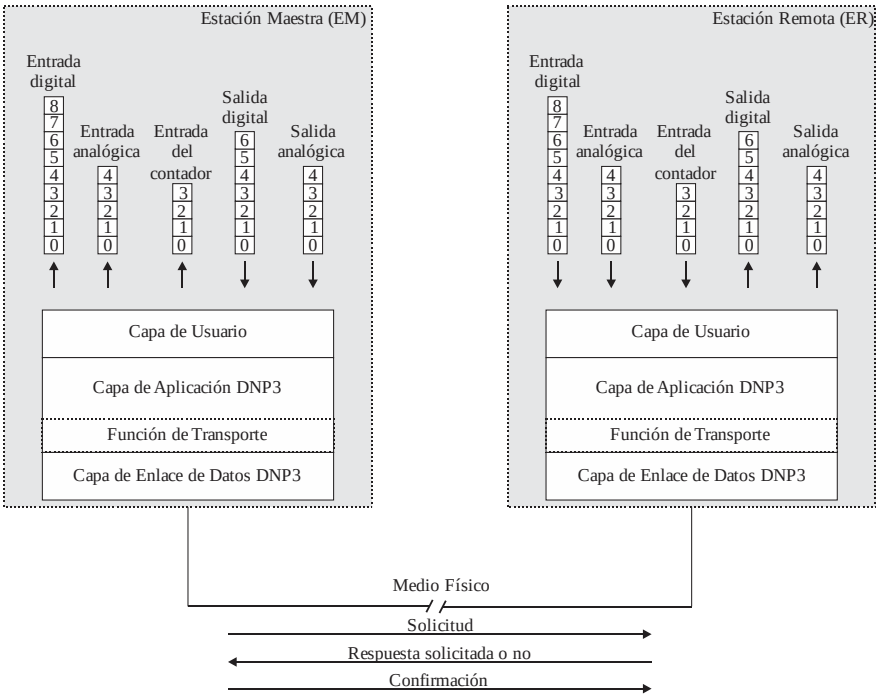


Figura 2.2. Esquema de la composición básica del protocolo DNP3.

En la Figura 2.3 se muestra cómo se construye la trama DNP a partir de un mensaje de la Capa de Aplicación. Dependiendo de su longitud, la Capa de Aplicación divide el mensaje en diferentes fragmentos (*fragment*) y a cada fragmento le añade una cabecera de la Capa de Aplicación (AH, *Application Header*). La Función de Transporte recibe los fragmentos y los divide en segmentos (*segment*), a los cuales les añade una cabecera de la Función de Transporte (TH, *Transport Header*)[5]. Por último, la Capa de Enlace de Datos toma cada segmento, le añade la cabecera de la Capa de Enlace de Datos (LH, *Link Header*) y calcula el CRC para conformar la trama DNP3

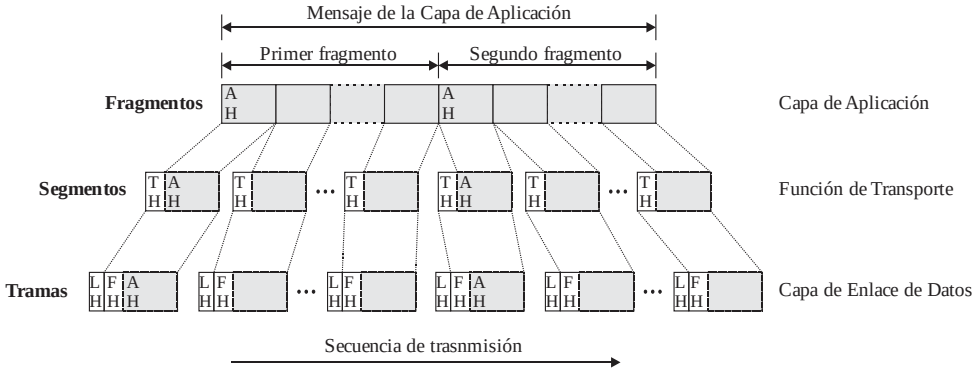


Figura 2.3. Secuencia de separación del mensaje para formar una trama DNP3.

[5] Cabe señalar que la Función de Transporte añade la TH a todos los segmentos y sólo el primer segmento de cada fragmento contará con la TH y la AH.

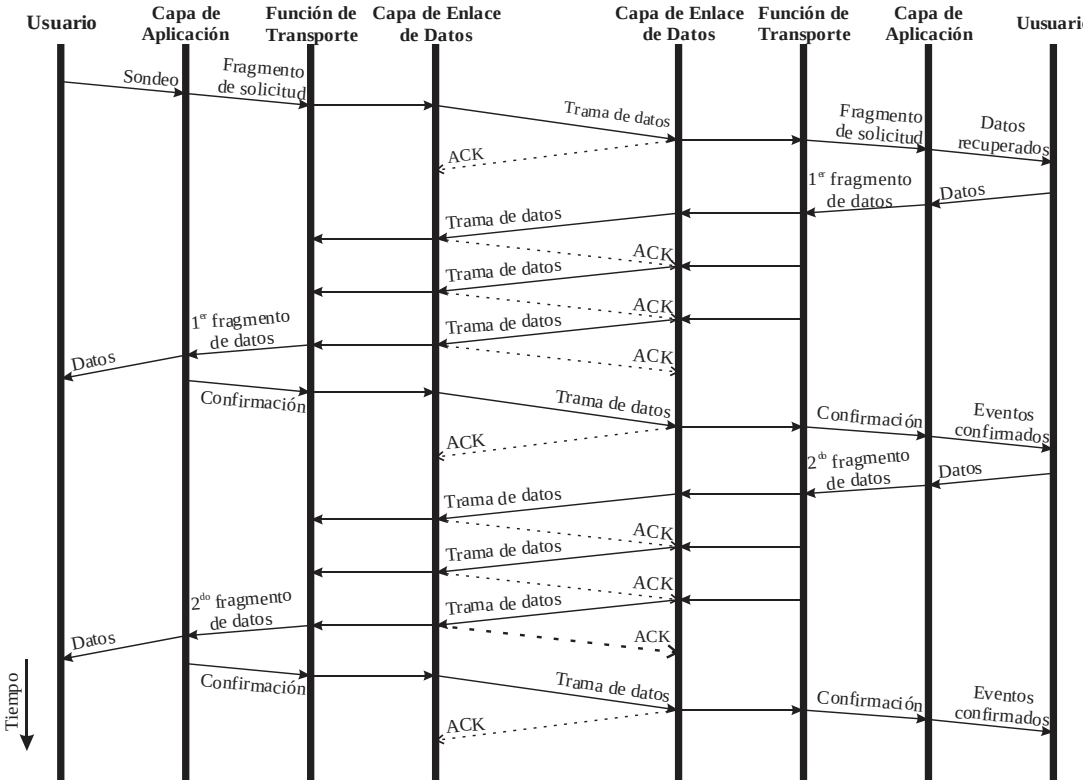


Figura 2.4. Ejemplo de la secuencia de envío de mensajes DNP3.

La Figura 2.4 muestra las secuencias de envío y recepción de una solicitud así como de su respuesta. La Figura 2.5 ilustra la forma en que se envía una respuesta no solicitada y su confirmación.

2.1. Capa de Aplicación DNP3

El modelo de referencia OSI define a la Capa de Aplicación como la interfaz entre el software del usuario (Capa de Usuario) y las capas inferiores del protocolo (Figura 2.6), así mismo proporciona funciones estandarizadas, formatos de datos y procedimientos para la transmisión eficiente de datos, atributos y órdenes de control [13].

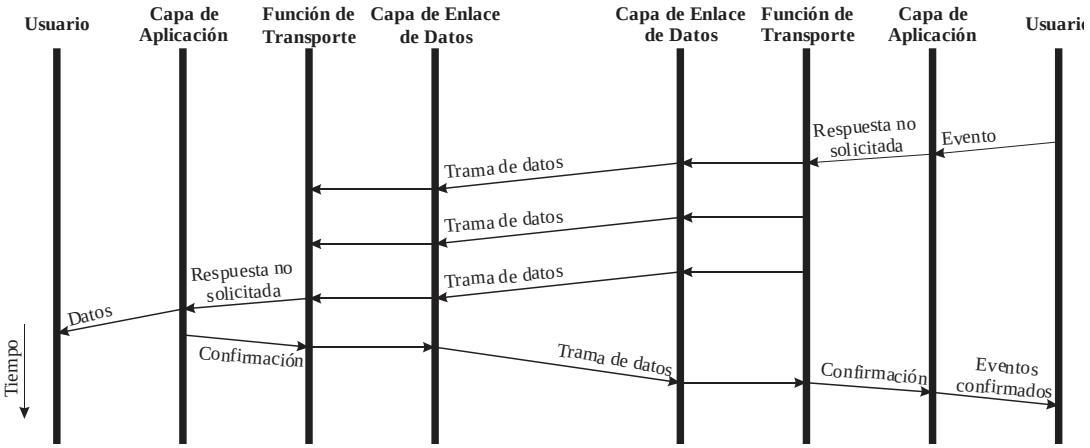


Figura 2.5. Ejemplo de la secuencia de envío de mensajes no solicitados DNP3.

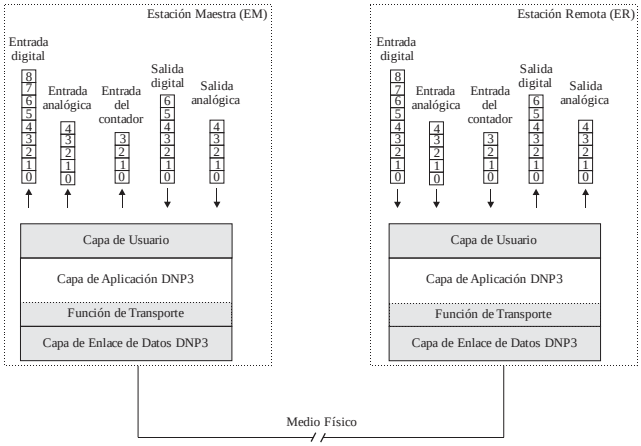


Figura 2.6. Capa de Aplicación dentro de las capas del protocolo DNP3.

2.1.1. Conceptos generales

Respecto a la Capa de Usuario, la literatura de DNP3 utiliza el término Punto [2] (*point*) o *tag* para asociar una entrada/salida analógica o binaria, o bien a un contador con el valor específico de su medición (Figura 2.7).

Los puntos se clasifican de acuerdo a sus características de funcionalidad, relación con el hardware o espacio lógico asociado. En DNP3 cada tipo de Punto se visualiza como un arreglo de Puntos independientes e indexados, en donde cada Punto es único e identificable. Además, se deben considerar las siguientes características:

- Cada Punto es identificable dentro del arreglo.
- Un Punto es usualmente un valor estático.
- Los puntos pueden generar eventos.

Para identificar un Punto se utilizan un índice (*index*) y un grupo (*group*), y para definir el tipo de datos se utiliza una variación (*variation*); una ER puede transportar cinco tipos de puntos como muestra la Figura 2.7, y en algunos casos en particular soporta la transmisión de archivos y otros tipos de datos [2].

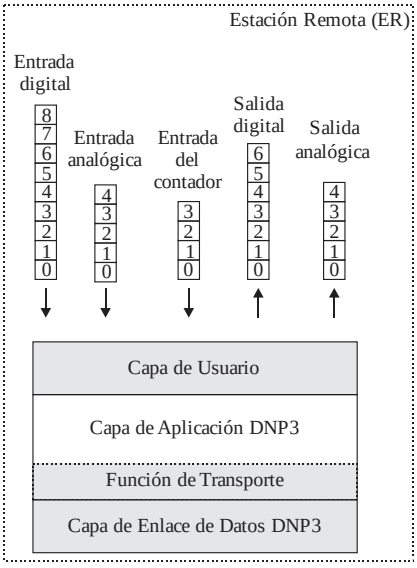


Figura 2.7. Tipos de arreglo de puntos.

DNP3 utiliza índices para identificar a los puntos que son del mismo tipo, los números de los índices corresponden a los números de los elementos del arreglo.

Los grupos clasifican el tipo o tipos de datos que contiene el mensaje; cada grupo indica las mediciones del mismo tipo y el método de generación del dato. La Tabla 2.1 lista la clasificación de grupos DNP3.

Tabla 2.1. Clasificación de grupos DNP3.

Grupo	Descripción
30	Valor actual de la medición
31	Valor fijo de la medición
32	Cambio del valor actual del evento
33	Cambio en el valor fijo del evento

Ocasionalmente, se utiliza el número de grupo para especificar el tipo de datos que se reporta (valor del temporizador, archivos de control, características de terminales virtuales, etcétera). DNP3 soporta diferentes tipos de datos, a esto se le conoce como variación, y cada grupo puede elegir entre las variaciones de la Tabla 2.2, en donde las banderas indican las siguientes condiciones: Fuente conectada, Fuente restablecida y Fuente fuera del rango de medición.

Tabla 2.2. Variaciones DNP3.

Variación	Descripción
1	Entero de 32 bits con bandera
2	Entero de 16 bits con bandera
3	Entero de 32 bits
4	Entero de 16 bits
5	Flotante de 32 bits con bandera
6	Flotante de 64 bits con bandera

Un objeto DNP3 (*DNP3 object*) se define como la representación codificada de la medición de un Punto o de otra estructura, de acuerdo al formato del grupo y a su variación, para ser transportada en el mensaje. Un mensaje puede contener múltiples objetos y cada objeto representa el valor de un Punto en un instante dado.

En DNP3 se emplea el término instancia para relacionar objetos del mismo tipo de evento y distinguirlos de otros; por ejemplo, en un mensaje con 6 bytes de longitud, el cual transporta el valor actual de 6 mediciones de entradas analógicas que son expresadas en enteros de 32 bit, con el número de grupo 30 y una variación de 3; entonces se dice que existen 6 instancias a objetos de entrada analógica de 32 bits.

En DNP3 un dato estático (*static*) hace referencia al valor actual de un Punto, que a su vez representa su medición más reciente (calculada u obtenida).

Por otro lado, se asocia un evento (*event*) a un cambio significativo que puede ser: la medición de un Punto que cruza un umbral definido, una entrada analógica que cambia su valor aun cuando no existe una fuente que provoque dicho cambio, un fenómeno transitorio o la disponibilidad de datos de reciente medición.

A continuación se listan algunas estructuras de información que otorga un evento:

- Tipo de evento (entradas binarias, entradas analógicas, etcétera).
- Valores (*on*, *off*, etcétera).
- Índice del Punto.
- Clase asignada.
- Representación de objetos que se encuentran en el búfer (*buffer*) de transmisión.

Se debe establecer la diferencia entre un evento DNP3 y un reporte de evento de algún objeto DNP3, debido a que algunos objetos DNP3 se conocen como “objetos de eventos”, sin embargo estos son únicamente una representación del evento y no la información real que se encuentra almacenada en la ER.

La información del evento sólo se puede eliminar una vez que la ER ha enviado la descripción del evento a la EM y ésta envíe la confirmación de que ha recibido la descripción del evento.

Respecto a un evento, DNP3 establece como necesario que:

- La información de un evento debe ser un asunto local.
- No está permitido perder o descartar la información almacenada en la estructura del evento hasta que la EM transmita y reconozca una representación del evento.
- Se debe evitar duplicar un evento.

DNP3 utiliza el concepto de clase (*class*) para organizar los valores actuales en las siguientes categorías:

- *class 0*: Hace referencia a los datos estáticos. Cuando la EM solicita la *class 0* a la ER, se refiere al valor medido más reciente del Punto correspondiente.
- *class 1, 2* y *3*: Se utilizan de manera estratégica para asignar prioridades.

2.1.2. Estructura del mensaje DNP3

La EM genera y envía solicitudes a la ER para devolver datos, ejecutar una orden (*command*) o realizar una actividad especial. Tras la recepción de la solicitud, la ER realiza la acción solicitada y genera un mensaje de respuesta, y transmite los datos, resultados o la información especial a la EM.

Cabe señalar que ocasionalmente la ER puede generar un mensaje de respuesta no solicitada para la EM.

2.1.2.1. Fragmento

Un fragmento (*fragment*) es un bloque de bytes que contiene una solicitud de información o respuesta entre la EM y la ER. Cada fragmento contiene un código de función que especifica al receptor cómo debe procesar el fragmento.

DNP3 limita la cantidad de memoria que se asigna para el envío y recepción de datos dependiendo del número de dispositivos activos en la red. Esto se logra mediante la especificación del tamaño máximo de cada fragmento y permite que los mensajes de respuesta se dividan en uno o varios fragmentos.

La Figura 2.8 muestra la estructura de los fragmentos de solicitud y de respuesta. Cada fragmento inicia con la cabecera de la Capa de Aplicación, la cual contiene la información del mensaje de control.

A menudo la cabecera de la Capa de Aplicación no es suficiente para transmitir la información del mensaje de control, por lo que se agregan una o más cabeceras de objetos o se incluyen objetos DNP3 después de la cabecera de la Capa de Aplicación. La cabecera de objeto especifica el tipo, formato e identificación de los objetos DNP3 que le siguen.

La Figura 2.9 muestra el formato de la cabecera de solicitud de la Capa de Aplicación y la Figura 2.10 el formato de la cabecera de respuesta de la Capa de Aplicación, la diferencia es que esta última contiene un campo adicional llamado indicaciones internas.

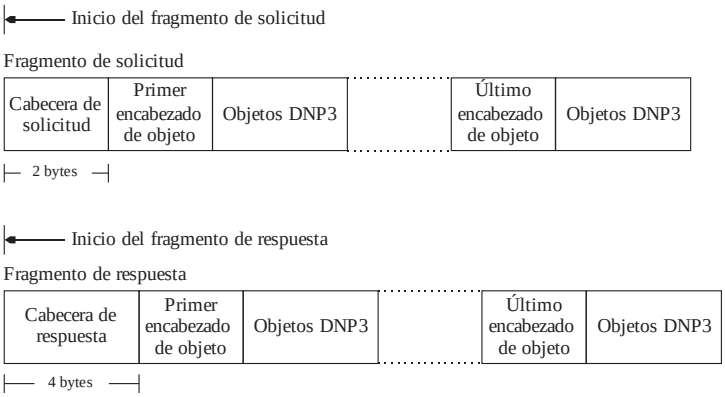


Figura 2.8. Estructura básica de los fragmentos.

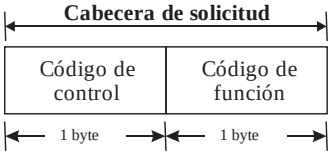


Figura 2.9. Cabecera de solicitud.

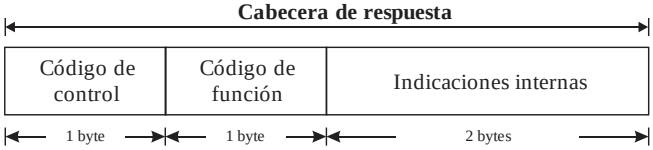


Figura 2.10. Cabecera de respuesta.

2.1.2.2. Código de control de la Capa de Aplicación

El código de control de la Capa de Aplicación proporciona la información necesaria para construir y reensamblar fragmentos de varios mensajes, e indicar sí el receptor debe devolver un mensaje desde la Capa de Aplicación. Además ofrece información para evitar duplicar mensajes.

El código de control se divide en cinco campos (Figura 2.11), los cuales se describen a continuación:

- *FIR*: Campo de un sólo bit; cuando *FIR*=1 indica que es el primer fragmento de un mensaje, si *FIR*=0 indica que no es el primer fragmento del mensaje.
- *FIN*: Campo de un sólo bit; cuando *FIN*=1 indica que es el fragmento final del mensaje, de lo contrario el mensaje aún no termina.
- *CON*: Campo de un sólo bit que indica sí el receptor debe devolver un mensaje de confirmación desde la Capa de Aplicación.
  Un mensaje de respuesta de la Capa de Aplicación es un mensaje breve que se utiliza para informar que un fragmento llegó completo.
  La ER debe habilitar el bit *CON* en los mensajes que contienen datos de un evento, esperando que la EM envíe la confirmación para que la ER pueda descartar la información almacenada en su búfer de transmisión, y así enviar el siguiente fragmento hacia la EM; también la ER debe habilitar este bit cuando ocurre un evento y es necesario enviar un mensaje de respuesta no solicitada.
  Si la EM no habilita el bit CON, significa que la EM no solicita confirmación de la Capa de Aplicación de la ER.

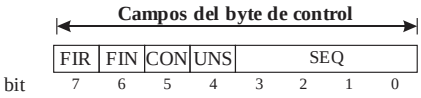


Figura 2.11. Campos del byte de control.

- *UNS*: Campo de un sólo bit que cuando se habilita indica que el mensaje contiene una respuesta no solicitada o la respuesta a un mensaje no solicitado, cuando el valor de *UNS*=0 indica que la secuencia de números está asociada a una solicitud o un mensaje de respuesta solicitada. La ER habilita este bit en fragmentos que contienen una respuesta no solicitada y lo inhabilita cuando envía una respuesta solicitada; mientras que la EM lo habilita en fragmentos de confirmación de la Capa de Aplicación para confirmar la recepción de una respuesta no solicitada y lo inhabilita en fragmentos de confirmación de la Capa de Aplicación al confirmar la recepción de una respuesta solicitada.
- *SEQ*: Este campo consta de 4 bits y se utiliza para verificar que los fragmentos sean recibidos en el orden correcto y para detectar fragmentos duplicados. El campo *SEQ* tiene un rango de valores de 0 a 15 que se incrementa mediante un contador que realiza operaciones de modulo 16. Lo anterior se realiza para evitar repetir fragmentos de solicitud de la EM, fragmentos subsecuentes después del primer fragmento en mensajes que contienen fragmentos múltiples y fragmentos de respuesta no solicitada.
  Todos los dispositivos en la red deben mantener la secuencia de números independientes para cada dispositivo con el que se comunica, en donde una secuencia se utiliza para las peticiones solicitadas, las respuestas y confirmaciones, y otra secuencia se utiliza para las respuestas no solicitadas y confirmaciones; cabe señalar que no existe relación entre ambos números de secuencia.

2.1.2.3. Código de función

El código de función identifica el propósito del mensaje; en la EM tiene un rango de 1 a 120 y la respuesta de la ER usa un rango de valores entre 129 y 255 como se muestra en la Tabla 2.3.

2.1.2.4. Indicaciones internas

El campo de indicaciones internas sólo aparece en la cabecera de respuesta de la Capa de Aplicación inmediatamente después del byte de código de función; este campo se divide en dos bytes como se muestra en la Figura 2.12. Los bits de estos dos bytes indican los estados de error de las ER.

La Tabla 2.4 muestra los códigos de indicaciones internas siguiendo la notación IINx.b[6].

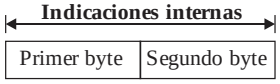


Figura 2.12. Bytes de indicaciones internas.

[6] En la notación IINx.b, la x es el primero o segundo byte de la Figura 2.12 y la letra b hace referencia al número del bit, por ejemplo, si se envía una indicación interna con el siguiente código IIN2.0 se refiere al primer bit del segundo byte.

Tabla 2.3. Códigos de función.

Tipo de mensaje	Código	Nombre	Descripción
Confirmación	0 (0x00)	Confirmación (*CONFIRM*)	La EM envía para confirmar la recepción de un fragmento para la Capa de Aplicación.
Solicitud	1 (0x01)	Lectura (*READ*)	La ER devolverá los datos específicos de los objetos solicitados.
Solicitud	2 (0x02)	Escritura (*WRITE*)	La ER deberá almacenar los datos específicos de los objetos enviados en la solicitud.
Solicitud	3 (0x03)	Selección (*SELECT*)	Una vez que la ER selecciona los puntos de salida especificados por la solicitud, estos están preparados para la siguiente operación. La ER no activa las salidas hasta que se recibe la solicitud de operación.
Solicitud	4 (0x04)	Operar (*OPERATE*)	En la ER se activan las salidas seleccionadas mediante una orden de selección.
Solicitud	5 (0x05)	Operación inmediata (*DIRECT_OPERATE*)	La ER ejecuta inmediatamente los puntos de salida especificados por la solicitud, no es necesaria una orden de selección.
Solicitud	6 (0x05)	Operación inmediata sin retorno (*DIRECT_OPERATE_NR*)	Es igual al código de función 5, pero en este caso la ER no envía una respuesta.
Solicitud	7 (0x07)	Congelación inmediata (*IMMED_FREEZE*)	La ER copia los valores de los puntos especificados por los objetos en la solicitud que se separaron por una retención en el búfer.
Solicitud	8 (0x08)	Congelación inmediata sin retorno (I*MMED_FREEZE_NR*)	Es igual al código de función 7, pero la ER no envía una respuesta.
Solicitud	9 (0x09)	Eliminación de la congelación (*FREEZE_CLEEN*)	La ER copia los datos de los puntos especificados por los objetos en la solicitud, los cuales fueron separados por congelación en un búfer. Después de esta operación, los valores se reinicializan a cero.
Solicitud	10 (0x0A)	Eliminación de la congelación sin retorno (*FREEZE_CLEEN_NR*)	Es igual al código de función 9, pero en este caso la ER no envía una respuesta.
Solicitud	11 (0x0B)	Congelamiento por tiempo (*FREZE_AT_TIME*)	La ER copia los datos de los puntos especificados por los objetos en la solicitud, los cuales fueron separados por congelación en un búfer en algún instante y/o en intervalos de tiempos definidos en un objeto especial de tiempo.
Solicitud	12 (0x0C)	Congelamiento por tiempo sin retorno (*FREZE_AT_TIME_NR*)	Es igual al código de función 11, pero en este caso la ER no envía una respuesta.
Solicitud	13 (0x0D)	Reinicio en frío (*COLD_RESTART*)	La ER debe reiniciar completamente el dispositivo.
Solicitud	14 (0x0E)	Reinicio en caliente (*WARM_RESTART*)	La ER sólo reinicia algunas partes del dispositivo.
Solicitud	15 (0x0F)	Inicialización de datos (*INITIALIZE_DATA*)	Obsoleto, no es utilizado en nuevos dispositivos.
Solicitud	16 (0x10)	Inicialización de aplicaciones (*INICIALIZE_APL*)	La ER cambia las aplicaciones a los estados enlistados en los objetos de solicitud.
Solicitud	17 (0x11)	Inicio de aplicaciones (*START_APPL*)	La ER ejecuta las aplicaciones que se especifican en los objetos de la solicitud.
Solicitud	18 (0x12)	Detención de los estados de	La ER detiene las aplicaciones que se especifican en los objetos de la solicitud.

		aplicaciones (*STOP_APPL*)	
Solicitud	19 (0x13)	Almacenar configuraciones (*SAVE_CONFIG*)	La ER almacena en la memoria no volátil el archivo de configuración ubicado en una memoria volátil. Este código de función no se utiliza en nuevos dispositivos.
Solicitud	20 (0x14)	Habilitar respuestas no solicitadas (*ENABLE_UNSOLICITED*)	Permite a la ER habilitar respuestas no solicitadas de puntos especificados en los objetos de la petición.
Solicitud	21 (0x15)	Deshabilitar respuestas no solicitadas (*DISABLE_UNSOLICITED*)	Evita a la ER iniciar respuestas no solicitadas de puntos especificados en los objetos de la petición.
Solicitud	22 (0x16)	Asignación de clases (*ASSIGN_CLASS*)	La ER asigna los eventos generados en los puntos solicitados por los objetos en una de las clases.
Solicitud	23 (0x17)	Medición de retardos (*DELAY_MEASURE*)	La ER informa el tiempo que le toma procesar e transmitir su respuesta, lo que permite a la EM calcular el retardo programado en el canal de comunicación y sincronizar tiempos[7].
Solicitud	24 (0x18)	Registrar el tiempo del último dato (*RECORD_CURRENT_TIME*)	La ER registra el instante en el que se recibe el último byte para la sincronización de tiempo en redes LAN.
Solicitud	25 (0x19)	Abrir un archivo (*OPEN_FILE*)	La ER abre un archivo.
Solicitud	26 (0x1A)	Cerrar un archivo (*CLOSE_FILE*)	La ER cierra un archivo.
Solicitud	27 (0x1B)	Eliminar un archivo (*DELETE_FILE*)	La ER elimina un archivo.
Solicitud	28 (0x1C)	Obtener información de un archivo (*GET_FILE_INFO*)	La ER obtiene información de un archivo.
Solicitud	29 (0x1D)	Autentificación de archivos (*AUTHENTICATE_FILE*)	La ER regresa la clave de autentificación del archivo.
Solicitud	30 (0x1E)	Cancelar transferencia de archivos (*ABOR_FILE*)	La ER cancela la transferencia de archivos.
Solicitud	31 (0x1F)	Activación de configuraciones (*ACTIVATE_CONFIG*)	La ER utiliza la configuración especificada por los objetos transmitidos en la solicitud.
-	32 (0x20) - 128 (0x80)	-	Reservados.
Respuesta	129 (0x81)	Respuesta (*RESPONSE*)	La EM interpreta este fragmento como una respuesta de la Capa de Aplicación a una solicitud de nivel de aplicación enviada anteriormente.
Respuesta	130 (0x82)	Respuesta no solicitada (*UNSOLICITED_RESPONSE*)	La EM interpreta este mensaje como una respuesta no solicitada, la cual no fue motivada por una solicitud explícita.
Respuesta	131 (0x83) - 255 (0xFF)	-	Reservados

[7] No se aplica en redes de área local (LAN, *Local Area Network*).

Tabla 2.4. Códigos de indicaciones internas.

Bit	Nombre	Descripción
IIN1.0	Todas las estaciones (*ALL_STATIONS*)	El mensaje fue recibido por todas las estaciones.
IIN1.1	Eventos de clase 1 (*CLASS_1_EVENT*)	La ER no reportó un evento de clase 1.
IIN1.2	Eventos de clase 2 (*CLASS_2_EVENT*)	La ER no reportó un evento de clase 2.
IIN1.3	Eventos de clase 3 (*CLASS_3_EVENT*)	La ER no reportó un evento de clase 3.
IIN1.4	Solicitud de tiempo (*NEED_TIME*)	Se requiere un tiempo de sincronización.
IIN1.5	Control local (*LOCAL_CONTROL*)	Uno o más puntos de la ER se encuentran en modo local.
IIN1.6	Problemas con el equipo (*DEVICE_TROUBLE*)	Indica una condición anormal en una parte de la ER.
IIN1.7	Reinicio de equipo (*DEVICE_RESTART*)	La ER se ha reiniciado.
IIN2.0	Código de función no soportado (*NO_FUNC_CODE_SUPPORT*)	La ER no soporta este código de función.
IIN2.1	Objeto desconocido (*OBJECT_UNKNOWN*)	La estación remota no soporta la operación solicitada en los objetos de la petición.
IIN2.2	Error de parámetros (*PARAMETER_ERROR*)	Se detectó un error de parámetros.
IIN2.3	Evento de desbordamiento de la memoria de datos (*EVENT_BUFFER_OVERFLOW*)	Existe un evento de desbordamiento de la memoria de datos en la ER y se ha perdido al menos un caso no confirmado.
IIN2.4	Ejecución en curso (*ALREADY_EXECUTING*)	Se está ejecutando la operación solicitada.
IIN2.5	Configuración incorrecta (*CONFG_CORRUPT*)	La ER detectó una configuración incorrecta.
IIN2.6	Reservado 2 (*RESERVED_2*)	Reservado para uso futuro
IIN2.7	Reservado 1 (*RESERVED_1*)	Reservado para uso futuro

2.1.2.5. Cabeceras de objetos

Este inciso presenta un ejemplo para ilustrar la importancia de las cabeceras de función.

Una EM quiere leer 20 datos de una RTU; al formular el mensaje de solicitud, la EM utiliza el código de función *READ*, la cabecera de objeto de la solicitud debe especificar:

- El Punto de entrada analógica y el tipo de dato correspondiente.
- El tipo de formato de datos que se requiere (entero, flotante, etcétera).
- Los 20 valores indexados del Punto.

Los objetos DNP3 no están incluidos en la solicitud, sólo una cabecera de objeto. Dado que la EM no está enviando valores, sólo envía información suficiente para que la ER conozca el formato y los valores que requiere. El formato de respuesta utiliza el código de función *RESPONSE*, que contiene la cabecera de objeto igual o similar al objeto recibido anteriormente (de la EM), seguido por los objetos DNP3. Cada objeto contiene un único valor indexado del Punto solicitado.

Las cabeceras de objeto consisten en un campo de tipo de objeto, un campo de clasificación y un campo de rango, este último depende del código en el campo de clasificación (Figura 2.13). El campo tipo de objeto consta de un campo de grupo seguido por campo de variación.

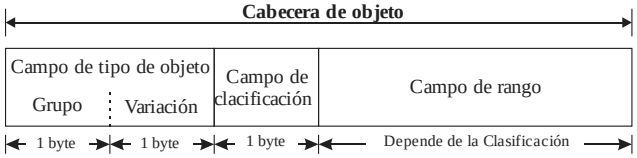


Figura 2.13. Campos de la cabecera de objetos.

2.1.2.6. Grupo

El grupo especifica el tipo de dato o valores solicitados por la EM o en una respuesta de la ER. Algunos ejemplos son: Entradas analógicas, valores actuales, entradas binarias, valores de eventos, Contador congelado, valor actual o valores de los contadores.

2.1.2.7. Variación

La variación especifica el formato de datos de los objetos DNP3. Por ejemplo, las variaciones existentes para informar sobre el valor actual de las entradas analógicas como enteros de 16 bits, enteros de 32 bits, las cantidades de decimales de un número flotante (corto o largo), los enteros pueden incluir un byte adicional para banderas. Cada formato tiene distintas alternativas de variación única.

2.1.2.8. Campos clasificación y rango

Estos dos campos se analizan juntos dado que la estructura y el contenido del campo de rango depende del valor del byte de clasificación como se muestra en Figura 2.13.

El byte de clasificación está dividido en tres campos como muestra la Figura 2.14:

- *Res*: Este campo está reservado para un uso futuro, por el momento utiliza el valor 0.
- *Código de objeto fijo*: Ocupa un tamaño de 3 bits para especificar la existencia de algún código fijo (*Prefix*) antes de cada uno de los objetos DNP3 que siguen a la cabecera de objeto (Figura 2.15). Los códigos *Prefix* contienen un número de índice o el tamaño del objeto, y se utilizan de la siguiente manera:
    - Para asociar el objeto de datos con un índice en la ER.
    - Para dar a conocer el tamaño del objeto (Tabla 2.5).
    - En algunos casos, como la solicitud de un índice de puntos de entrada binario no secuencial, la EM debe transmitir una lista de los puntos indexados a la ER, en donde la EM no transmite objetos de datos reales. En este caso, el uso del objeto *Prefix* 1, 2 ó 3 envía objetos nulos (código 0) para un índice de puntos, como en la solicitud que muestra la Figura 2.16.
    - En otras situaciones, los datos de los índices de Punto son contiguos. El campo de rango especifica un índice inicial y uno final, entonces no es necesario un objeto *prefix* porque el índice corresponde con la posición de los datos de la respuesta, se almacena el ancho de banda para no incluir valores redundantes de índices y se utiliza el código 0 para especificar que no aparecerán códigos *Prefix* antes de cada objeto.
- *Código de especificación de rango*: Indica si se utiliza el campo de rango, de ser así, indica su contenido y su tamaño; la Tabla 2.6 lista los valores del código de especificación de rango.

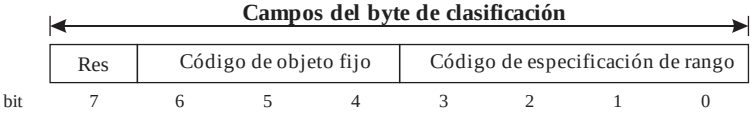


Figura 2.14. Byte del campo clasificación.

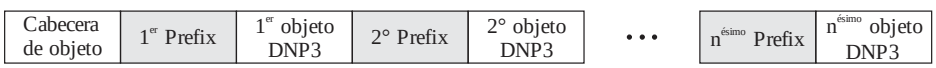


Figura 2.15. Códigos de objeto fijo.

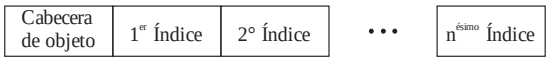


Figura 2.16. Lista de índices.

Tabla 2.5. Valores de los códigos de objeto fijo.

Código (Hex)	Descripción	Tamaño de los objetos *Prefix* (Bytes)
0	Los objetos *Pre*fix son empaquetados sin índice.	-
1	Los objetos *Prefix* se han fijado como un índice.	1
2	Los objetos *Prefix* se han fijado como un índice.	2
3	Los objetos *Prefix* se han fijado como un índice.	4
4	Los objetos *Prefix* se han fijado como un tamaño de objeto.	1
5	Los objetos *Prefix* se han fijado como un tamaño de objeto.	2
6	Los objetos *Prefix* se han fijado como un tamaño de objeto.	4
7	Está reservado para un uso futuro.	-

Tabla 2.6. Valores de los códigos de especificación de rango.

Código (Hex)	Descripción	No. de Bytes	Aplicación
0	El campo rango contiene un byte de inicio y un byte de parada de índices.	2	Estos códigos se utilizan cuando los objetos DNP3 se empaquetan de manera ordenada en un índice.
1	El campo rango contiene dos bytes de inicio y dos bytes de parada de índices.	4	
2	El campo rango contiene cuatro bytes de inicio y cuatro bytes de parada de índices.	8	
3	El campo rango contiene un byte de inicio y un byte de parada de direcciones virtuales.	2	Estos códigos se utilizan para especificar lugares contiguos de direcciones específicas de espacio de memoria virtual (depende del proveedor). Estos códigos no se utilizan, pero cuando un proveedor los aplica, a menudo se utiliza con objetos enteros de 8 bits sin signo.
4	El campo rango contiene dos bytes de inicio y dos bytes de parada de direcciones virtuales.	4	
5	El campo rango contiene cuatro bytes de inicio y cuatro bytes de parada de direcciones virtuales.	8	
6	Este campo rango especifica a todos los valores, pero no se emplea actualmente.	0	Indica que la EM requiere los valores de todos los puntos especificados por el grupo de objetos. No hay campo rango.
7	El campo rango contiene un byte que indica la cantidad de objetos.	1	Indica que el campo rango es la cantidad de datos de objeto o índices de objeto. La ER puede responder con un número menor de objetos si la solicitud de la EM utiliza los códigos de clasificación 7, 8 ó 9; ésta es una razón para elegir un número menor de datos. Una ER debe enviar al menos un objeto de datos sí tiene el tipo de dato solicitado.
8	El campo rango contiene dos bytes que indican la cantidad de objetos.	2	
9	El campo rango contiene cuatro bytes que indican la cantidad de objetos.	4	
A	Reservado para un uso futuro.	-	Reservado para un uso futuro.
B	Calificador de formato variable. El campo rango contiene un byte que indica la cantidad de objetos.	1	Indica que los datos tienen un formato de longitud variable que no puede ser predefinido para todos los grupos y variaciones de los objetos.
C	Reservado para un uso futuro.	-	Reservado para un uso futuro.
D	Reservado para un uso futuro.	-	Reservado para un uso futuro.
E	Reservado para un uso futuro.	-	Reservado para un uso futuro.
F	Reservado para un uso futuro.	-	Reservado para un uso futuro.

La Tabla 2.7 lista los códigos calificadores más utilizados y la Tabla 2.8 lista las reglas de los fragmentos de la Capa de Aplicación.

Tabla 2.7. Códigos calificadores más utilizados.

Código (Hex)	Uso en la solicitud de la EM	Uso en la respuesta de la ER
00, 01	Se solicita un sólo Punto o un rango de puntos estáticos.	En la respuesta se envían objetos estáticos.
06	Se solicitan todos los puntos.	En la respuesta no está permitido este código.
07, 08	Se utiliza en la solicitud de una cantidad limitada de eventos. En una cantidad única no tiene índice (por ejemplo: la fecha y la hora).	Una cantidad única no tiene índice (por ejemplo, la fecha y la hora).
17, 28	Se solicita el código para utilizar los controles u otras funciones, tales como la lectura de varios objetos, donde los índices no son secuenciales.	La respuesta consiste en objetos de eventos (por lo general uno o más Puntos no relacionados)
5B	Se emplea el código para transmitir los objetos cuyo tamaño puede ser desconocido para el receptor (por ejemplo, abrir un archivo).	Se emplea para transmitir los objetos cuyo tamaño puede ser desconocido para el receptor (por ejemplo, archivos de datos).

2.1.3. Respuestas no solicitadas

Cuando ocurre un evento, las ER envía respuestas no solicitadas sin una solicitud específica de la EM (Figura 2.5). El protocolo DNP3 incluye el soporte a las respuestas solicitadas como característica opcional, en donde las EM y ER no están obligadas a aplicar el mensaje de respuesta no solicitada.

2.1.4. Máquinas de estados de la Capa de Aplicación

2.1.4.1. Máquina de estados de la ER

El propósito de la máquina de estados es especificar el comportamiento de la ER en la recepción y transmisión de los fragmentos. Por cada fragmento recibido, la ER examina los bits FIR, FIN y UNS, además del valor del campo SEQ en el byte de control y el código de función de la Capa de Aplicación. También debe almacenar los valores SEQ de los bytes de los fragmento de solicitud que se aceptan y de los fragmentos de respuesta que se transmiten.

La ER sólo acepta fragmentos de solicitud que contengan solicitudes válidas y que cumplan con los criterios establecidos en la tabla de estados (Tabla 2.9), de lo contrario se descartan los fragmentos y se mantiene en el mismo estado.

La ER configura las siguientes variables locales para cada EM con la que se comunica:

- *Aceptación del primer valor solicitado (FirstValidRequestAccepted)*: Esta variable booleana se utiliza para sincronizar el procesamiento de los valores del campo SEQ en las solicitudes válidas. El valor de esta variable es puesta a 0 o *false* en el arranque o inmediatamente después de un reinicio. Se establece en 1 o *true* cuando se recibe la primera solicitud válida.
- *Espera de la confirmación de un número de secuencia (ECSN, Expected Confirm Sequence Number)*: Esta variable se utiliza para confirmar el número de secuencia. Si la ER está esperando la confirmación de una respuesta solicitada, el valor de esta variable contiene el número de secuencia que aparece en el byte de control de la Capa de Aplicación; de lo contrario, tiene un valor NE, lo que significa que no se espera una confirmación. Solamente se procesan las confirmaciones solicitadas que se reciben con un número de secuencia correspondiente ECSN, y todas las demás son descartadas. Esta variable no tiene ningún significado para las respuestas no solicitadas, aunque su valor se puede establecer o examinar a la espera de una confirmación de una respuesta no solicitada.

Tabla 2.8. Reglas de los fragmentos.

Regla	Descripción
1	El tamaño máximo de un fragmento para su transmisión es de 2,048 bytes.
2	El tamaño mínimo de un fragmento para su transmisión es de 2 bytes.
3	Las ER pueden recibir fragmentos con un tamaño mínimo de 249 bytes y las EM pueden recibir fragmentos con un tamaño mínimo de 2,048 bytes.
4	La EM sólo debe enviar las solicitudes que se ajusten dentro de un sólo fragmento.
5	La EM debe aceptar fragmentos de respuesta múltiple.
6	Una ER debe ser capaz de devolver todos los objetos de un evento y los datos estáticos en conjunto, dentro de una sola respuesta. Si es necesario, utilizará fragmentos múltiples para transmitir toda la respuesta.
7	Cada fragmento se debe analizar y procesar en forma individual. Un fragmento completo sólo debe contener objetos DNP3 y no partes de un objeto, es decir, los objetos no se pueden dividir en dos o más fragmentos.
8	El bit FIR se debe habilitar en el fragmento que comienza un mensaje.
9	El bit FIN se debe habilitar en el fragmento que termina un mensaje.
10	Un mensaje puede consistir en un sólo fragmento, en donde se habilitan los bits FIR y FIN.
11	La EM no solicita la confirmación de la Capa de Aplicación, es decir, no debe habilitar el bit CON en los mensajes de petición (En la actualidad esta regla es obsoleta).
12	Las ER que reciben un fragmento con el bit CON habilitado, inmediatamente debe responder con un mensaje de confirmación de la Capa de Aplicación (permite la compatibilidad con versiones anteriores).
13	Cuando las EM reciben un fragmento con el bit CON habilitado, debe responder con un mensaje de confirmación de la Capa de Aplicación antes de enviar cualquier mensaje de solicitud a las ER.
14	Por cada reintento de envío de fragmentos de solicitud, la EM debe incrementar el número del campo SEQ utilizando la ecuación (n+1)%64, donde *n* es el número de secuencia del fragmento de la solicitud anterior.
15	El primer fragmento de respuesta de una solicitud debe tener el mismo número de secuencia que el de la solicitud. Si la respuesta necesita múltiples fragmentos, los fragmentos posteriores deberán incrementar el número del campo SEQ utilizando la ecuación (n+1)%64.
16	Una EM puede reenviar un mensaje de solicitud si se utiliza el mismo número del campo SEQ y los otros bytes deben coincidir con el mensaje de solicitud original. Hay excepciones en la que se prohíbe el reenvío de solicitudes si en éstas se envía: Códigos de función DIRECT_OPERATE o DIRECT_OPERATE_NR, códigos de función DELAY_MEASURE o RECORD_CURRENT_TIME o una función WRITE con un objeto de tiempo absoluto o con el último objeto almacenado.
17	Una ER puede reenviar mensajes de respuesta no solicitados y no reenvía mensajes de respuesta solicitados.
18	Un fragmento de solicitud que contiene el código de función CONFIRM debe utilizar el número de secuencia y el estado del bit UNS del fragmento cuando se inicia la confirmación.
19	Las ER deben ignorar el campo SEQ en los mensajes de solicitud de difusión.
20	Las respuestas no solicitadas tienen un número del campo SEQ diferente, el cual no tiene relación con el número del campo SEQ de las respuestas solicitadas.
21	Una ER que envía respuestas no solicitadas puede elegir cualquier número en el campo SEQ en su primer mensaje después de un reinicio. La EM debe aceptar estos mensajes.
22	Una ER que se reinicia debe omitir el número del campo SEQ en la primera solicitud que recibe de una EM después de la reanudación, de lo contrario ejecutará la solicitud. A partir de entonces, la ER deberá examinar los números del campo SEQ en las solicitudes recibidas.
23	Una ER debe solicitar una confirmación de la Capa de Aplicación cuando se envía un fragmento que contiene objetos de un evento. La recepción del mensaje de confirmación, de la EM a la ER, permite saber que la información llegó a la EM y por tanto la ER puede descartar los datos del búfer de eventos.
24	Se requiere una confirmación de la Capa de Aplicación para cada fragmento de un mensaje de múltiples fragmentos, excepto el último fragmento. La solicitud de confirmación de la Capa de Aplicación es opcional para el último fragmento de un mensaje de múltiples fragmentos a menos que exista otra razón por la cual la confirmación sea obligatoria, ya que el último fragmento también contiene eventos. La recepción de la confirmación significa que la ER puede enviar el siguiente fragmento.
25	La solicitud de confirmación de la Capa de Aplicación es obligatoria en mensajes de respuesta no solicitados. La ER no debe desechar la información hasta que reciba un mensaje de confirmación de la EM.
26	En algunas ocasiones, la EM envía una solicitud de difusión a todas las ER y se les solicita una conformación de la Capa de Aplicación. La misma dirección que utiliza la EM para enviar el mensaje a todas las estaciones determina la necesidad de confirmación.

- El valor del campo SEQ de los fragmentos aceptados recientemente de una solicitud válida.
- Los bytes de los fragmentos aceptados recientemente de una solicitud válida de la Capa de Aplicación.
- Los bits FIR, FIN, CON y el valor del campo SEQ de los fragmentos que recientemente transmitió la respuesta solicitada.
- Los bytes de los fragmentos de la Capa de Aplicación que recientemente transmitió la respuesta solicitada.
- Los bits FIR, FIN, CON y el valor del campo de SEQ de los fragmentos recientemente transmitidos de una respuesta no solicitada.
- Los fragmentos recientemente transmitidos de una respuesta no solicitada de la Capa de Aplicación.

El software de la ER requiere de los siguientes estados para una correcta recepción:

- *Estado ocioso*: El software está a la espera de un fragmento de solicitud o de un evento que puede dar lugar a una respuesta no solicitada. En algunos casos la solicitud puede retrasarse hasta que la entrada a este estado esté disponible como consecuencia de las acciones de otros estados. Cuando existe una solicitud disponible y ésta está retrasada, debe procesar las solicitudes aplazadas de inmediato como si se acabaran de recibir. El software de la ER siempre se inicia en el estado ocioso.
- *Estado de espera de una confirmación solicitada (WaitSolCfm)*: La EM recibe de la ER una petición para enviar una confirmación (eventos o múltiples fragmentos) y la ER está a la espera de la confirmación.
- *Estado de espera de una confirmación no solicitada (WaitUnsolCfm)*: La ER transmite una respuesta no solicitada por parte de la EM.

La Tabla 2.9 ilustra el comportamiento de la ER de acuerdo a las solicitudes o eventos que ocurren en ésta. En la Figura 2.17 se muestra la representación gráfica de la Tabla 2.9.

2.1.4.2. Máquina de estados de la EM para fragmentos de solicitud

El propósito de la máquina de estados de la EM es especificar su comportamiento cuando se reciben fragmentos con el bit UNS=0 en el byte de control de la Capa de Aplicación. El software de la EM necesita examinar los bits FIR y FIN, y el valor del campo SEQ en el byte de control de la Capa de Aplicación. También es necesario almacenar dichos bits junto con todos los bytes del último fragmento aceptado del envío de una solicitud de respuesta. La EM acepta el fragmento sí cumple con los criterios establecidos en la Tabla 2.10, de lo contrario se descarta el fragmento. La EM no necesita almacenar los bytes de control de la Capa de Aplicación o los valores de cualquier byte de los fragmentos descartados. La EM también debe recordar el valor del campo SEQ a partir del último fragmento de solicitud que se envió a una ER.

Una EM debe mantener un conjunto separado de variables para cada ER con la que se comunica, las variables incluyen:

- Los bits FIR y FIN, y el valor del campo SEQ a partir del fragmento del último fragmento aceptado de una respuesta solicitada.
- Los bytes del fragmento recientemente aceptados de una respuesta solicitada.
- El valor del campo SEQ del fragmento de solicitud recientemente transmitido.

Tabla 2.9. Tabla de estados de la ER.

Estado actual	Evento que desencadena una acción y la posible transición				Acción		Estado de transición.	
A	B	C			D		E	
Estado del software	Evento	El fragmento recibido contiene			Acción a realizar		Siguiente estado	
		UNS	SEQ	Código de función				
Ocioso	[RESTART] Ocurrió un reinicio y la ER se está configurando para enviar las respuestas solicitadas.	-	-	-	Enviar una respuesta no solicitados con los bits UNS=NULL, CON=NULL y el campo SEQ es igual a cualquier valor válido además de utilizar la IIN1.7.		WaitUnsolCfm	1
	[DEFERRED_READ_EXISTS] Se retrasó una petición de lectura en otro estado.	-	-	-	El proceso de leer la solicitud y enviar la respuesta.		Si txCON=0, estado Ocioso; si no WaitSolCfm.	2
	[UNSOL_TRIGGER] La ER está configurada como ECNS==NE, y se origina una nueva secuencia de respuesta no solicitada.	-	-	-	Se realiza un incremento M (%16) y se envía una respuesta no solicitada con los bits UNS y CON habilitados.		WaitUnsolCfm	3
	[BROADCAST_FRAG_RCVD] Fragmento recibido con la dirección de difusión.	0	X	*REQUEST*	Se acepta el fragmento y el proceso de la solicitud. No se envía una respuesta.		Ocioso	4
	[FIRST_FRAG_RCVD] El primer fragmento se recibió después de un reinicio.	0	X	*REQUEST*	Se acepta el fragmento y el proceso de solicitud. Se habilita la variable FirstValidRequestAccepted. Se envía la respuesta si es solicitada. Si txCON==1, se asigna a la variable ECSN el número de secuencia del fragmento transmitido.		Si no hay respuesta o txCON=0 entonces se pasa a Ocioso de lo contrario a WaitSolCfm.	5
	[NEW_FRAG_RCVD] Fragmento de solicitud recibido.	0	!=N	*REQUEST*	Se acepta el fragmento y el proceso de la solicitud. Se envía la respuesta si es solicitada. Si txCON==1, se asigna el número de secuencia del fragmento transmitido a la variable ECSN.		Si no hay respuesta o txCON==0 entonces se pasa a ocioso de lo contrario a WaitSolCfm.	6
	[REPEAT_FRAG_RCVD] Solicitud recibida y repetida.	0	N	*REQUEST*	Comparar byte a byte con el fragmento de solicitud anterior ¿los bytes coinciden?		-	7
					Sí	Se acepta el fragmento y se envía la misma respuesta, no se procesa la solicitud.	Ocioso	8
					No	Se aceptan el fragmento y el proceso de la solicitud. Se envía la respuesta si es solicitada. Si txCON==1, se asigna a la variable ECSN el número de secuencia del fragmento transmitido.	Si no hay respuesta o txCON==0 entonces se pasa a Ocioso de lo contrario a WaitSolCfm.	9
	[CONFIRM_RCVD] Confirmación recibida.	X	X	*CONFIRM*	Se desecha el fragmento de confirmación, no se elimina ningún evento.		Ocioso	10
WaitSolCfm	[*MATCHING_SOL_CONFIRM_RCVD*] Confirma el fragmento recibido (N tiene el valor de SEQ).	0	== ECSN	*CONFIRM*	Se acepta el mensaje, se establece ECSN=NE y se confirma el proceso.		Si no hay respuesta o txCON==0 entonces se pasa a Ocioso de lo contrario a WaitSolCfm.	11
	[NON-MATCHING_SOL_CONFIRM_ RCVD] Confirma el fragmento recibido (N no tiene el valor de SEQ esperado).	0	!= ECSN	*CONFIRM*	Se descarta el fragmento y no se elimina ningún fragmento.		WaitSolCfm	12
	[UNSOL_CFM_RCVD] Recepción de una confirmación no solicitada.	1	X	*CONFIRM*	Se descarta el fragmento y no se elimina ningún fragmento.		WaitSolCfm	13
	[BROADCAST_FRAG_RCVD]	0	X	*REQUEST*	Se asume que la confirmación fracasó y se establece		Ocioso	14

	Recepción de un fragmento con la dirección de difusión.				ECSN=NE. Se acepta el fragmento de solicitud y se realiza el proceso.			
	[NEW_FRAG_RCVD] Recepción del fragmento de solicitud (N es el número de secuencia de la última solicitud valida recibida).	0	!=N	*REQUEST*	Se asume que la confirmación fracasó y se establece ECSN=NE. Se envía la respuesta si es requerida. Si txCON==1, se establece ECSN con el número de secuencia en el fragmento de transmisión.		Si no hay respuesta o txCON==0 entonces se pasa a Ocioso de lo contrario a WaitSolCfm.	15
	[REPEAT_FRAG_RCVD] Recepción del fragmento de solicitud (N es el número de secuencia de la última solicitud valida recibida).	0	N	*REQUEST*	Comparar byte a byte con el fragmento de solicitud anterior ¿los bytes coinciden?		-	16
					Sí	Se acepta el fragmento y se envía la misma respuesta. No se realiza ningún proceso.	WaitSolCfm.	17
					No	Se acepta el fragmento y su proceso, se envía una respuesta si es solicitada. Si txCON==1, se establece ECSN con el número de secuencia en el fragmento de transmisión.	Si no hay respuesta o txCON==0 entonces se pasa a Ocioso de lo contrario a WaitSolCfm.	18
	[CONFIRM_RCVD] Confirmación recibida.	-		-	Se descarta el fragmento y no se elimina ningún evento.		Ocioso	19
WaitUnsolCfm	[MATCHING_SOL_CONFIRM_ RCVD] Solicitud de confirmación recibida, SEQ==ECSN.	0	== ECSN	*CONFIRM*	Se realiza el proceso de confirmación con ECSN=NE.		WaitUnsolCfm	20
	[NON-MATCHING_SOL_ CONFIRM_RCVD] Solicitud de confirmación recibida, SEQ!=ECSN.	0	!= ECSN	*CONFIRM*	Se descarta la confirmación del fragmento, no se remueven los eventos.		WaitUnsolCfm	21
	[MATCHING_UNSOL_CFM_RCVD] Recepción de una confirmación no solicitada.	1	M	*CONFIRM*	Se acepta el fragmento y se procesa la confirmación.		Si ECSN==NE entonces se pasa a ocioso de los contrario a WaitSolCfm	22
	[NON-MATCHING_UNSOL_CFM_ RCVD] Recepción del fragmento de confirmación.	1	!=M	*CONFIRM*	Se descarta el fragmento de confirmación.		WaitUnsolCfm.	23
	[BROADCAST_FRAG_RCVD] Recepción de fragmentos con una dirección de difusión.	0	X	*REQUEST*	A la variable ECSN se asigna el valor NE. Si existe una solicitud *READ* diferida, ésta se elimina. Se acepta el fragmento y el proceso solicitado. No se envía respuesta.		WaitUnsolCfm	24
	[READ_REQ_RCVD] Solicitud READ recibida.	0	X	REQUEST	A la variable ECSN se asigna el valor NE. Si existe una solicitud diferida, se reemplaza con la nueva solicitud. Se aplaza la construcción de la respuesta mediante el congelamiento de la solicitud hasta que el software pasa al estado ocioso.		WaitUnsolCfm.	25
	[NON-RD_REQ_RCVD] Solicitud NO READ recibida.	0	!=N	*REQUEST* no valido	A la variable ECSN se asigna el valor de NE. Si existe una solicitud *READ* diferida, ésta se elimina, se acepta el fragmento y se envía la respuesta. Si txCON==1, entonces ECSN=SEQ.		WaitUnsolCfm	26
	[REPEAT_NON-RD_ RCVD] Solicitud NO READ recibida.	0	N	*REQUEST* no valido	Si existe una solicitud *READ* diferida, ésta se elimina; se compara byte por byte con la solicitud anterior ¿coinciden?		-	27
					Sí	Se acepta el fragmento, se envía la misma respuesta. No se procesa la solicitud.	WaitUnsolCfm	28
					No	A la variable ECSN se asigna el valor de NE. Se acepta el fragmento y el proceso solicitado. La respuesta es enviada si es solicitada. Si txCON==1, entonces ECSN=SEQ.	WaitUnsolCfm	29
	[SOL_CONFIRM_TIMEOUT] Tiempo de espera agotado de la solicitud de	-	-	-	A la variable ECSN se asigna el valor de NE, no se elimina ningún evento y tampoco se envía la eliminación de los		WaitUnsolCfm	30

	confirmación del temporizador.				eventos.			
	[UNSOL_CONFIRM_TIMEOUT_ DEFER] Tiempo de espera agotado en espera de la confirmación. La lectura de la respuesta no está diferida.	-	-	-	Se debe considerar la falta de confirmación y cancelación de la espera de una confirmación no solicitada. Finaliza la serie de respuesta no solicitadas.		Ocioso	31
	[UNSOL_CONFIRM_TIMEOUT_NO_ DEFER] Tiempo de espera agotado en espera de la confirmación. La lectura de la respuesta no está diferida.	-	-	-	Se debe considerar la falta de confirmación. ¿Tiene configurado el número de reintentos de confirmaciones no solicitadas?		-	32
					Sí	Se asume que la confirmación fracasó y se poner fin a espera de una confirmación no solicitada. Se Termina la serie respuestas no solicitadas.	Si ECSN==NE entonces se pasa a Ocioso de los contrario a WaitSolCfm	33
					No	Se hace una transmisión idéntica o se regenera un reintento de una respuesta no solicitada.[8]	WaitUnsolCfm	34

[8] Claves para entender las tablas de las máquinas de estado:

- Los títulos de las tablas hacen referencia a los eventos desencadenantes que inician una acción y la posible transición a un estado.
- En la columna B, las etiquetas que se encuentran entre corchetes están generando eventos y el nombre indica el estado específico que lo provocó.
- La letra X significa el estado no importa.
- El símbolo == indica “es igual que”.
- El símbolo **!=** indica “es diferente que”.
- El símbolo - indica “no es aplicable” (NE).
- Las letras N y M representan los números de secuencia del byte de control de la Capa de Aplicación.
  - La letra N indica el número del campo de secuencia de una solicitud valida de la EM o de su respuesta (no de una difusión).
  - La letra M indica el número del campo de secuencia de la respuesta no solicitada o con la más recientemente enviada por parte de la ER.
- La variable txCON en las columnas D y E, se refiere que el bit de confirmación se establece en 0 o en 1 en el byte de control de la Capa de Aplicación de la ER.
- Cuando aparece la frase “sin respuesta” en la columna E, indica que se ha recibido una petición que requiere una respuesta por parte de la ER.
- Cada vez que se transmite una respuesta solicitada que requiere confirmación, se inicializa el contador de respuesta solicitada. Cada vez que la ER transmite una respuesta no solicitada, se inicializa el temporizador de respuestas no solicitadas. Dichos temporizadores se ejecutan de manera concurrente.
- Una vez entrando al estado Ocioso, las solicitudes READ diferidas se manejan como si se hubiesen recibido inmediatamente.
- El símbolo - indica “no es aplicable”.
- N es cualquier número de secuencia valido.
- N+1 es calculado con la Tabla 2.8.
- El símbolo ‡ representa un dato perdido o duplicado.
- SAME significa que ese número es idéntico al número de secuencia del segmento inmediatamente anterior a este segmento.
- +1 significa que el número de secuencia se incrementa en uno (utilizando la Tabla 2.14.
- + M, es el número de secuencia se incrementa en más de uno y menos de 64 bytes (1 <M <64) del número de secuencia del segmento anterior.
- El campo *Sequence* se abreviará como SEQ.
- Los círculos en la Figura 2.33 representan los estados y las líneas representan la transición a un nuevo estado. Los números de las líneas corresponden al número de fila de la Tabla 2.20.
- Las siglas NFCB significan bit de conteo para la siguiente trama. Es el 1 o 0 estado de la siguiente bite FCB para ser incluidos durante la transmisión de una nueva trama de la EP a la ES con el bit FCV habilitado.
- El símbolo ▲ de la Tabla 2.20 indica que la variable SecondaryStationIsReset se establece en verdadero (1).
- El símbolo ▼ de la Tabla 2.20 indica que la variable SecondaryStationIsReset se establece en falso (0).
- Excepto para las filas 27 y 28 de la Tabla 2.20 no se muestra el comportamiento cuando la respuesta de la ES establece el bit de DFC. Si esto ocurre, la EP tiene la opción de:
  - En espera de un plazo razonable antes de transmitir los datos del usuario.
  - Envío de un mensaje REQUEST_LINK_STATES y luego continuar con el estado UR–LinkStatusWait SecResetIdle si la variable se establece en false. Estado R-LinkStatusWait si el estado SecResetIdle variable se establece en true.
  - A la variable SecResetIdle se le da el valor de 0 y se dirige al estado SecUnResetIdle.

- En los campos en donde aparecen las instrucciones "depende de la implementación" o "depende de la aplicación", significa que la causa de la acción es una razón privada o la acción a realizar depende del diseño de software.
- Los valores de las columnas de "Func" de la Tabla 2.21 se refieren a los números de código de la función en los respectivos bytes recibidos o transmitidos en el enlace.
- El doble punto (..) significa que es un rango consecutivo entre el número de la izquierda y el número de la derecha.
- Las siglas EFBC representan al contador de espera de tramas y representa el estado del bit FCB que se espera en la siguiente trama desde la EP con el bit FCV habilitado.
- El símbolo ▲ en la Tabla 2.21 indica que la variable LinkIsReset se le asigna el valor 1 (true).

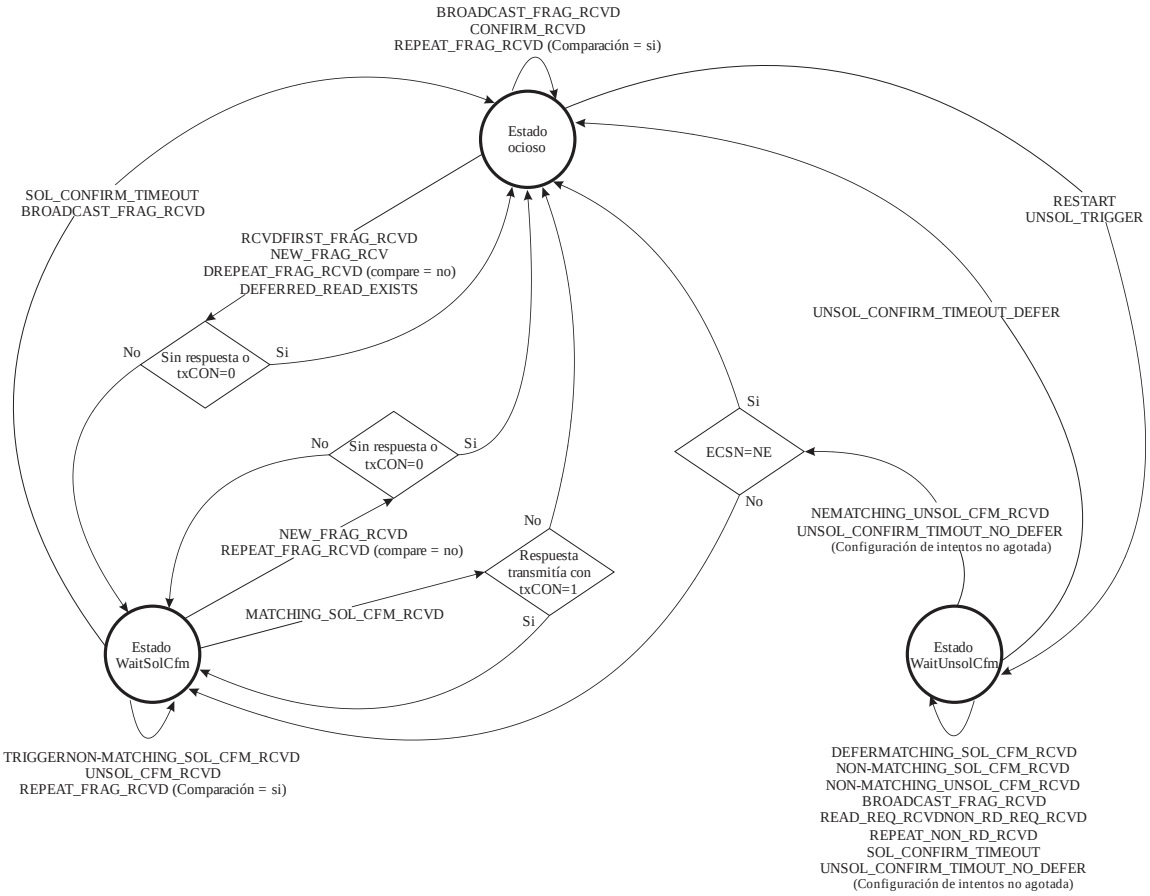


Figura 2.17. Máquina de estados de la ER.

La EM mantiene un conjunto separado e independiente de las variables de respuestas solicitadas y no solicitadas. Las máquinas de estado se deben ejecutar de manera concurrente en las EM para dar soporte a las respuestas no solicitadas.

La EM requiere de tres estados para el manejo de las respuestas solicitadas:

- *Estado ocioso*: La EM está esperando al software del usuario DNP3 para iniciar una solicitud. La EM siempre inicializa en este estado después de un reinicio.
- *Estado espera del primero (AwaitFirst)*: El software está esperando a que llegue el primer fragmento de la respuesta solicitada desde la ER.
- *Estado de ensamble*: En este estado, la EM está a la espera de más fragmentos de una respuesta de fragmentos múltiples.

La Figura 2.18 representa la información descrita en la Tabla 2.10, en la cual se muestra el comportamiento de la EM en la solicitud de respuestas.

2.1.4.3. Máquina de estados de la EM con una respuesta no solicitada

El propósito de la máquina de estados de recepción de una respuesta no solicitada es especificar el comportamiento de una EM cuando los fragmentos son recibidos con UNS=1 en el byte de control de la Capa de Aplicación. El software de la EM debe examinar el valor del campo SEQ en el byte de control de la Capa de Aplicación. También es necesario recordar el valor del campo SEQ y de todos los bytes del fragmento aceptado de una respuesta no solicitada.

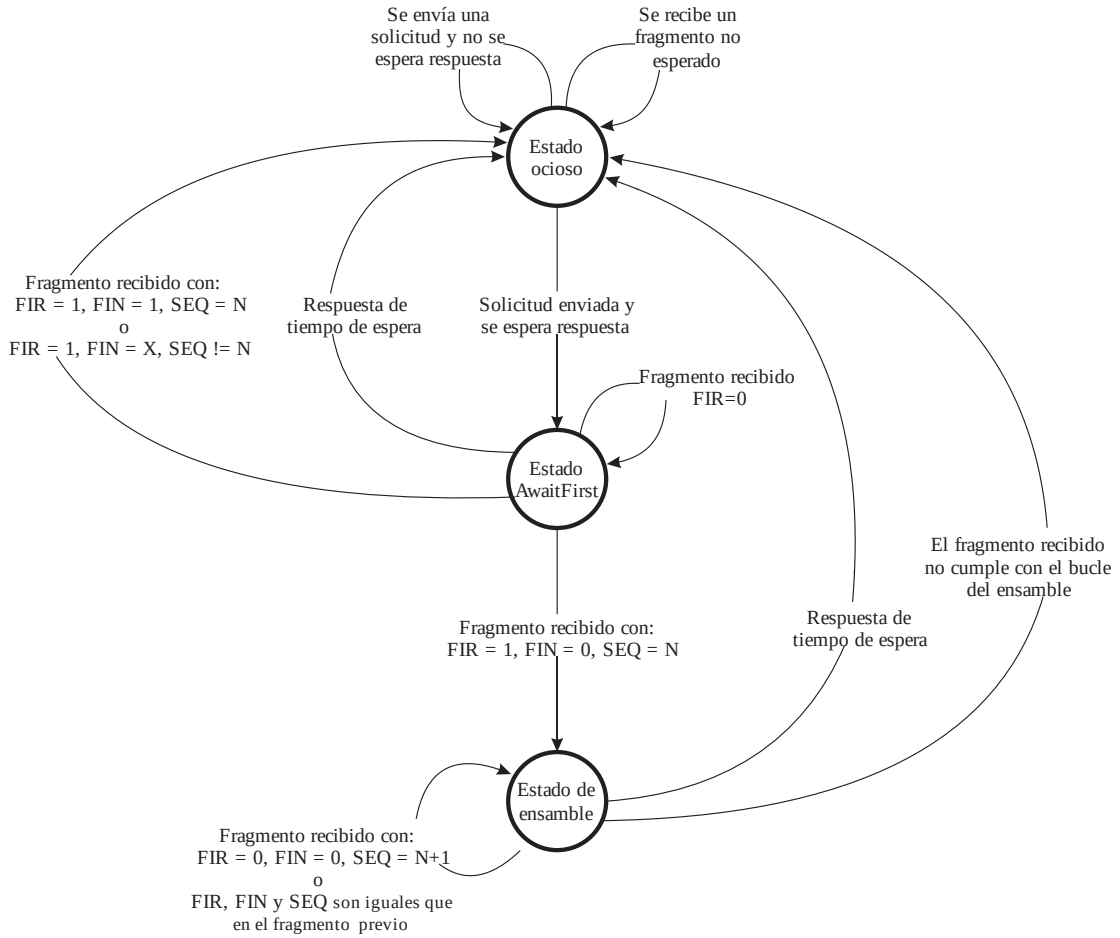


Figura 2.18. Máquina de estados de la EM al realizar solicitudes.

Una EM debe mantener un conjunto de variables para cada ER con la que se comunica. Las variables incluyen el valor del campo SEQ del fragmento de respuesta no solicitada aceptado y los bytes aceptados del fragmento de respuesta no solicitada.

El software de la EM necesita tres estados para el manejo de las respuestas no solicitadas (Tabla 2.11):

- *Inicio*: La EM acaba de encender después de un reinicio, por cualquier razón no se ha realizado un sondeo inicial integral.
- *FirstUR*: La EM inició, se completó un sondeo inicial integral y está esperando la primera respuesta no solicitada de la ER.
- *Ocioso*: El software está a la espera de un fragmento de respuesta no solicitada.

La Figura 2.19 representa la información descrita en la Tabla 2.11.

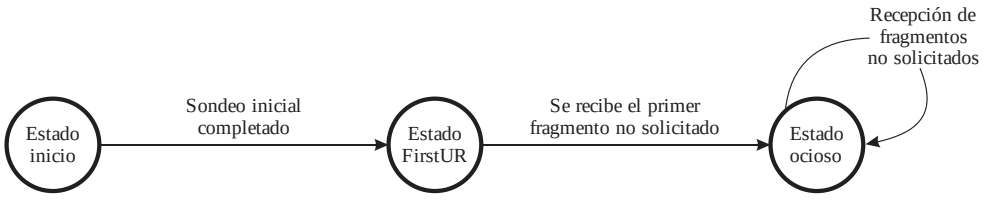


Figura 2.19. Máquina de estados de la EM al recibir una respuesta no solicitada.

Tabla 2.10. Tabla de estados de la EM de solicitud de respuestas.

Estado actual	Evento que genera una acción y una posible transición							Acción a realizar	Estado siguiente	
A	B			C			D	E	F	
	FIR	FIN	SEQ	FIR	FIN	SEQ	SEQ			
Ocioso	X	X	X	X	X	X	X	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	1
	El usuario inicia una solicitud			-	-	-	-	Se transmite la solicitud del usuario y si se espera una respuesta, el fragmento de inicio recibe el temporizador.	Si se espera una respuesta, AwaitFirst; si no Ocioso.	2
AwaitFirst	0	X	X	X	X	X	X	Se descarta el fragmento y no se realiza una confirmación.	AwaitFirst	3
	1	0	N	X	X	X	N	Se envía la confirmación, se acepta y procesa el fragmento, y el fragmento de inicio recibe el temporizador.	AwaitFirst	4
	1	X	!=N	X	X	X	N	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	5
	1	1	N	X	X	X	N	Se envía la confirmación, se acepta y procesa el fragmento.	Ocioso	6
	Tiempo de respuesta agotado			X	X	X	X	Se considera la falta de respuesta.	Ocioso	7
Ensamble	0	0	N	1	0	N	X	Se compara byte a byte con el fragmento aceptado; si coinciden se envía la confirmación y se empieza a recibir el fragmento del temporizador; si los bytes no coinciden, se desecha el fragmento y no hay confirmación.	Si los bytes coinciden Ensamble; si no Ocioso	8
	0	0	N	0	0	N	X	El fragmento se descarta y no se realiza una confirmación.	Ensamble	9
	0	0	N+1	X	0	N	X	Se envía la confirmación, se acepta y procesa el fragmento, y el fragmento de inicio recibe el temporizador.	Ocioso	10
	0	0	!=N y != N+1	X	0	N	X	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	11
	0	1	N+1	X	0	N	X	Se envía la confirmación, se acepta y procesa el fragmento.	Ocioso	12
	0	1	!= N+1	X	0	N	X	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	13
	1	0	N	1	0	N	N	Se compara byte a byte con el fragmento aceptado; si coinciden se envía la confirmación y se empieza a recibir el fragmento del temporizador; si los bytes no coinciden, se desecha el fragmento y no hay confirmación.	Si los bytes coinciden Ensamble; si no Ocioso	14
	1	0	!=N	0	0	N	X	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	15
	1	1	X	X	0	N	X	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	16
				X	X	X	X	Se descarta el fragmento y no se realiza una confirmación.	Ocioso	17
	Tiempo de respuesta agotado			X	X	X	X	Se considera la falta de respuesta.	Ocioso	18

Tabla 2.11. Tabla de estados de la EM ante una respuesta no solicitada.

Estado actual	Evento que genera una acción y posible transición		Evento	Estado siguiente	
A	B	C	D	E	
Inicio	X	-	Se descarta el fragmento y no se envía confirmación.	Inicio	1
	Sondeo inicial completado	-	Se prepara para recibir una respuesta no solicitada.	FirstUR	2
FirstUR	X	-	Se envía una respuesta de confirmación, se acepta y procesa el fragmento.	Ocioso	3
	X y IIN1.7	-	Se envía una respuesta de confirmación, se acepta y procesa el fragmento, se envía una solicitud de reinicio con IIN1.7 y se completa el sondeo.	Ocioso	4
Ocioso	N	N	Se envía una respuesta de confirmación, se compara byte por byte con el fragmento anterior, si coinciden no se realiza ninguna acción, de lo contrario se acepta y procesa el fragmento, y se completa el sondeo.	Ocioso	5
	N+1	N	Se envía una respuesta de confirmación, se acepta y procesa el fragmento.	Ocioso	6
	!=(N+1)	N	Se envía una respuesta de confirmación, se acepta y procesa el fragmento, y se completa el sondeo.	Ocioso	7
	X y IIN1.7	N	Se envía una respuesta de confirmación, se acepta y procesa el fragmento, se envía una solicitud de reinicio con IIN1.7 y se completa el sondeo.	Ocioso	8

2.2. Función de transporte

La Función de Transporte es una subcapa de la Capa de Aplicación (Figura 2.20). Todos los mensajes hacia y desde la Capa de Aplicación pasan a través de la Función de Transporte en su camino desde y hacia la otra estación.

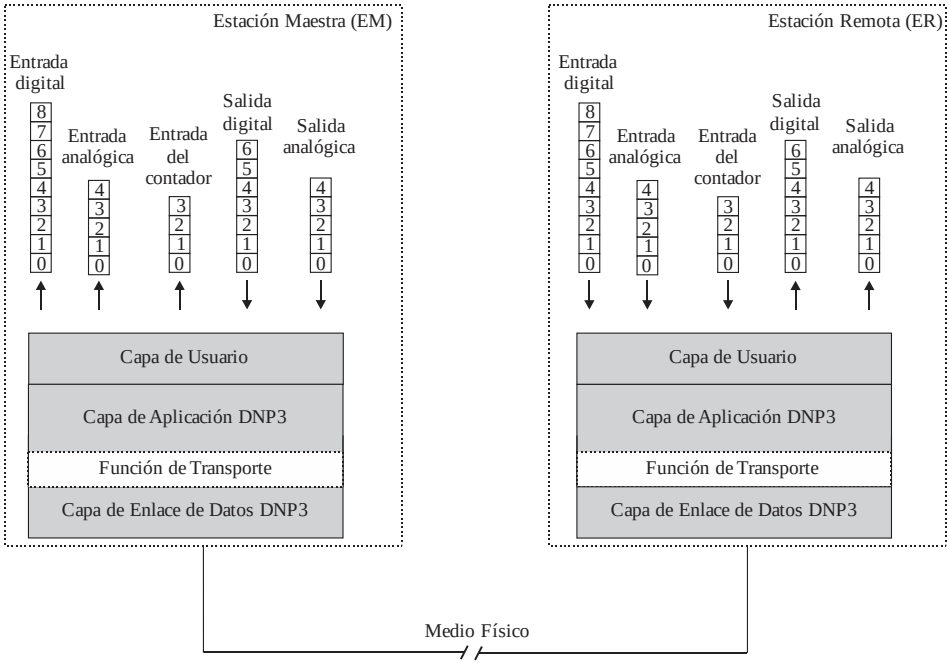


Figura 2.20. Subcapa Función de Transporte dentro de las capas del protocolo DNP3.

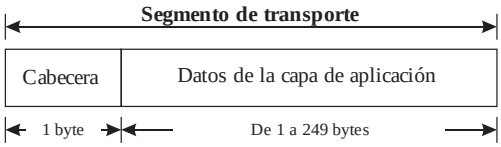


Figura 2.21. Segmento de transporte.

El tamaño de un fragmento de mensaje de la Capa de Aplicación DNP3 puede ser mayor que el número de bytes permitido en una trama de la Capa de Enlace de Datos, por ello la Función de Transporte desensambla los fragmentos de la Capa de Aplicación DNP3 en unidades de datos de la Función de Transporte (segmentos) para su transmisión y ensamblaje del fragmento original en la estación destino.

En la estación origen[9], los fragmentos de la Capa de Aplicación DNP3 se dividen en segmentos y se les añade una cabecera que contiene información de secuencia para cada segmento. La cabecera y los datos de la aplicación forman un segmento de transporte (Figura 2.21) que se pasa a la Capa de Enlace de Datos, los segmentos de transporte siempre se pasan de uno en uno y en secuencia FIFO.

2.2.1. Descripción de la Función de Transporte

La cabecera de la Función de Transporte se compone de un sólo byte. La cabecera es el primer byte en un segmento de transporte. A cada dato fragmentado de la Capa de Aplicación se le antepone una cabecera y después pasa a la Capa de Enlace de Datos para su transmisión.

El byte de la cabecera de la Función de Transporte se compone de tres campos (Figura 2.22):

- *Bit de finalización (FIN)*: Campo de un bit de longitud que cuando se activa indica que éste es el último segmento que se enviará del fragmento en turno.
- *Bit de inicio (FIR)*: Campo de un bit de longitud que cuando se activa indica que éste es el primer segmento del fragmento en turno.
- *Secuencia (Sequence)*: Campo de 6 bits de longitud, utilizado para verificar que los segmentos se reciben correctamente (control de flujo) y para proteger contra la duplicación o falta de segmentos recibidos, tiene un rango de 0 a 63. La secuencia de los números incrementa de uno en uno con %64 para cada segmento de transporte en una serie de segmentos de transporte de un fragmento de la Capa de Aplicación. Cada fragmento de la Capa de Aplicación se transmite como una serie de segmentos consecutivos.

La Tabla 2.12 muestra las reglas de la Función de Transporte para su correcto funcionamiento y la Figura 2.23 ilustra el funcionamiento de la Función de Transporte.

La Tabla 2.13 muestra las reglas que deben cumplir los datos de la Capa de Aplicación.

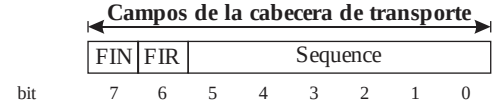


Figura 2.22. Campos de la cabecera de transporte.

[9] La estación origen es la que genera el fragmento, dado que tanto la EM como la ER pueden generar fragmentos.

Tabla 2.12. Reglas de la Función de Transporte.

Regla	Descripción
1	Se puede transmitir una serie de segmentos sólo si el primer segmento de la serie tiene habilitado el bit FIR.
2	La transmisión de una serie de segmentos finaliza cuando en un segmento se habilita el bit FIN.
3	Cuando se ha recibido una serie de segmentos sin finalizar y se recibe un segmento con el bit FIR habilitado, se descarta la serie de segmentos previa y se inicia una nueva serie con el segmento recibido como el primer elemento de la serie.
4	Si no existe una transmisión en proceso de una serie segmentos, se descartan los segmento recibidos sin previa habilitación del bit FIR.
5	Un segmento con el bit FIR habilitado tiene que tener un número de secuencia entre 0 y 63 sin importar el historial de recepción, no obstante se recomienda un incremento del número de secuencia con un número más allá de la secuencia del segmento previamente transmitido como lo indica la ecuación n%64.
6	En la transmisión de los siguientes segmentos se debe realizar lo siguiente: • Cada segmento recibido debe tener un número de secuencia que se incremente en uno con respecto al segmento anterior utilizando la ecuación n%64. • Se descarta todo segmento con el bit FIN desactivado y con un número de secuencia idéntico al del segmento anterior. • Se eliminan los segmentos con el bit FIN habilitado y con un número de secuencia idéntico al del segmento anterior o con cualquier número de secuencia excepto el esperado.
7	Se puede transmitir un único segmento habilitando los bits FIR y FIN.
8	Una vez completada y ensamblada la serie de segmentos, los datos de la Capa de Aplicación pasan a la Función de Transporte.

Tabla 2.13. Reglas de los datos de la Capa de Aplicación.

Regla	Descripción
1	Cada segmento de transporte puede contener desde 1 hasta 249 bytes de la Capa de Aplicación. Los segmentos pueden contener menos de 249 bytes de la Capa de Aplicación. No es necesario que todos los segmentos tengan el mismo tamaño. El receptor debe aceptar los segmentos de transporte de distintos tamaños.
2	La función de transporte conserva el orden del byte del fragmento de la capa de aplicación (Figura 2.23). Para que en el extremo receptor los fragmentos de la Capa de Aplicación se vuelvan a ensamblar en el mismo orden: • Los datos de la Capa de Aplicación se dividen en dimensiones adecuadas de conjuntos de bytes. • La primera porción de los datos de la Capa de Aplicación, con el número de byte 0, se transporta en el primer segmento, este segmento tiene el bit FIR habilitado en la cabecera de la Función de Transporte y un número de serie *n*. • A la siguiente porción de datos de la Capa de Aplicación (si existe), se asigna al segmento un número de secuencia sucesivo calculado con la ecuación n%64. • El patrón de colocar las porciones de la Capa de Aplicación con números de bytes inferiores antes de las porciones con los números de bytes más altos en los segmentos, continúa hasta la última parte que se envía. • En la cabecera del último segmento se habilita el bit FIN.

2.2.2. Máquina de estados de la Función de Transporte

Si se considera que el software de la Función de Transporte soporta un búfer del mismo tamaño del fragmento (búfer de fragmento), en donde los segmentos recibidos se almacenan temporalmente antes de pasar el fragmento a la Capa de Aplicación, el software requiere los siguientes dos estados para su correcto funcionamiento:

- *Estado ocioso*: El software está en espera de un segmento con el bit FIR activado.
- *Estado ensamble*: El búfer de fragmentos contiene datos de la Capa de Aplicación de al menos un segmento, mientras el software espera segmentos adicionales que completen el fragmento.

La Tabla 2.14 muestra los estados de recepción en la Función de Transporte.

Tabla 2.14. Estados de recepción en la Función de Transporte.

Estado actual	Evento que desencadena una acción y la posible transición				Acción	Estado de transición	
A	B	C			D	E	
		FIR	FIN	SEQ			
Ocioso	No es el 1er segmento	0	X	X	Se desecha el segmento.	Ocioso	1
	Un único segmento	1	1	X	Se borra el búfer de fragmento para almacenar los datos del segmento y el fragmento pasa a la Capa de Aplicación.	Ocioso	2
	Primer segmento de múltiples segmentos	1	0	X	Se borra el búfer de fragmento y el segmento recibido coloca los datos en el búfer de fragmento.	Ensamble	3
Ensamble	Igual número de SEQ al estado anterior. Se esperan segmentos	0	0	SAME	Se descarta el segmento.	Ensamble	4
	Igual número de SEQ al anterior, segmento final.	0	1	SAME	Se descarta el segmento.	Ocioso	5
	Se ha recibido el segmento esperado, se esperan más segmentos.	0	0	+1	Anexa los datos del segmento a los datos contenidos en el búfer de fragmento.	Ensamble	6
	Se ha recibido el segmento esperado, segmento final.	0	1	+1	Anexar los datos del segmento al contenido del búfer de fragmento y pasarlo a la Capa de Aplicación.	Ocioso	7
	El número SEQ está fuera de secuencia.	0	X	+M	Se descarta el segmento.	Ocioso	8
	Primer segmento de múltiples segmentos	1	0	X	Se borra el búfer de fragmento y el segmento recibido coloca los datos en el búfer de fragmento.	Ensamble	9
	Un único segmento	1	1	X	Se borra el búfer de fragmento y el segmento recibido coloca los datos en el búfer.	Ocioso	10

La Figura 2.24 muestra la representación gráfica de la Tabla 2.14.

2.3. Capa de Enlace de Datos DNP3

La Capa de Enlace de Datos proporciona una interfaz entre la Función de Transporte y el canal de comunicación (medio físico) o capa de transporte de red (Figura 2.25). Las principales funciones de la Capa de Enlace de Datos son el direccionamiento de estaciones y la detección de errores. Esta capa añade los últimos bytes de la cabecera DNP3 que se transmite por el canal de comunicación.

El protocolo ha sido diseñado para funcionar en un modo orientado a bytes en un canal de comunicación, así como a través de redes orientadas a paquetes, utilizando TCP/IP (sistemas orientados a conexión) y UDP/IP (sistemas no orientados a conexión).

La Capa de Enlace de Datos DNP3 tiene dos objetivos principales:

- Dirigir el transporte de los datos desde la Capa de Aplicación del dispositivo origen a través del canal de comunicación hasta la Capa de Aplicación del dispositivo destino.

El software de la Capa de Enlace de Datos codifica los segmentos de la Función de Transporte que provienen de una capa superior en tramas de la Capa de Enlace de Datos y finalmente se envía la trama por el canal de comunicación.

- Gestionar las variables asociadas a la sincronización de tramas, controlar los errores y proporcionar indicadores de estado del enlace.

La capa de enlace de datos proporciona los siguientes servicios:

- Encapsulación de los segmentos de la Función de Transporte en tramas de la Capa de Enlace de Datos para su transmisión por el canal de comunicación.
- Decodificación de las tramas recibidas desde el canal de comunicación en segmentos.
- Detección de errores, en DNP3 cada trama transmitida tiene una palabra[10] dedicada al CRC por cada 16 bytes de datos y otro para el encabezado de la trama de enlace de datos. Esta palabra del CRC es verificada en cada bloque de la trama recibida para que los datos válidos pasen a la Función de Transporte.
- DNP3 requiere de las direcciones de la estación origen y de la estación destino para permitir que varias EM y múltiples ER puedan compartir el canal de comunicación.
- Confirmación de cada trama de enlace de datos.
- Detección de pérdida o repetición de tramas de la Capa de Enlace de Datos de una solicitud.

2.3.1. Modelo de operación

La Capa de Enlace de Datos realiza operaciones para transmitir los datos de la capa de Enlace de Datos a otro dispositivo DNP3, además de realizar las funciones para la gestión del enlace. Una operación consiste en uno o dos mensajes:

- Un mensaje de solicitud del dispositivo de la transacción, llamado Estación Primaria (EP), a otro dispositivo, llamado Estación Secundaria (ES).
- Opcionalmente, un mensaje de respuesta de la ES a la EP.

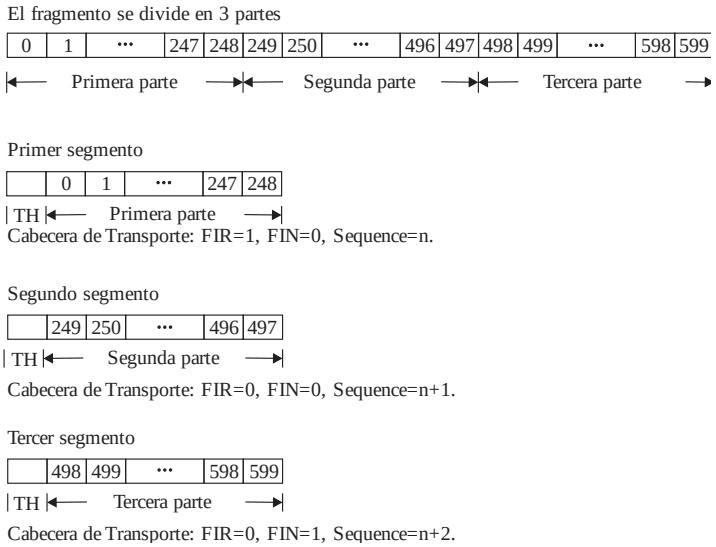


Figura 2.23. Ejemplo de la segmentación de un fragmento de 600 datos.

[10] En DNP3 se usa el término “palabra” para referirse a una longitud de datos de 16 bits.

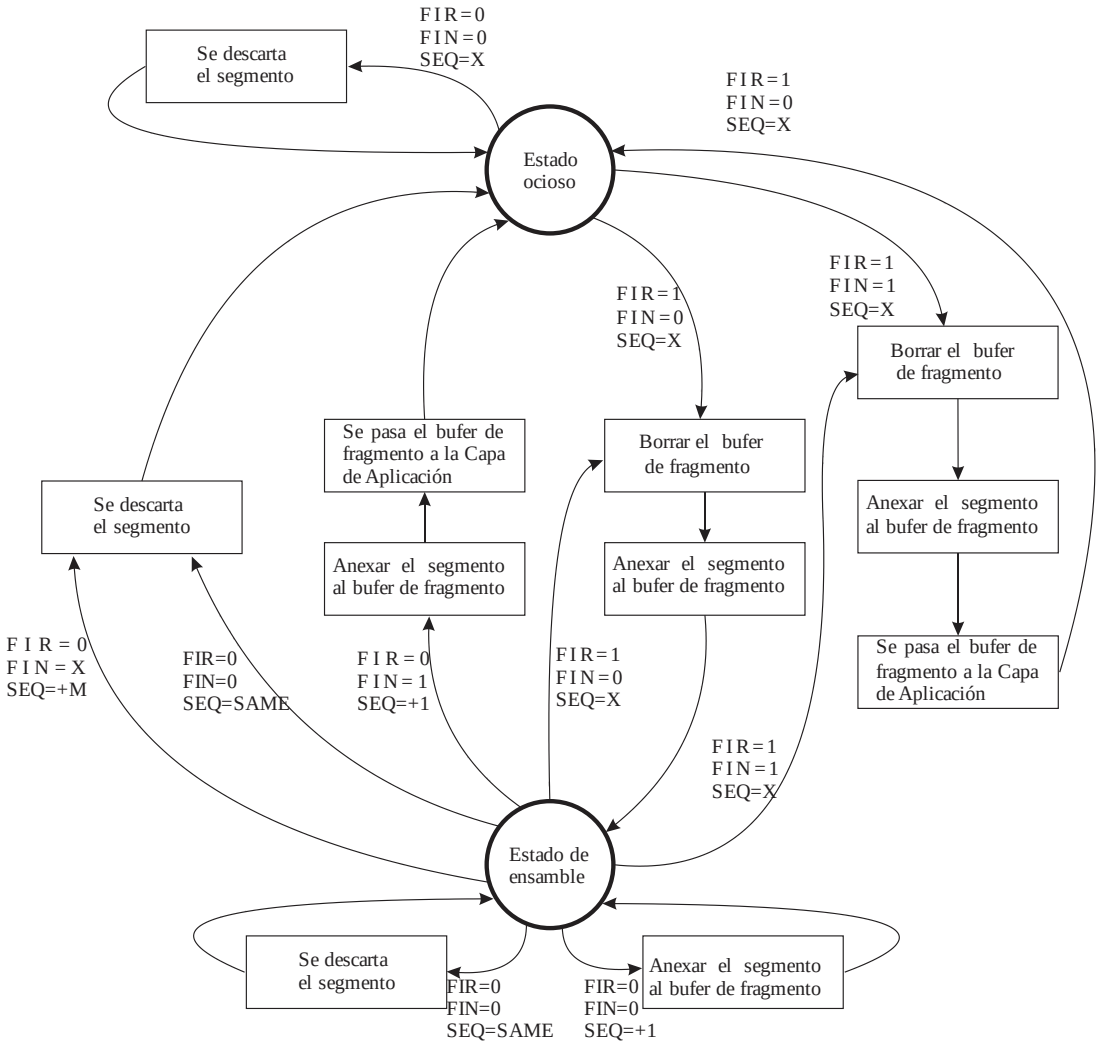


Figura 2.24. Diagrama de estados de la Función de Transporte.

Las solicitudes y respuestas de la Capa de Enlace de Datos son distintas de las solicitudes y respuestas de la Capa de Aplicación.

En los mensajes de solicitud de la EP a la ES, el código de función está incluido en el mensaje para especificar la orden (*command*). Algunos códigos de función están destinados a la gestión del enlace. En este punto, el contenido del mensaje es privado entre las dos Capas de Enlace de Datos y no son transmitidos hacia las capas superiores.

Algunos códigos de función de los mensajes de la Capa de Enlace de Datos de la EP a la ES requieren de una respuesta. Sólo se puede enviar una notificación de la recepción del mensaje si éste fue recibido por la EP o contiene información de la Capa de Enlace de Datos de la ES, pero jamás contiene datos de las capas superiores.

La Capa de Enlace de Datos está dividida en una EP y una ES. El comportamiento de la EP es independiente del comportamiento de la ES. Las EP y ES son empleadas para el envío y recepción-confirmación de los datos entre dispositivos (EM y ER) (Figura 2.26).

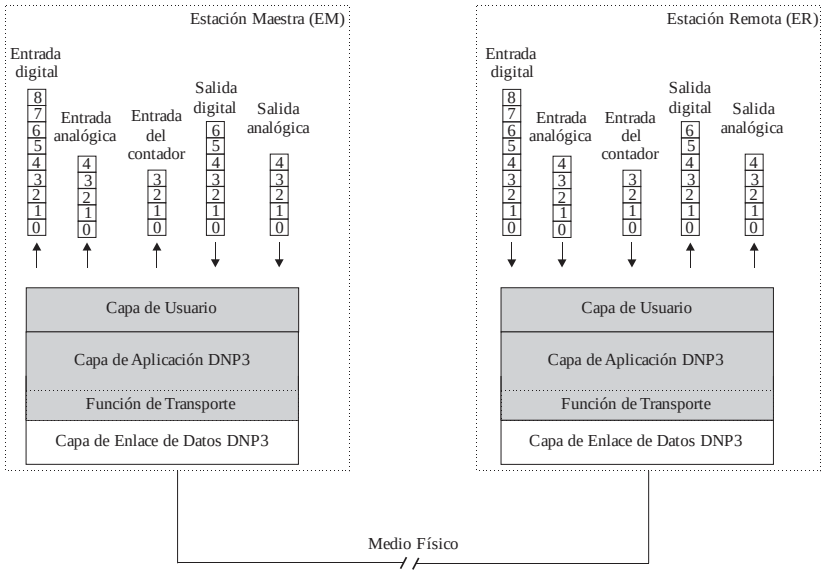


Figura 2.25. Capa de Enlace de Datos dentro de las capas del protocolo DNP3.

2.3.2. Formato de la trama DNP3

La trama de la Capa de Enlace de Datos tiene un bloque de longitud fija de la cabecera en el Bloque 0, seguido de los bloques de datos (opcionales). Cada bloque termina con un CRC de 16 bits como se muestra en la Figura 2.27.

La Figura 2.27 muestra los campos del Bloque 0:

- *Inicio*: Tiene una longitud de 2 bytes, con los valores hexadecimales 0x05 y 0x64 respectivamente.
- *Longitud (Len)*: Tiene una longitud de un byte y define el número de bytes que siguen en los bloques de cabecera y de datos, sin incluir los del campo CRC. Incluye los campos Control, Destino y Fuente de la cabecera, además de los datos de usuario. El valor mínimo para este campo es de 5, indicando sólo la cabecera actual y el valor máximo es de 255.

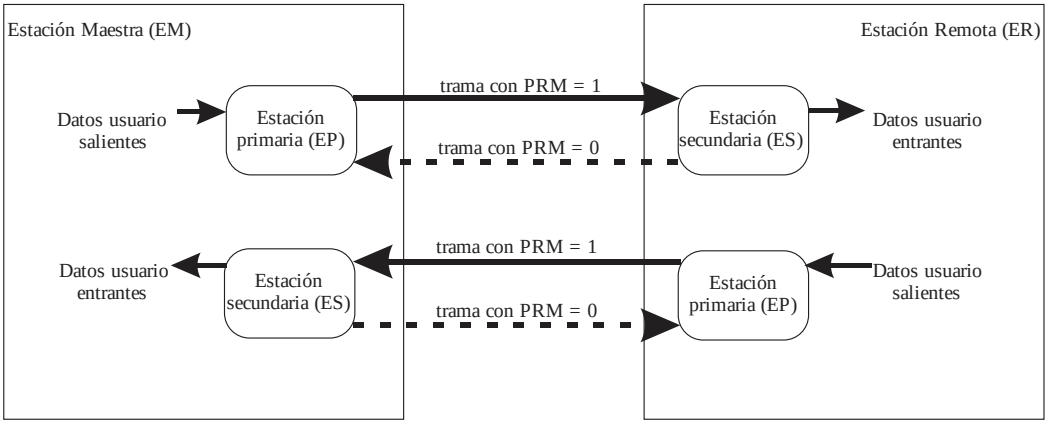


Figura 2.26. Diagrama del modelo de operación de la Capa de Enlace de Datos.

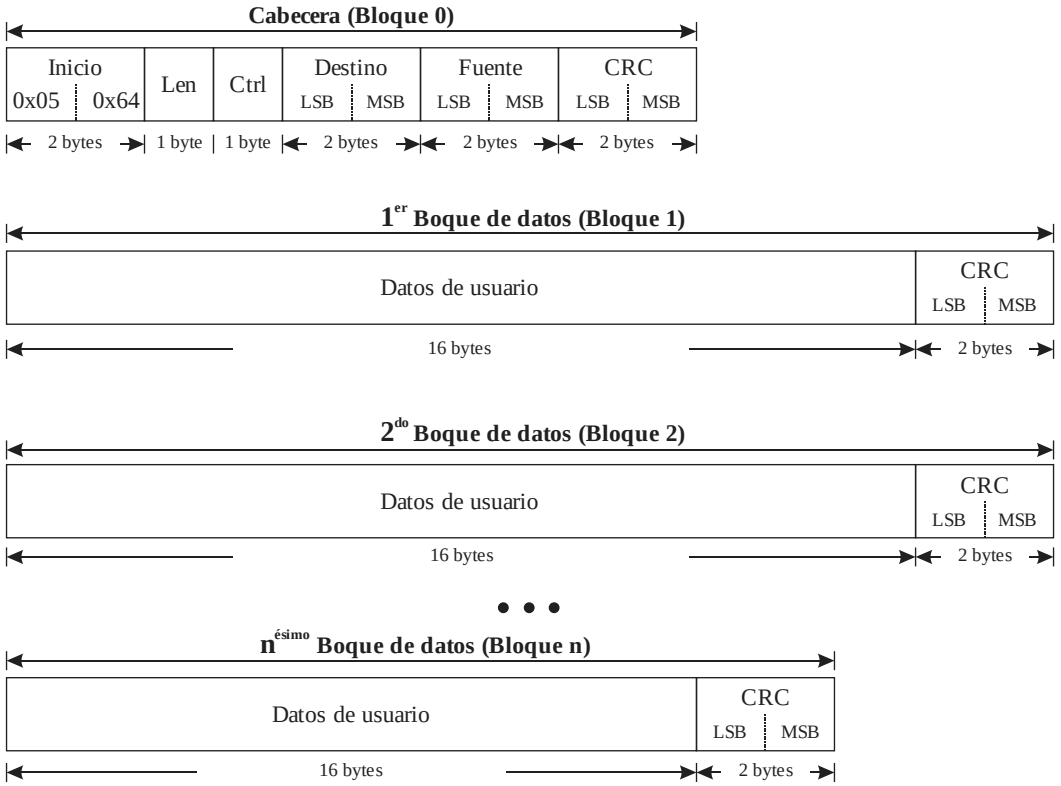


Figura 2.27. Formato de la trama DNP3.

- *Control (Ctrl)*: Con una longitud de un byte y contiene información sobre (Figura 2.28):
  - *Dirección (DIR)*: Indica el origen físico de la transmisión de la trama:
    - DIR=1 indica que la trama fue enviada por la EM.
    - DIR=0 indica que la trama fue enviada por la ER.
  - *Bit de mensaje de la EP (PRM)*: Indica la dirección de la trama con respecto a la estación origen.
    - PRM=1 indica que una operación está siendo iniciada por una EM o una ER, el código de la función se elige de la Tabla 2.15. Se habilitan los campos FCV y FCB.
    - PRM=0 indica la finalización de una operación, ya sea una EM o una ER, el código de la función se elige de la Tabla 2.16. Se habilita el campo DFC.
  - *Bit de conteo de tramas (FCB)*: Sólo es válido cuando una solicitud se envía desde la EP a la ES y con el bit FCV habilitado.
  - *Bit de conteo de tramas válidas (FCV):* Este bit aparece cuando se transmiten tramas de la EP a la ES y especifica si la ES debe examinar el bit FCB.
    - FCV=1 indica que el valor del bit del FCB es válido, se debe comprobar el valor del bit del FCB en el mensaje recibido.
    - FCV=0 indica que se debe ignorar el valor del bit FCB.
  - *Bit de control del flujo de datos (DFC)*: Este bit se emplea en las respuestas de la ES sin tener en cuenta el byte del código de función. Este bit también se emplea para indicar la falta de espacio en los búfers de la Capa de Aplicación para almacenar los fragmentos recibidos:

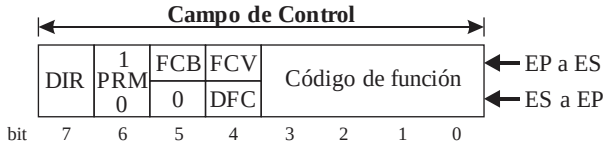


Figura 2.28. Byte de control de la cabecera de las tramas.

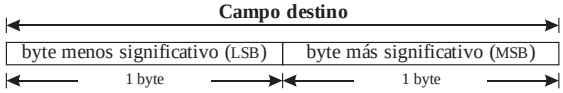


Figura 2.29. Formato de los campos Destino y Fuente.

- DFC=1 indica que los búfers se encuentran llenos o que la ES está ocupada.
- DFC=0 indica que los búfers y la ES están aptos para recibir un fragmento.

- *Código de función:* Indica la función o servicio asociado a la trama, el valor de este campo depende de la secuencia del mensaje, si es de la EP a la ES se utilizan los códigos de función de la Tabla 2.15 o si es de la ES a la EP se emplean los códigos de la Tabla 2.16.
- *Destino*: Tiene una longitud de 2 bytes en los que se especifica la dirección de la estación a la que se envía la trama (Figura 2.29).
- *Fuente*: Tiene una longitud de 2 bytes en los que se especifica la dirección de la estación que envío la trama (Figura 2.29).

Tabla 2.15. Códigos de función de la comunicación de la EP a la ES (PRM=1).

Código de función de la EP	Nombre del código de función	Servicio	Bit FCV	Códigos de respuesta permitidos para la ES
0	RESET_LINK_STATES	Reinicio del enlace remoto.	0	0 ó 1
1	-	Obsoleto.	-	15 o sin respuesta
2	TEST_LINK_STATES	Función prueba de enlace.	1	0 ó 1 (se acepta que no se responda si el enlace está en estado UnReset)
3	CONFIRMED_USER_DATA	Entrega del dato de aplicación. Se solicita una confirmación.	1	0 ó 1
4	UNCONFIRMED_USER_DATA	Entrega del dato de aplicación. No se solicita una confirmación.	0	Sin respuesta
5	-		-	15 o sin respuesta
6	-		-	15 o sin respuesta
7	-		-	15 o sin respuesta
8	-		-	15 o sin respuesta
9	REQUEST_LINK_STATUS	Solicitud del estado del enlace.	0	11
10	-		-	15 o sin respuesta
11	-		-	15 o sin respuesta
12	-		-	15 o sin respuesta
13	-		-	15 o sin respuesta
14	-		-	15 o sin respuesta
15	-		-	15 o sin respuesta

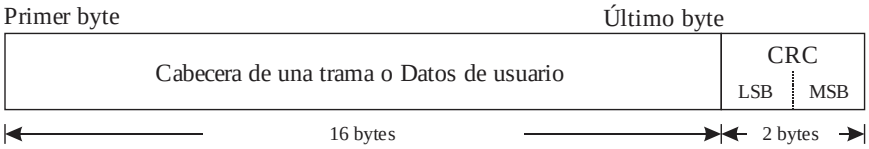


Figura 2.30. Ordenamiento del CRC.

Tabla 2.16. Códigos de función de la comunicación de la ES a la EP (PRM=0).

Código de función de la ES	Nombre del código de función	Servicio
0	ACK	Reconocimiento confirmado.
1	NACK	Reconocimiento no confirmado.
2	-	Reservado.
3	-	Reservado.
4	-	Reservado.
5	-	Reservado.
6	-	Reservado.
7	-	Reservado.
8	-	Reservado.
9	-	Reservado.
10	-	Reservado.
11	LINK_STATUS	Estado del enlace.
12	-	Reservado.
13	-	Reservado.
14	-	Obsoleto.
15	NOT_SUPPORTED	Sin soporte del enlace.

2.3.2.1. Datos de usuario

Los datos de usuario se refieren a los datos que pertenecen a las capas superiores que utilizan la Capa de Enlace de Datos. Una filosofía que hay detrás del diseño de la Capa de Enlace de Datos es la independencia de dicha capa, ésta proporciona un servicio a las capas superiores sin la necesidad de saber cuál es el contenido de los datos que transporta.

Si se observa la Figura 2.27, los campos siguientes al Bloque 0 son los datos del usuario (carga útil). El número máximo de bytes de datos de usuario que una trama puede transportar es de 250 (los 5 bytes de la Cabecera de Enlace de Datos se incluyen en esta cuenta dejando así un máximo de 250 bytes para datos de usuario). Cada bloque de datos de usuario, excepto el último, debe contener exactamente 16 bytes de datos de usuario; el último bloque tiene los bytes restantes si el número total de octetos de datos de usuario no es divisible por 16. Por último, cada bloque de datos de usuario tiene un CRC de dos bytes.

Cabe señalar que no todas las tramas transmiten los datos del usuario, algunas se utilizan sólo para la gestión del enlace.

2.3.2.2. CRC

El CRC del protocolo de comunicaciones DNP3 tiene una longitud de 2 bytes, los cuales se añaden al final de cada bloque para conformar una trama (Figura 2.30). Para calcular el CRC [URL8] del Bloque 0 se ocupan los campos: Inicio, Len, Ctrl, Destino y Fuente (Figura 2.27) y se emplea el polinomio de la Ecuación 2.1. Una vez calculado el CRC, éste se invierte antes de ser colocado en el bloque de transmisión.

$$x^{16}+x^{13}+x^{12}+x^{11}+x^{10}+x^{8}+x^{6}+x^{5}+x^{2}+1$$ **Ecuación 2.1**

La Figura 2.31 y la Figura 2.32 muestran el algoritmo empleado para el cálculo del CRC del protocolo de comunicaciones DNP3. Para la creación del CRC es necesario ordenar los datos invirtiendo byte por byte (Figura 2.31) para que sean procesados por la función que genera el CRC, esto se hace mediante aritmética módulo 2 entre los datos y el polinomio de DNP3 (Ecuación 2.1); esta operación se realiza sucesivamente mediante corrimientos hasta que los datos ya no sean divisibles por el polinomio. Una vez realizadas las operaciones descritas se niega el polinomio y los datos son regresados con el MSB seguido del LSB (Figura 2.32), esto último se realiza para cumplir con lo establecido en el orden del CRC (Figura 2.30).

Para la comprobación del CRC se realiza el proceso anterior de manera inversa.

Cada trama de la Capa de Enlace de Datos debe cumplir con las reglas descritas en el apartado 2.3.5.

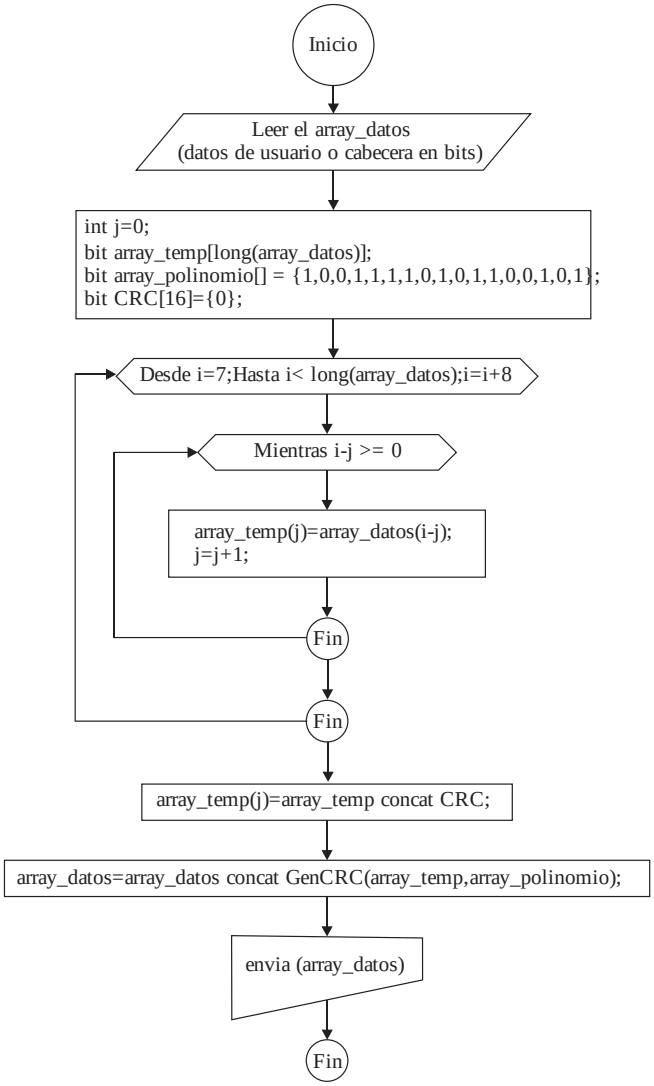


Figura 2.31. Diagrama de flujo de la preparación de los datos para el CRC.

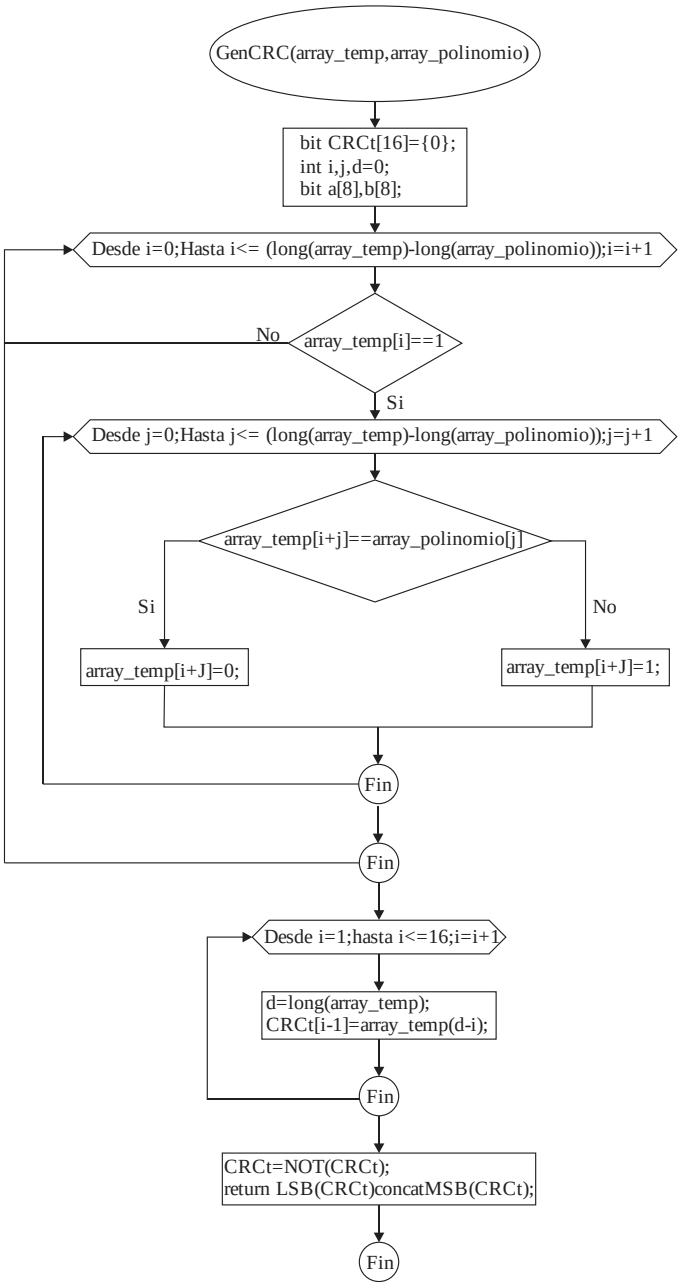


Figura 2.32. Diagrama de flujo de la función que genera el CRC.

2.3.3. Reglas de direccionamiento

Cada EM y ER dentro de una red debe utilizar una dirección DNP en el rango de 0x0000 a 0xFFEF y debe ser única para cada dispositivo en el enlace.

Las direcciones que se encuentran en el rango de 0xFFF0 a 0xFFFF (Tabla 2.17) están reservadas para uso especial del protocolo DNP3 y no deben asignarse a los dispositivos.

2.3.4. Variables para el control del enlace

Cada dispositivo debe mantener un conjunto de variables independientes para cada dispositivo con el que se comunicará mediante la confirmación de los servicios de Capa de Enlace de Datos. Las variables principales se muestran en la Tabla 2.18 y en la Tabla 2.19.

Tabla 2.17. Direccionamiento de protocolo de comunicaciones DNP3.

Dirección	Uso
0xFFFF	Difusión, confirmación de la Capa de Aplicación con opción de borrar la IIN 1.0.
0xFFFE	Difusión, confirmación de la Capa de Aplicación para borrar la IIN 1.0.
0xFFFD	Difusión, confirmación de la Capa de Aplicación sin borrar la IIN 1.0.
0xFFFC	Direccionamiento automático.
0xFFF0 a 0xFFFB	Reservado.

Tabla 2.18. Variables de la EP.

Variable	Descripción
SecondaryStationIsReset	Es una variable de tipo booleana que indica que la EP envió una trama con la función RESET_LINK_STATES y ha recibido una confirmación ACK.
SoftwareOperatingState	Indica el estado actual de la EP (Tabla 2.20).
NFCB (*Next FCB*)	Contiene el valor del siguiente FCB que será incluido durante la transmisión de una nueva trama de la EP a la ES con el bit FCV habilitado.

Tabla 2.19. Variables de la ES.

Variable	Descripción
LinkIsReset	Es una variable de tipo booleana que indica que el enlace se restableció por medio de la recepción de una trama que contiene la función RESET_LINK_STATES desde la EP.
SoftwareOperatingState	Indica el estado actual de la EP (Tabla 2.21).
EFCB (*Expected FCB*)	Contiene el valor del bit FCB esperado en la siguiente trama desde la EP con el bit FCV habilitado.

2.3.5. Detección de tramas erróneas

La Capa de Enlace de Datos de la ER debe revisar las tramas antes de su aceptación, para ello debe examinar los bloques de las tramas empleando las siguientes reglas y descartando aquellos bloques que no las cumplan:

- Los bits del campo Inicio deben ser 0x05 y 0x64 como se muestra en la Figura 2.27.
- La dirección destino debe coincidir con una de las direcciones que se programó en la Capa de Enlace de Datos para ser aceptada, las cuales son las direcciones reservadas o direcciones especiales de la Tabla 2.17.
- La dirección fuente debe coincidir con una de las direcciones programadas en la Capa de Enlace de Datos. Un dispositivo puede optar por aceptar las tramas de cualquier dirección Fuente.
- Los valores correctos del CRC deben aparecer en los lugares indicados dentro de la trama.
- La longitud real de la trama debe coincidir con el valor indicado en el campo Len.
- Los códigos de función deben cumplir con las funciones indicadas de acuerdo a la dirección del mensaje como se muestra en la Tabla 2.15 y en la Tabla 2.16.
- El bit FCV se debe habilitar en las tramas que se envían de la EP a la ES de acuerdo al código de función.

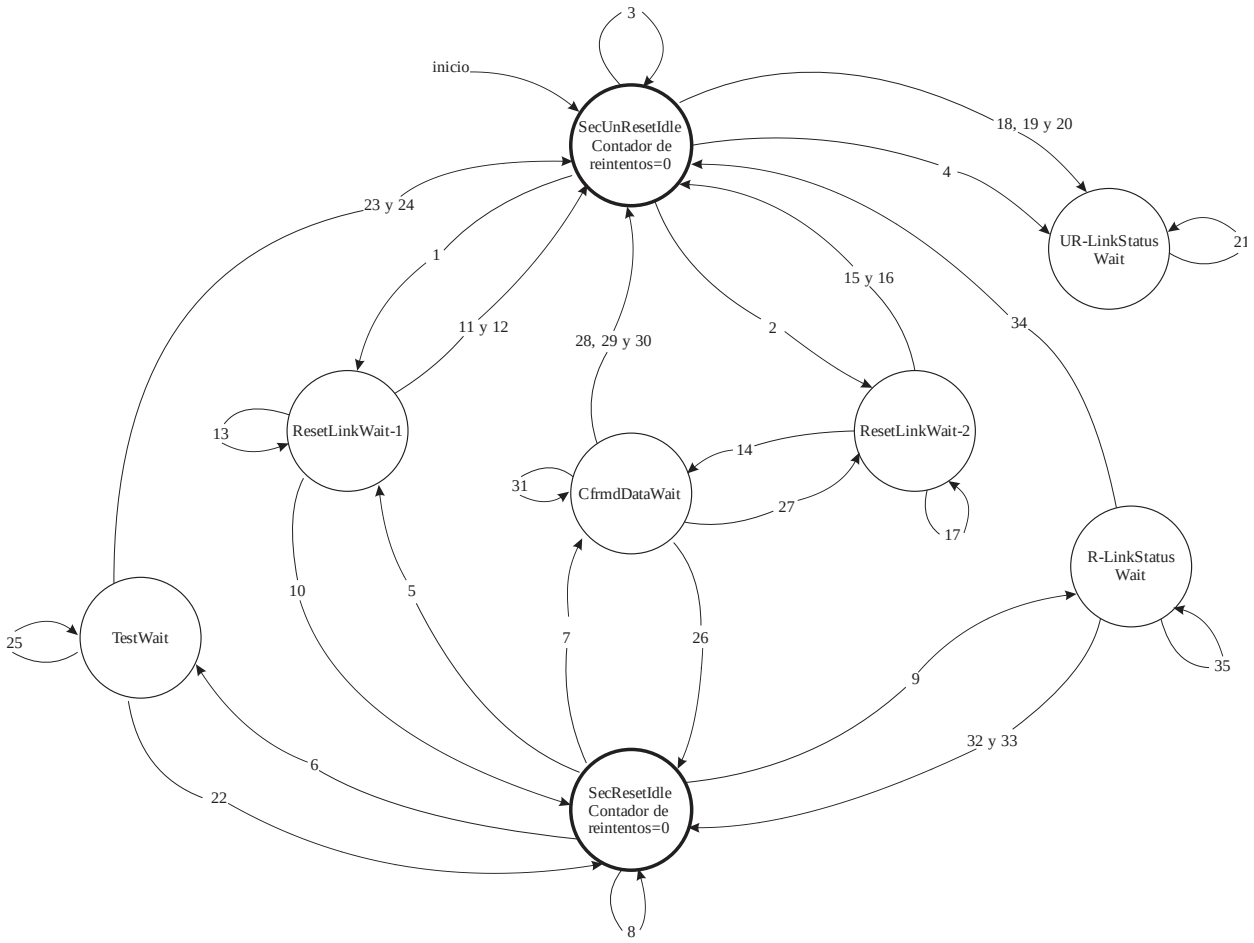


Figura 2.33. Diagrama de estados de la EP.

2.3.6. Anulación de colisiones

En el funcionamiento de algunos sistemas se requiere de un dispositivo para detectar si la transmisión se produce mientras otro dispositivo está transmitiendo. Si esto llegara a suceder, las transmisiones colisionan y casi con seguridad es una causa errores de comunicación en el dispositivo o dispositivos receptores. A fin de evitar colisiones, un dispositivo que detecta que la transmisión de otro dispositivo está en marcha, debe retrasar su transmisión y colocarla en espera para evitar colisiones.

Los dispositivos pueden detectar colisiones de dos maneras:

- La detección de la portadora en el medio.
- La detección de la recepción de los bits de datos.

Un dispositivo que está listo para transmitir debe esperar un tiempo de retardo de retención después de detectar que un dispositivo está transmitiendo. La demora debe constar de dos partes que se suman, un tiempo configurable fijo y un tiempo aleatorio. Estos tiempos dependen del sistema y no están especificados por DNP3. La retención de tiempo comienza cuando se detectan transmisiones de otro dispositivo, en ese instante puede empezar antes de que el dispositivo tenga un mensaje listo para su transmisión. Cabe señalar que muchos enlaces no requieren de prevención de colisiones y su aplicación es opcional.

Tabla 2.20. Tabla de estados de la EP.

Estado actual	Evento que desencadena una acción y la posible transición	Acción		Estado de transición	
A	**B**	**C**	**D**	**E**	
SecUnResetIdle	Depende de la implementación.	Contador de reintentos=0	0 RESET_LINK_STATES	ResetLinkWait-1	1
	El usuario requiere una confirmación.	Contador de reintentos=0	0 RESET_LINK_STATES	ResetLinkWait-2	2
	El usuario no requiere una confirmación.	-	4 UNCONFIRMED_USER_DATA	SecUnResetIdle	3
	Se mantiene activa o implementa la prevención de datos en ejecución.	Contador de reintentos=0	9 REQUEST_LINK_STATUS	UR–LinkStatusWait	4
SecResetIdle	Depende de la implementación.	Contador de reintentos=0	0 RESET_LINK_STATES	ResetLinkWait-1	5
	Solicitud del usuario para pruebas de enlace.	Contador de reintentos=0	2 TEST_LINK_STATES	TestWait	6
	Solicitud del usuario para enviar datos confirmados.	Contador de reintentos=0	3 CONFIRMED_USER_DATA	CfmdDataWait	7
	Solicitud del usuario para no enviar datos confirmados.	-	4 UNCONFIRMED_USER_DATA	SecResetIdle	8
	Se mantiene activa o implementa la prevención de datos en ejecución.	Contador de reintentos=0	9 REQUEST_LINK_STATUS	R–LinkStatusWait	9
ResetLinkWait-1	Se recibe código de función 0 (ACK).	▲ NFCB=1	-	SecResetIdle	10
	Código de función recibido diferente a 0.	▼	-	SecUnResetIdle	11
	Tiempo de espera agotado, sin respuesta y se hace reintento.	▼Información de usuario fallida	-	SecUnResetIdle	12
	Tiempo de espera agotado, sin respuesta y no se hace un reintento.	Incrementar el contador de reintentos	0 RESET_LINK_STATES	ResetLinkWait-1	13
ResetLinkWait-2	Se recibe código de función 0 (ACK).	▲ NFCB=1	3 CONFIRMED_USER_DATA	CfmdDataWait	14
	Código de función recibido diferente a 0.	▼	-	SecUnResetIdle	15
	Tiempo de espera agotado, sin respuesta y no se hace un reintento.	▼Información de usuario fallida	-	SecUnResetIdle	16
	Tiempo de espera agotado, sin respuesta y se hace reintento.	Incrementar el contador de reintentos	0 RESET_LINK_STATES	ResetLinkWait-2	17
UR–LinkStatusWait	Se recibe el código de función 11 (LINK_STATUS).	Depende de la implementación	-	SecUnResetIdle	18
	Código de función recibido	Nota de respuesta errónea	-	SecUnResetIdle	19

	diferente a 11.				
	Tiempo de espera agotado, sin respuesta y no se hace un reintento.	Informar a los usuarios del fallo	-	SecUnResetIdle	20
	Tiempo de espera agotado, sin respuesta y se hace reintento.	Incrementar el contador de reintentos	9 REQUEST_LINK_STATUS	UR–LinkStatusWait	21
TestWait	Se recibe código de función 0 (ACK).	Activar NFCB	-	SecResetIdle	22
	Código de función recibido diferente a 0.	▼	-	SecUnResetIdle	23
	Tiempo de espera agotado, sin respuesta y no se hace un reintento.	▼Información de usuario fallida	-	SecUnResetIdle	24
	Tiempo de espera agotado, sin respuesta y se realiza un reintento.	Incrementar el contador de reintentos	2 TEST_LINK_STATES	TestWait	25
CfmdDataWait	Se recibe código de función 0 (ACK).	Activar NFCB	-	SecResetIdle	26
	Se recibe código de función 1 (NACK) DFC=0.	▼Contador de reintentos=0	0 RESET_LINK_STATES	ResetLinkWait-2	27
	Se recibe código de función 1 (NACK) DFC=1.	▼	-	SecUnResetIdle	28
	Código de función recibido diferente a 0 y 1.	▼	-	SecUnResetIdle	29
	Tiempo de espera agotado, sin respuesta y no se hace reintento.	▼Información de usuario fallida	-	SecUnResetIdle	30
	Tiempo de espera agotado, sin respuesta y se hace un reintento.	Incrementar el contador de reintentos	3 CONFIRMED_USER_DATA	CfmdDataWait	31
R–LinkStatusWait	Se recibe el código de función 11 (LINK_STATUS).	Depende de la implementación	-	SecResetIdle	32
	Código de función recibido diferente a 11	Nota de respuesta errónea	-	SecResetIdle	33
	Tiempo de espera agotado, sin respuesta y no se hace un reintento.	▼Información de usuario fallida	-	SecUnResetIdle	34
	Tiempo de espera agotado, sin respuesta y se hace reintento.	Incrementar el contador de reintentos	9 REQUEST_LINK_STATUS	R–LinkStatusWait	35

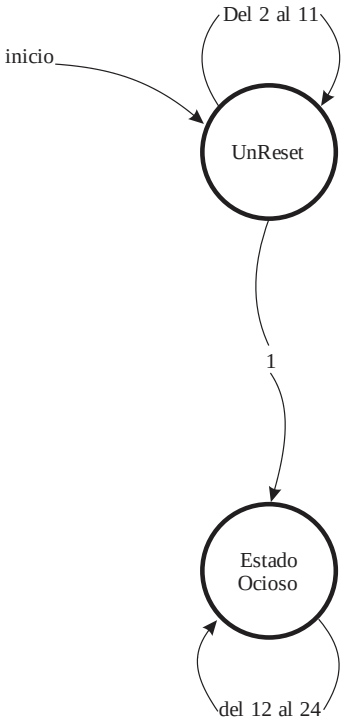


Figura 2.34. Diagrama de estados de la ES.

2.3.7. Máquinas de estado de la Capa de Enlace de Datos

2.3.7.1. Requerimientos de los estados de la EP

En este índice se describen los requisitos de los estados para que una EP transmita una solicitud a la ES de otro dispositivo los cuales son descritos en la Tabla 2.20 y en la Figura 2.33.

2.3.7.2. Requerimientos de los estados de la ES

En este inciso se describen los requerimiento de estados cuando una ES recibe un mensaje de otro dispositivo con el bit PRI=1 los cuales se describen en Tabla 2.21 y en la Figura 2.34.

El software requiere de dos estados para su correcto funcionamiento:

- *Estado UnReset*: La ES se reinició, se perdió la alimentación de voltaje o se detectó una pérdida de comunicación con la EP.
- *Estado ocioso*: El software se encuentra a la espera de que se le envíe un mensaje por parte de la EP.

Se devuelve a la EP el correcto valor de bit DFC, que se incluye en el byte de control. En la Tabla 2.21 no se indica si con el bit PRI=0 en la solicitud recibida las acciones no se realizan y la respuesta no se transmite. El bit PRI siempre se deshabilita en las respuestas enviadas a la EP.

Tabla 2.21. Tabla de estados de la ES.

Estado Actual	Evento que desencadena una acción y la posible transición			Acción			Estado de transición	
A	B			C	D		E	
	FCB	FCV	FUNC		FUNC	Comentarios	Ocioso	
UnReset	X	0	0	▲ Se establece EFCB = 1.	0	ACK.	UnReset	1
	X	1		No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	UnReset	2
	X	X	1, 5 .. 8 10 .. 15	No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	UnReset	3
	X	0	2	No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	UnReset	4
	X	1		No se realiza ninguna acción.	1	No es posible transmitir nada de manera opcional.	UnReset	5
	X	0	3	No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	UnReset	6
	X	1		No se realiza ninguna acción.	1	No es posible transmitir nada de manera opcional.	UnReset	7
	X	0	4	Se envía una solicitud a la aplicación de análisis de rutina.	No se transmite nada.		UnReset	8
	X	1		No se realiza ninguna acción.			UnReset	9
	X	0	9	No se realiza ninguna acción.	11	El estado del enlace	UnReset	10
	X	1		No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	UnReset	11
Ocioso	X	0	0	A la variable EFCB se le asigna el valor de 1.	0	ACK.	Ocioso	12
	X	1		No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	Ocioso	13
	X	X	1, 5 .. 8 10 .. 15	No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	Ocioso	14
	X	0	2	No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	Ocioso	15
	== EFCB	1		Se activa la variable EFCB con 1	0	ACK.	Ocioso	16
	!= EFCB			No se realiza ninguna acción.	-	Se retransmite la puesta más reciente que contiene código de función 0 (ACK) o 1 (NACK).	Ocioso	17
	X	0	3	No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	Ocioso	18
	== EFCB	1		Se envía una solicitud a la aplicación de análisis de rutina y EFCB=1.	0	ACK.	Ocioso	19
	!= EFCB			No se realiza ninguna acción.	0	ACK.	Ocioso	20
	X	0	4	Se envía una solicitud a la aplicación de análisis de rutina.	No se transmite nada.		Ocioso	21
	X	1		No se realiza ninguna acción.			Ocioso	22
	X	0	9	No se realiza ninguna acción.	11	El estado del enlace.	Ocioso	23
	X	1		No se realiza ninguna acción.	15	No es posible transmitir nada de manera opcional.	Ocioso	24

3. Especificación del protocolo DNP3 con SDL

En este capítulo se presentan los resultados del trabajo de tesis, obtenidos durante el desarrollo de la especificación del protocolo de comunicaciones DNP3 empleando SDL [9,10]. Se definen los requerimientos y el alcance del sistema, además se realiza una descripción de los tipos de datos y los objetos definidos en las especificaciones, estática y dinámica, del sistema final.

Para fines del modelado del sistema, a partir de ahora se hará referencia a la Capa de Aplicación como AL, a la Función de Transporte como TF y a la Capa de Enlace de Datos como DLL.

3.1. Requerimientos del sistema

Con base en las características del protocolo de comunicaciones DNP3 (Capítulo 2) se definen las siguientes restricciones respecto a su especificación:

- Este trabajo se centrará en la especificación de la DLL y de la TF.
- El sistema estará compuesto por una EM y una ER.
- Debido a que el medio físico no está definido en la especificación del protocolo de comunicaciones DNP3, se realizará una especificación mínima para poder realizar la ejecución de la especificación.
- Se realizará una especificación mínima para el búfer de recepción y el búfer de transmisión de mensajes (datos).

3.2. Objetos de la especificación

A continuación se describen los objetos creados para el desarrollo de la especificación del sistema, entre ellos los bloques, sub-bloques y procesos que constituyen la especificación estática del protocolo DNP3. Los objetos se presentan en orden descendente (arquitectura del sistema, arquitectura de bloque y nivel de procesos), definiendo el diseño arquitectural de la metodología de diseño. La Figura 3.1 muestra los niveles jerárquicos mediante un diagrama de árbol.

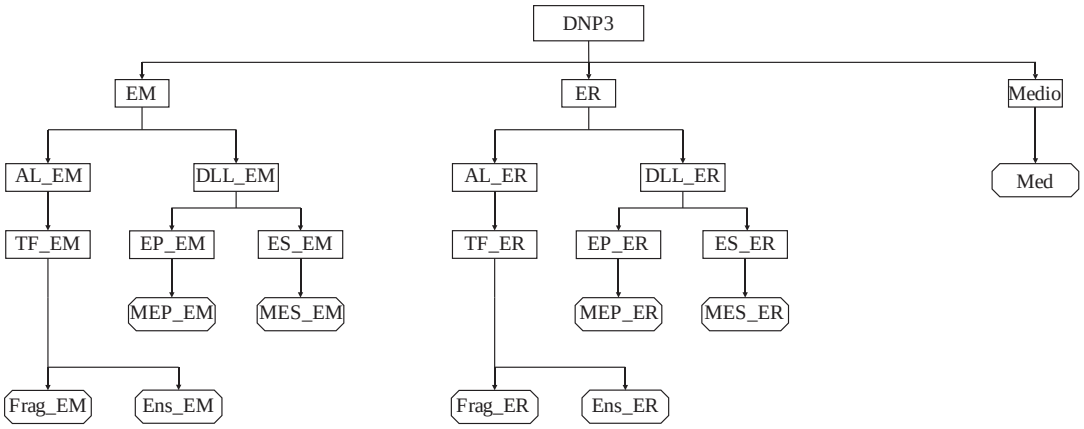


Figura 3.1. Jerarquía de los objetos de la especificación[11].

En la Tabla 3.1 se resumen los objetos SDL del sistema, así como los bloques, procesos y una descripción de las acciones que realizan.

Tabla 3.1. Objetos SDL del sistema.

Objeto SDL				**Descripción**
Bloques			**Procesos**	
Medio			Med	Realiza las funciones del medio físico.
EM	AL_EM	TF_EM	Frag_EM	Proceso de la máquina de estados de la TF que fragmenta los segmentos en la EM.
			Ens_EM	Proceso de la máquina de estados de la TF que reensamblar los segmentos en la EM.
	DLL_EM	EP_EM	MEP_EM	Proceso de la máquina de estados del DLL que transmite las tramas a la ES de la ER.
		ES_EM	MES_EM	Proceso de la máquina de estados del DLL que recibe las tramas de la EP de la ER.
ER	AL_ER	TF_ER	Frag_ER	Proceso de la máquina de estados de la TF que fragmenta los segmentos en la EM.
			Ens_ER	Proceso de la máquina de estados de la TF que reensamblar los segmentos en la EM.
	DLL_ER	EP_ER	MEP_ER	Proceso de la máquina de estados del DLL que transmite las tramas a la ES de la EM.
		ES_ER	MES_ER	Proceso de la máquina de estados del DLL que recibe las tramas de la EP de la EM.

[11] Lista de acrónimos empleada en la especificación de objetos SDL:

EM: Estación Maestra
AL_EM: AL de la EM
DLL_EM: DLL de la EM
EP_EM: EP de la EM
ES_EM: ES de la EM
Frag_EM: Proceso de fragmentación de la EM
Ens_EM: Proceso de ensamble de la EM
MEP_EM: Máquina de estados de la EP de la EM
MES_EM: Máquina de estados de la ES de la EM
ER: Estación Remota
AL_ER: AL de la ER
DLL_ER: DLL de la ER
EP_ER: EP de la ER
ES_ER: ES de la ER
Frag_ER: Proceso de fragmentación de la ER
Ens_ER: Proceso de ensamble de la ER
MEP_ER: Máquina de estados de la EP de la ER
MES_ER: Máquina de estados de la ES de la ER
Medio: Capa física
M1: Proceso de la Capa Física para el envío de datos

El sistema está constituido por los siguientes elementos, representados por objetos SDL:

- Un medio físico que representa una topología Maestro/Esclavo.
- Una ER que realiza las funciones de cualquier dispositivo electrónico inteligente (IED).
- Una EM que realiza el envío de solicitudes a la ES.
- Cada estación con su respectiva configuración de acuerdo al protocolo de comunicaciones DNP3.

3.3. Especificación de datos

Los tipos de datos [7] usados para la especificación del protocolo DNP3 se han declarado dentro de la librería (*package*) DNP3_Datos en el nivel de entorno del sistema, lo cual permite que sean accesibles en los demás niveles de abstracción, además que se utilizó la norma ASN.1 [8].

3.3.1. Tipos de datos relacionados con el control del protocolo DNP3

Los datos que controlan el protocolo están divididos en dos grupos principales, los que controlan la capa DLL y los que controlan la TF.

3.3.1.1. Tipos de datos para el procesamiento de tramas

Estos tipos de datos (Figura 3.2) permiten la creación y manipulación de las tramas y de las estructuras que las generan. Los siguientes tipos de datos son la base del protocolo DNP3 ya que con estos se permite crear la mayoría de los datos empleados:

- Byte: Este tipo de datos crea una cadena de 8 bits para la creación de los datos del protocolo.
- Bit_t: Este tipo de datos crea un arreglo de un bit para identificar el campo de control de cada capa del protocolo.
- Bit_4: Este tipo de datos crea un arreglo de 4 bits para los códigos de función del protocolo.
- Reintentos: Este tipo de dato se emplea para crear los datos relacionados con los reintentos que realizan las EP en la transmisión de la información.
- TCRC: Este tipo de datos crea un arreglo de 16 bits para almacenar el CRC de cada trama.
- KEY: Este tipo de datos crea un arreglo de 17 bits para el polinomio generador del CRC de la trama.

3.3.1.2. Datos para el control del DLL

Los tipos de datos que se muestran en la Figura 3.3 generan la estructura de las cabeceras del DLL, así como las estructuras de los códigos de control de las EP y ES:

- C_EP: Esta estructura contiene los datos del código de control de la EP, el cual está compuesto por: DIR_EP, PRM_EP, FCB, FCV y Cfun_EP.
- C_ES: Esta estructura contiene los datos del código de control de la EP, el cual está compuesto por: DIR_ER, PRM_ER, DFC, cero y Cfun_ER.
- H_DLL: Esta estructura contiene los datos que generan la cabecera del DLL, la cual está compuesta por: LSB_INI, MSB_IN, LEN, CTRL, LSB_Dest, MSB_Dest, LSB_Fuent, MSB_ Fuent.

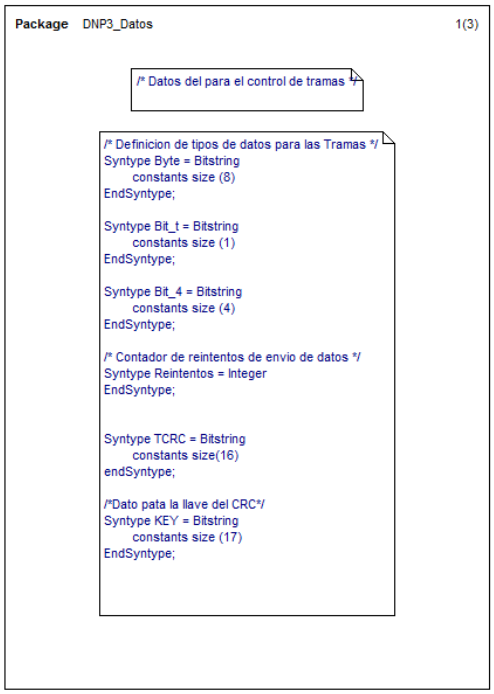


Figura 3.2. Tipos de datos para el procesamiento de tramas.

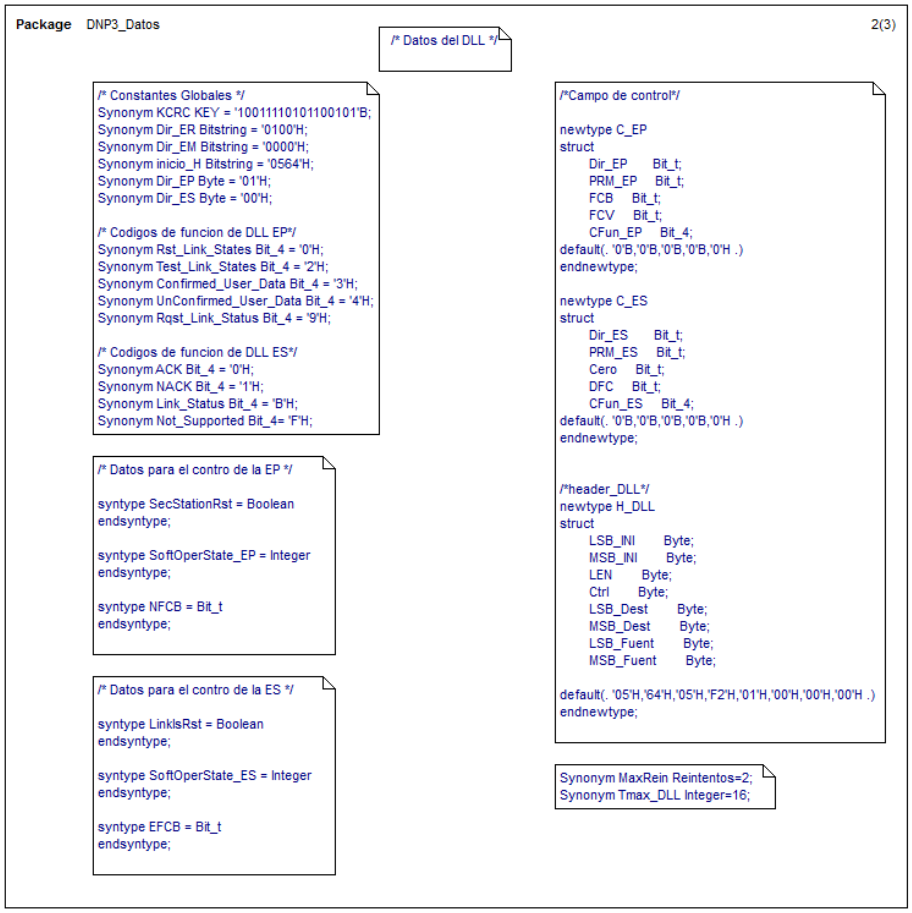


Figura 3.3. Tipos de datos para el control del DLL.

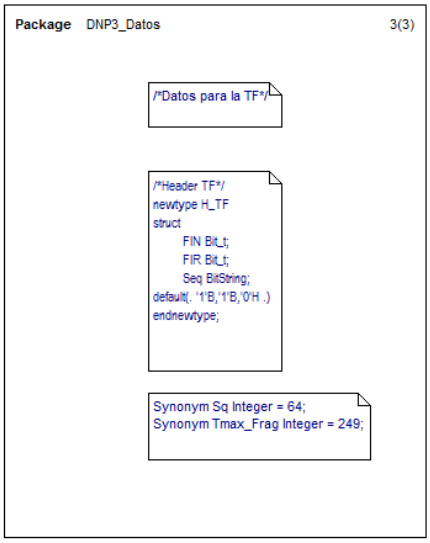


Figura 3.4. Dato de control de la TF.

3.3.1.3. Datos para el control de la TF

Los datos empleados en la Figura 3.4 definen el control de la TF, principalmente H_TF que contiene los datos que generan la cabecera de la TF, la cual está conformada por FIN, FIR y Seq.

3.3.2. Tipos de datos para la especificación del tiempo

Los datos para la especificación del tiempo (Figura 3.5) definen los retardos o el tiempo de espera de la respuesta de una trama. Estos tipos de datos son:

- uT: Este tipo de dato especifica una duración del tiempo en microsegundos y se emplea en el retardo generado para el proceso del medio físico.
- mT: Este tipo de dato especifica una duración del tiempo en milisegundos y se emplea para el tiempo de espera de la repuesta de una trama.

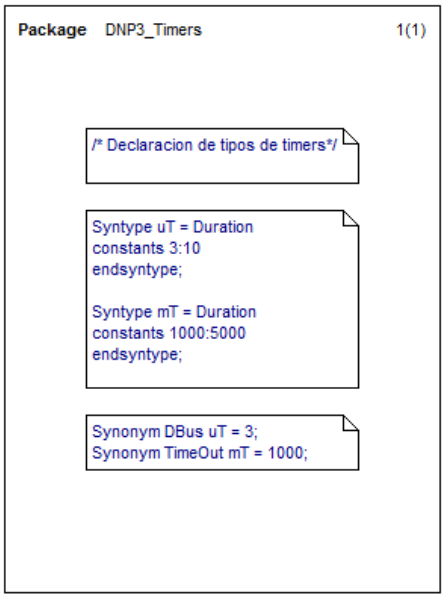


Figura 3.5. Datos para la especificación del tiempo.

3.4. Especificación estática

Los objetos identificados para el desarrollo del protocolo de comunicaciones DNP3 son utilizados para construir la especificación estática del sistema (Tabla 3.1). Como se muestra en la Figura 3.2, esta especificación está dividida en varios niveles de abstracción, con lo cual se define la arquitectura del sistema mediante la distribución de los objetos que la componen y el flujo de información entre ellos empleando señales.

3.4.1. Especificación de canales, rutas de señal y compuertas

La especificación de los objetos SDL se puede realizar utilizando una representación remota basada en tipos (*type*) o representaciones remotas basadas en señales específicas. Para el desarrollo de los objetos del sistema se utilizan especificaciones remotas basadas en señales específicas, excepto el proceso M1. 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 el intercambio de señales; esto se lleva a cabo mediante la especificación de compuertas.

3.4.1.1. Especificación de señales

Las señales son la base para establecer la comunicación entre las entidades del sistema y se transportan a través de canales o rutas de señal especificadas. La Figura 3.6 muestra las señales que se emplean para la especificación del protocolo. Todas las señales son parametrizadas[12], es decir, transportan información necesaria para procesamiento o control del protocolo.

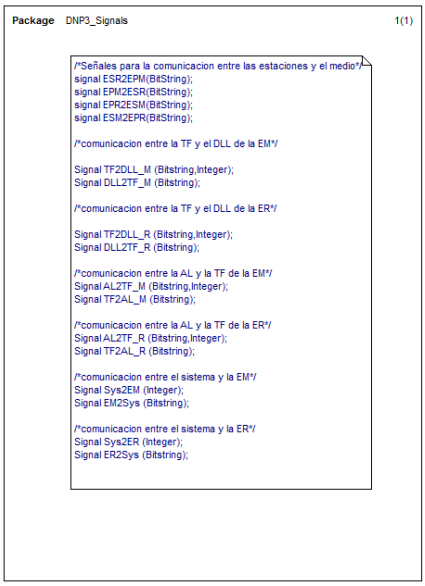


Figura 3.6. Declaración de las señales del sistema.

[12] El término parametrización se puede definir como determinar el modo de ejecución de una acción considerando la configuración de las variables o señales.

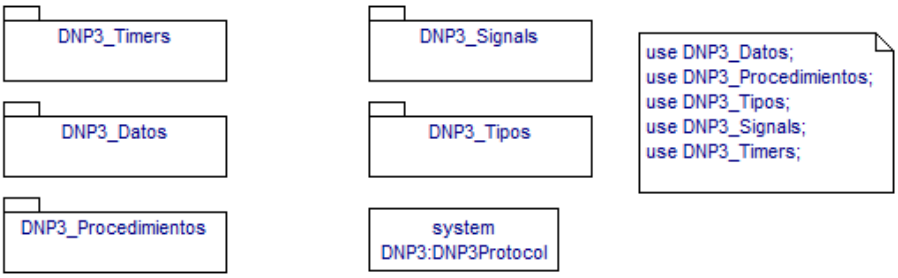


Figura 3.7. Librerías de la especificación del protocolo DNP3.

3.4.2. Sistema DNP3

Los tipos de datos, constantes de tiempo, señales y procedimientos se declaran en librerías (Figura 3.7). El sistema consiste en una instancia del tipo DNP3Protocol, el cual representa el menor nivel de abstracción. La declaración del sistema DNP3Protocol está conformado por los bloques EM y ER, y por una instancia al bloque tipo M1 (Figura 3.8). La Figura 3.9 muestra dichos bloques a nivel de sistema.

3.4.2.1. Bloque medio

El bloque medio representa el medio físico para la transmisión de datos empleando una topología Maestro/Esclavo, mediante una conexión punto a punto, con lo que se establece un medio físico para la comunicación entre la EM y la ER.

El bloque medio se compone de un único proceso que es una instancia de tipo P_Medio:M1, este proceso sólo realiza el paso de las señales mediante las rutas de señal 2gm1, 2gm2, 2gm3 y 2gm4 que se encargan de enviar la información de manera correcta a los dispositivos correspondientes. Con el modelado del medio se asegura que puedan conectarse más de una ER al medio (con fines de un desarrollo futuro).

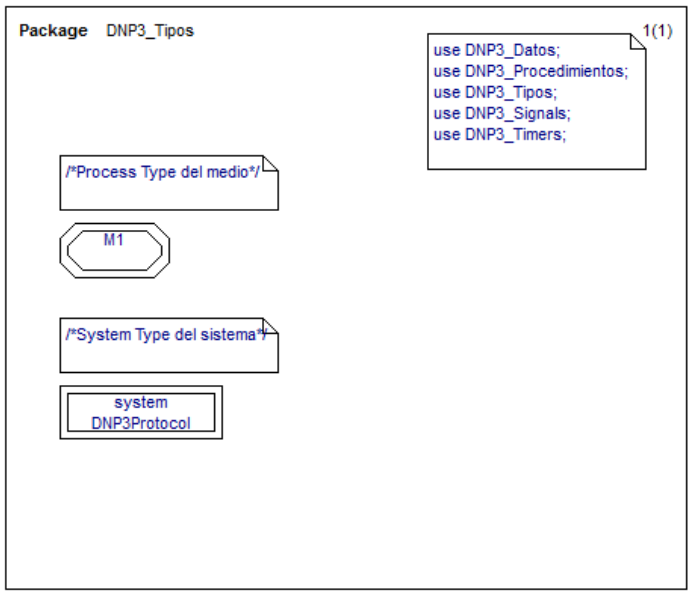


Figura 3.8. Definición de los objetos en SDL.

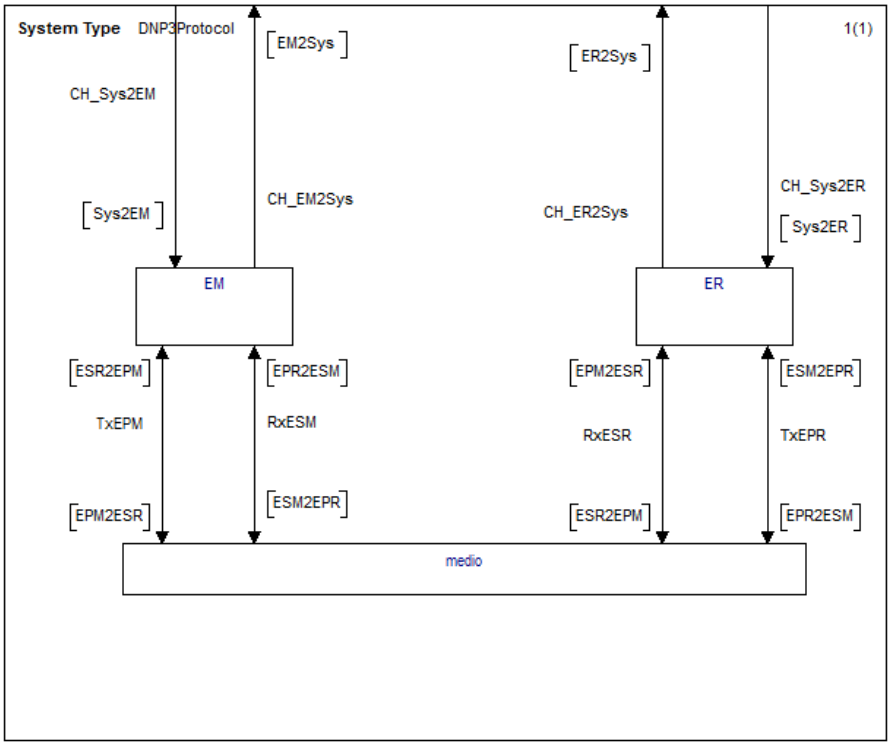


Figura 3.9. Especificación del sistema DNP3.

3.4.3. Bloque EM

El bloque EM define la arquitectura de una EM (Figura 3.11), con las restricciones descritas en el subcapítulo 3.1. El bloque EM está compuesto por los bloques AL_EM y DLL_EM, los cuales representan a la AL y a la DLL de la EM respectivamente.

Este bloque se encarga de generar y enviar solicitudes al bloque ER de acuerdo a lo solicitado por el usuario, el cual procesa y realiza las operaciones solicitadas, es decir es la interfaz entre el usuario y la EM.

Dado que DNP3 es un protocolo basado en la topología Maestro/Esclavo, el usuario envía la función a realizar por medio de la señal Sys2EM al bloque EM, tomando en cuenta los códigos de función descritos en la Tabla 2.3.

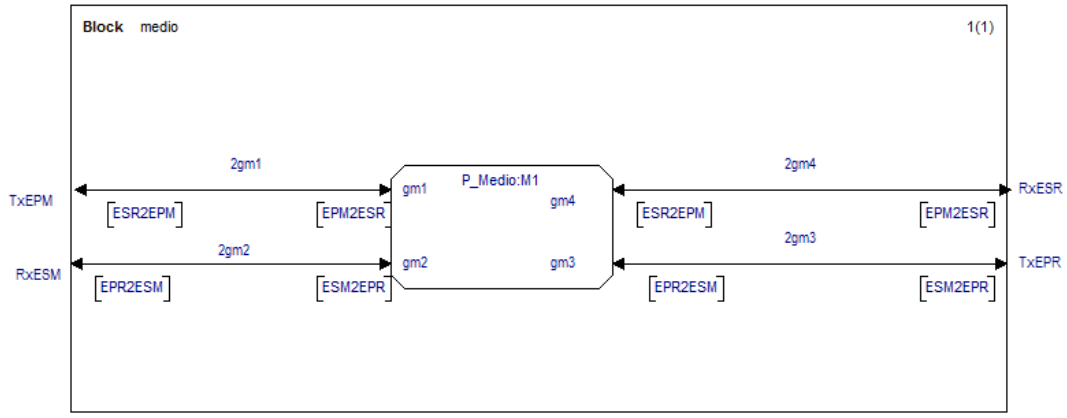


Figura 3.10. Diagrama SDL del bloque medio.

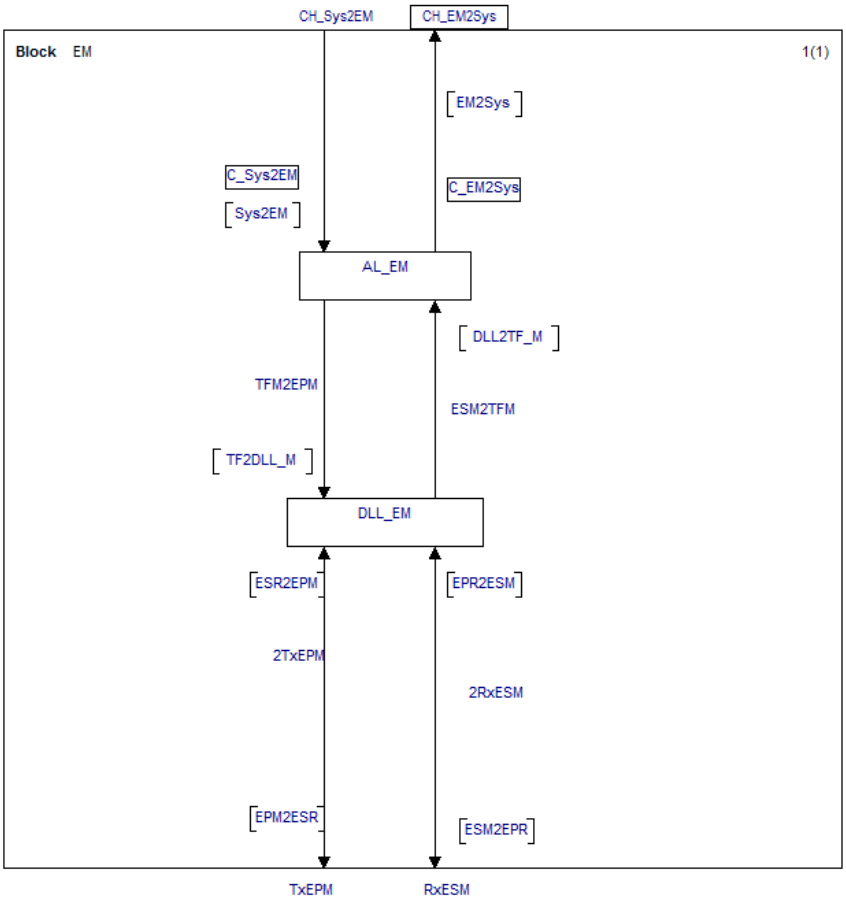


Figura 3.11. Diagrama SDL del bloque EM.

3.4.3.1. Bloque AL_EM

El bloque AL_EM define la arquitectura SDL de la AL de la EM y está compuesta por los procesos P1_M y P2_M, los cuales representan las máquinas de estado de la AL (procesos definidos para especificaciones futuras) además del bloque TF_EM que define la TF de la EM (Figura 3.12).

Este bloque se definió de acuerdo a lo especificado en el subcapítulo 2.2, en donde se describe la relación entre la AL y la TF.

3.4.3.1.1. Bloque TF_EM

La Figura 3.13 muestra la arquitectura SDL de la TF, la cual está compuesta por los procesos Frag_EM y Ens_EM, que tienen la función de dividir los segmentos de la AL en fragmentos y de ensamblar los mismos para regenerar los segmentos, respectivamente. Aunque en el subcapítulo 2.2 no se realiza una definición concreta de la máquina de estados para proceso de fragmentación, se realizó la implementación del proceso Frag_EM que realiza dicha operación considerando las restricciones de tamaño máximo de los fragmentos (250 bytes) de la FT.

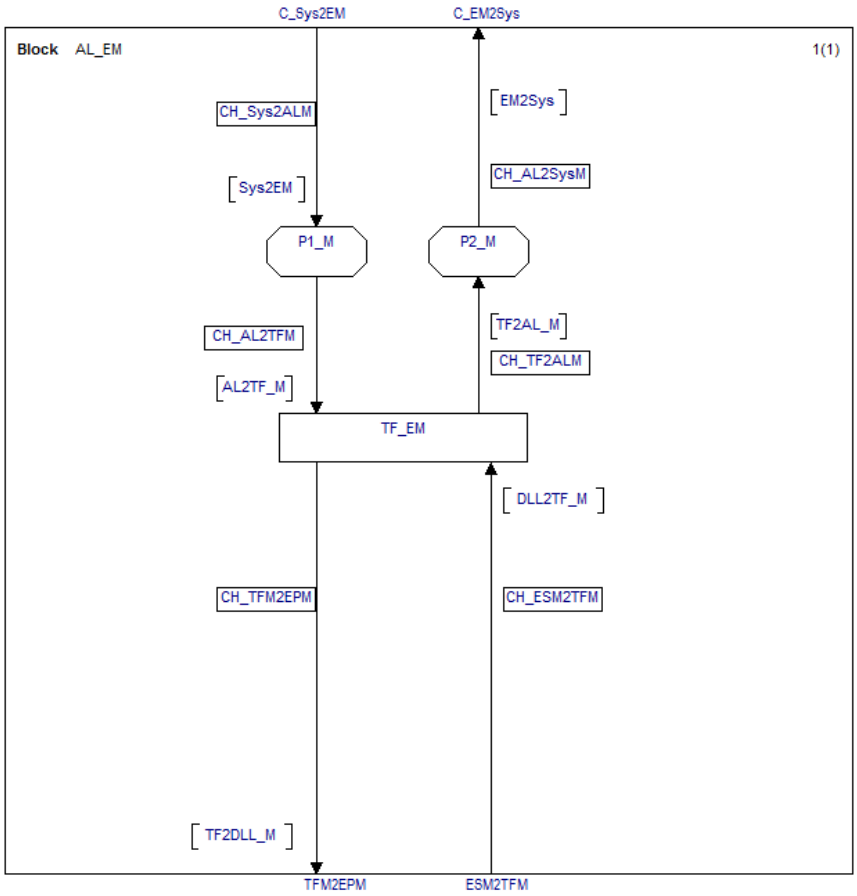


Figura 3.12. Diagrama SDL del bloque AL_EM.

3.4.3.2. Bloque DLL_EM

La Figura 3.14 muestra la arquitectura SDL de la DLL, en donde el bloque DLL_EM está compuesto por los bloques EP_EM y ES_EM, los cuales representan a la EP y la ES de la EM. Estas estaciones se encargan del envío y recepción de los datos provenientes de las capas superiores o confirmaciones de otros dispositivos (EP) así como de la recepción de respuestas no solicitadas (ES) por parte de las ER, también este bloque interactúa directamente con el medio físico por medio de las señales.

3.4.3.2.1. Bloque EP_EM

La Figura 3.15 describe la arquitectura SDL del bloque EP_EM, que representa la EP de la EM descrito en el apartado 2.3.1; este bloque está conformado por el proceso MEP_EM que realiza la máquina de estados de esta estación, además este bloque es el que se encarga de interactuar directamente con el medio físico del sistema.

3.4.3.2.2. Bloque ES_EM

La Figura 3.16 describe la arquitectura SDL del bloque ES_EM, que representa la ES de la EM descrito en el apartado 2.3.1; este bloque está conformado por el proceso MES_EM con el cual se realiza la máquina de estados de dicha estación.

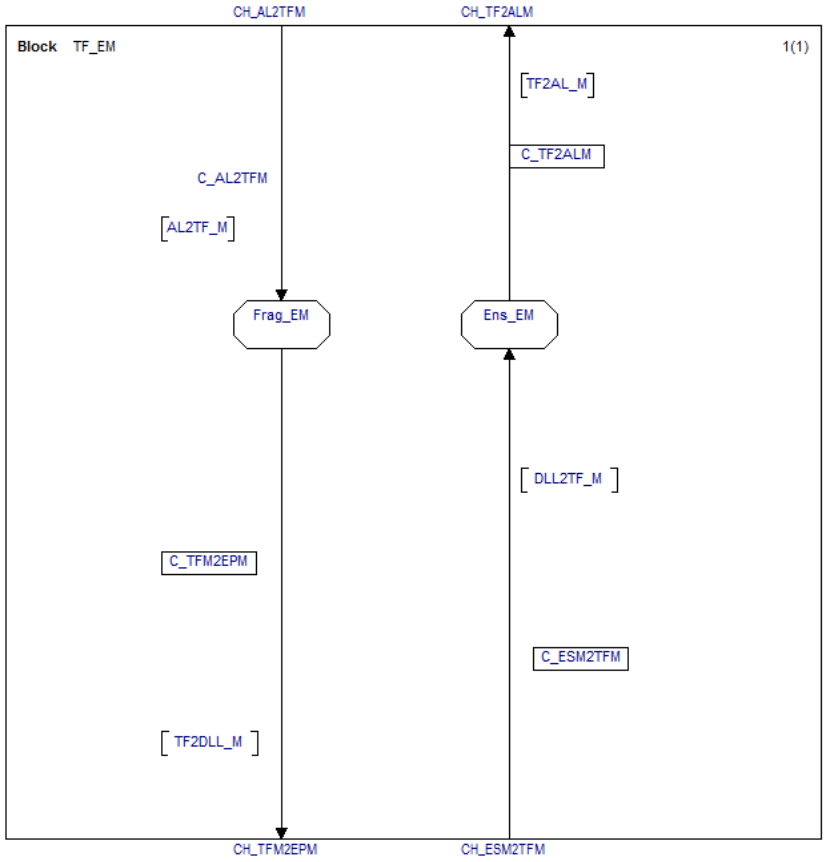


Figura 3.13. Diagrama SDL del bloque TF_EM.

3.4.4. Bloque ER

El bloque ER se diseñó de la misma forma que el bloque EM, y por ello sus bloques y sus procesos están conformados de la misma manera, también tienen un nombre muy similar, sin embargo la diferencia ocurre en la parte dinámica de las estaciones (subcapitulo 3.5) y en las señales que reciben.

3.5. Especificación dinámica

La especificación dinámica describe el comportamiento del sistema y se encuentra dentro de los procesos que conforman la especificación. Dado que la funcionalidad de los procesos ya se ha definido en la especificación estática, a continuación se detalla el funcionamiento interno basado en máquinas de estado.

Debido a que la especificación de los procesos es muy extensa, sobre todo en los procesos MEP_EM, MES_EM, MEP_ER, MES_ER, Ens_EM y Ens_ER, 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 señales y la 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 que corresponden a los estados y a las señales de entrada.

Para una revisión exhaustiva de la especificación formal de DNP3, se entregan los siguientes documentos conjuntamente con este trabajo de tesis:

- *Especificación formal en formato cbf*: Formato de la especificación generada por la herramienta Cinderella SDL.
- *Especificación formal en formato pdf*: Para que los lectores que no disponen de la herramienta Cinderella SDL, puedan visualizar la especificación formal en el Anexo A.
- *Diagrama MSC en formato pdf*: Los diagramas MSCs (*Message Sequence Chart*) representan el resultado de la simulación como una secuencia de mensajes ordenada en el tiempo, los cuales se abordan en el capítulo 4 para documentar la evaluación de resultados.

3.5.1. Proceso P1_M

Una vez que el proceso P1_M (Figura 3.17) ejecuta el símbolo de inicio, se procede a enviar la configuración inicial hacia el bloque TF_EM mediante la señal AL2Tf_M, con esto la EM inicializa los parámetros para establecer la comunicación con la ER. Posteriormente pasa al estado ocioso en espera de recibir el código de función (Tabla 2.3) por parte del usuario mediante la señal Sys2EM. Una vez recibida la señal con la solicitud del usuario (código de función), se envía al bloque TF_EM junto con el valor de la variable D_AL (con fines de funcionalidad).

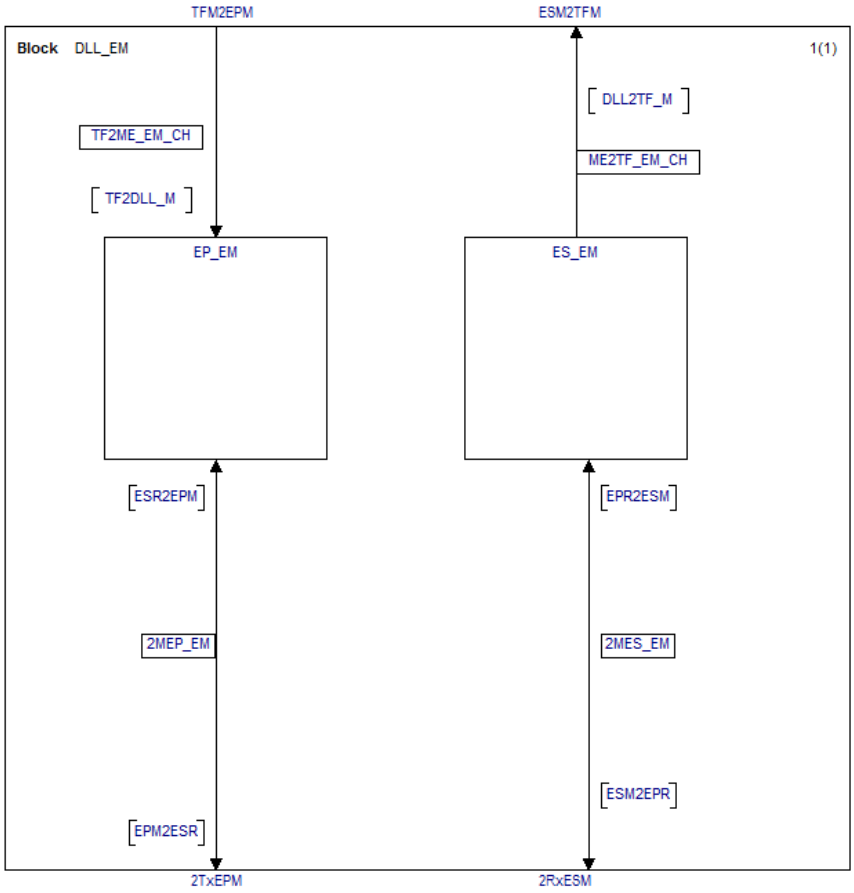


Figura 3.14. Diagrama SDL del bloque DLL_EM.

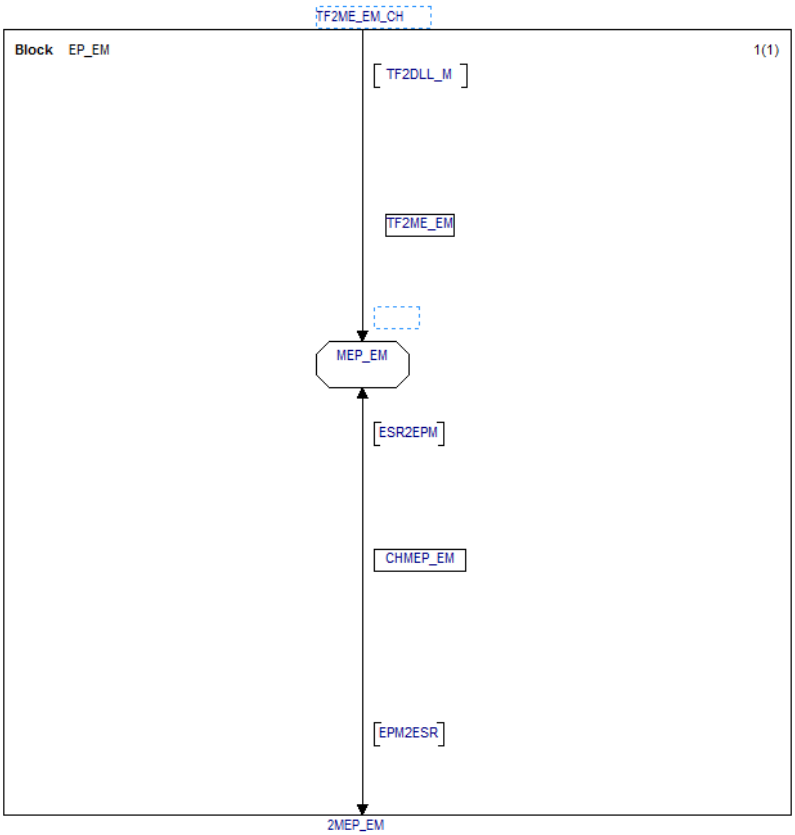


Figura 3.15. Diagrama SDL del bloque EP_EM.

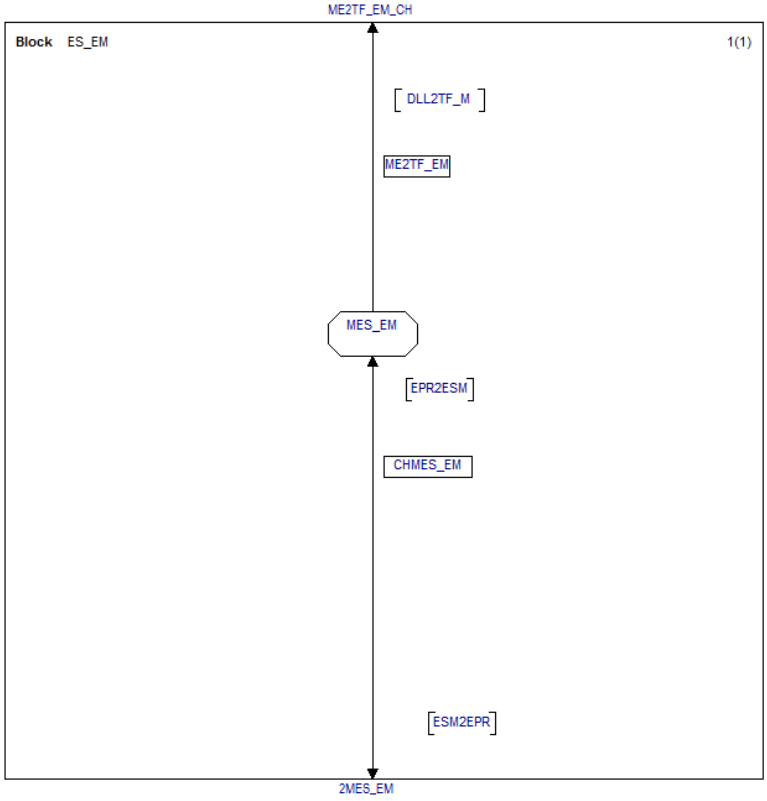


Figura 3.16. Diagrama SDL del bloque ES_EM.

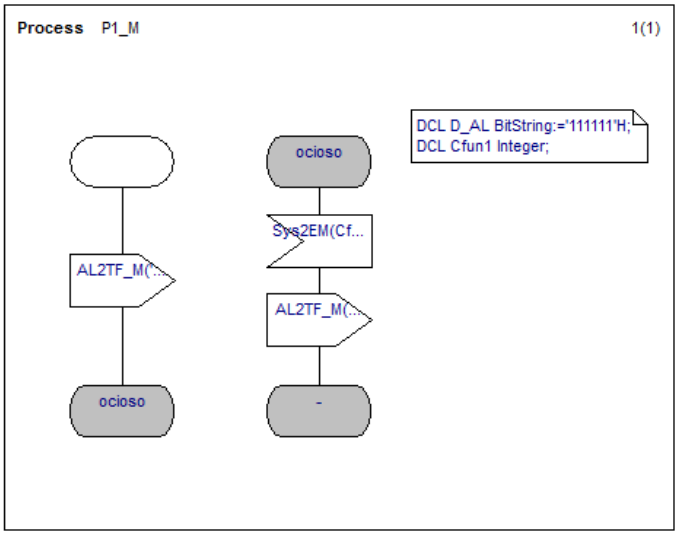


Figura 3.17. Descripción del proceso P1_M.

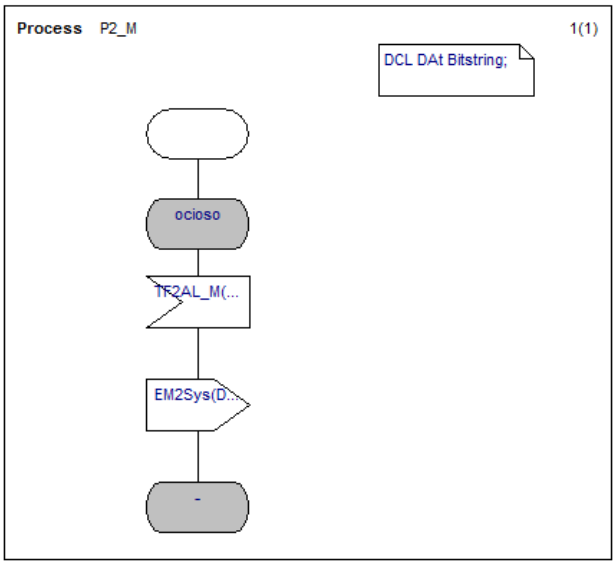


Figura 3.18. Descripción del proceso P2_M.

3.5.2. Proceso P2_M

La Figura 3.18 muestra el proceso P2_M; una vez que se ejecuta el símbolo de inicio, el proceso pasa al estado ocioso en espera de recibir los datos de alguna solicitud de la AL o alguna respuesta no solicitada por parte de alguna ER. Para fines de funcionalidad del protocolo sólo se pasan al usuario los datos recibidos.

3.5.3. Proceso Frag_EM

Una vez que se ejecuta el símbolo de inicio en el proceso Frag_EM (Figura 3.19), se llega al estado Fragmentar que se encuentra en espera de la recepción de datos provenientes del proceso P1_M de la EM. Este proceso se encarga de acondicionar la información, es decir, separa los segmentos de la máquina de estados de la AL en bloques de 249 bytes (en caso de ser necesario), posteriormente se agregan los datos del byte de la cabecera de la TF (Figura 2.22) para generar los fragmentos. A pesar de que esta máquina de estados no está definida en la especificación del

protocolo de comunicaciones DNP3, se implementó de acuerdo a lo descrito en el apartado 2.2.2; este proceso además de enviar los fragmentos también envía el código de función para el correcto funcionamiento de la DLL.

3.5.4. Proceso Ens_EM

Al igual que el proceso anterior, el proceso Ens_EM (Figura 3.20) necesita ejecutar el símbolo de inicio para posteriormente pasar al estado ocioso, este estado está a la espera de recibir información por parte de la DLL de la EM. De acuerdo a los campos de la cabecera de la TF recibida (FIR, FIN y SEQ), se realiza el ensamble de los segmentos como se indica en la Figura 2.24 y en la Tabla 2.14, para ser enviados al proceso P2_M (Figura 3.18) perteneciente a la AL de la EM.

3.5.5. Proceso MEP_EM

El proceso descrito en la Figura 3.21 es uno de los procesos principales del desarrollo de la especificación del protocolo, describe la máquina de estados y la tabla de estados del inciso 2.3.7.1; cabe señalar que este proceso fue diseñado para la comunicación entre una EM y una ER.

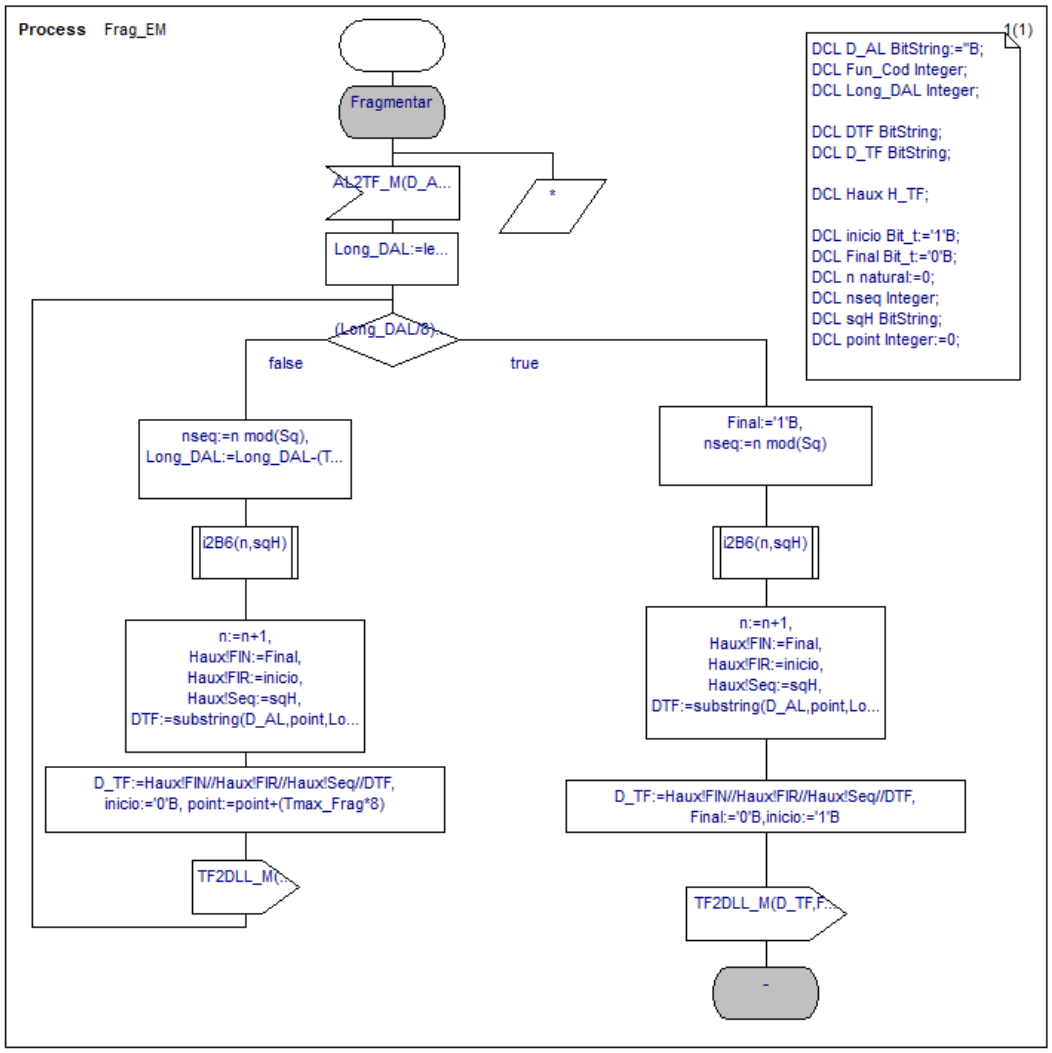


Figura 3.19. Descripción del proceso Frag_EM.

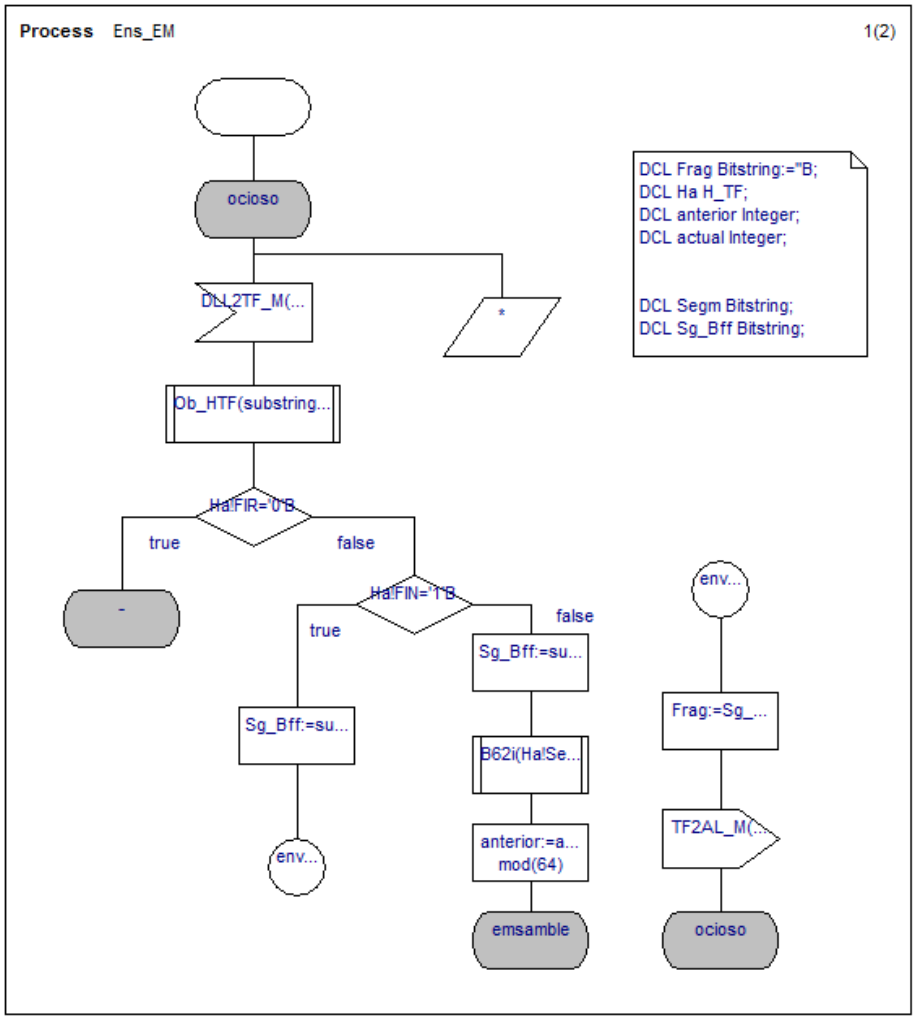


Figura 3.20. Descripción del proceso Ens_EM.

Una vez que se ejecuta el símbolo de inicio se traslada al estado SecUnResetIdle, que se encuentra a la espera de recibir información por parte de la TF mediante la señal TF2DLL_M; se realiza la acción de acuerdo al código de función elegido por el usuario (excepto en el inicio o reinicio de la EM). Si se recibe algún código de función en que se requiere de una respuesta por parte de la ER se activa el contador de reintentos y el temporizador de espera para la respuesta de la DLL; una vez realizados los pasos anteriores se procede a conformar la trama con la llamada a los procedimientos Gen_Trm_H (genera tramas para las cabeceras de la DLL de la EM) o Gen_Trm_D (genera la trama de los datos provenientes de la TF de la EM) (apartado 3.5.14). Finalmente, se envía la trama al bloque medio mediante la señal EPM2ESR y de acuerdo al código de función recibido se realiza la transición de estado; si es necesario se puede esperar la respuesta de la ER y realizar otro cambio de estado como ocurre en la mayoría de los estados, excepto en los estados SecUnResetIdle y SecResetIdle. Los códigos de función que puede transmitir la EM se encuentran en la Tabla 2.15, estos se codifican en cadenas de bits para su mejor manejo por parte de las estaciones y de los procedimientos, implícitamente en el proceso se realiza la separación de los datos provenientes de la TF en datos de 16 bytes para crear el CRC de cada trama.

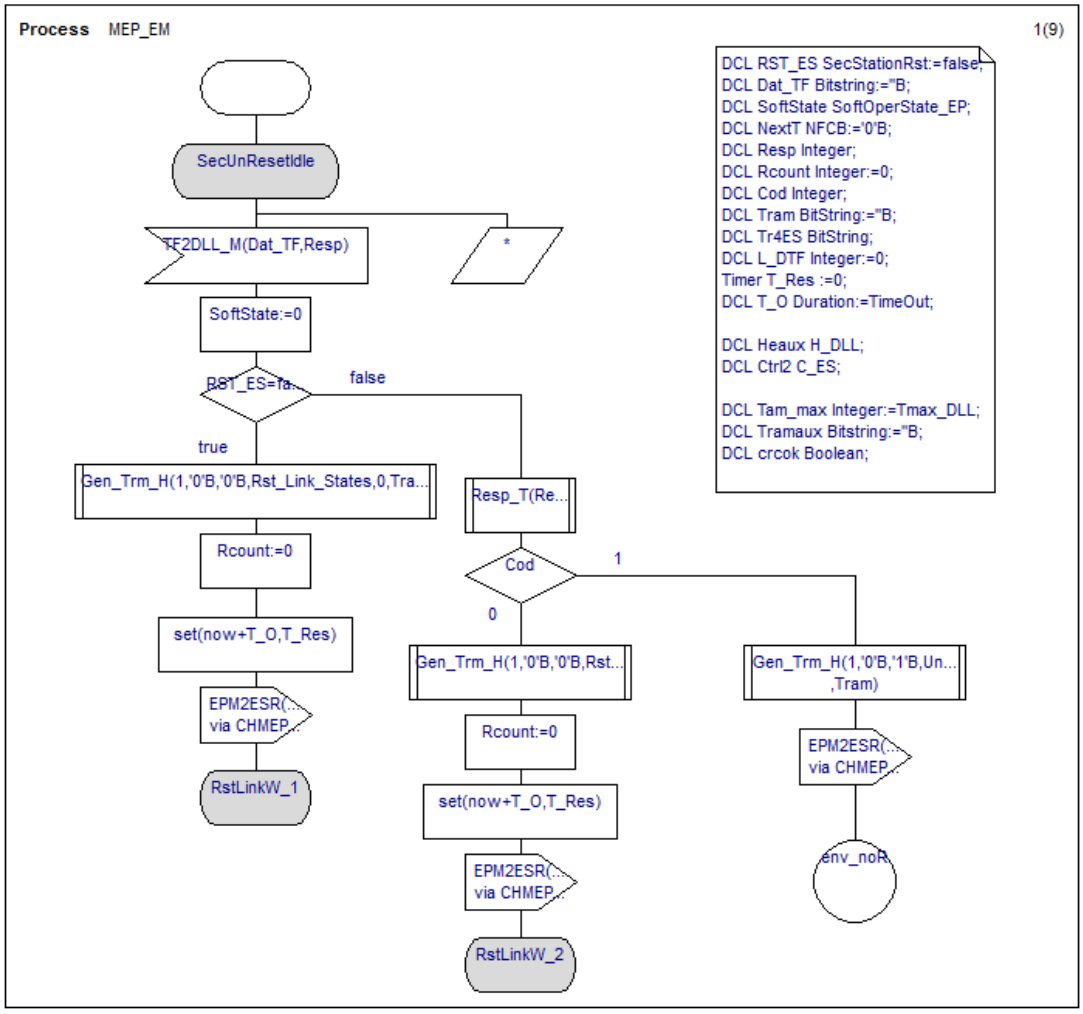


Figura 3.21. Descripción del proceso MEP_EM.

3.5.6. Proceso MES_EM

Al igual que el proceso anterior, el proceso MES_EM (Figura 3.22) define el comportamiento de la ES (Tabla 2.21 y Figura 2.34) perteneciente a la DLL de la EM.

Una vez ejecutado el símbolo de inicio se procede a realizar la transición al estado Ocioso_M, el cual está a la espera de recibir alguna señal proveniente de la EP de la ER (respuestas no solicitadas o respuestas a peticiones por parte de la EM); una vez recibida la señal se obtiene la cabecera de la DLL para corroborar la dirección destino de la trama; en caso de que no sea la dirección destino la misma que la del dispositivo, se desecha la trama; posteriormente se verifica la trama ocupando el proceso inverso al CRC, si éste es correcto se verifica el código de función de la cabecera para realizar la solicitud correspondiente y se envía una trama de acuerdo a la solicitud realizada, de lo contrario se envía una trama de respuesta a la EP con el código de función NACK (las posibles respuestas de la ES se describen en la Tabla 2.16). Para este proceso se incluyeron dos estados que no están especificados en la descripción del protocolo de comunicaciones DNP3 y se encargan de actuar como un búfer en el caso de múltiples tramas (fragmentos mayores a 16 bytes). Dichos estados son: Dat2TF (Datos recibidos con el código de función Confirmed_User_Data) y Dat2TF_NR (Datos recibidos con el código de función UnConfirmed_User_Data).

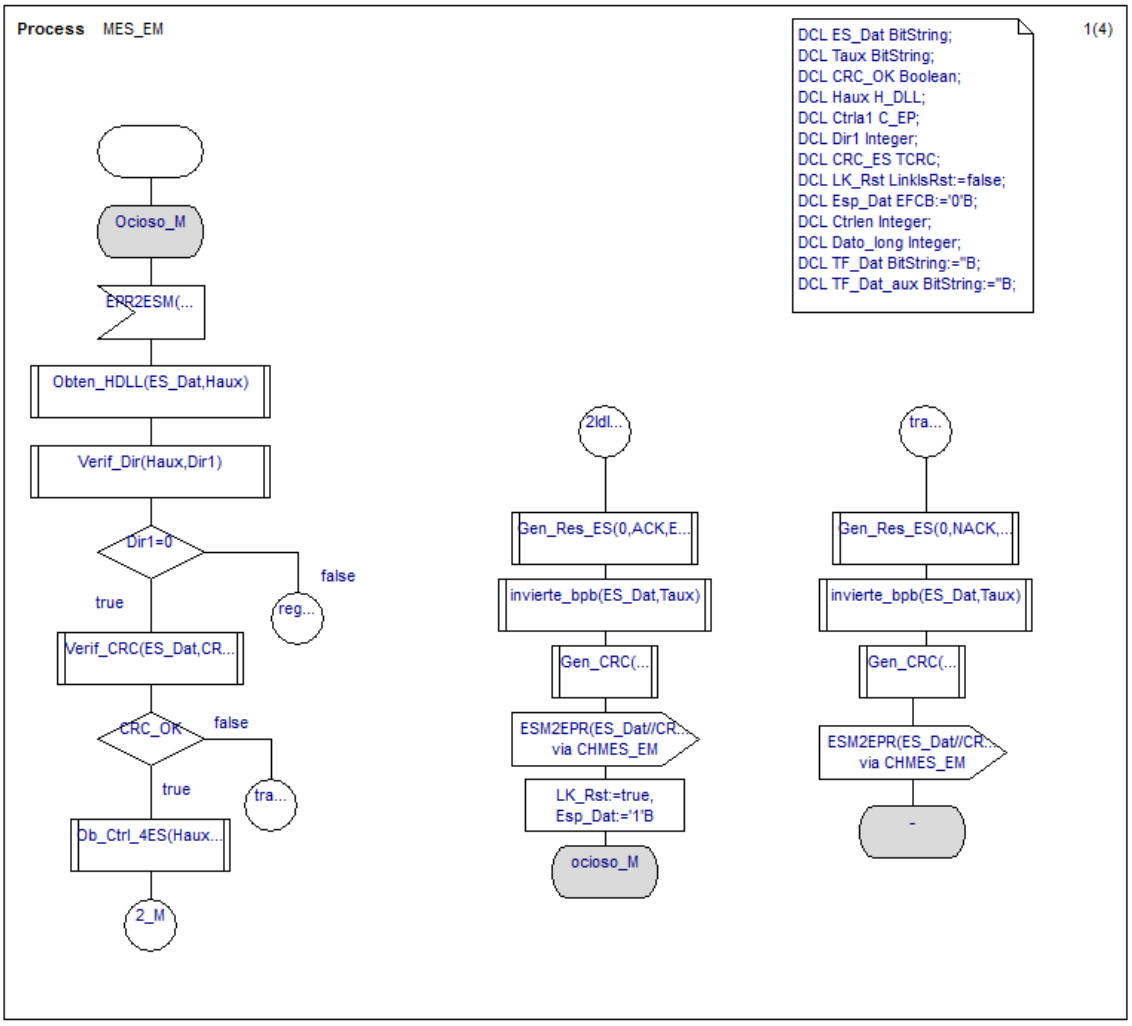


Figura 3.22. Descripción del proceso MES_EM.

3.5.7. Proceso tipo M1

La Figura 3.23 muestra la descripción del proceso M1; una vez ejecutado el símbolo de inicio se pasa al estado recibe, el cual está a la espera de recibir algún dato por parte de la EM o de la ER mediante las señales EPM2ESR, ESR2EPM, EPR2ESM y ESM2EPR (definidas de acuerdo a lo descrito en el apartado 2.3.1), cuando esto ocurre se habilita el temporizador y se realiza la transición al estado correspondiente a la señal recibida (espera1, espera2, espera3 y espera4), al estar en cualquier estado se espera el envío de la señal por parte del temporizador habilitado (TBus) y se envían los datos recibidos para regresar al estado recibe.

3.5.8. Proceso MEP_ER

El proceso MEP_ER (Figura 3.24) es similar al descrito en el apartado 3.5.5 dado que se trató de estandarizar los bloques en las estaciones, la diferencia ocurre principalmente en la dirección física del dispositivo, en las señales con las que interactúa (TF2DLL_R y EPR2ESM) y la función real del dispositivo (enviar respuestas solicitadas y no solicitadas), también cumple con las características necesarias para el envío de datos proceso MES_EM (Figura 3.16) de la EM que ésta no necesita de la configuración inicial para establecer la comunicación por los eventos de mayor importancia que realiza la EP de la ER.

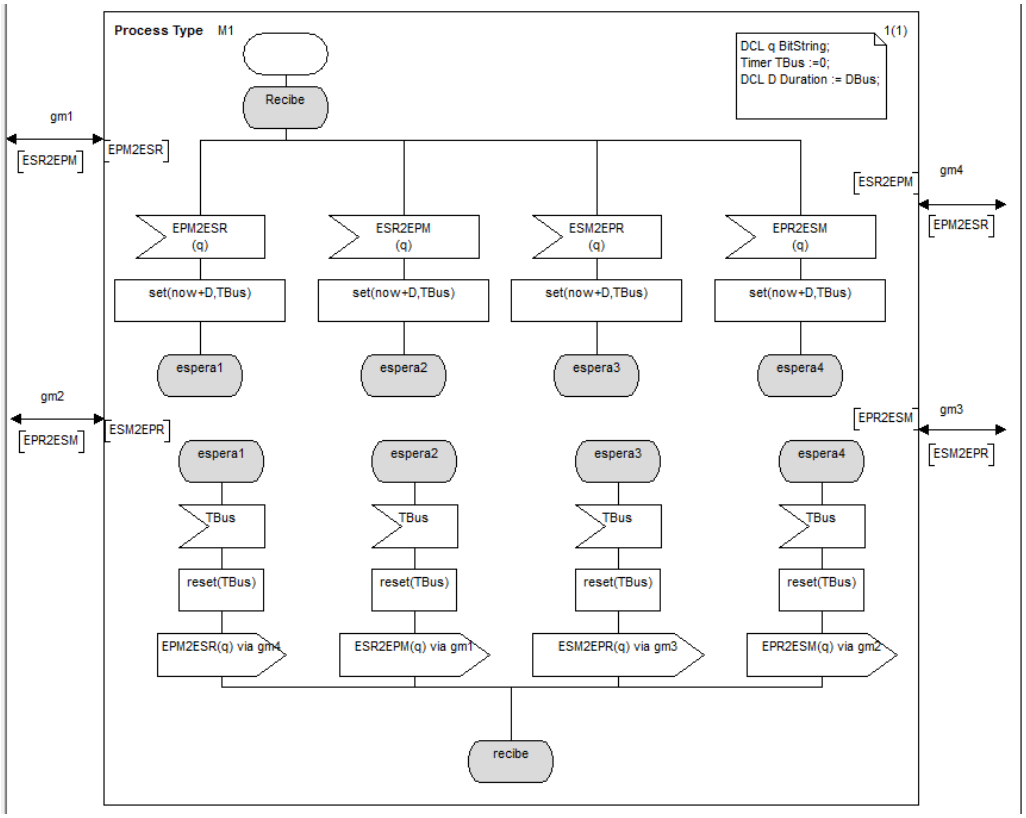


Figura 3.23. Descripción del proceso M1.

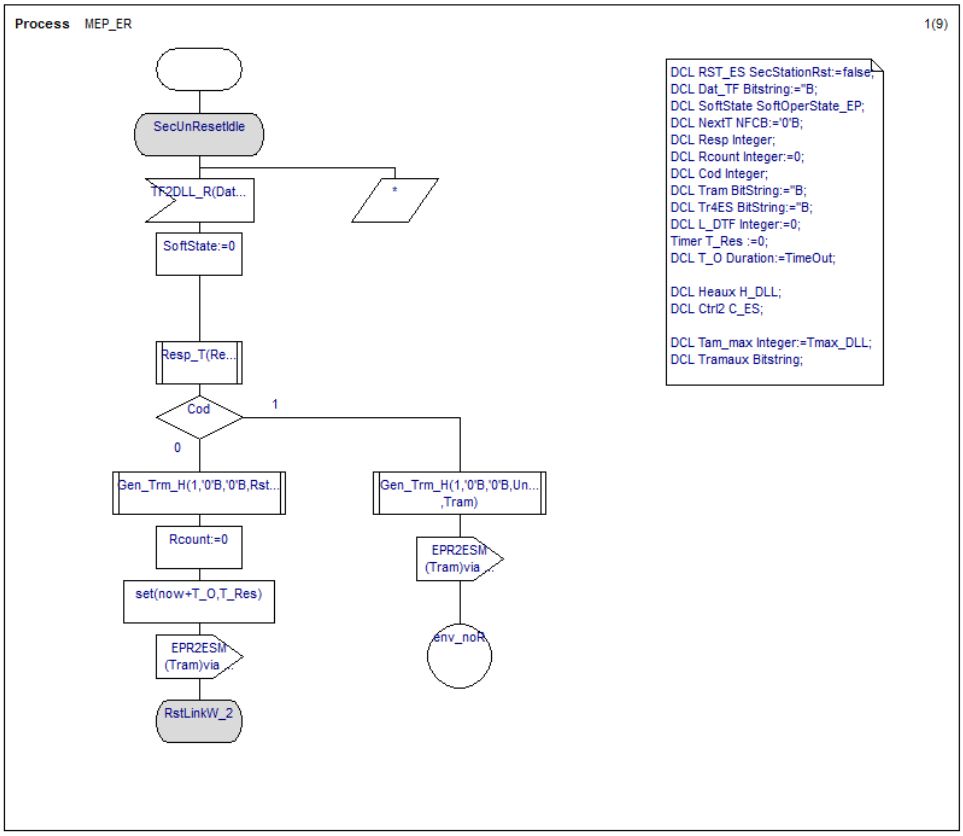


Figura 3.24. Descripción del proceso MEP_ER.

3.5.9. Proceso MES_ER

La Figura 3.25 muestra el proceso MES_ER. Una vez que se ejecuta el símbolo de inicio se realiza la transición al estado unreset, este estado está a la espera de la llegada de la señal EPM2ESR en la que se envía la solicitud para habilitar la ES desde la ER mediante el código de función Rst_Link_States. Una vez recibido dicho código de función, se envía la respuesta ACK y se realiza la transición al estado ocioso, en este estado se realiza la recepción de cualquier otro código de función así como las tramas de datos (en este proceso también se incluyen los estados Dat2TF y Dat2TF_NR).

3.5.10. Proceso Frag_ER

El proceso Frag_ER define la máquina de estados (Figura 3.26) que procesa un segmento proveniente de AL de la ER. Este proceso realiza su comportamiento similar al descrito en el apartado 3.5.3, con la diferencia en las señales con las que interactúa el proceso (AL2TF_R y TF2DLL_R) propias de la ER.

3.5.11. Proceso Ens_ER

Al igual que el proceso anterior, el proceso Ens_ER (Figura 3.27) presenta un comportamiento similar que el descrito en el apartado 3.5.4, en el que se describe el comportamiento del proceso Ens_EM, la diferencia radica en las señales con las que interactúa este proceso (DLL2TF_R y TF2AL_R).

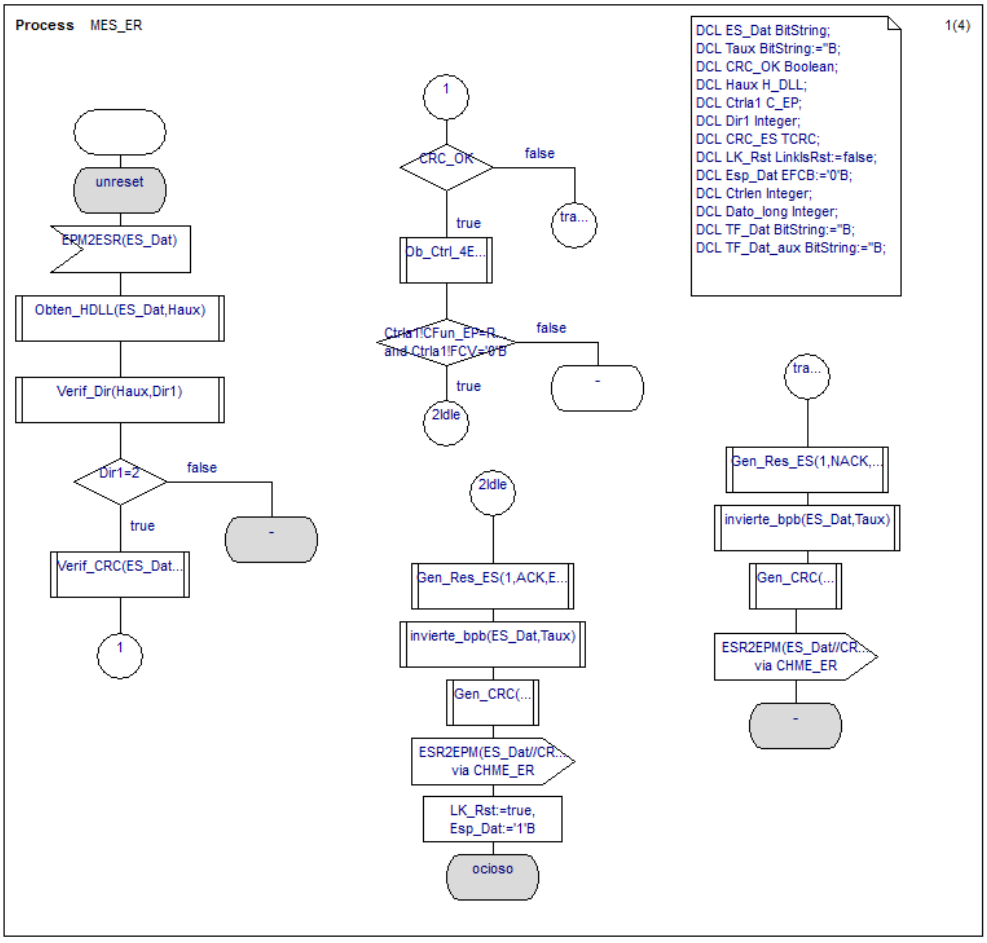


Figura 3.25. Descripción del proceso MES_ER.

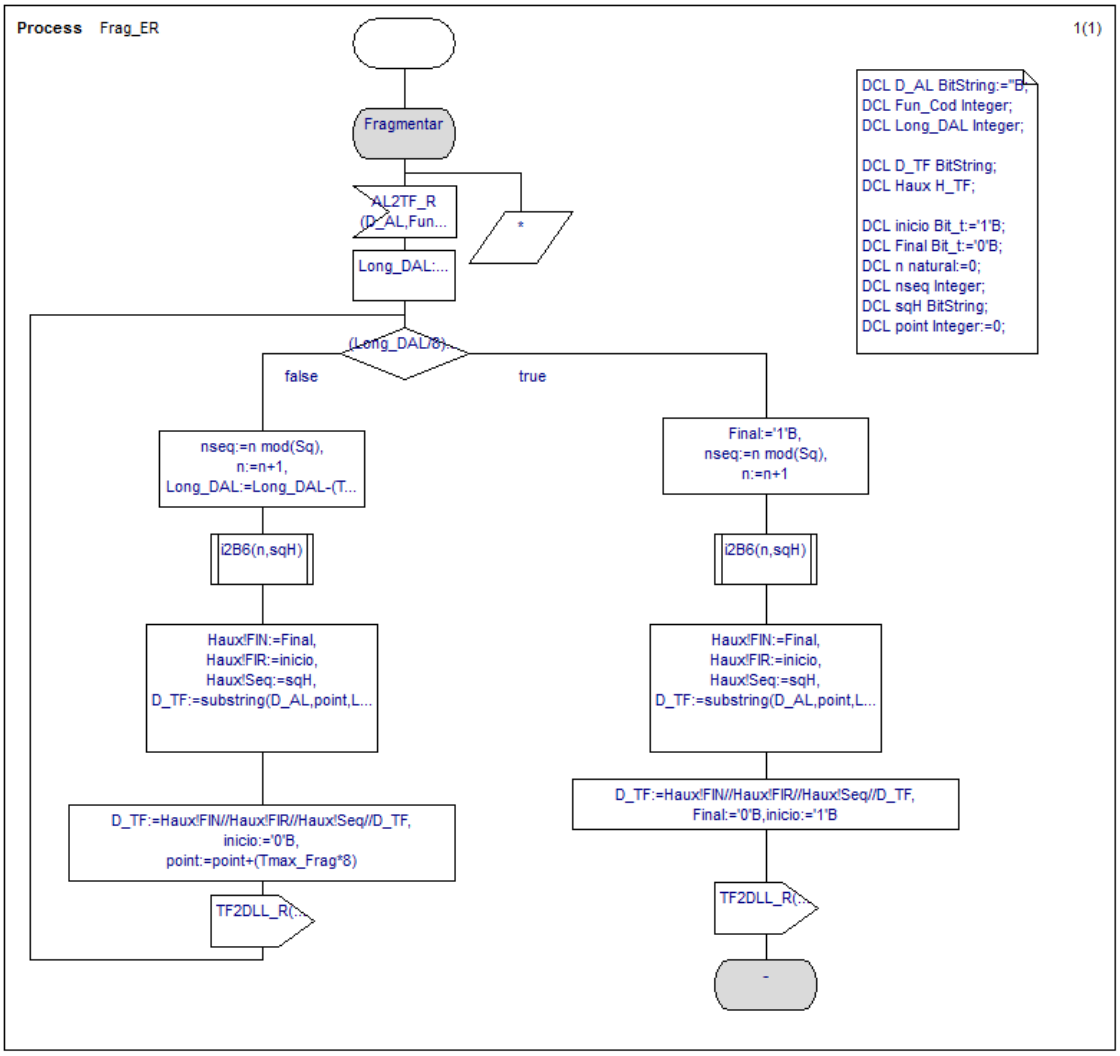


Figura 3.26. Descripción del proceso Frag_ER.

3.5.12. Proceso P1_R

Una vez que se ejecuta el símbolo de inicio, el proceso P1_R (Figura 3.28) realiza la transición al estado ocioso que está a la espera de recibir alguna información (evento) por medio de la señal Sys2ER, después se envía la información mediante la señal AL2TR_R para su acondicionamiento.

Este proceso se creó para fines de funcionalidad del protocolo y para desarrollos futuros de la Capa de Aplicación de la ER.

3.5.13. Proceso P2_R

Una vez que ejecuta el símbolo de inicio, el proceso P2_R (Figura 3.29) pasa al estado ocioso que está a la espera de recibir información por medio de la señal TF2AL y ésta se envía al sistema, (en este caso sería la capa de usuario de la ER), por medio de la señal ESR2Sys.

Este proceso se desarrolló, al igual que el anterior, con fines de funcionalidad del protocolo y especificándolo para el desarrollo futuro del mismo.

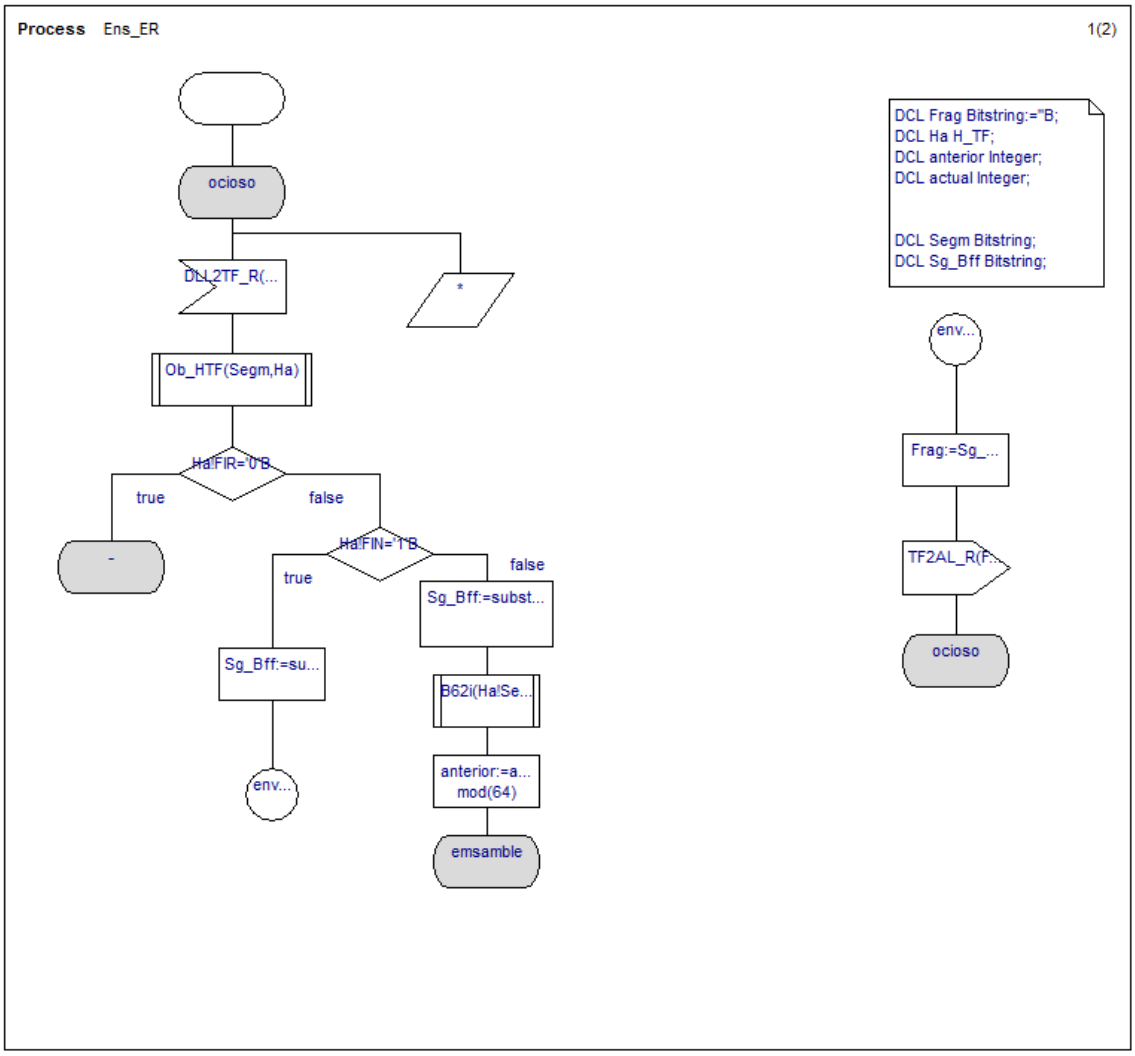


Figura 3.27. Descripción del proceso Ens_ER.

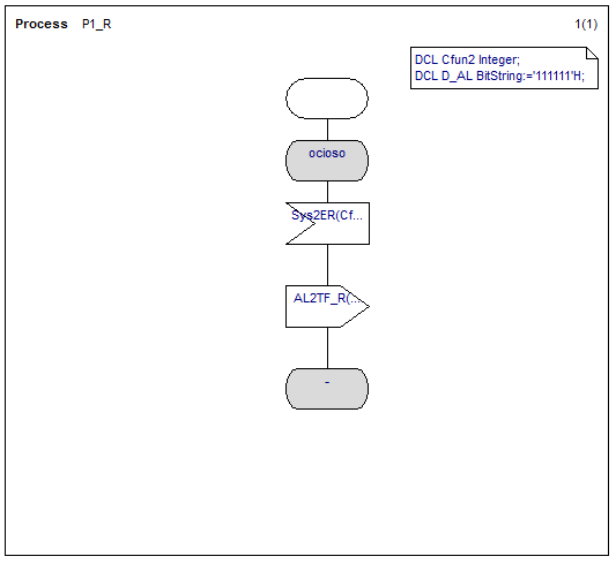


Figura 3.28. Descripción del proceso P1_R.

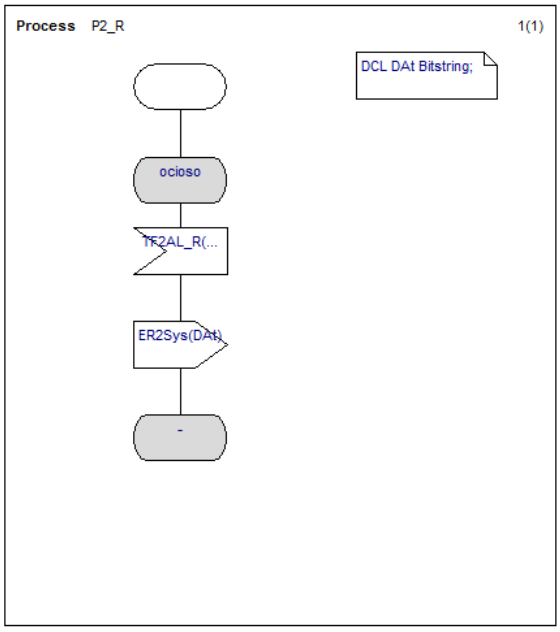


Figura 3.29. Descripción del proceso P2_R.

3.5.14. Procedimientos

En este apartado se describen los procedimientos más importantes, los cuales son responsables de generar y verificar el CRC de la trama; para entender estos procedimientos se debe considerar el algoritmo descrito en el inciso 2.3.2.2.

3.5.14.1. Procedimiento invierte_bpb

El procedimiento invierte_bpb (Figura 3.30) es el encargado de invertir byte por byte cualquier cadena, (sin importar si es alguna cabecera o dato), como se describe en el algoritmo encargado de acondicionar los datos (Figura 2.31).

Este procedimiento recibe un bitstring (cadena de bits) y devuelve un bitstring; una vez que se ejecuta el inicio del procedimiento, se calcula la longitud de la cadena recibida y se compara la longitud con un contador. Posteriormente se inicia el procedimiento de inversión de los datos, los cuales se almacenan en la cadena auxiliar dw (de 16 bytes); una vez finalizada la inversión se realiza una división de la cadena dw (substring) y se le asigna a la cadena que se regresa (dat_inv).

3.5.14.2. Procedimiento mod2

El procedimiento mod2 (Figura 3.31) realiza la función de desplazamiento, descrita en la Figura 2.32, mediante el empleo de la operación XOR a nivel de bits.

Este procedimiento recibe un bitstring y devuelve lo mismo; una vez que se ejecuta el inicio del procedimiento se calcula la longitud de la cadena recibida, posteriormente se localizan los 1´s dentro de la cadena recibida (cadena previamente invertida) mediante una búsqueda incremental, cuando algún 1 es localizado se realiza la operación XOR con la cadena actual y la variable KEY que contiene el valor del polinomio generador del CRC (inciso 2.3.2.2). Cuando se finaliza este procedimiento se regresa la cadena dat_mod.

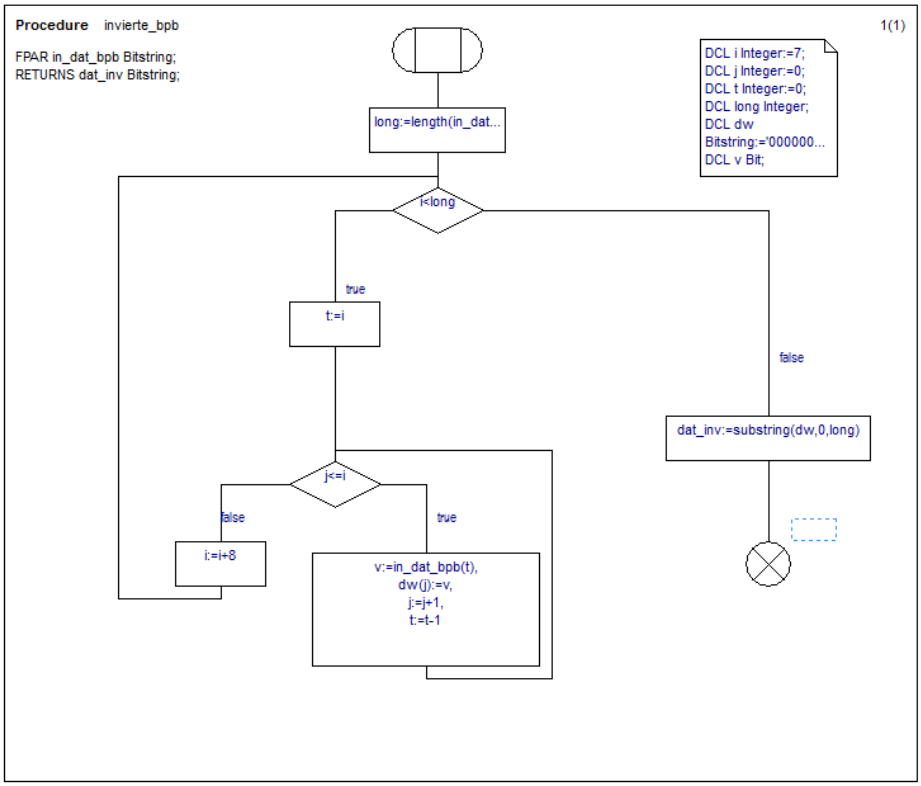


Figura 3.30. Descripción del procedimiento invierte_bpb.

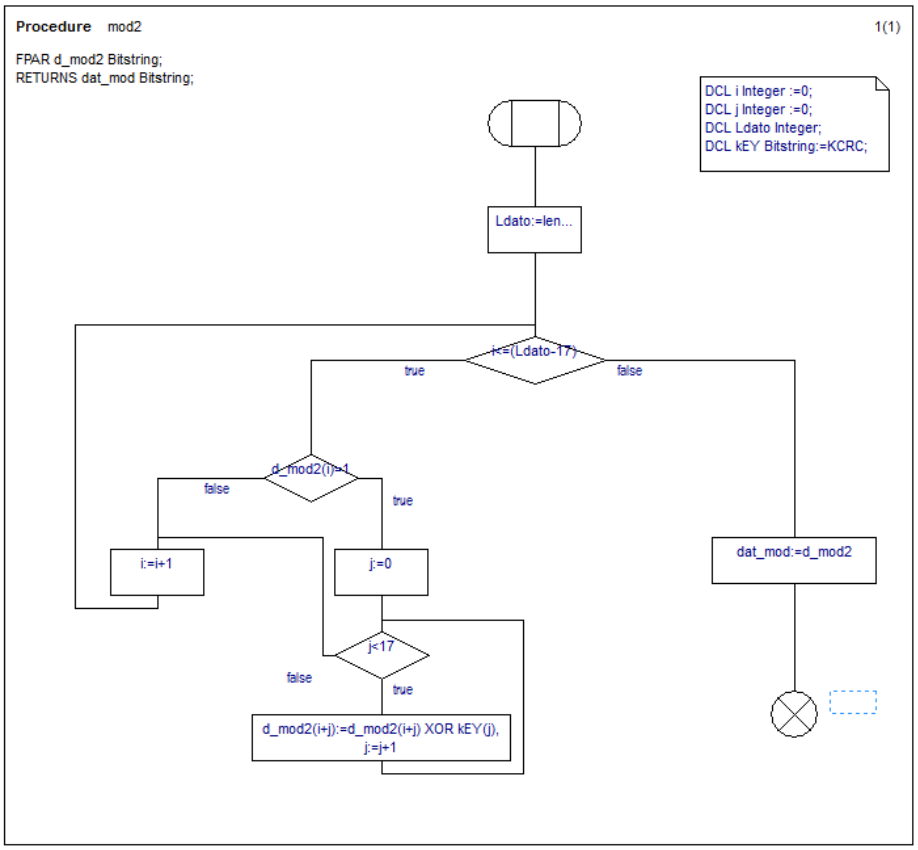


Figura 3.31. Descripción del procedimiento mod2.

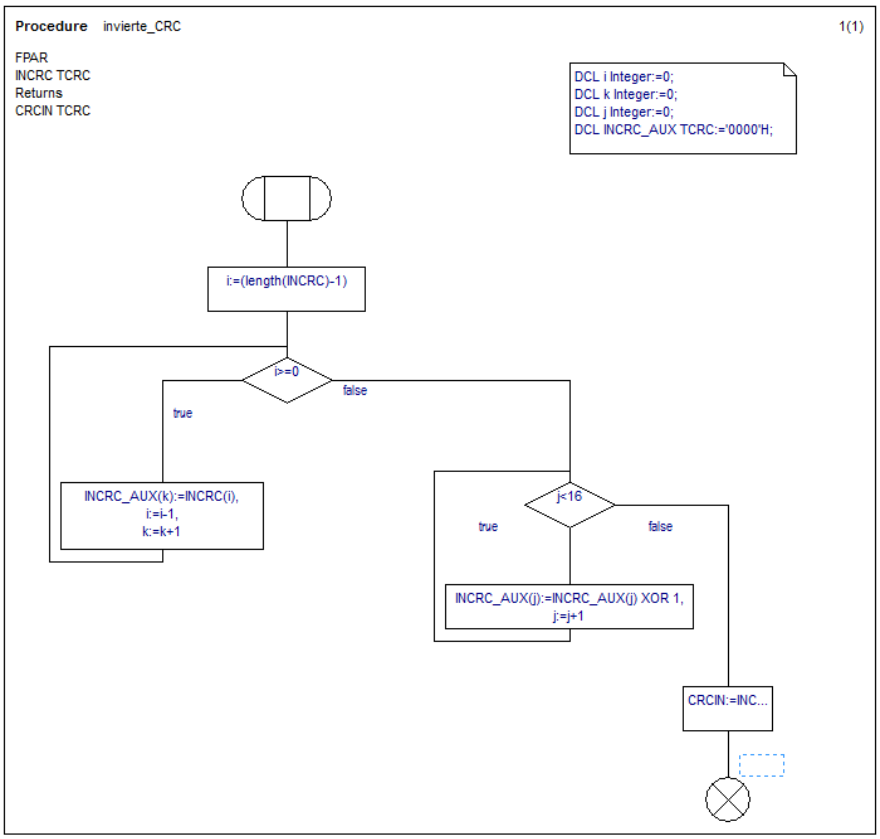


Figura 3.32. Descripción del procedimiento invierte_CRC.

3.5.14.3. Procedimiento invierte_CRC

El procedimiento invierte_CRC (Figura 3.32) es el encargado de realizar la operación de invertir el polinomio y de realizar la operación de XOR con 1 (Figura 2.32).

Este procedimiento recibe una cadena de bits de tipo TCRC y devuelve lo mismo. Una vez que se ejecuta el inicio del procedimiento, primero se realiza la inversión de la cadena recibida (INCRC) y posteriormente se realiza una operación XOR entre los valores de la cadena invertida y 1. Finalmente se devuelve el resultado de realizar todas las operaciones mediante la variable CRCIN.

Tanto este procedimiento como los anteriores, se emplean tanto para generar como para verificar el CRC.

3.5.14.4. Procedimiento Gen_CRC

Una vez descritos los procedimientos empleados para realizar el procedimiento Gen_CRC (Figura 3.33), a continuación se describe el procedimiento completo de la Figura 2.32.

Este procedimiento recibe una cadena de datos de tipo bitstring, (previamente invertida), y retorna una variable de tipo TCRC. Después de ejecutar el símbolo de inicio del procedimiento, se concatena a la variable DatosI una variable de tipo TCR (CRCaux) y se analiza la longitud de la cadena resultant, posteriormente se envía la cadena DatosI al procedimiento mod2 y posteriormente, se divide el resultado mediante la operación substring; finalmente se envía el resultado al procedimiento inv_CRC y se coloca el byte más significativo en el lugar del menos significativo y viceversa.

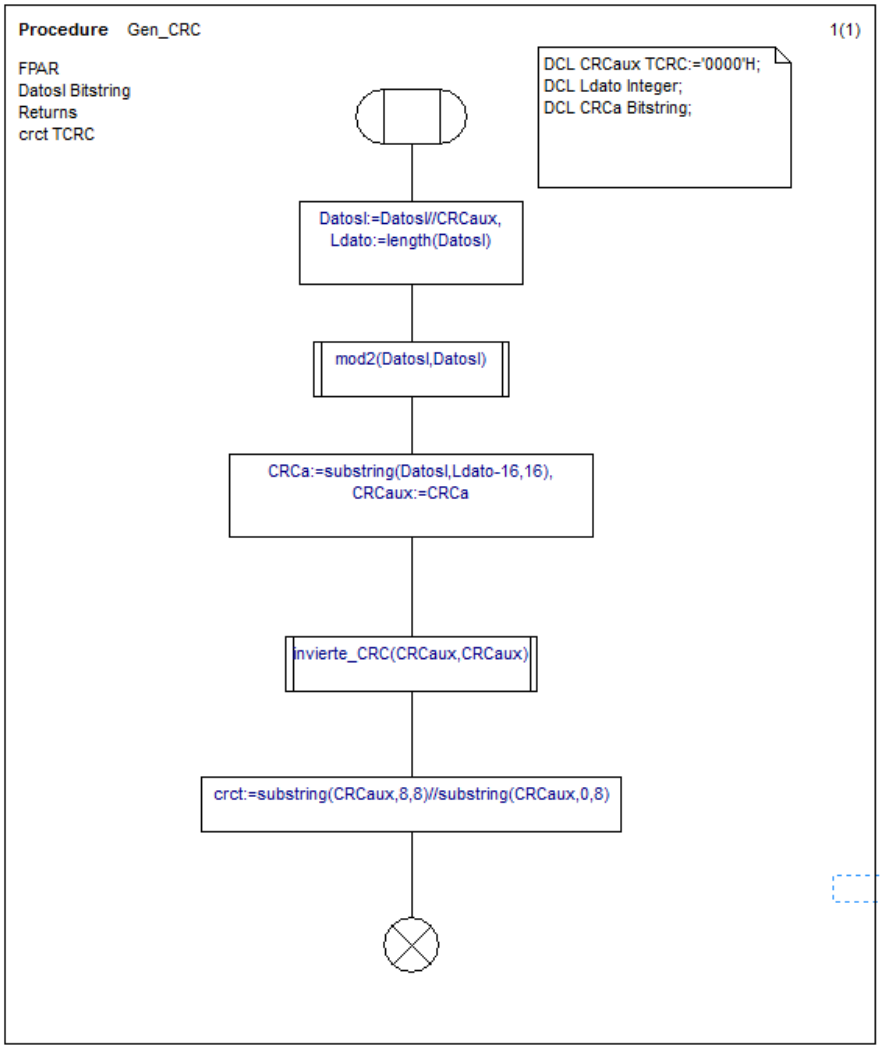


Figura 3.33. Descripción del procedimiento Gen_CRC.

3.5.14.5. Procedimiento Verif_CRC

El procedimiento Verif_CRC (Figura 3.34) se encarga de realizar la operación inversa a la anteriormente descrita, es decir verificar la trama recibida por la ES o EP. Recibe una cadena de tipo bitstring (T_Chek) y retorna una variable de tipo boolean. Una vez que el procedimiento ejecuta el inicio del procedimiento se calcula el tamaño de la trama, después se subdivide para su tratamiento utilizando los procedimientos descritos, finalmente se emplea el procedimiento Chek_0 que se encarga de verificar que todos los bits del CRC sean cero, con esto regresa el valor de true, de lo contrario regresaría un valor false.

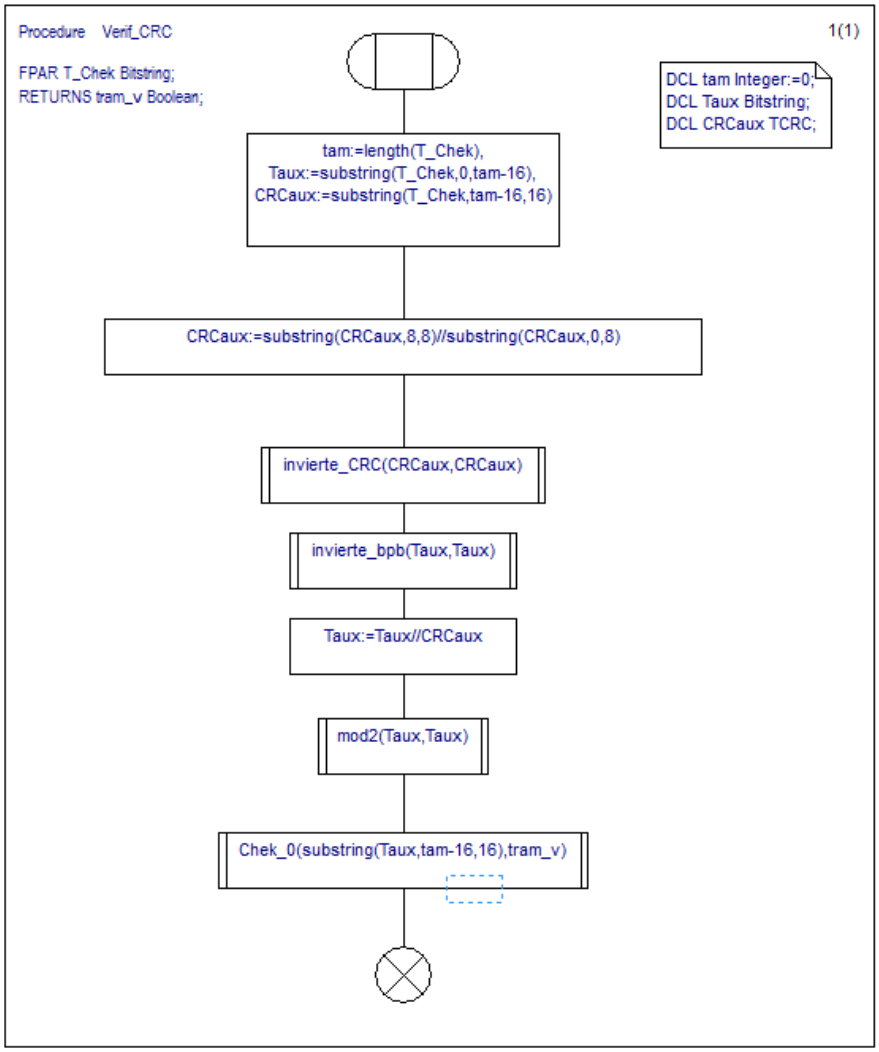


Figura 3.34. Descripción del procedimiento Verif_CRC.

4. Conclusiones y trabajos futuros

La etapa de diseño que define la metodología de la Figura 1.1 se ha detallado en los subcapítulos 3.4 y 3.5, cuyo resultado principal es la especificación formal del protocolo de comunicaciones DNP3 (capítulo 3). El presente capítulo presenta la etapa de diseño detallado, la cual consistió en la simulación, depuración y refinamiento de la especificación obtenida, y que dio lugar a establecer las conclusiones del presente trabajo de tesis.

Como parte de la metodología de desarrollo, una vez obtenida la especificación formal, ésta debe verificarse mediante la simulación, depuración y refinamiento, las cuales se realizan con ayuda de herramientas de software especializadas [3]. En este caso, no se dispuso de herramientas adicionales, por lo que dichas actividades se realizaron en forma secuencial y durante el desarrollo de la especificación empleando las propiedades de edición, análisis (sintáctico y semántico), simulación integral y generación de MSCs de la herramienta Cinderella SDL.

Las actividades de simulación y depuración se aplican a la especificación formal para corregir posibles errores de diseño y comprobar que se satisfagan los requerimientos de comportamiento. La actividad de refinamiento consistió en pruebas de comportamiento y tiene como objetivo verificar que la especificación obtenida reaccione ante los eventos de manera adecuada y se obtenga un grado elevado de confiabilidad del sistema. Debido a la complejidad de esta última actividad, se requiere el dominio del lenguaje TTCN-3 (*Testing and Test Control Notation*, tercera versión) [5] y el uso de herramientas de validación y verificación. Cabe señalar que no se cuenta con dichas herramientas y por ello no se realizaron formalmente las pruebas de comportamiento.

4.1. Actividades de simulación y depuración

La especificación SDL del protocolo de comunicaciones DNP3 se realizó mediante la construcción de un sistema compuesto por bloques, los cuales contienen los procesos que describen el funcionamiento de dicho protocolo. Para realizar la actividad de simulación, o prueba del sistema (prueba de caja negra), el usuario del sistema (Figura 3.17 y Figura 3.28) envió códigos de función pertenecientes a la Capa de Aplicación (Tabla 2.15 y Tabla 2.16), obteniendo los siguientes resultados:

- Correcta generación y comprobación del CRC.
- Generación de las cabeceras pertenecientes a cada capa diseñada.

- Recepción correcta de las tramas que establecen la comunicación entre EM y ER.
- Recepción, transmisión y procesamiento de los datos pertenecientes a capas superiores (datos).
- Una vez realizados los puntos anteriores, se enviaron tramas con cada código de función de la Capa de Aplicación para revisar el comportamiento de las capas inferiores, cabe señalar que la construcción de dichas tramas y el comportamiento de las capas inferiores son resultados de la especificación.

Durante la simulación de la especificación del protocolo de comunicaciones DNP3 se llevó a cabo su depuración mediante la supervisión constante del explorador de Cinderella SDL (Figura 4.1), el cual visualiza el estado en que se encuentra el proceso que se está simulando y los valores de las variables correspondientes a dicho proceso. Durante la simulación se analizó la reacción del proceso ante la recepción de las señales que constituyen su alfabeto de entrada.

Cinderella SDL permite realizar la depuración, ya que tiene la posibilidad de establecer puntos de ruptura donde se requiere 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 es el caso de la mayoría de los procesos de la especificación del protocolo).

Los errores detectados durante la simulación consistieron 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 parametrizarón algunas señales para poder determinar el origen de los errores.

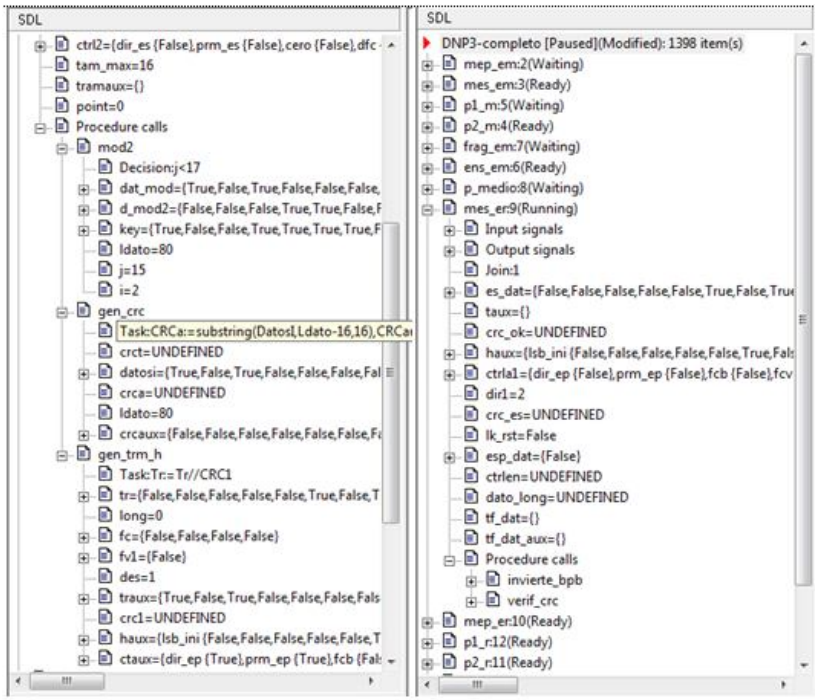


Figura 4.1. Exploración de los procesos durante la simulación.

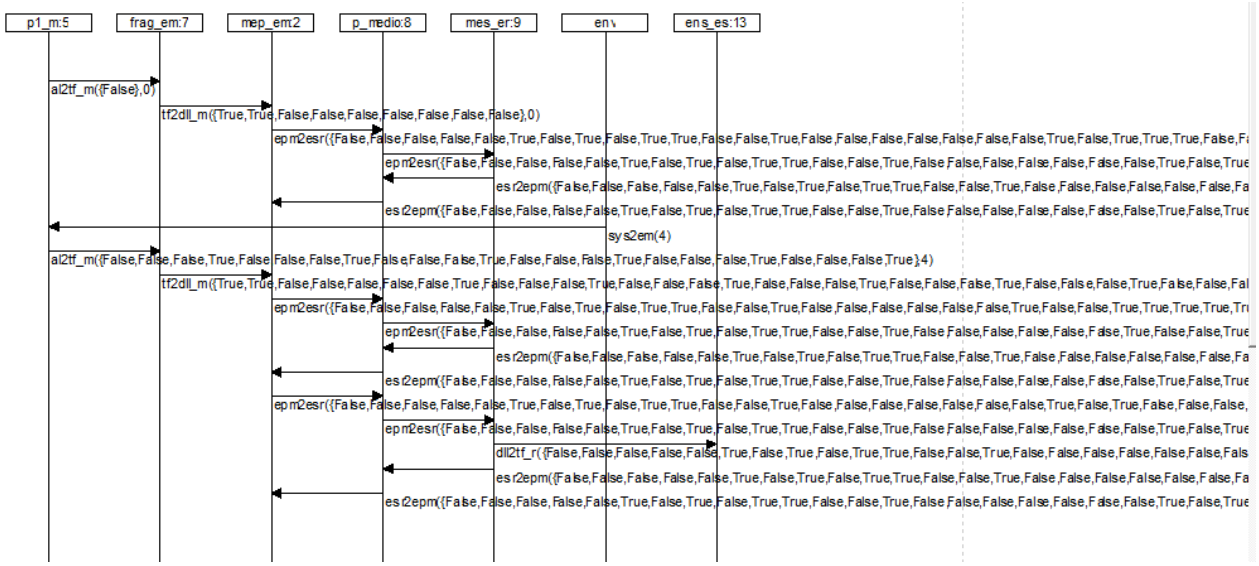


Figura 4.2. Ejemplo de los casos de uso (MSCs) generados con Cinderella SDL.

La Figura 4.2 muestra el aspecto de un conjunto de señales dentro del esquema MSC[13], con lo cual se pudo observar la evolución de la simulación como una secuencia de eventos ordenados en el tiempo. 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). 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 la depuración se realizó de forma manual y que los errores más complejos de solucionar se originaron debido al tiempo en la simulación del sistema.

La validación de la especificación del protocolo DNP3 se realizó mediante la verificación de la detección de errores, empleando el cálculo del CRC, así como las funciones realizadas en cada uno de los procesos del sistema; también se analizaron los datos enviados por medio de señales a cada uno de los procesos definidos en la especificación del protocolo.

4.2. Conclusiones

En el presente trabajo de tesis se realizó una investigación en dos áreas importantes en el desarrollo tecnológico e industrial, por un lado el protocolo de comunicaciones DNP3 y por otro la FDT SDL. Resultado de dicho trabajo es la especificación ejecutable del protocolo DNP3 como caso de estudio, objetivo central de la presente tesis, y deriva el siguiente conjunto de aportaciones:

- Se estudió la norma SDL, detallada en la recomendación Z.100, lo cual fue fundamental para lograr el desarrollo del protocolo DNP3.
- Se estudió a detalle la arquitectura de protocolos DNP3 (capítulo 2).
- Fue necesario estudiar el lenguaje ASN.1, detallado en la recomendación Z.105, durante las etapas de pruebas y desarrollo, ya que el procedimiento para calcular los CRC tomaba demasiado tiempo en su ejecución y con la manipulación correcta de esos tipos de datos se redujo de manera significativa el tiempo empleado en el cálculo los CRC.

[13] Para generar el MSC se interrumpió temporalmente la simulación mientras se analizaba el MSC.

- Se obtuvo una especificación formal SDL de la Capa de Enlace de Datos y la Función de Transporte del protocolo DNP3 sin ambigüedades incluyendo una reducida descripción de la Capa de Aplicación.
- Se definió el algoritmo para el cálculo del CRC, dado que éste no está definido de manera concreta en la especificación del protocolo de comunicaciones DNP3.
- Es importante destacar que la especificación estática del protocolo de comunicaciones DNP3 no está descrita completamente en la especificación del protocolo, sólo indica una idealización de la misma, por lo que fue necesario el desarrollo de bloques y procesos que no están descritos en el protocolo (MED, Frag_EM, Frag_ER, Dat2TF y Dat2TF_NR).

Finalmente se puede realizar la valoración final sobre la utilidad de aplicar de las FDTs en el diseño de sistemas de comunicaciones y protocolos, siendo ésta 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 establecidos, así como su ejecución, simulación, validación y pruebas.

Con base en el trabajo realizado, se proponen los siguientes trabajos futuros:

- Continuar con la especificación del protocolo de comunicaciones DNP3, particularmente con la inclusión de la Capa de Aplicación y la Capa Física.
- Realizar la simulación del protocolo encapsulándolo para la transmisión bajo el protocolo TCP/IP.
- Realizar la simulación de la especificación con más de una ER.
- Realizar en estudio detallado del comportamiento del protocolo utilizando las demás topologías que permite DNP3 para su comunicación.
- Realizar un estudio detallado de la eficacia del CRC implementado por el protocolo DNP3 respecto a los CRC utilizados en protocolos similares.
- Implementar el protocolo en FPGAs y microcontroladores para el diseño de una red que contenga al menos una EM y una ER.

Bibliografía

[1] Chamú, C., Desarrollo de un Sistema Educativo para Enseñanza del Protocolo de Comunicaciones CAN, Universidad Tecnológica de la Mixteca, Tesis de Licenciatura, Abril de 2005.

[2] DNP3 SPECIFICATION, Version 2.01, 3 February 2007.

[3] Ellsberger, J., Formal Object-oriented Language for Communicating Systems, Prentice Hall, 1997.

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

[5] González, R., Especificación del Protocolo FLEXRAY utilizando un Lenguaje de Descripción Formal, Universidad Tecnológica de la Mixteca, Tesis de Licenciatura, Julio 2008.

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

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

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

[9] Doldi, L., Validation of Communications Systems with SDL, Wiley, 2003.

[10] Doldi, L., Visual Design Executable Models, May 2001, ISBN 2-9526600-06.

[11] Nogueira, J., Metodologías de Especificación Formal aplicadas a Modelos Normalizados de Protocolos de Comunicación Industriales, Universidad de Vigo, E.T.S.I. Telecomunicación, Tesis Doctoral, Octubre 2000.

[12] Merino, P., Lenguajes de Especificación SDL y MSC, Universidad de Málaga, Depto. de Lenguajes y Ciencias de la Computación, Curso 2002/2003.

[13] Stallings, W., Comunicaciones y Redes de computadoras, Séptima edición, Prentice Hall, 2004.

[14] Turnner K., Using Formal Description Techniques: An introduction to Estelle, Lotos and SDL. John Wile&Sons, 1993.

Sitios de internet

[URL1] <http://www.cinderella.dk/> Página web de la empresa “CinderellaAps”, 06/09/2010.

[URL2] <http://www.dnp.org> Página oficial del protocolo DNP, 09/2010.

[URL3] <http://www.dnp.org/Modules/Library/Default.aspx?CategoryID=3> Página de descarga de los manuales del protocolo DNP3, 09/2010.

[URL4] <http://www.itu/ITU-T/> Página web de la ITU, 10/2010.

[URL5] < http://www.sdl-forum.org/> Foro de SDL, 10/2010.

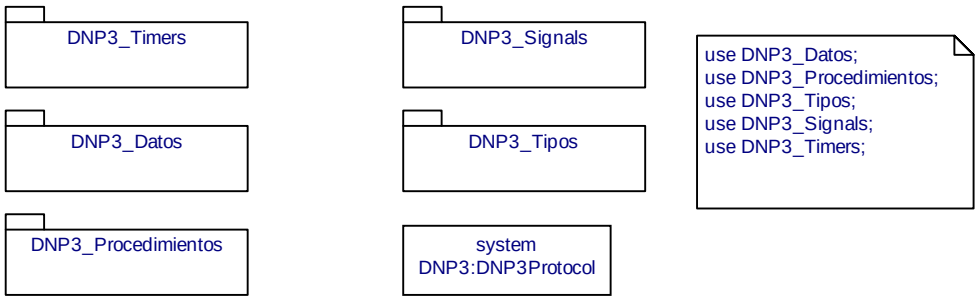
[URL6] <http://my.epri.com>Página web del EPRI, 09/2010.

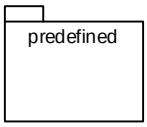
[URL7]<http://www.harris.com> Página oficial de GE-Harris, 01/11/2010.

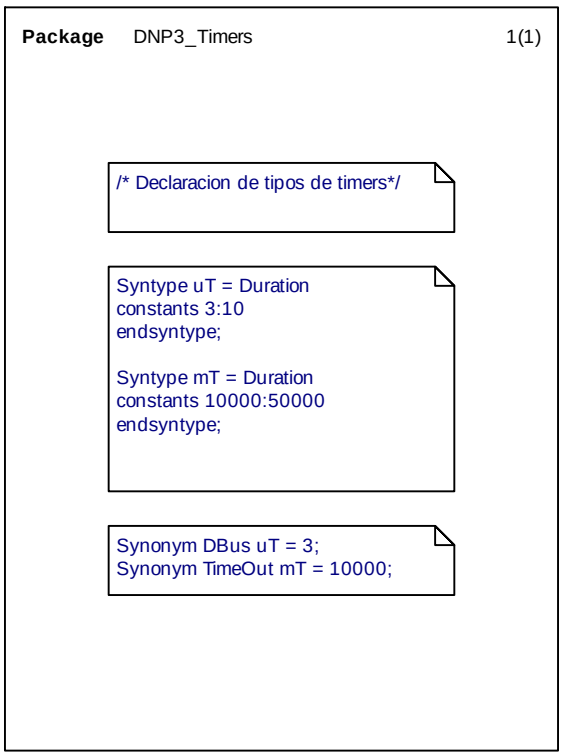
[URL8]< http://www.lammertbies.nl/forum/viewtopic.php?t=1327> Página que indica el cálculo CRC de DNP3, 07/03/2011.

[URL9]< http://www.lcc.uma.es/~pedro/docencia/sc/UsingDataTypes.pdf> Página que muestra los tipos de datos del estándar Z.105, 08/03/2011.

Anexo A. Especificación formal del protocolo DNP3 con SDL


DNP3_Timers
DNP3_Signals
use DNP3_Datos;
use DNP3_Procedimientos;
use DNP3_Tipos;
use DNP3_Signals;
use DNP3_Timers;
DNP3_Datos
DNP3_Tipos
DNP3_Procedimientos
system
DNP3:DNP3Protocol


predefined


Package DNP3_Timers 1(1)
/* Declaracion de tipos de timers*/
Syntype uT = Duration
constants 3:10
endsyntype;
Syntype mT = Duration
constants 10000:50000
endsyntype;
Synonym DBus uT = 3;
Synonym TimeOut mT = 10000;

Package DNP3_Signals 1(1)

```
/*Señales para la comunicacion entre las estaciones y el medio*/
signal ESR2EPM(BitString);
signal EPM2ESR(BitString);
signal EPR2ESM(BitString);
signal ESM2EPR(BitString);

/*comunicacion entre la TF y el DLL de la EM*/

Signal TF2DLL_M (Bitstring,Integer);
Signal DLL2TF_M (Bitstring);

/*comunicacion entre la TF y el DLL de la ER*/

Signal TF2DLL_R (Bitstring,Integer);
Signal DLL2TF_R (Bitstring);

/*comunicacion entre la AL y la TF de la EM*/
Signal AL2TF_M (Bitstring,Integer);
Signal TF2AL_M (Bitstring);

/*comunicacion entre la AL y la TF de la ER*/
Signal AL2TF_R (Bitstring,Integer);
Signal TF2AL_R (Bitstring);

/*comunicacion entre el sistema y la EM*/
Signal Sys2EM (Integer);
Signal EM2Sys (Bitstring);

/*comunicacion entre el sistema y la ER*/
Signal Sys2ER (Integer);
Signal ER2Sys (Bitstring);
```

Package DNP3_Datos 1(3)

/* Datos del para el control de tramas */

```
/* Definicion de tipos de datos para las Tramas */
Syntype Byte = Bitstring
       constants size (8)
EndSyntype;

Syntype Bit_t = Bitstring
       constants size (1)
EndSyntype;

Syntype Bit_4 = Bitstring
       constants size (4)
EndSyntype;

/* Contador de reintentos de envio de datos */
Syntype Reintentos = Integer
EndSyntype;


Syntype TCRC = Bitstring
       constants size(16)
endSyntype;

/*Dato pata la llave del CRC*/
Syntype KEY = Bitstring
       constants size (17)
EndSyntype;
```

Package DNP3_Datos

2(3)

/* Datos del DLL */

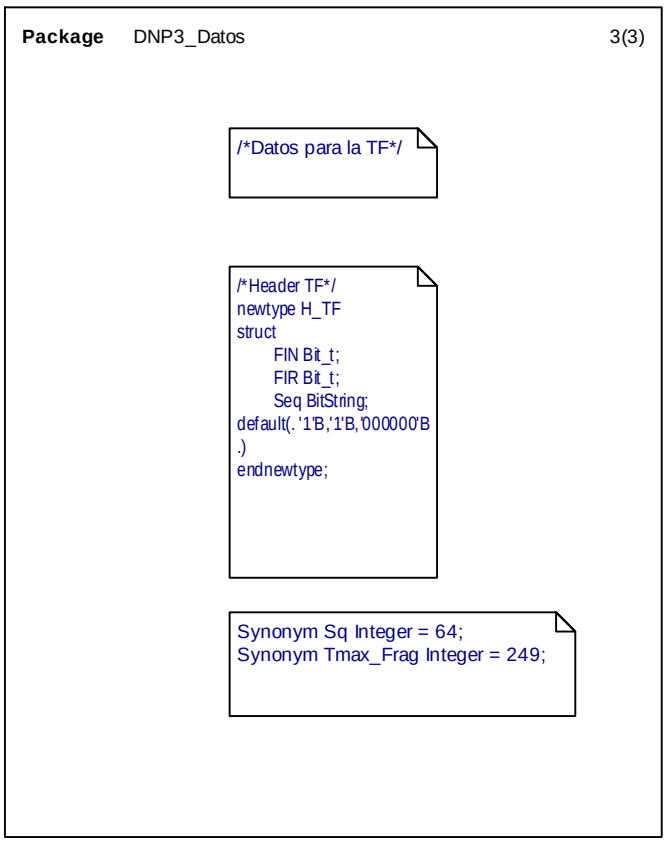
```
/* Datos para el contro de la EP */

syntype SecStationRst = Boolean
endsyntype;

syntype SoftOperState_EP = Integer
endsyntype;

syntype NFCB = Bit_t
endsyntype;
```

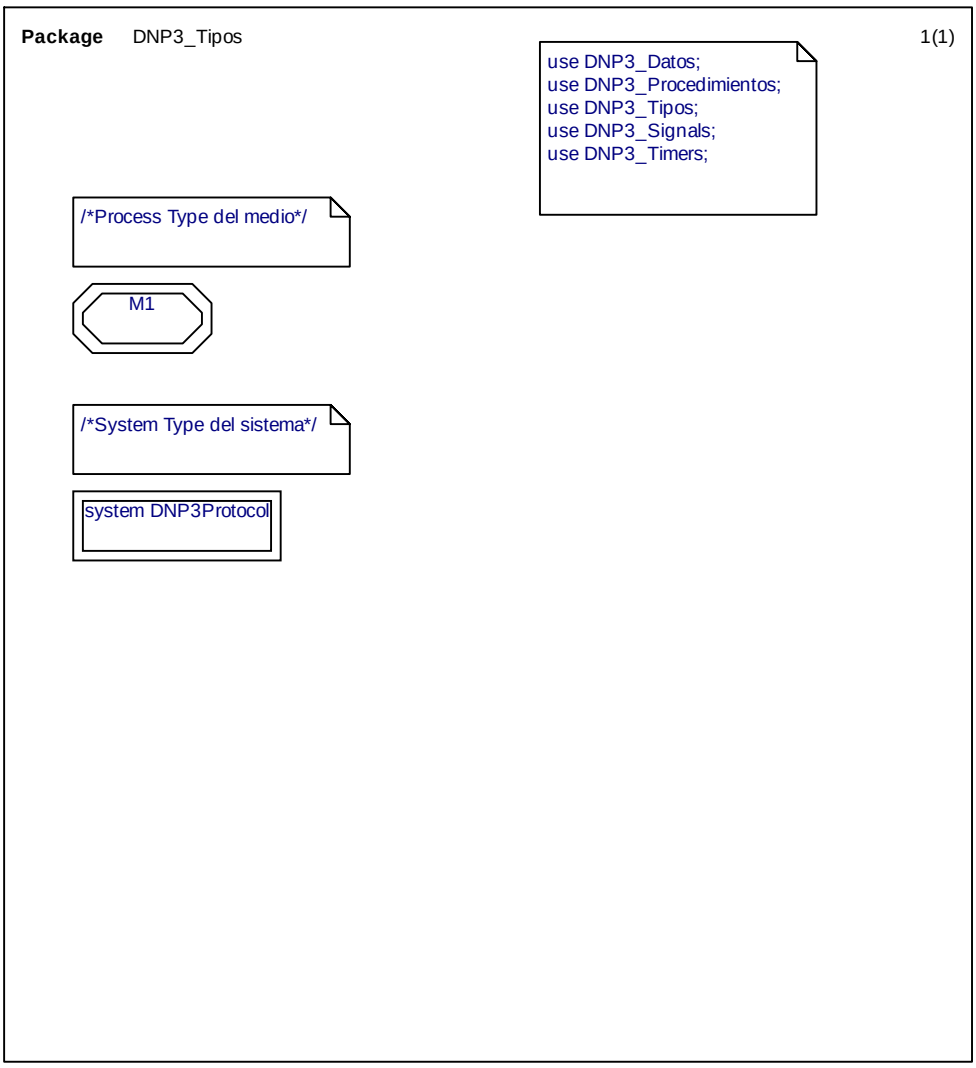
```
/* Datos para el contro de la ES */

syntype LinkIsRst = Boolean
endsyntype;

syntype SoftOperState_ES = Integer
endsyntype;

syntype EFCB = Bit_t
endsyntype;
```

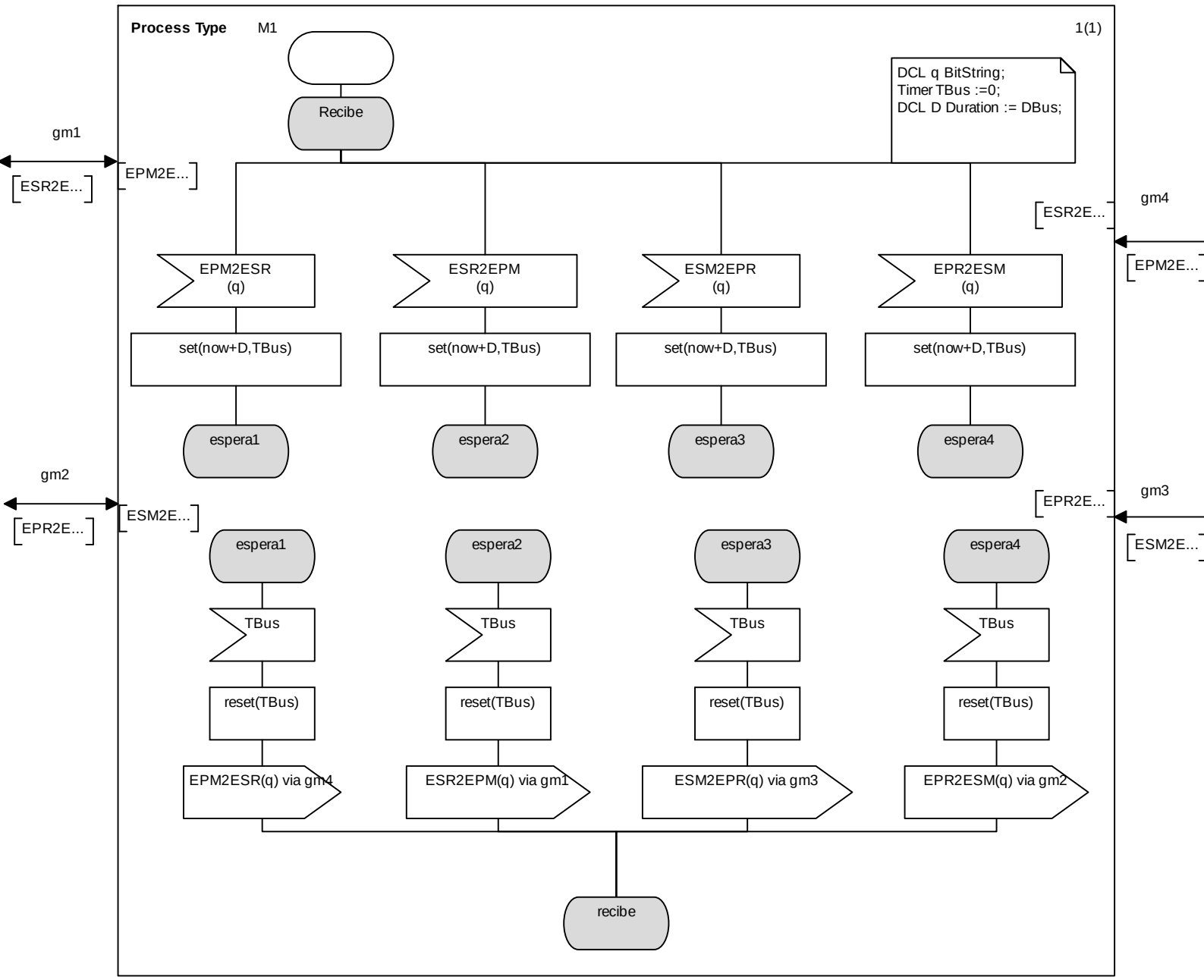
```
/* Constantes Globales */
Synonym KCRC KEY = '10011110101100101'B;
Synonym Dir_ER Bitstring = '0100'H;
Synonym Dir_EM Bitstring = '0000'H;
Synonym inicio_H Bitstring = '0564'H;
Synonym Dir_EP Byte = '01'H;
Synonym Dir_ES Byte = '00'H;

/* Codigos de funcion de DLL EP*/
Synonym Rst_Link_States Bit_4 = '0'H;
Synonym Test_Link_States Bit_4 = '2'H;
Synonym Confirmed_User_Data Bit_4 = '3'H;
Synonym UnConfirmed_User_Data Bit_4 = '4'H;
Synonym Rqst_Link_Status Bit_4 = '9'H;

/* Codigos de funcion de DLL ES*/
Synonym ACK Bit_4 = '0'H;
Synonym NACK Bit_4 = '1'H;
Synonym Link_Status Bit_4 = 'B'H;
Synonym Not_Supported Bit_4= 'F'H;
```

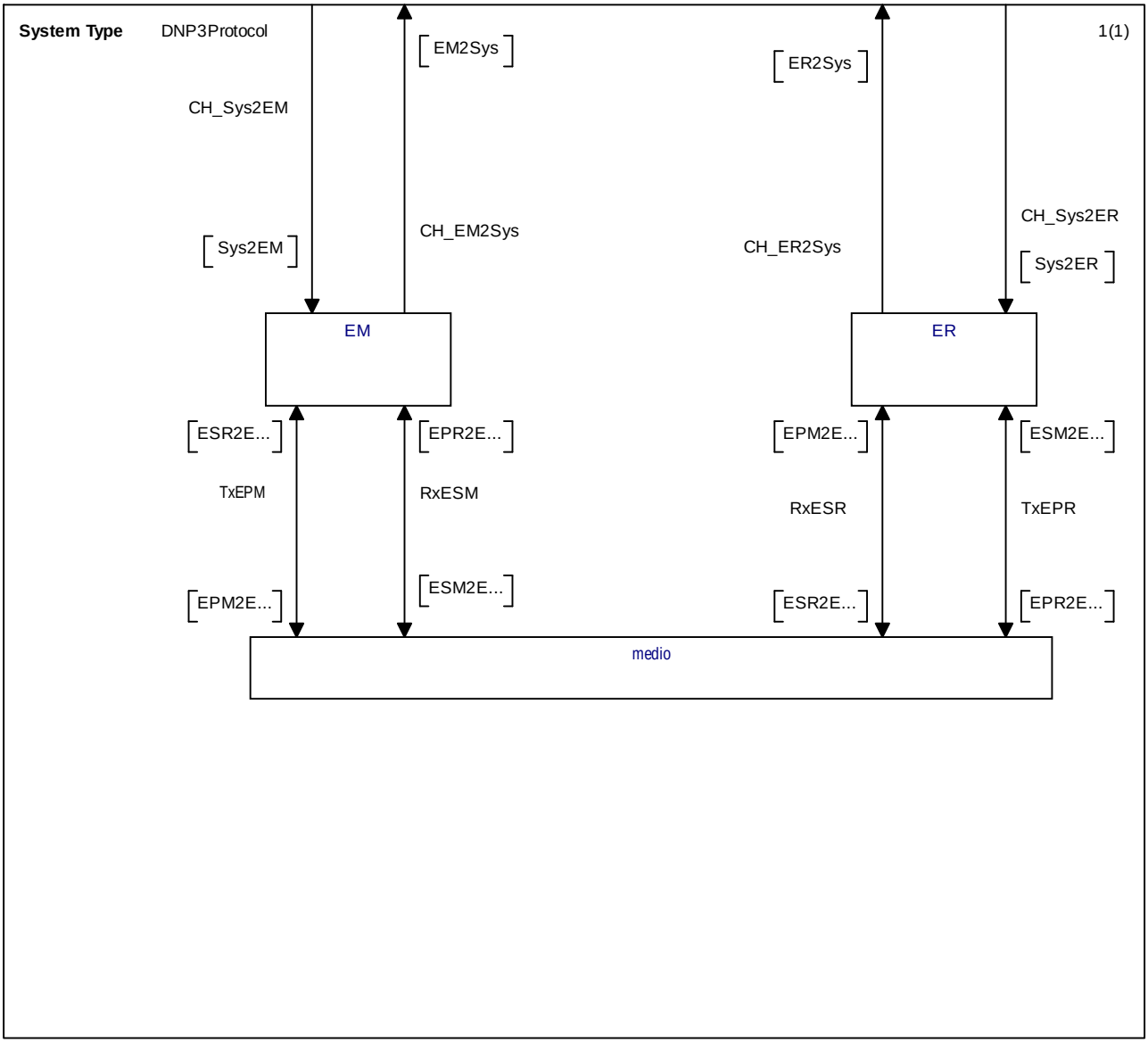
```
/*Campo de control*/

newtype C_EP
struct
        Dir_EP       Bit_t;
        PRM_EP      Bit_t;
        FCB      Bit_t;
        FCV      Bit_t;
        CFun_EP      Bit_4;
default(. '0'B,'0'B,'0'B,'0'B,'0'H .)
endnewtype;

newtype C_ES
struct
        Dir_ES       Bit_t;
        PRM_ES      Bit_t;
        Cero      Bit_t;
        DFC      Bit_t;
        CFun_ES      Bit_4;
default(. '0'B,'0'B,'0'B,'0'B,'0'H .)
endnewtype;


/*header_DLL*/
newtype H_DLL
struct
        LSB_INI        Byte;
        MSB_INI        Byte;
        LEN        Byte;
        Ctrl        Byte;
        LSB_Dest        Byte;
        MSB_Dest        Byte;
        LSB_Fuent        Byte;
        MSB_Fuent        Byte;

default(. '05'H,'64'H,'05'H,'F2'H,'01'H,'00'H,'00'H,'00'H .)
endnewtype;
```

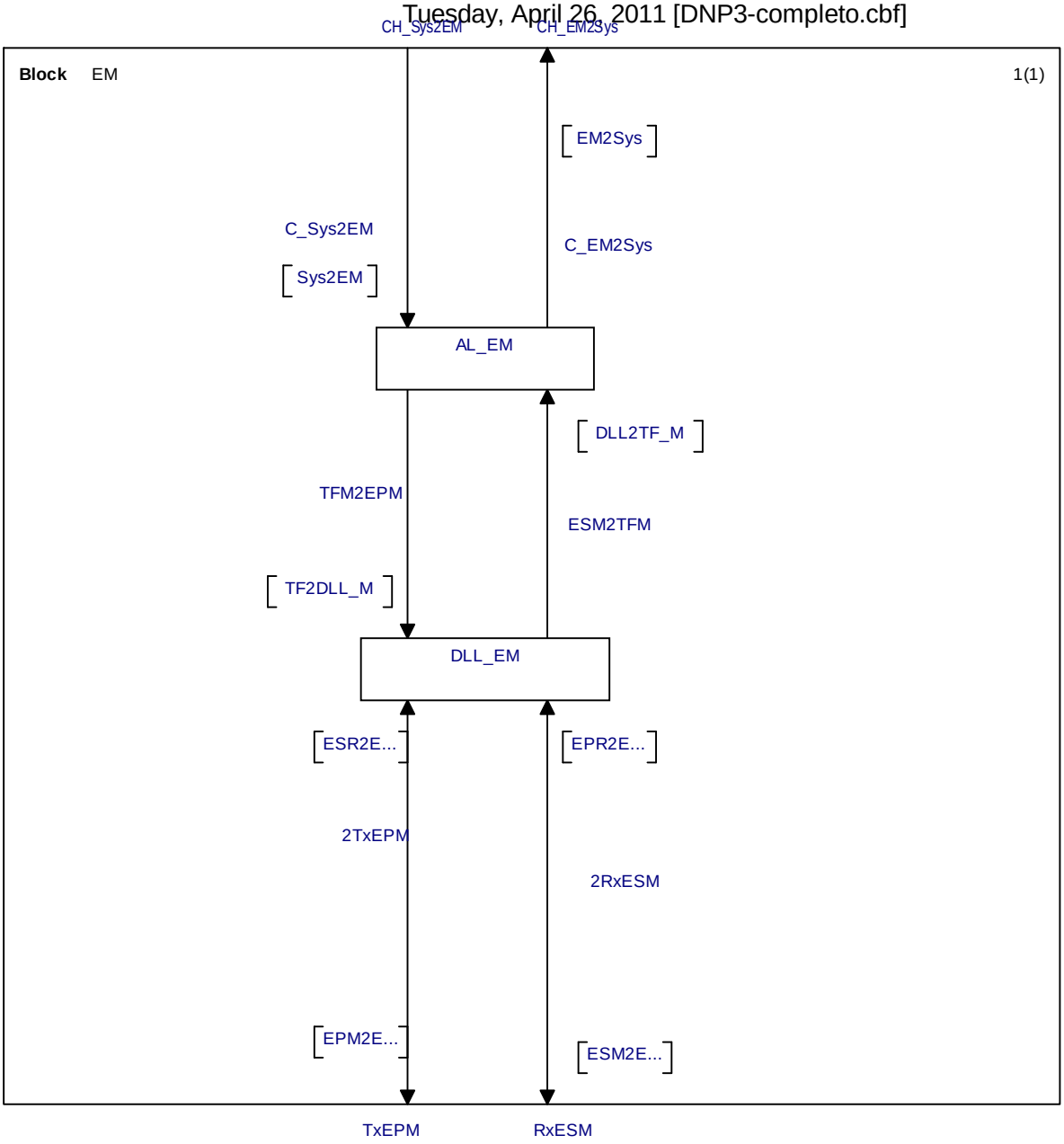
```
Synonym MaxRein Reintentos=2;
Synonym Tmax_DLL Integer=16;
```

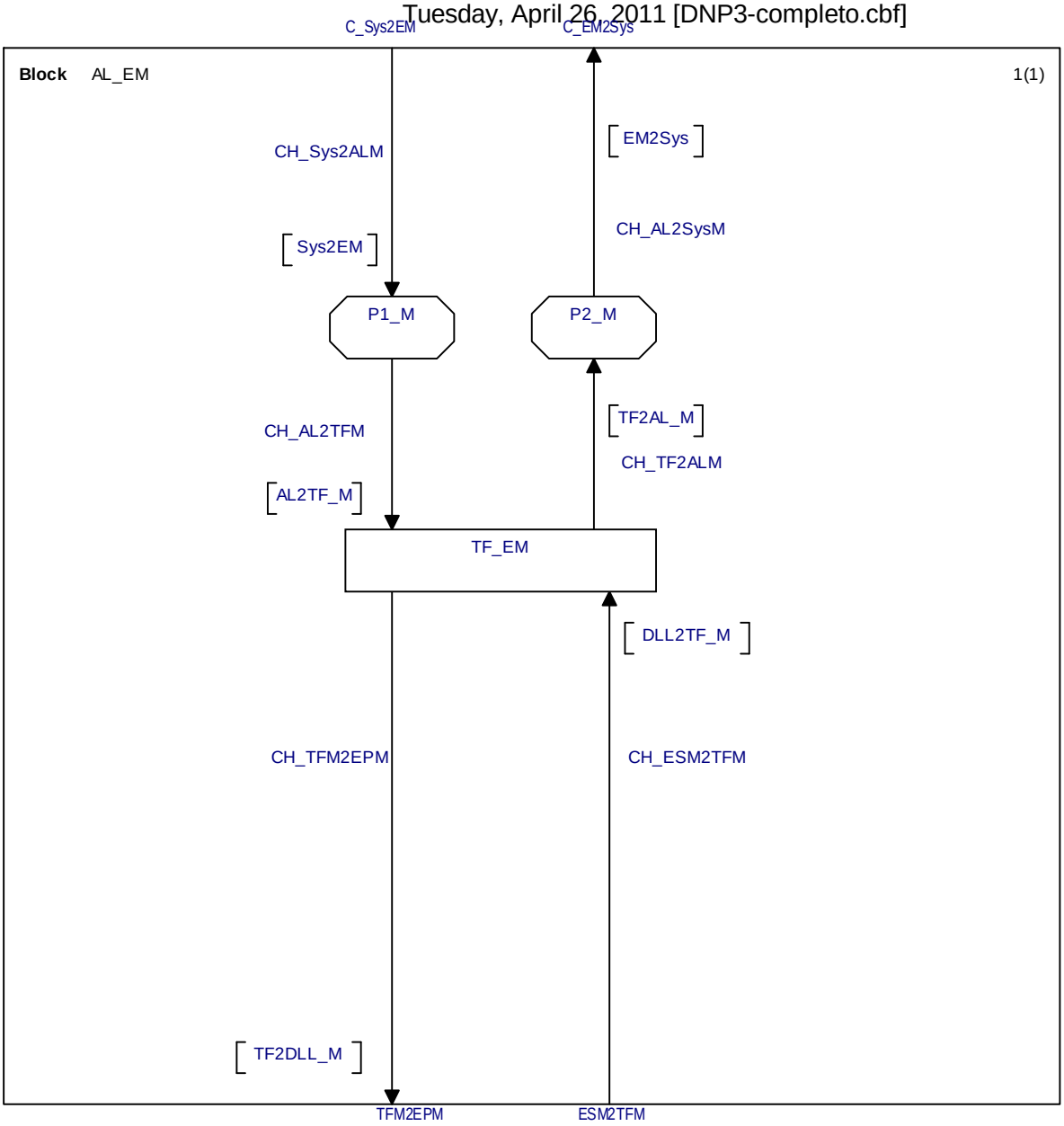

Package DNP3_Datos 3(3)
/*Datos para la TF*/
/*Header TF*/
newtype H_TF
struct
FIN Bit_t;
FIR Bit_t;
Seq BitString;
default(. '1'B,'1'B,'000000'B
.)
endnewtype;
Synonym Sq Integer = 64;
Synonym Tmax_Frag Integer = 249;

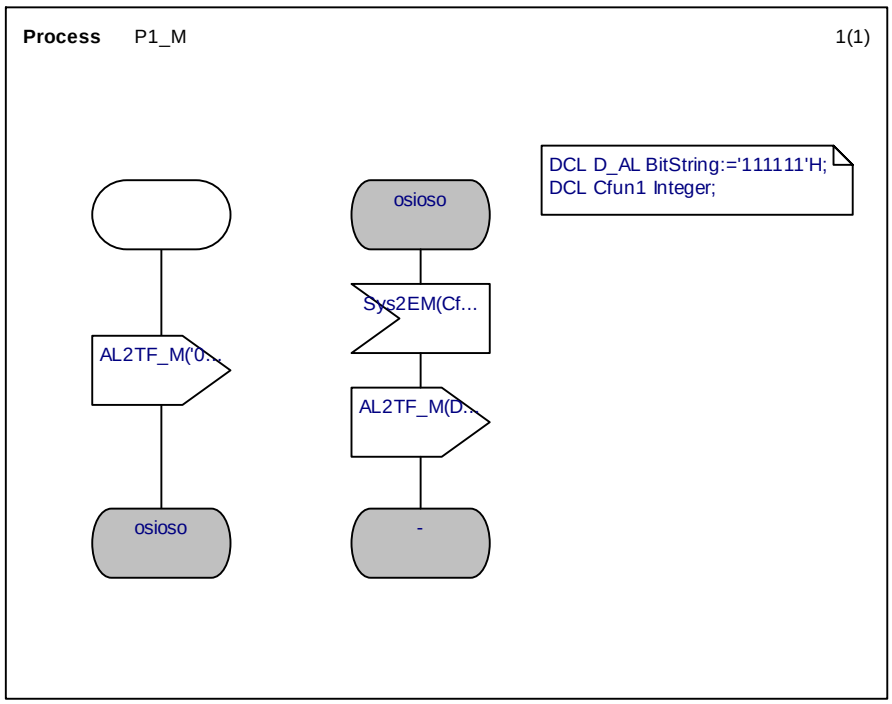

Package DNP3_Tipos
1(1)
use DNP3_Datos;
use DNP3_Procedimientos;
use DNP3_Tipos;
use DNP3_Signals;
use DNP3_Timers;
/*Process Type del medio*/
M1
/*System Type del sistema*/
system DNP3Protocol

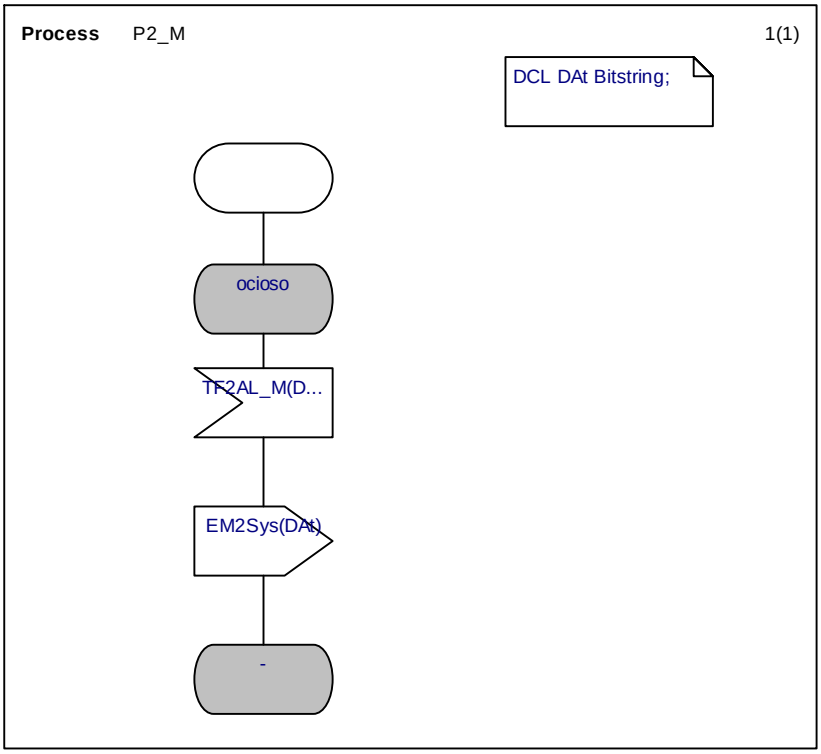

Process Type M1
1(1)
Recibe
DCL q BitString;
Timer TBus :=0;
DCL D Duration := DBus;
gm1
ESR2E...
EPM2E...
gm4
ESR2E...
EPM2E...
EPM2ESR (q)
ESR2EPM (q)
ESM2EPR (q)
EPR2ESM (q)
set(now+D,TBus)
set(now+D,TBus)
set(now+D,TBus)
set(now+D,TBus)
espera1
espera2
espera3
espera4
gm2
EPR2E...
ESM2E...
EPR2E...
gm3
ESM2E...
espera1
espera2
espera3
espera4
TBus
TBus
TBus
TBus
reset(TBus)
reset(TBus)
reset(TBus)
reset(TBus)
EPM2ESR(q) via gm4
ESR2EPM(q) via gm1
ESM2EPR(q) via gm3
EPR2ESM(q) via gm2
recibe

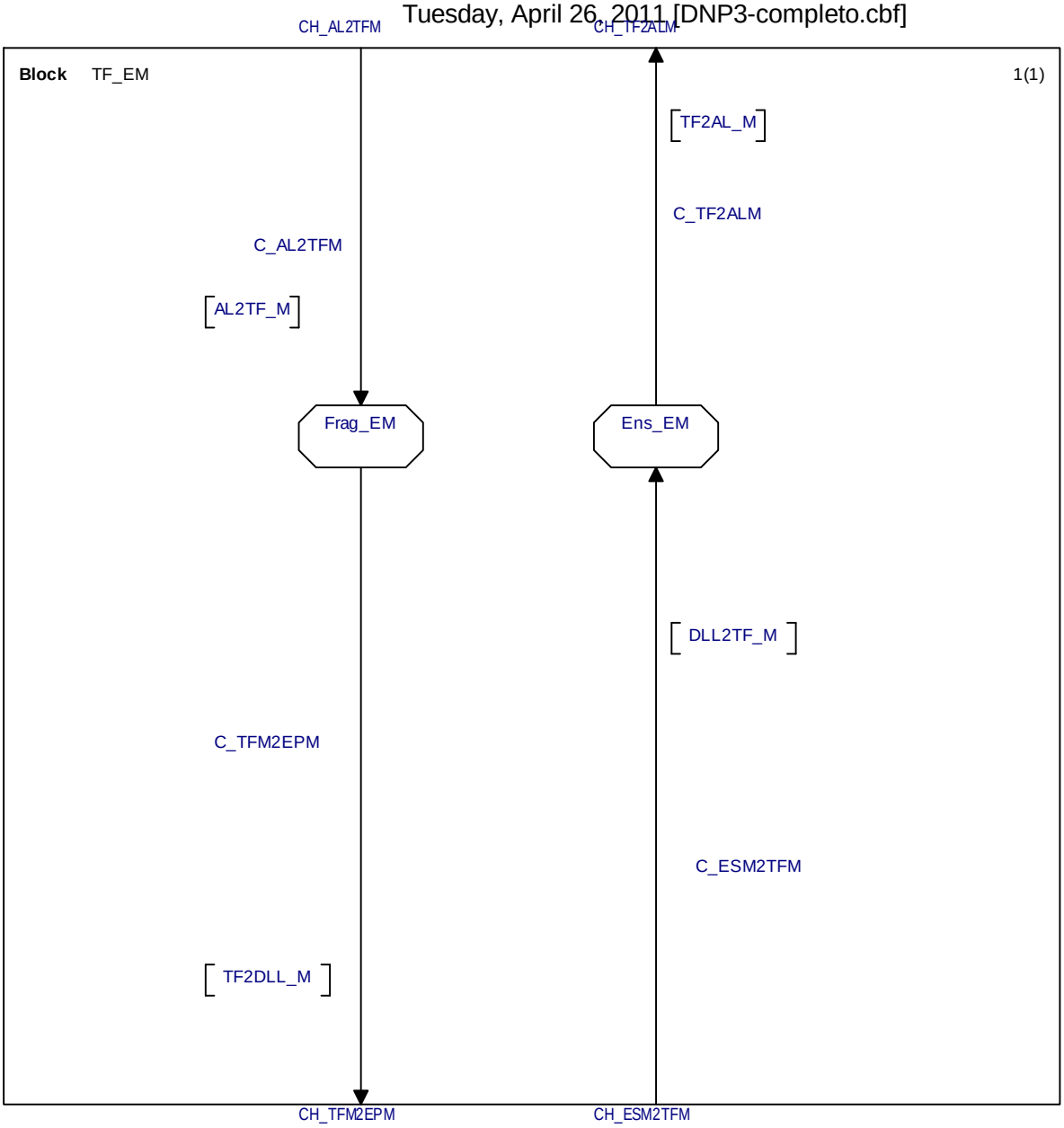

System Type
DNP3Protocol
1(1)
EM2Sys
ER2Sys
CH_Sys2EM
CH_Sys2ER
Sys2EM
CH_EM2Sys
CH_ER2Sys
Sys2ER
EM
ER
ESR2E...
EPR2E...
EPM2E...
ESM2E...
TxEPM
RxESM
RxESR
TxEPR
EPM2E...
ESM2E...
ESR2E...
EPR2E...
medio

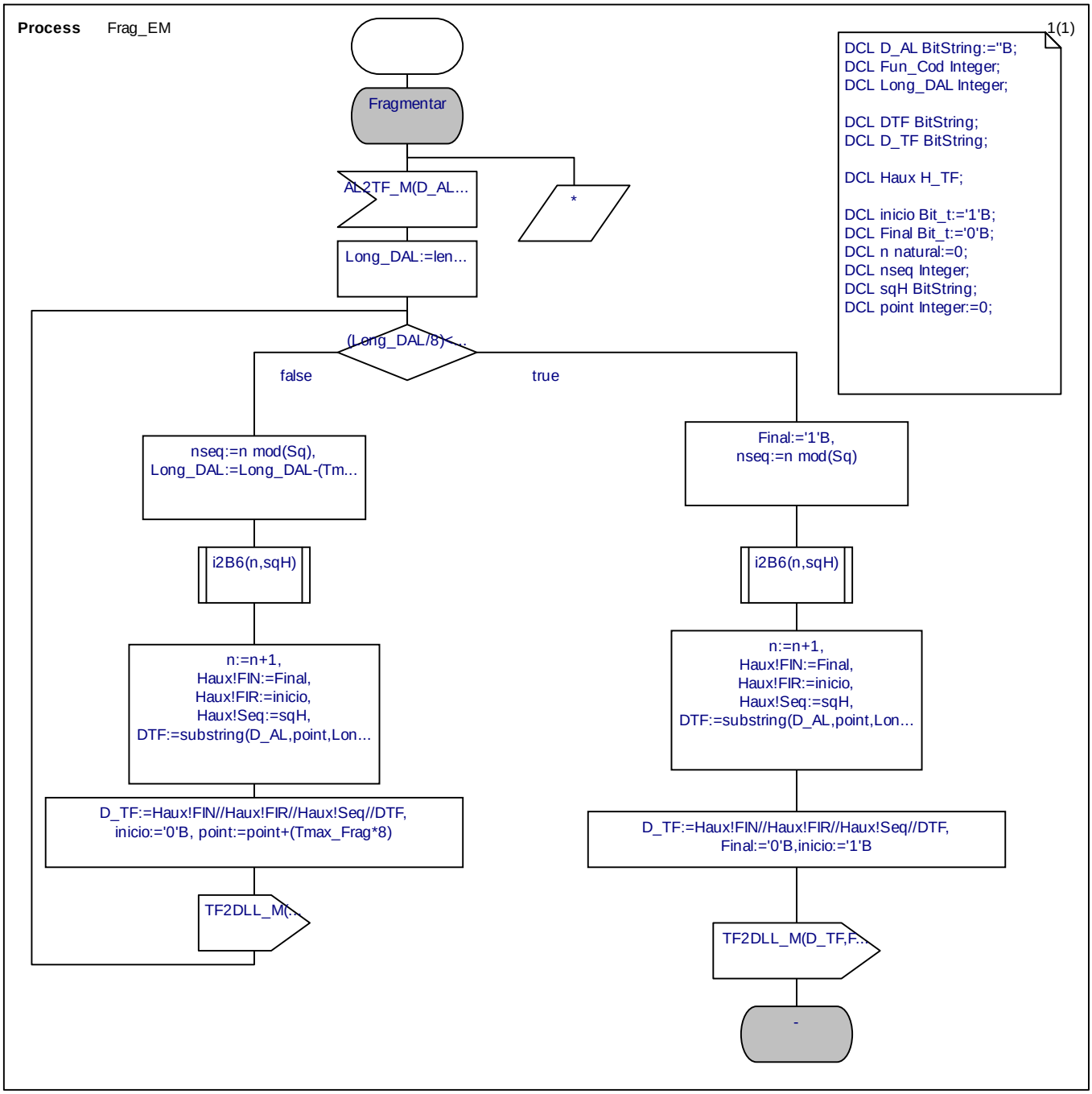

CH_Sys2EM
CH_EM2Sys
Block EM
1(1)
EM2Sys
C_Sys2EM
C_EM2Sys
Sys2EM
AL_EM
DLL2TF_M
TFM2EPM
ESM2TFM
TF2DLL_M
DLL_EM
ESR2E...
EPR2E...
2TxEPM
2RxESM
EPM2E...
ESM2E...
TxEPM
RxESM


C_Sys2EM
C_EM2Sys
Block AL_EM
1(1)
CH_Sys2ALM
EM2Sys
CH_AL2SysM
Sys2EM
P1_M
P2_M
CH_AL2TFM
TF2AL_M
CH_TF2ALM
AL2TF_M
TF_EM
DLL2TF_M
CH_TFM2EPM
CH_ESM2TFM
TF2DLL_M
TFM2EPM
ESM2TFM

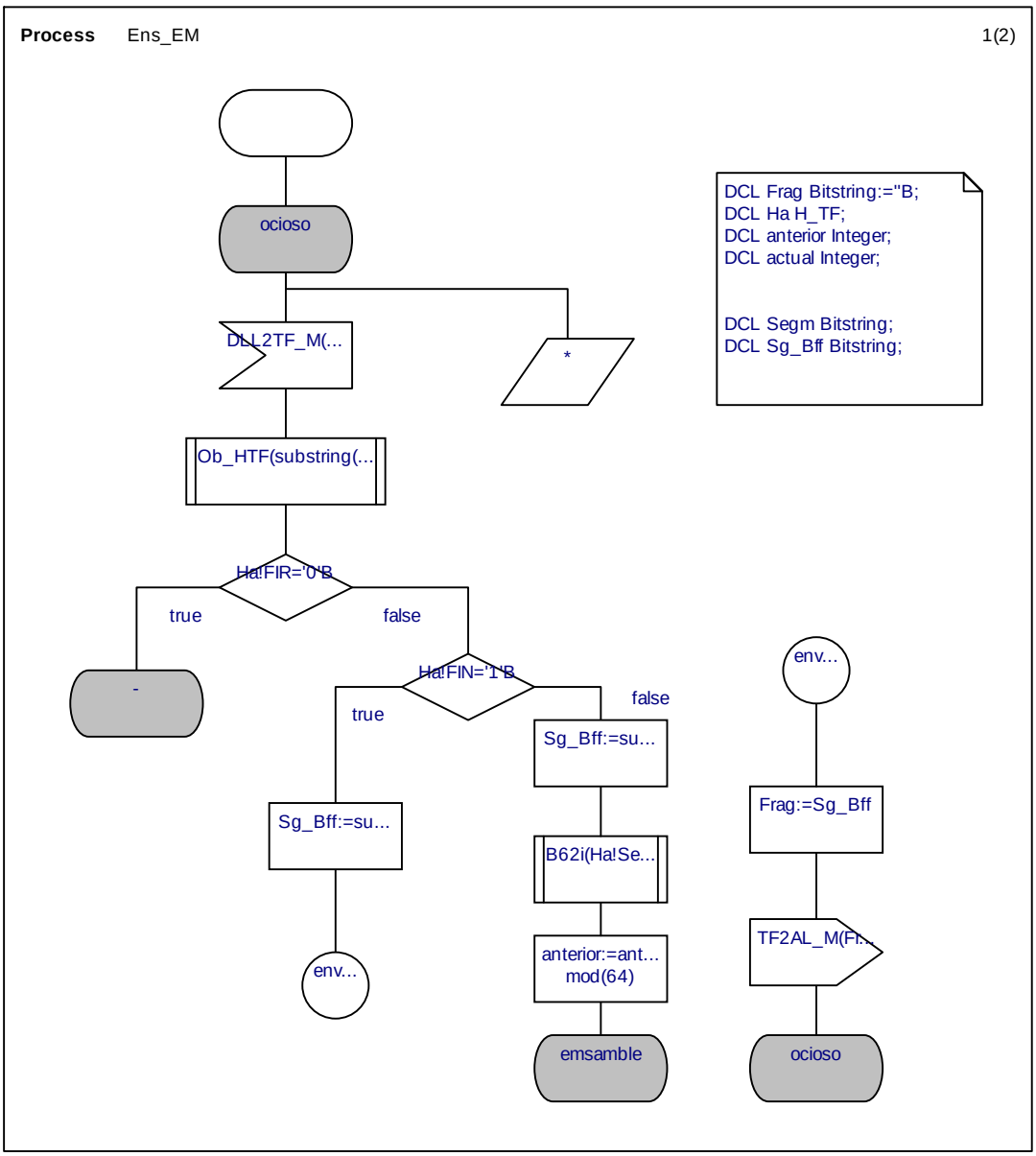

Process P1_M
1(1)
DCL D_AL BitString:='111111'H;
DCL Cfun1 Integer;
osioso
Sys2EM(Cf...
AL2TF_M('0...
AL2TF_M(D...
osioso
-


Process P2_M
1(1)
DCL DAt Bitstring;
ocioso
TF2AL_M(D...
EM2Sys(DAt)
-

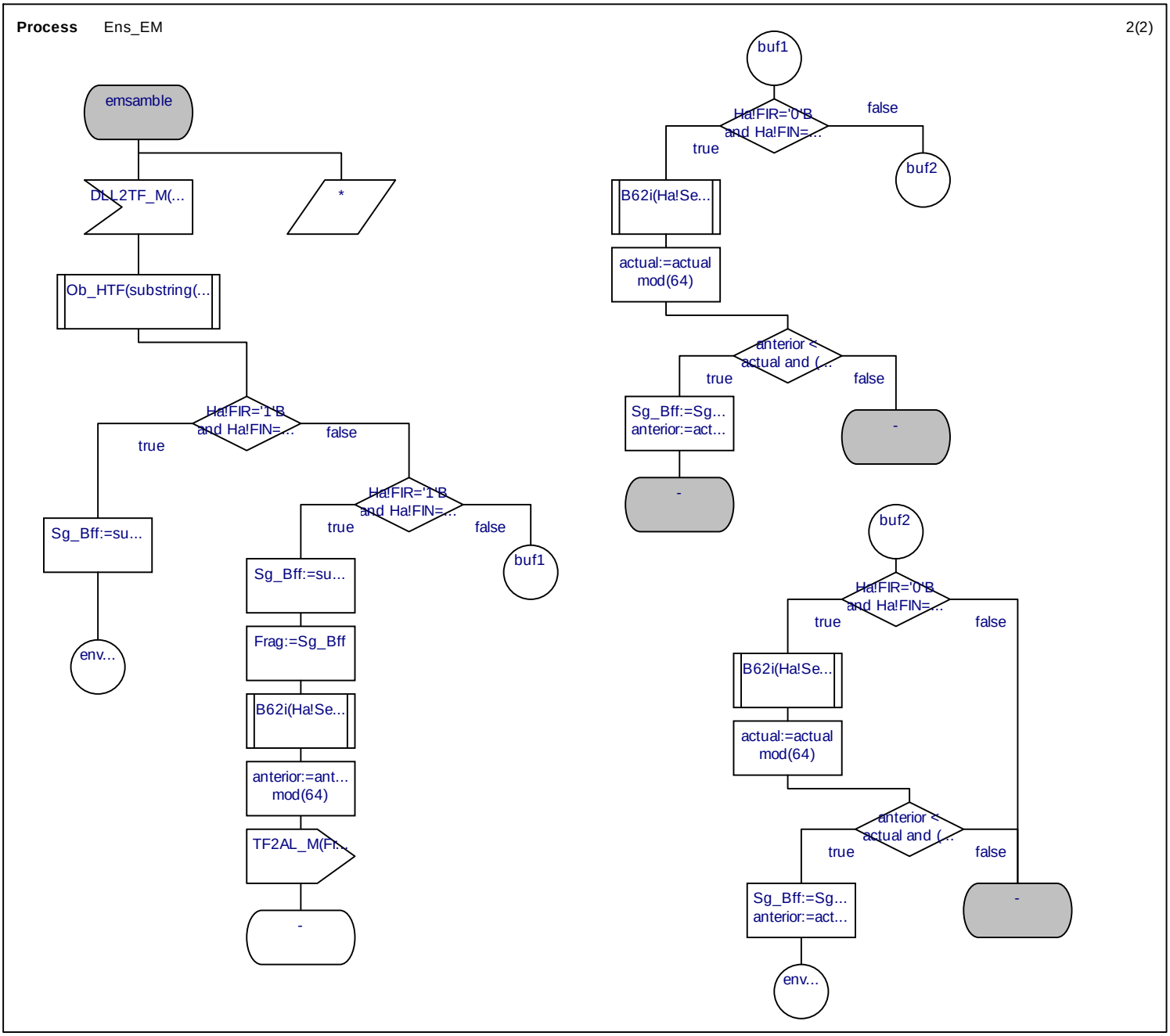

CH_AL2TFM
CH_TF2ALM
Block TF_EM
1(1)
TF2AL_M
C_TF2ALM
C_AL2TFM
AL2TF_M
Frag_EM
Ens_EM
DLL2TF_M
C_TFM2EPM
C_ESM2TFM
TF2DLL_M
CH_TFM2EPM
CH_ESM2TFM

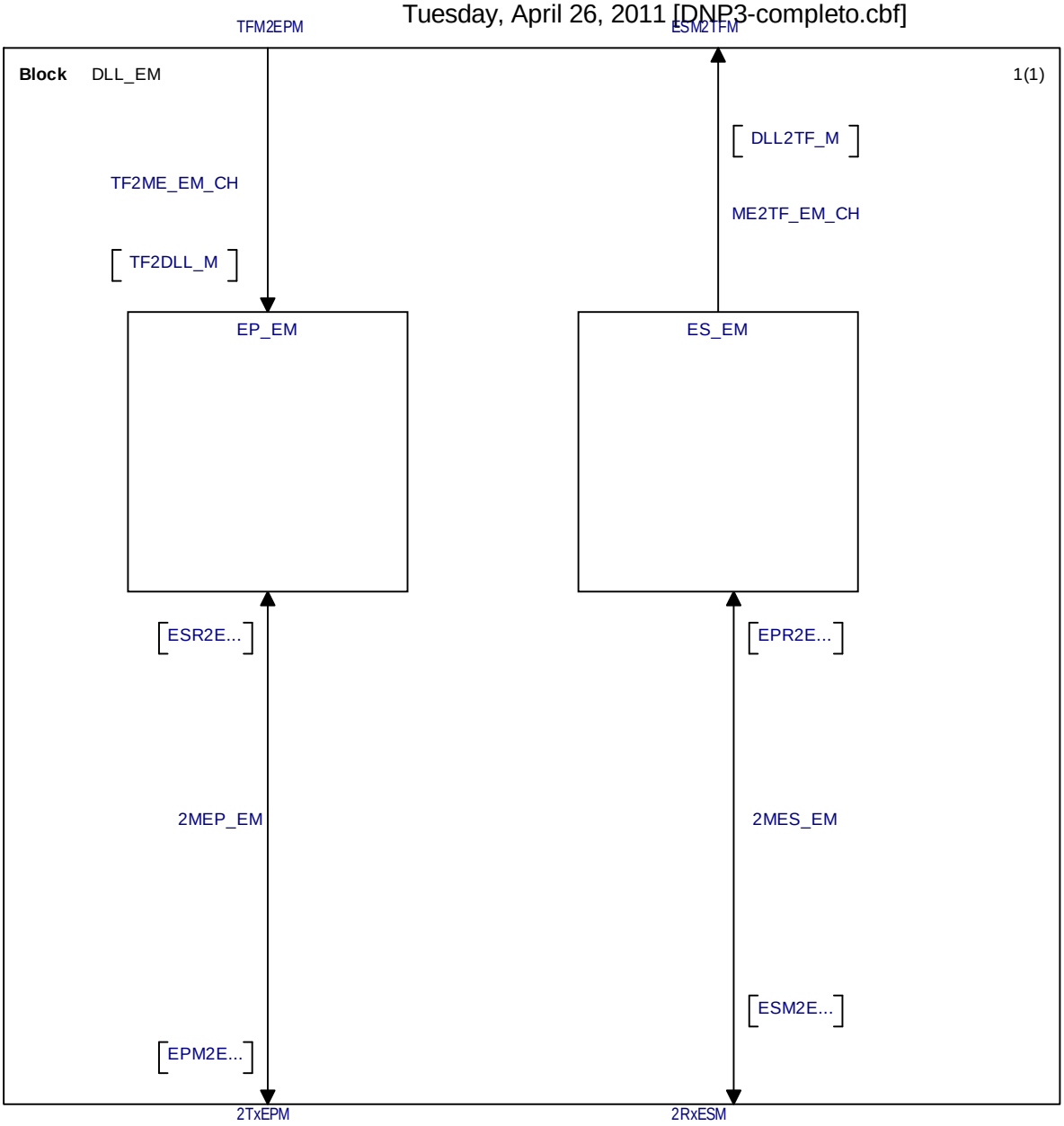

Process Frag_EM
1(1)
DCL D_AL BitString:="B;
DCL Fun_Cod Integer;
DCL Long_DAL Integer;
DCL DTF BitString;
DCL D_TF BitString;
DCL Haux H_TF;
DCL inicio Bit_t:='1'B;
DCL Final Bit_t:='0'B;
DCL n natural:=0;
DCL nseq Integer;
DCL sqH BitString;
DCL point Integer:=0;
Fragmentar
AL2TF_M(D_AL...
*
Long_DAL:=len...
(Long_DAL/8)<...
false
true
nseq:=n mod(Sq),
Long_DAL:=Long_DAL-(Tm...
i2B6(n,sqH)
n:=n+1,
Haux!FIN:=Final,
Haux!FIR:=inicio,
Haux!Seq:=sqH,
DTF:=substring(D_AL,point,Lon...
D_TF:=Haux!FIN//Haux!FIR//Haux!Seq//DTF,
inicio:='0'B, point:=point+(Tmax_Frag*8)
TF2DLL_M(...
Final:='1'B,
nseq:=n mod(Sq)
i2B6(n,sqH)
n:=n+1,
Haux!FIN:=Final,
Haux!FIR:=inicio,
Haux!Seq:=sqH,
DTF:=substring(D_AL,point,Lon...
D_TF:=Haux!FIN//Haux!FIR//Haux!Seq//DTF,
Final:='0'B,inicio:='1'B
TF2DLL_M(D_TF,F...
-

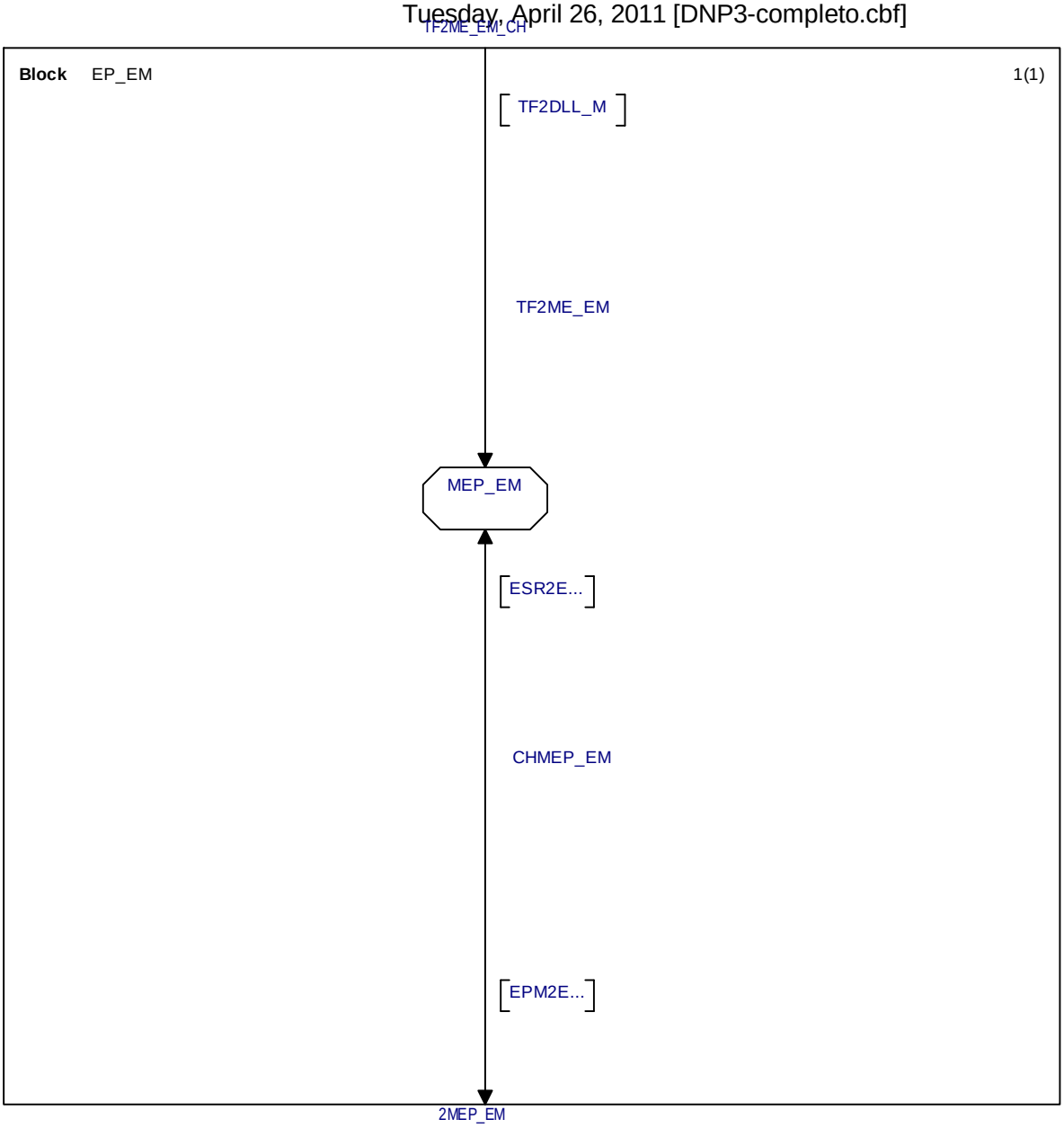
Process Ens_EM 1(2)



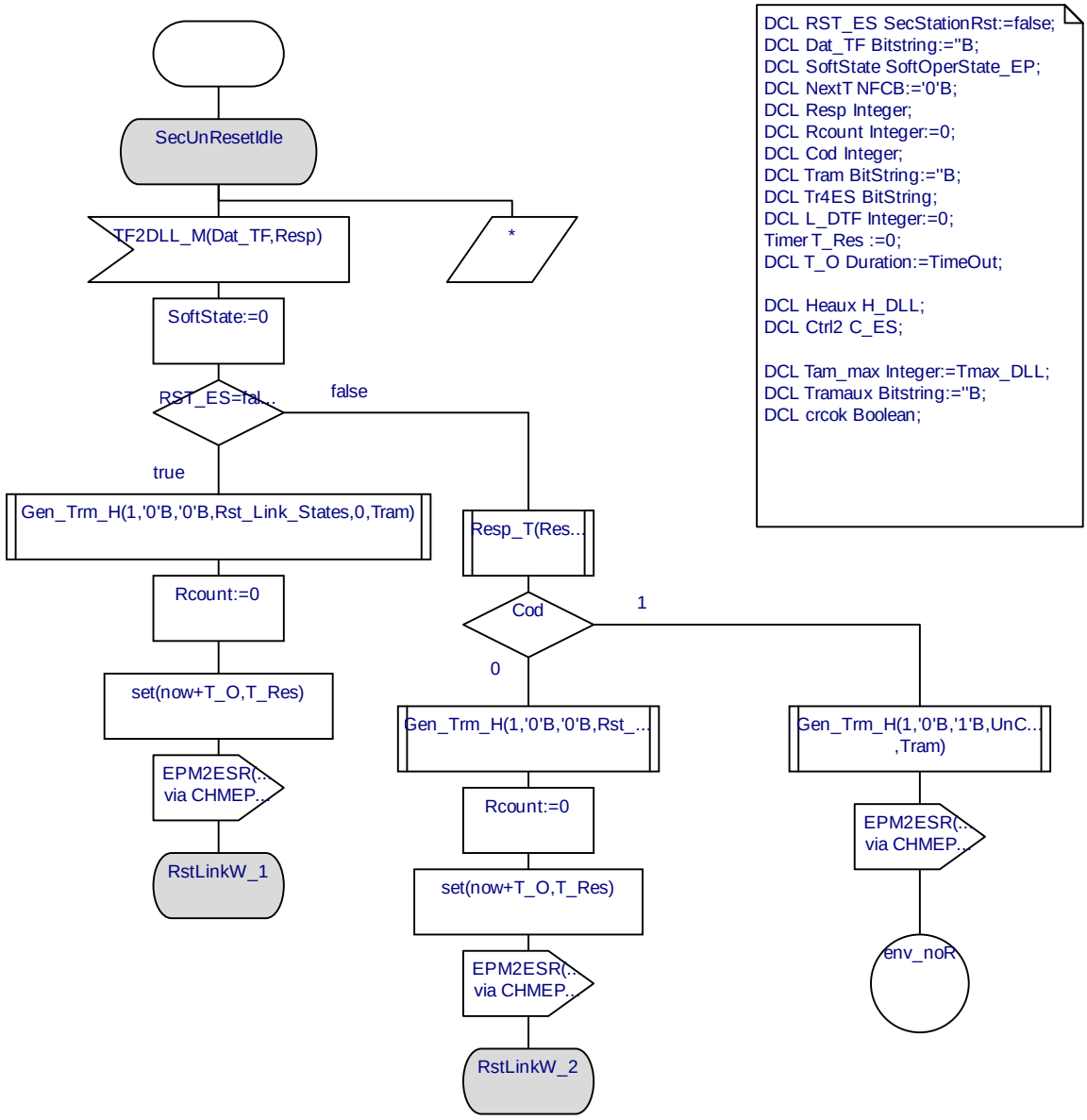
Process Ens_EM 2(2)



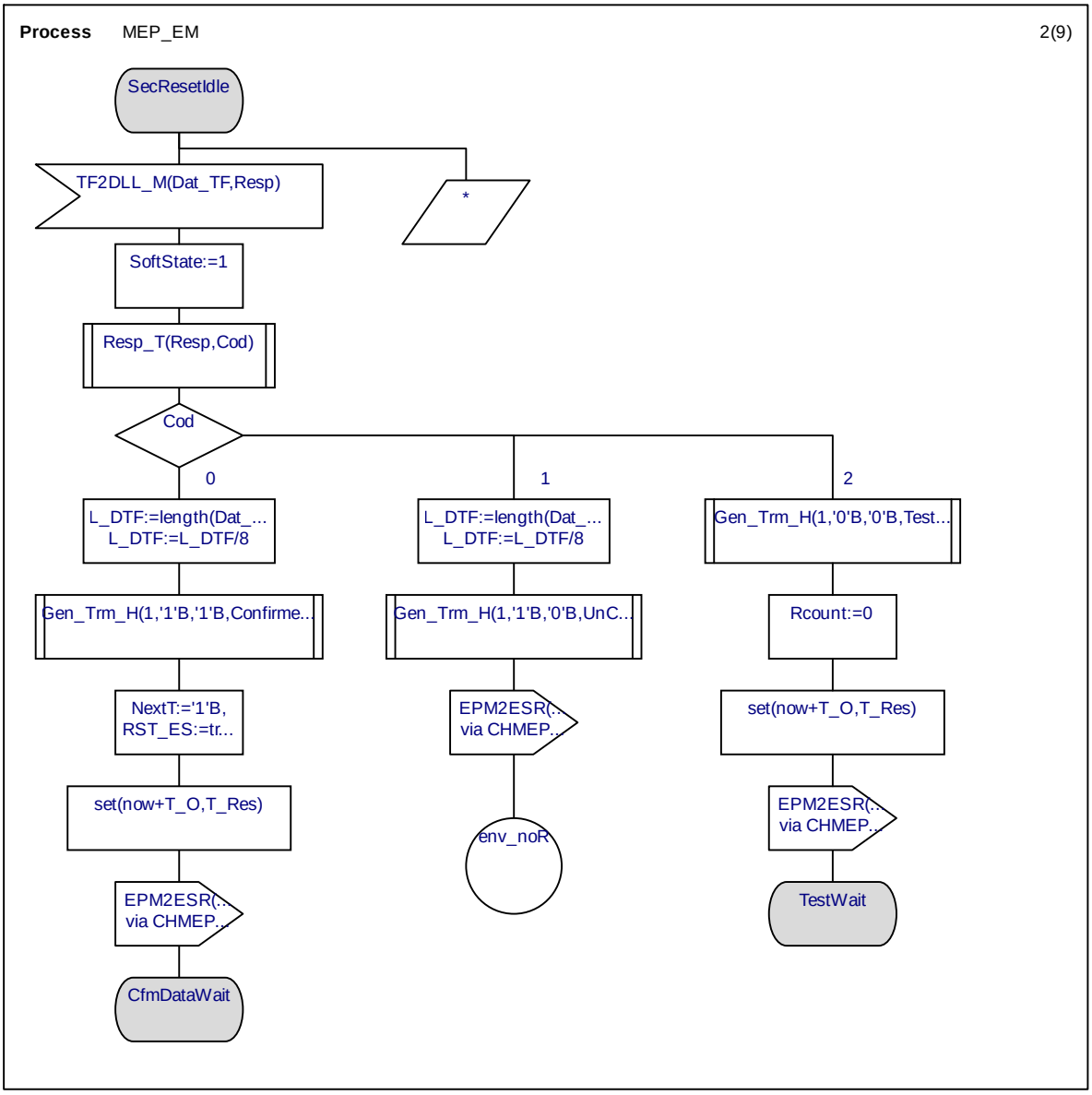

TFM2EPM
ESM2TFM
Block DLL_EM
1(1)
DLL2TF_M
TF2ME_EM_CH
ME2TF_EM_CH
TF2DLL_M
EP_EM
ES_EM
ESR2E...
EPR2E...
2MEP_EM
2MES_EM
ESM2E...
EPM2E...
2TxEPM
2RxESM


TF2ME_EM_CH
Block EP_EM
1(1)
TF2DLL_M
TF2ME_EM
MEP_EM
ESR2E...
CHMEP_EM
EPM2E...
2MEP_EM

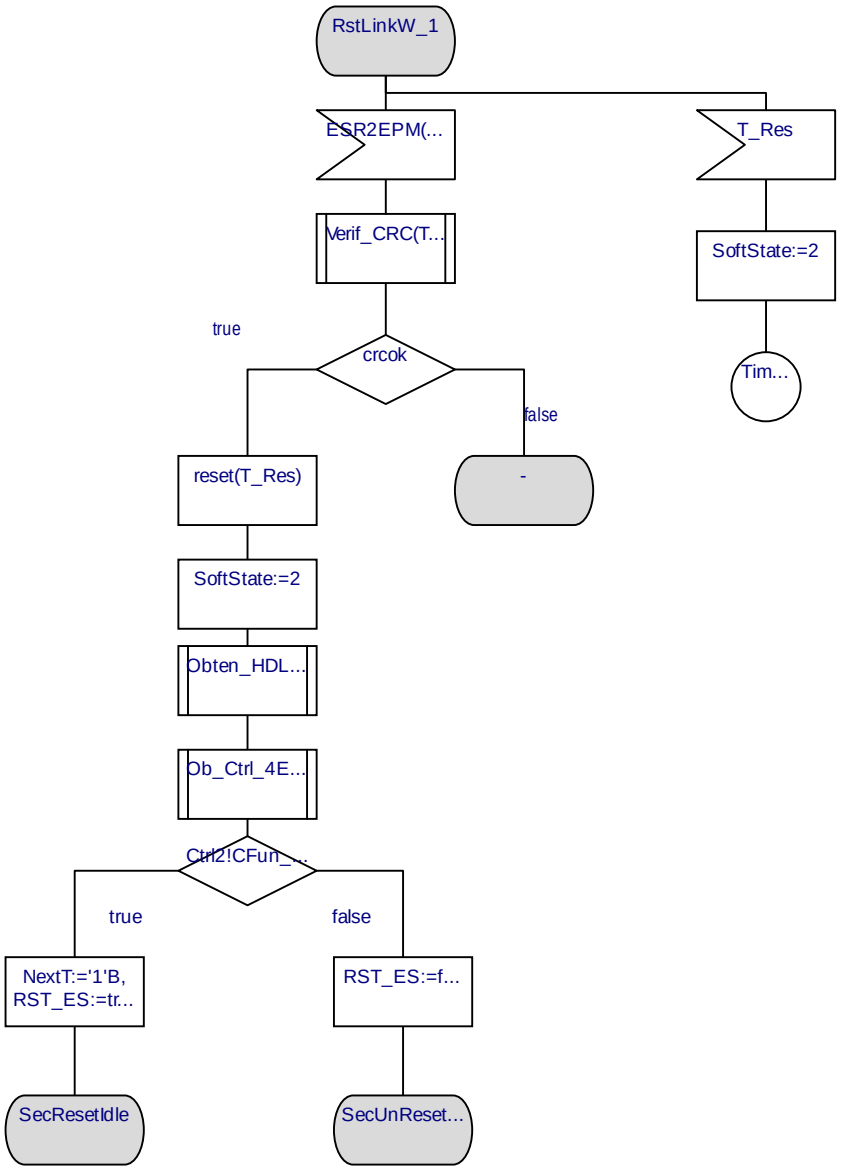
Process MEP_EM 1(9)



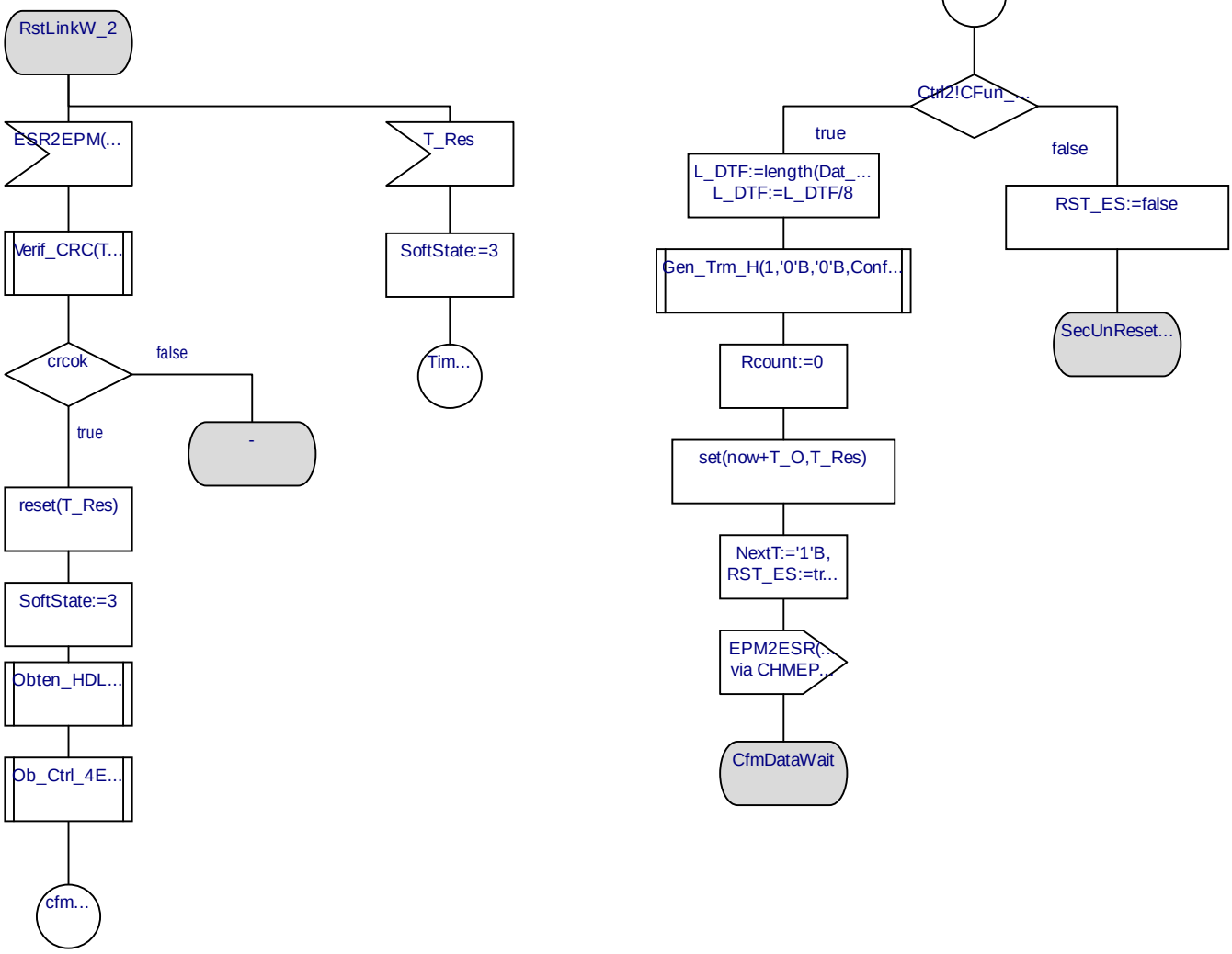
Process MEP_EM 2(9)



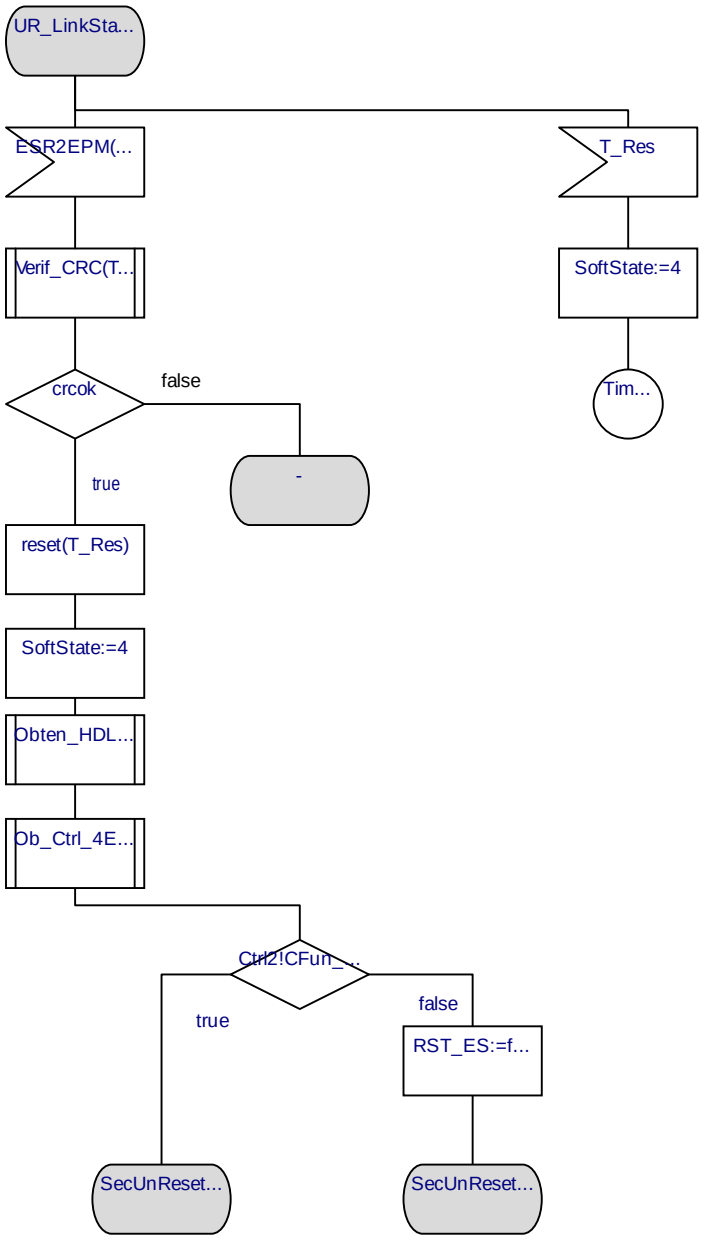
Process MEP_EM 3(9)



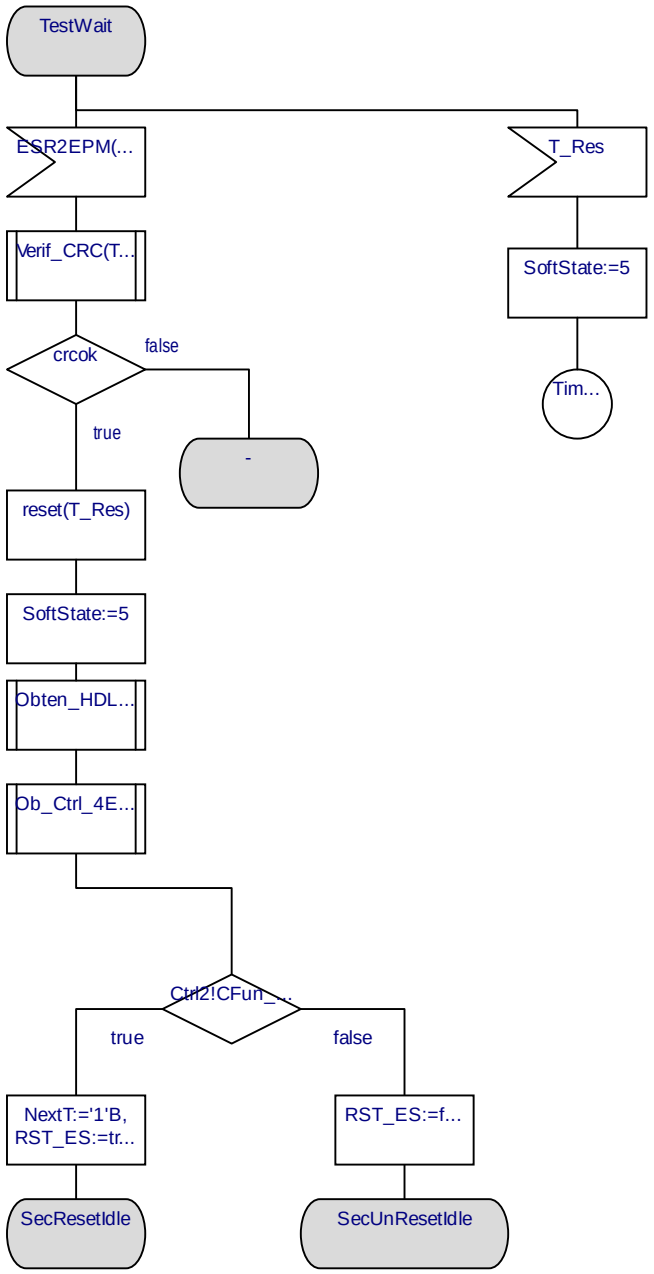
Process MEP_EM 4(9)



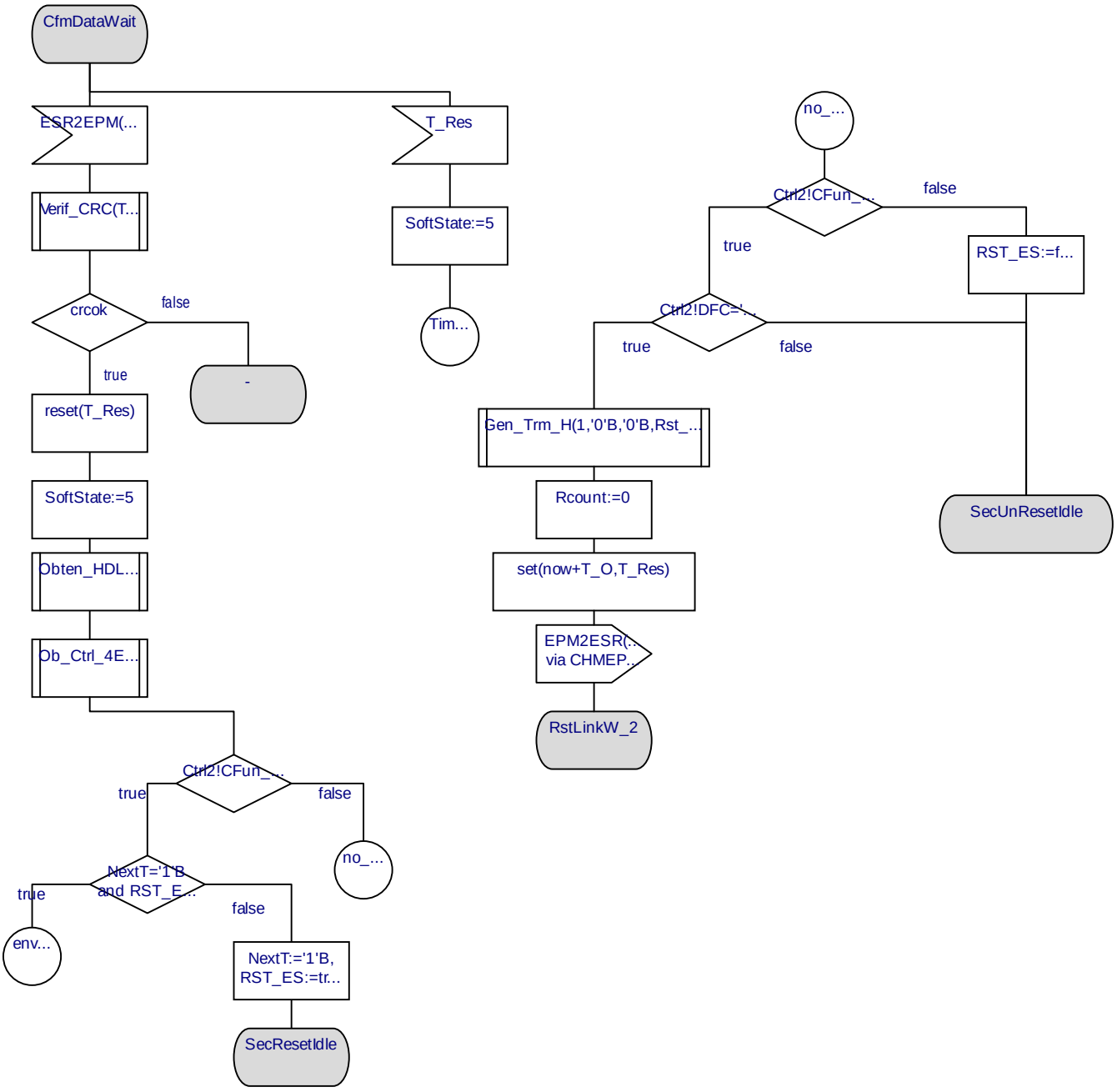
Process MEP_EM 5(9)



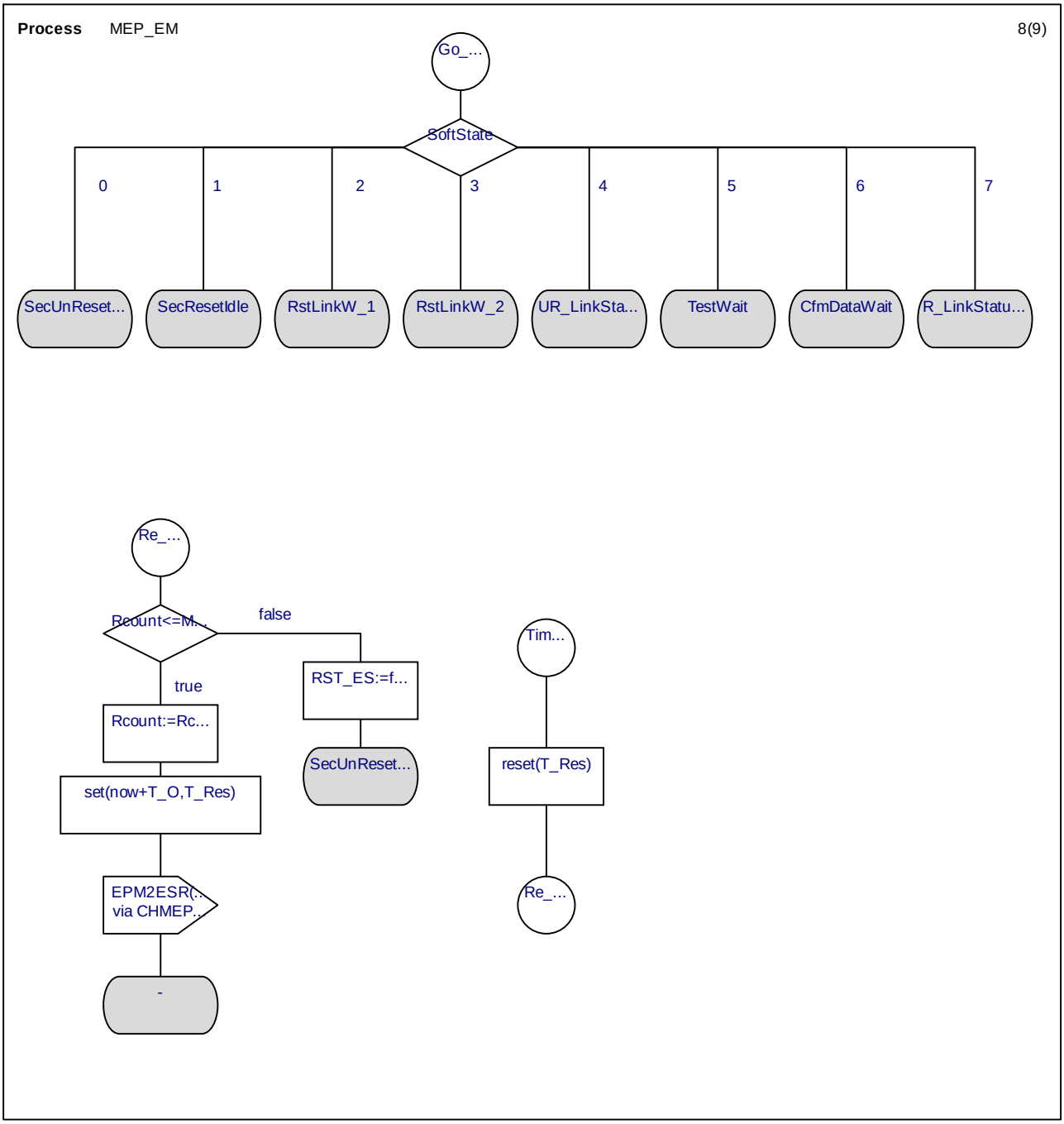
Process MEP_EM 6(9)



Process MEP_EM 7(9)

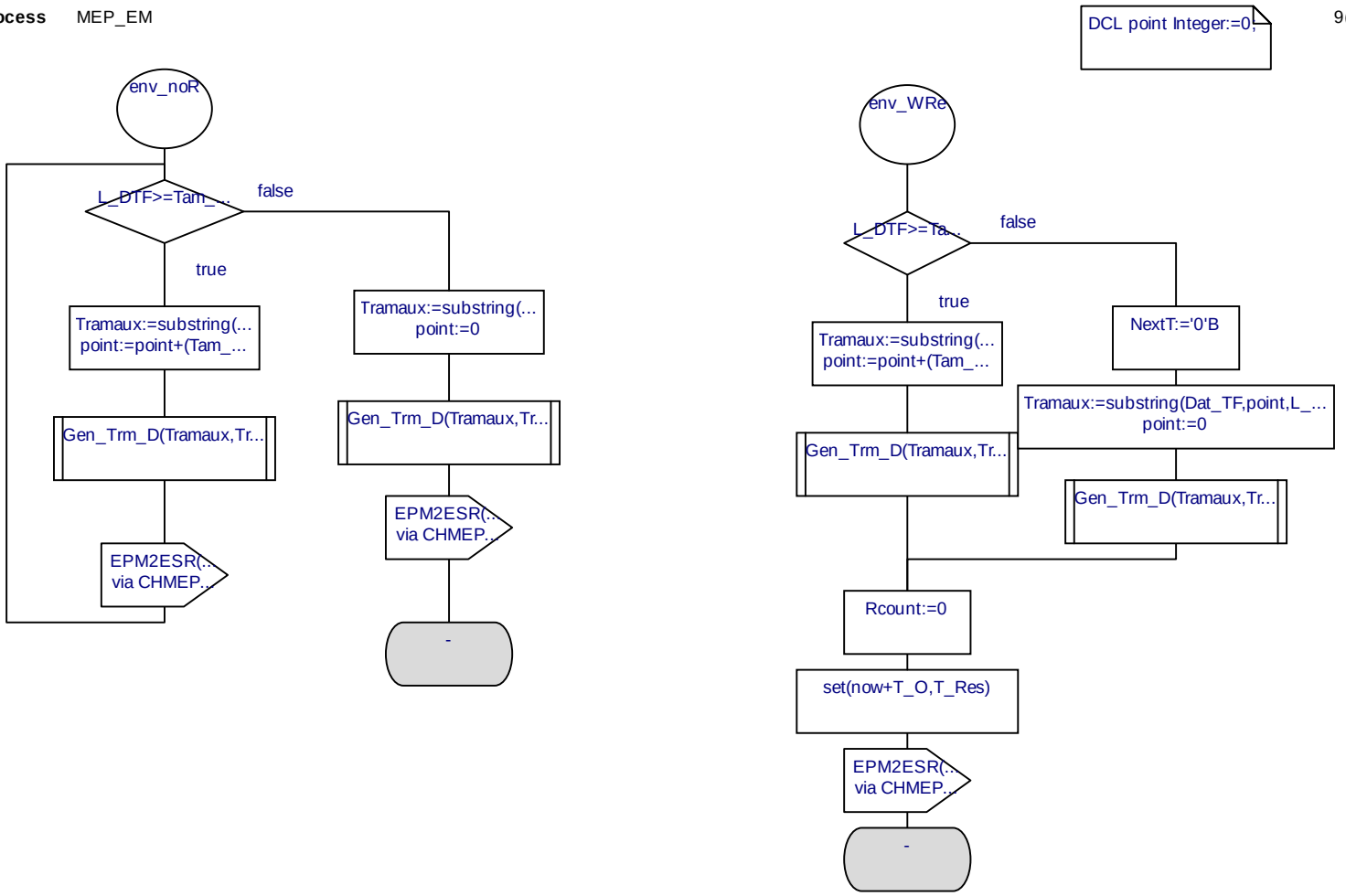


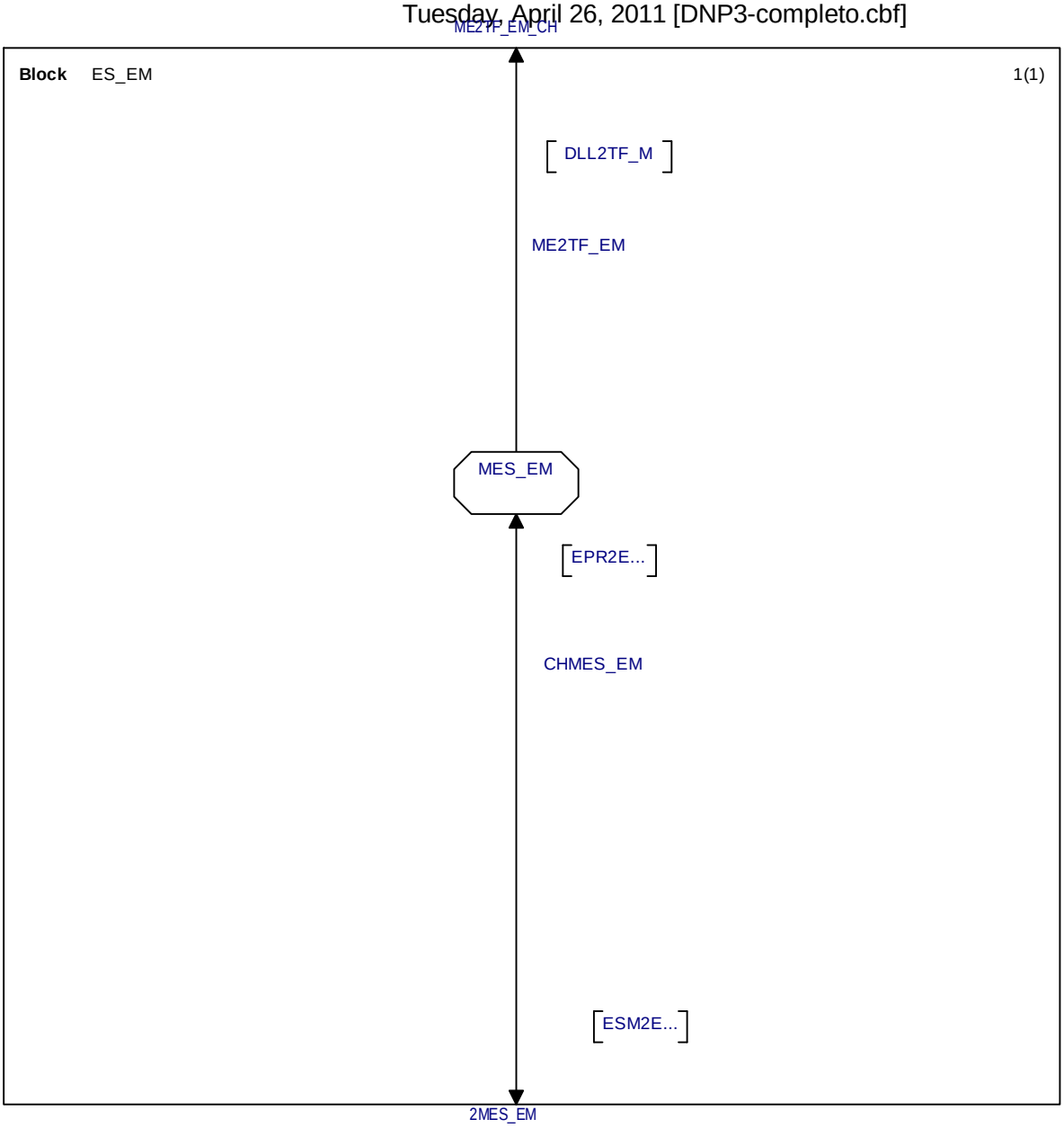
Process MEP_EM 8(9)



Process MEP_EM

9(9)

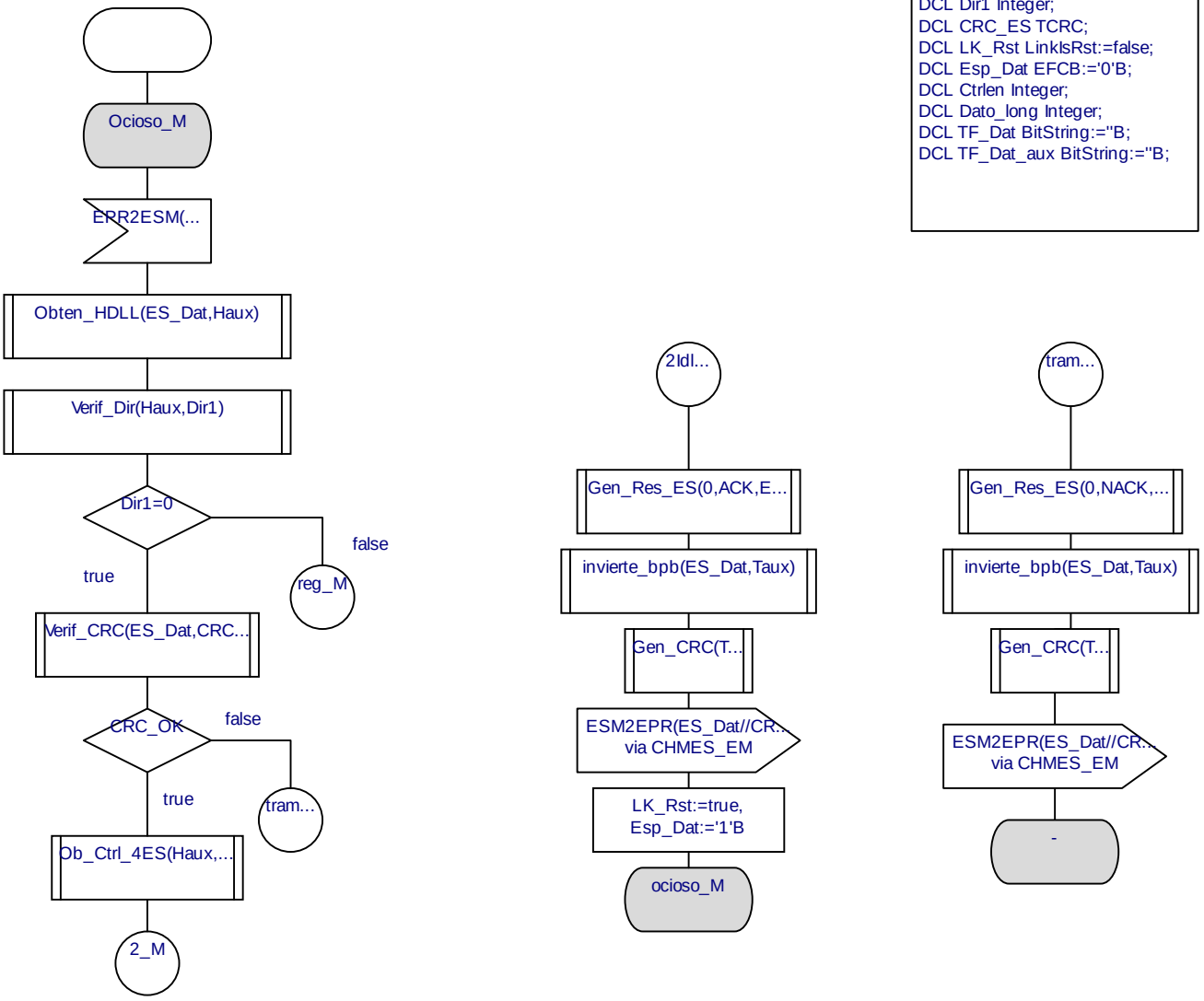



ME2TF_EM_CH
Block ES_EM
1(1)
DLL2TF_M
ME2TF_EM
MES_EM
EPR2E...
CHMES_EM
ESM2E...
2MES_EM

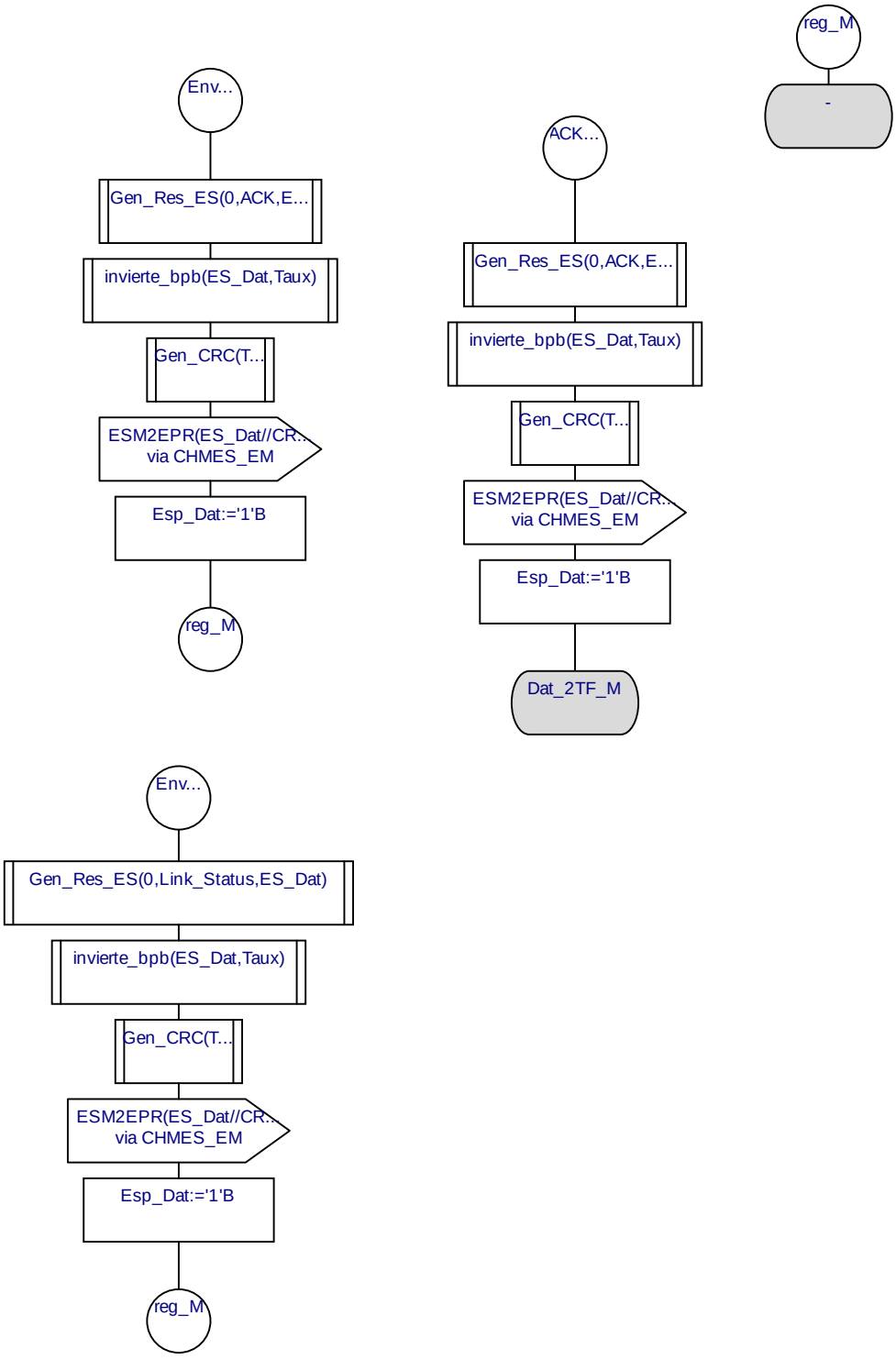
Process MES_EM

1(4)

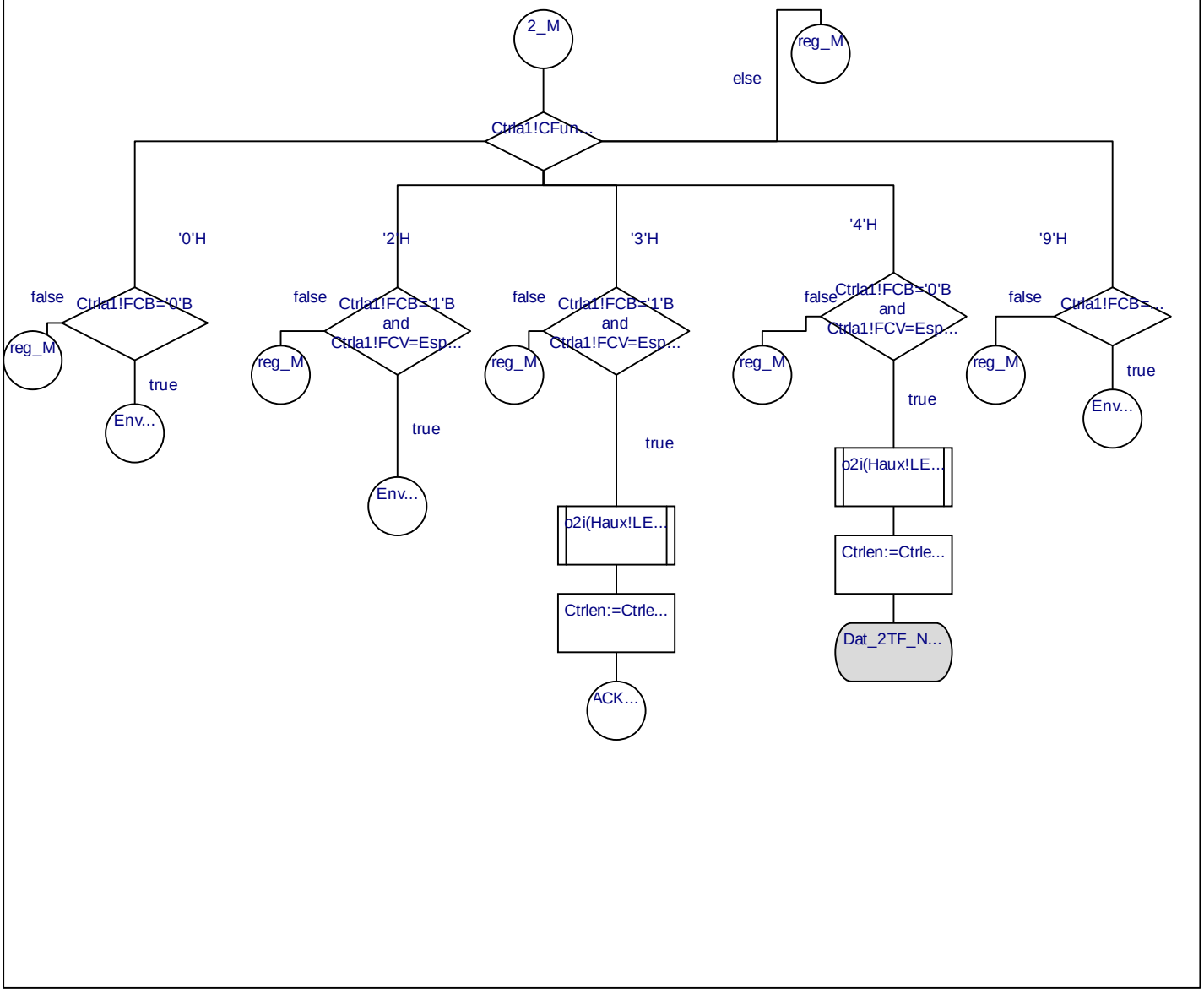
```
DCL ES_Dat BitString;
DCL Taux BitString;
DCL CRC_OK Boolean;
DCL Haux H_DLL;
DCL Ctrla1 C_EP;
DCL Dir1 Integer;
DCL CRC_ES TCRC;
DCL LK_Rst LinkIsRst:=false;
DCL Esp_Dat EFCB:='0'B;
DCL Ctrlen Integer;
DCL Dato_long Integer;
DCL TF_Dat BitString:=''B;
DCL TF_Dat_aux BitString:=''B;
```



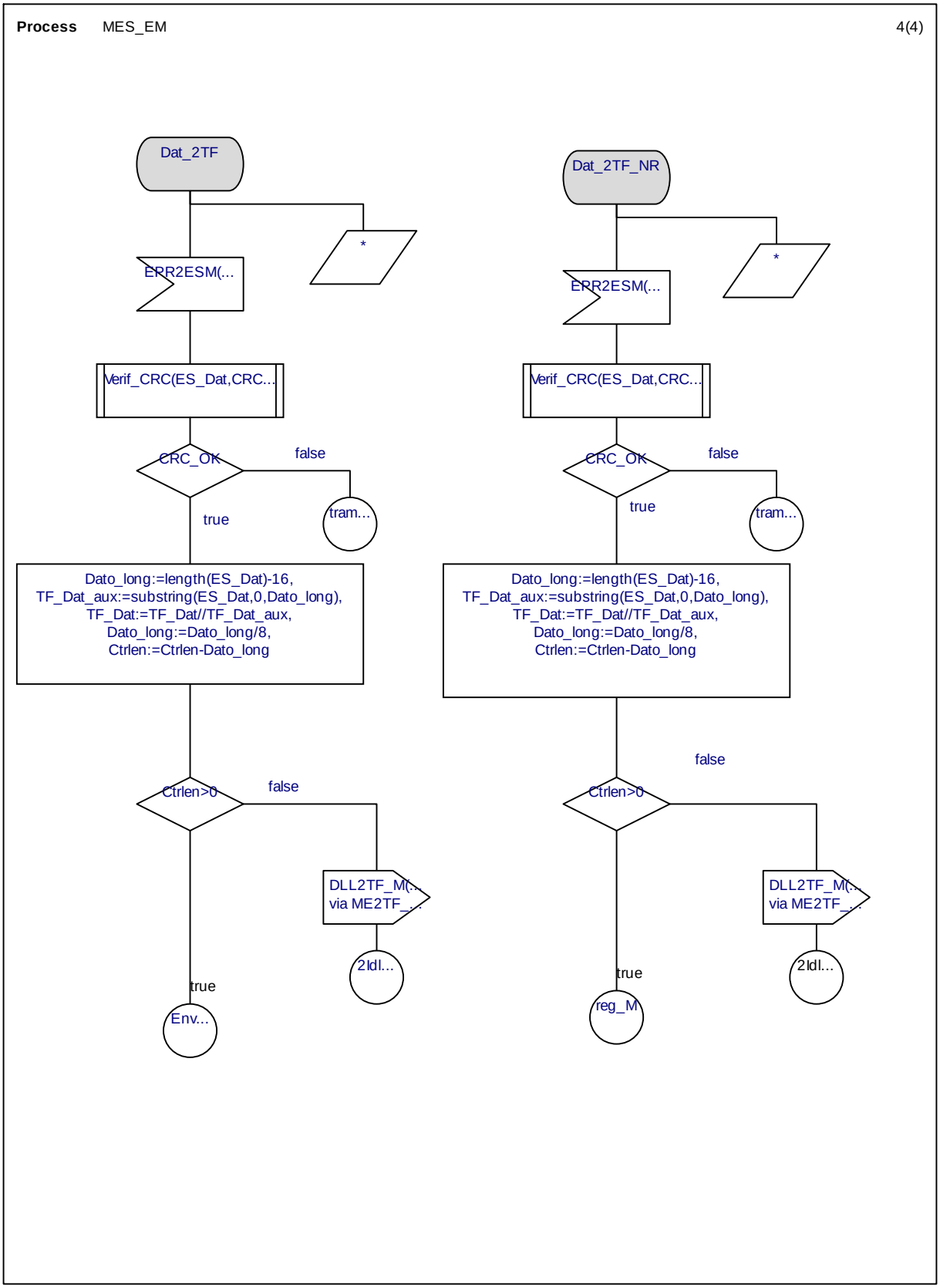
Process MES_EM 2(4)

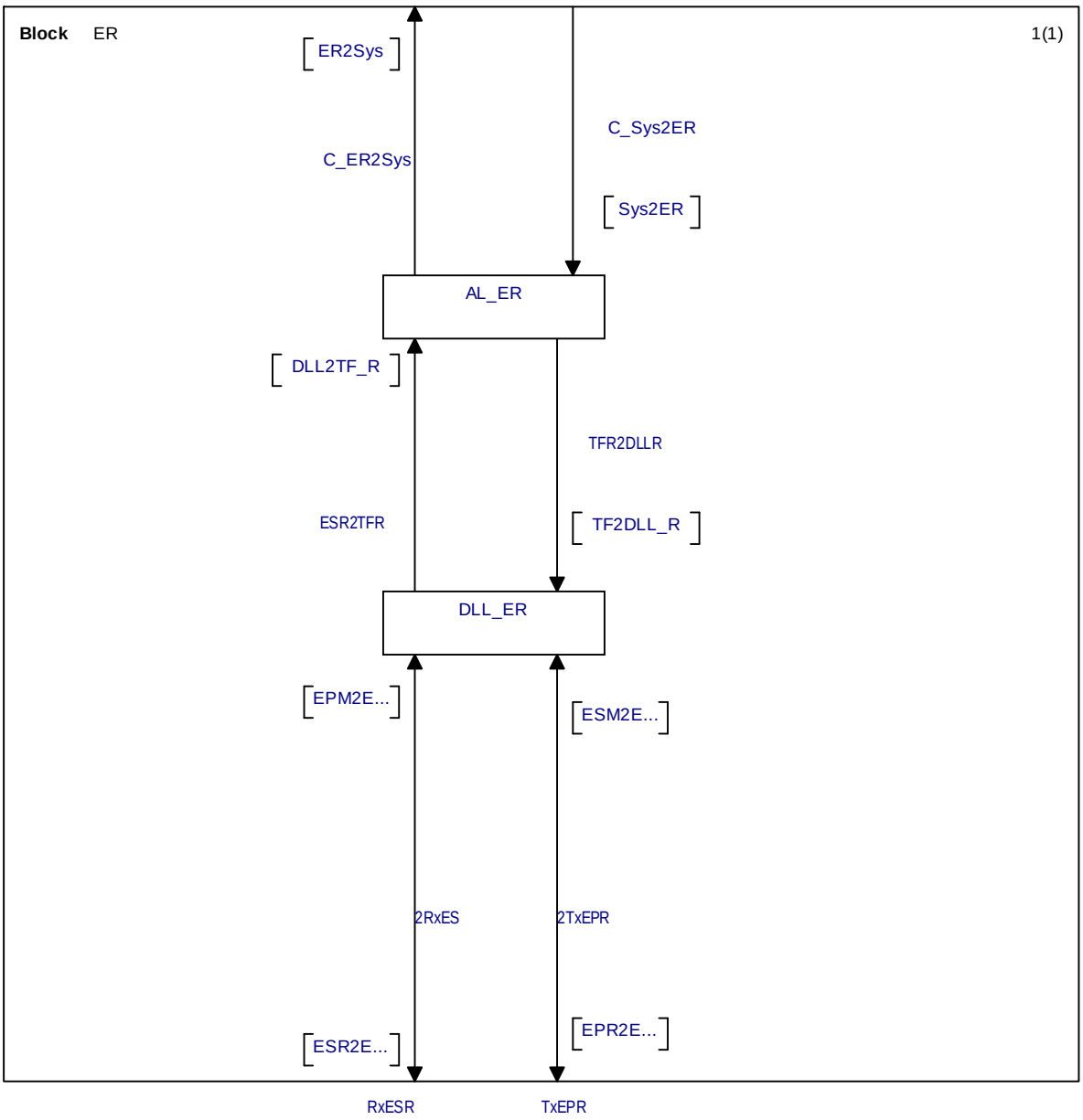


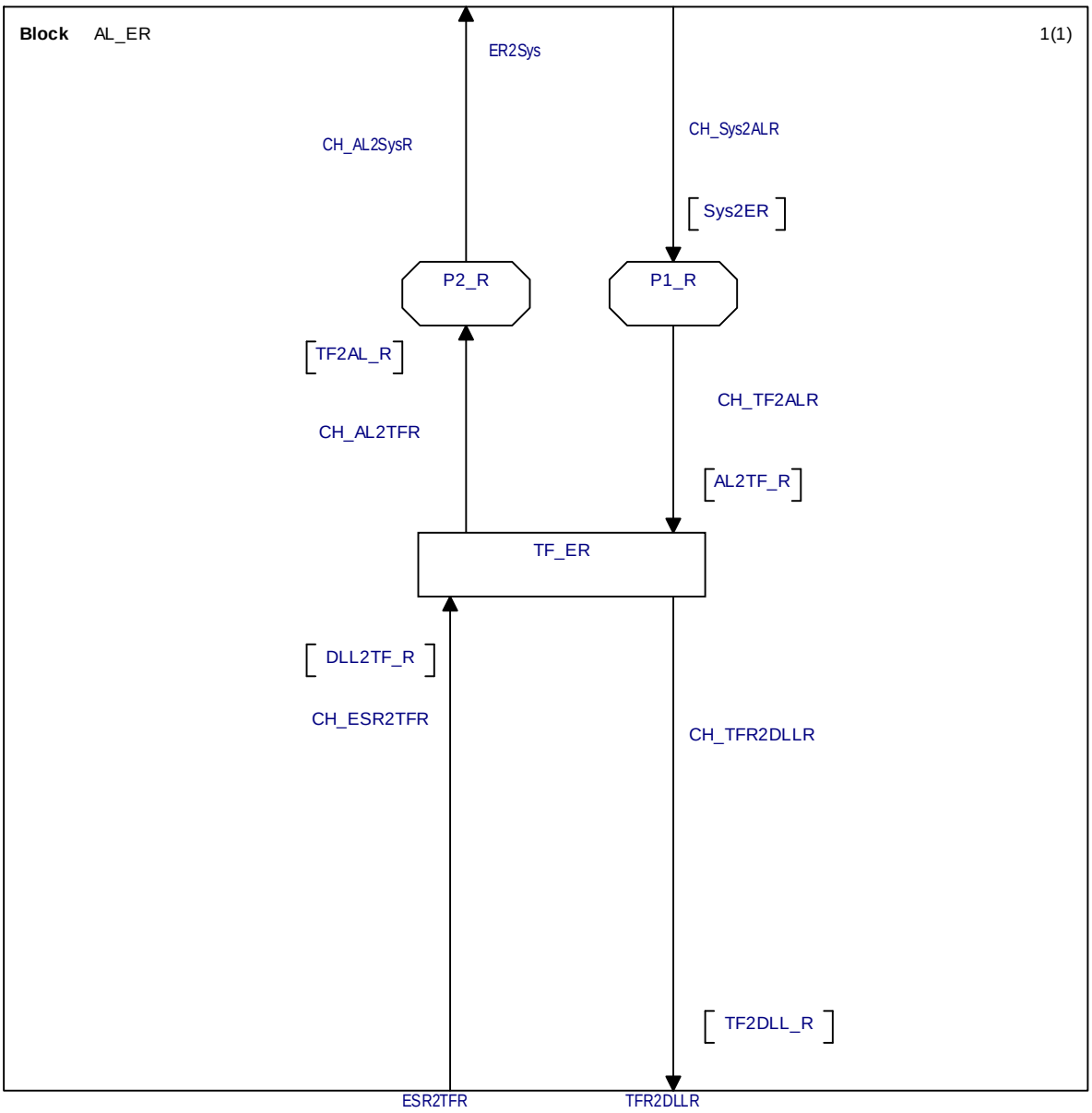
Process MES_EM 3(4)



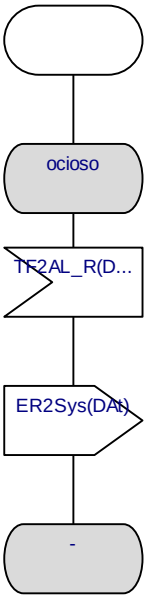
Process MES_EM 4(4)




Block ER
1(1)
ER2Sys
C_Sys2ER
C_ER2Sys
Sys2ER
AL_ER
DLL2TF_R
TFR2DLLR
ESR2TFR
TF2DLL_R
DLL_ER
EPM2E...
ESM2E...
2RxES
2TxEPR
ESR2E...
EPR2E...
RxESR
TxEPR

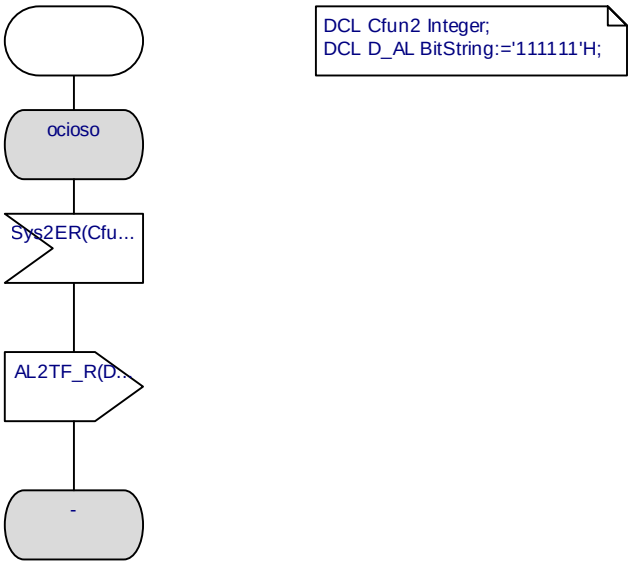

Block AL_ER
1(1)
ER2Sys
CH_AL2SysR
CH_Sys2ALR
Sys2ER
P2_R
P1_R
TF2AL_R
CH_TF2ALR
CH_AL2TFR
AL2TF_R
TF_ER
DLL2TF_R
CH_ESR2TFR
CH_TFR2DLLR
TF2DLL_R
ESR2TFR
TFR2DLLR

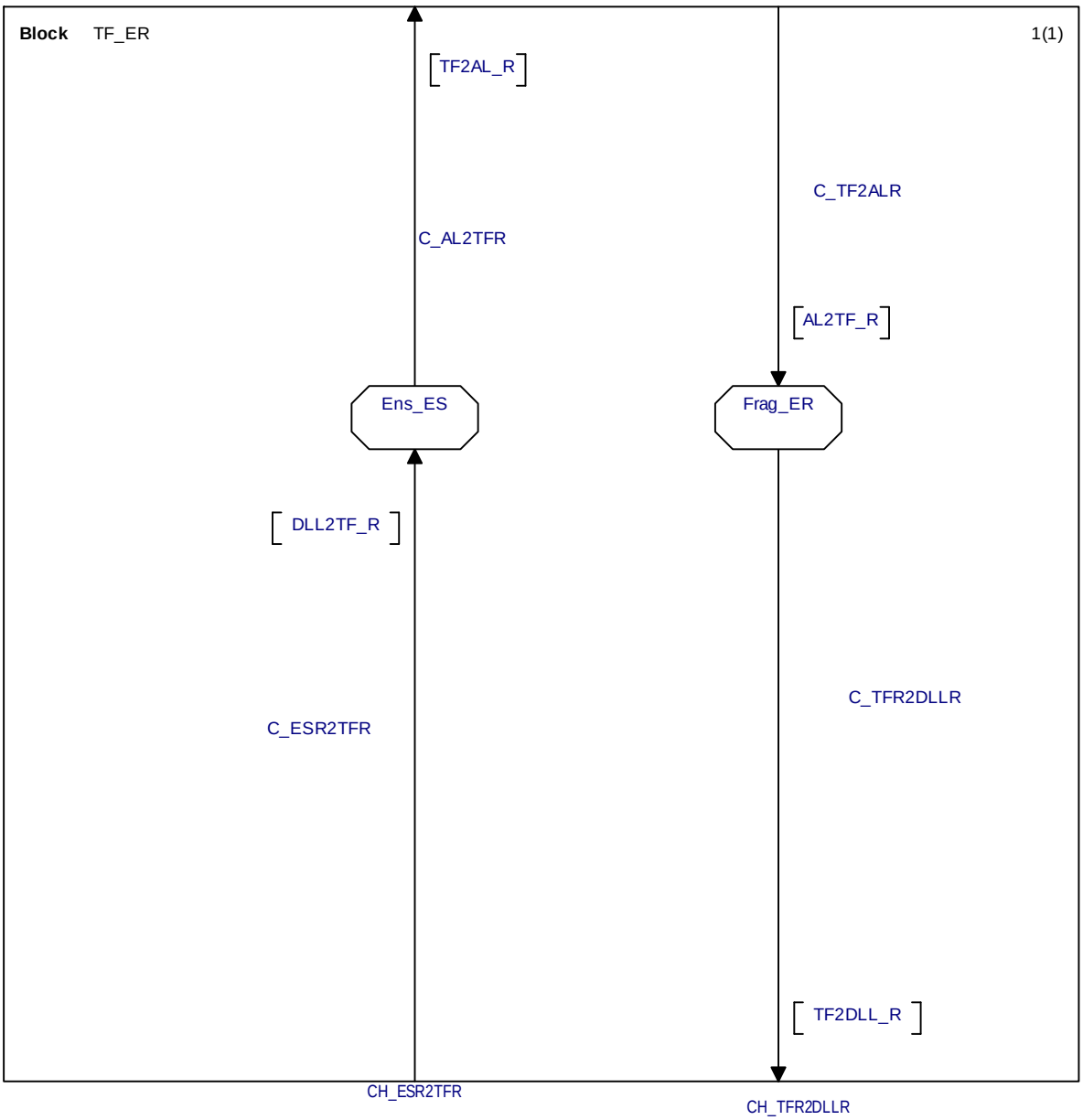
Process P2_R 1(1)



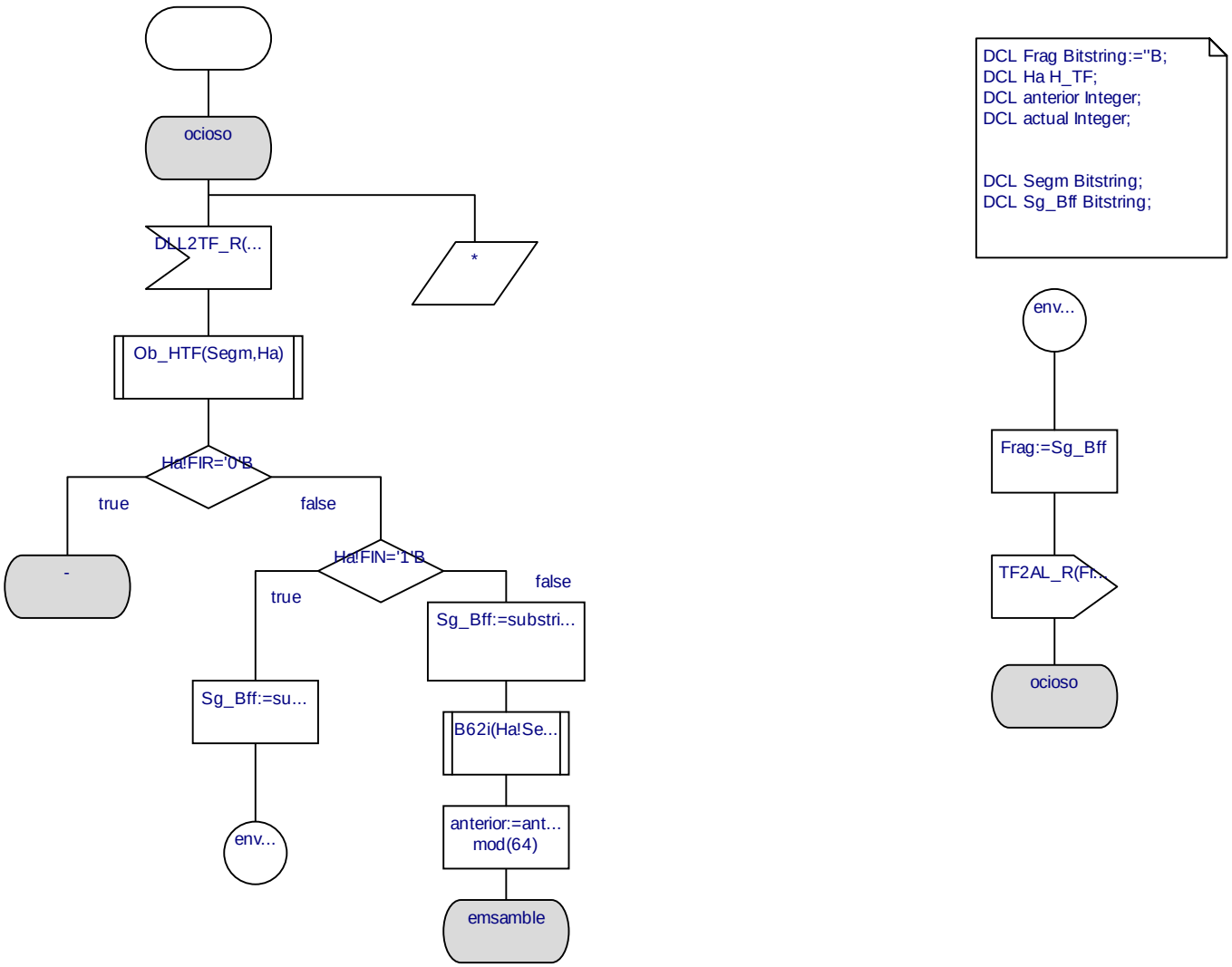
DCL DAt Bitstring;

Process P1_R 1(1)

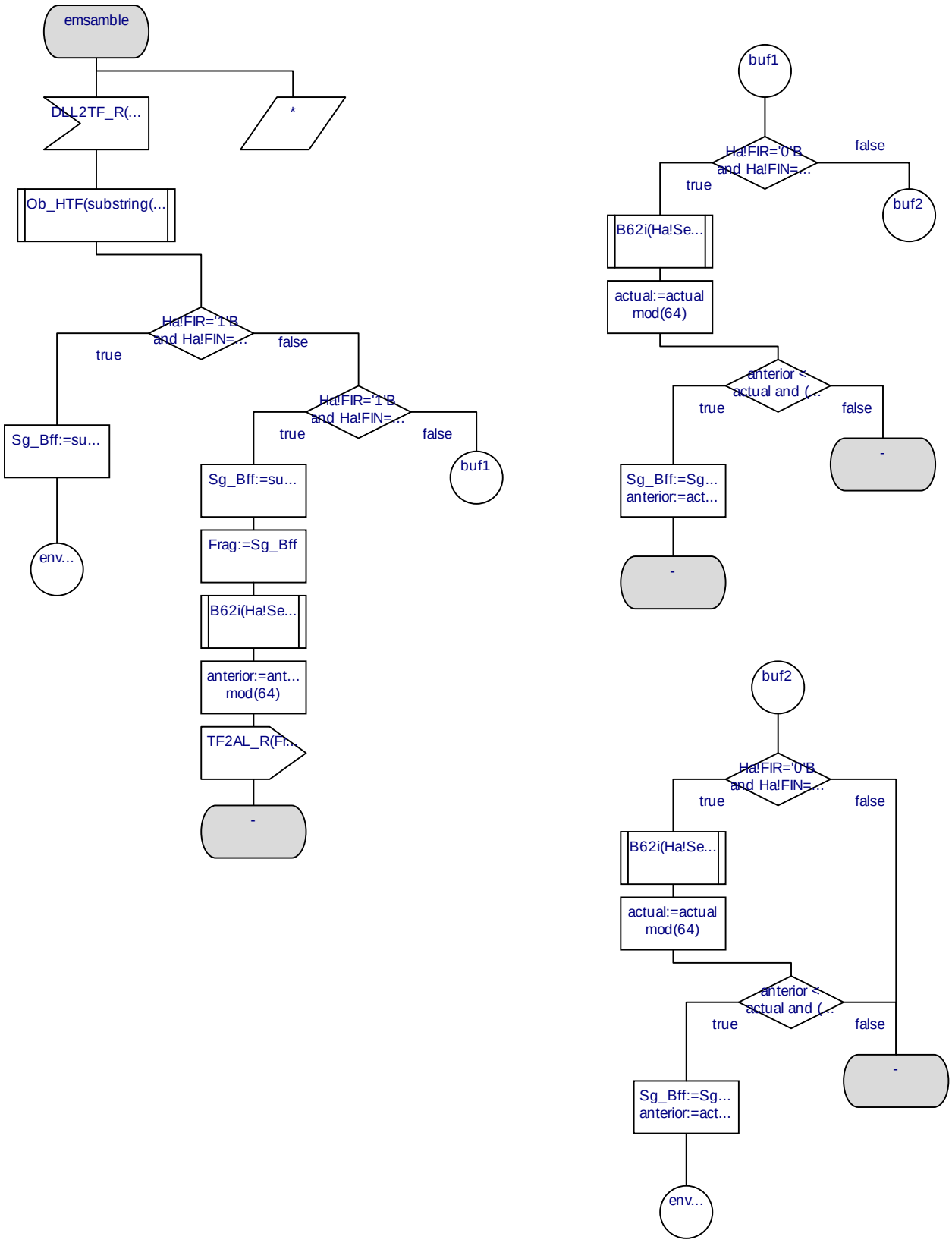



Block TF_ER
1(1)
TF2AL_R
C_AL2TFR
C_TF2ALR
AL2TF_R
Ens_ES
Frag_ER
DLL2TF_R
C_ESR2TFR
C_TFR2DLLR
TF2DLL_R
CH_ESR2TFR
CH_TFR2DLLR

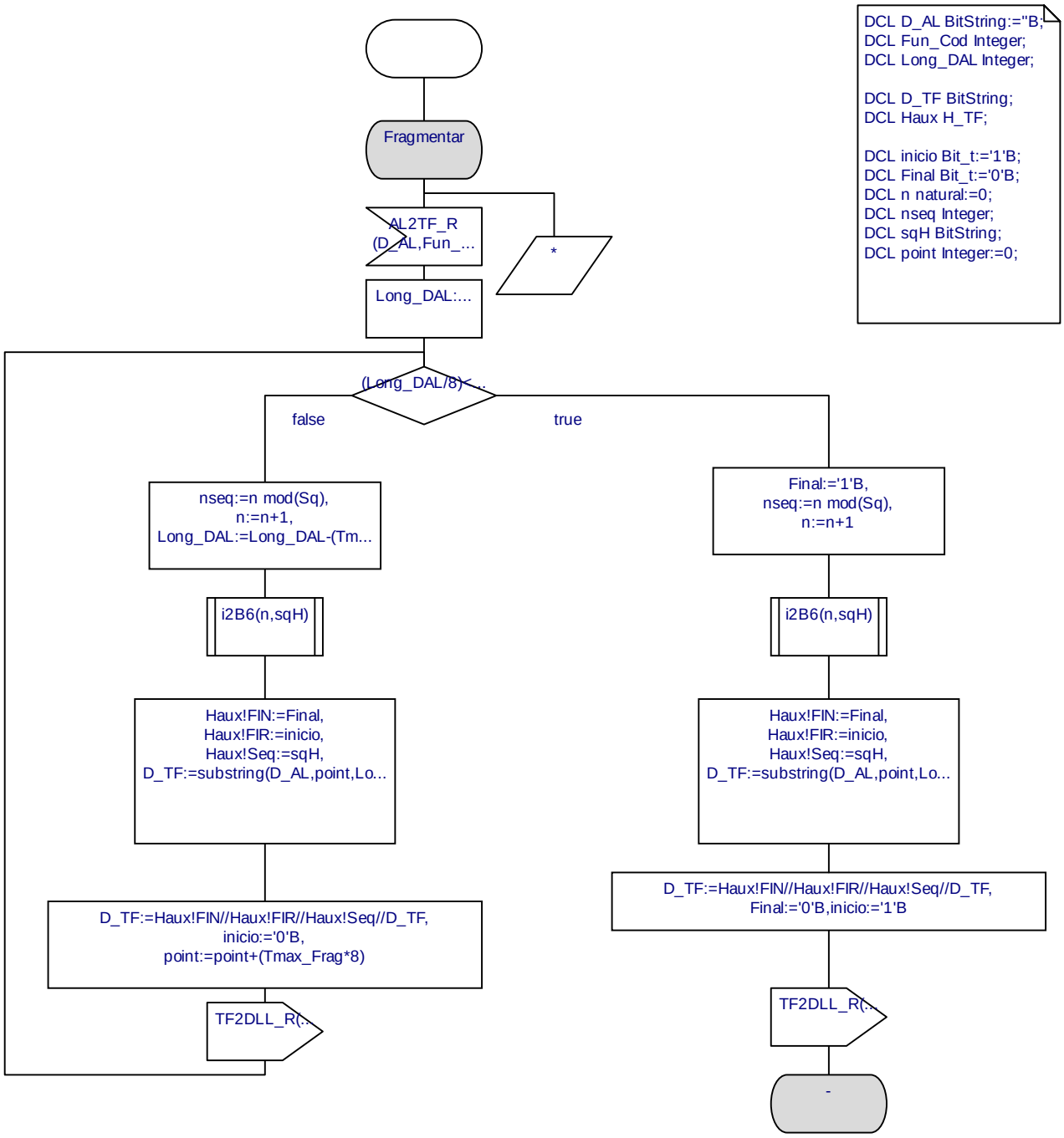
Process Ens_ES 1(2)

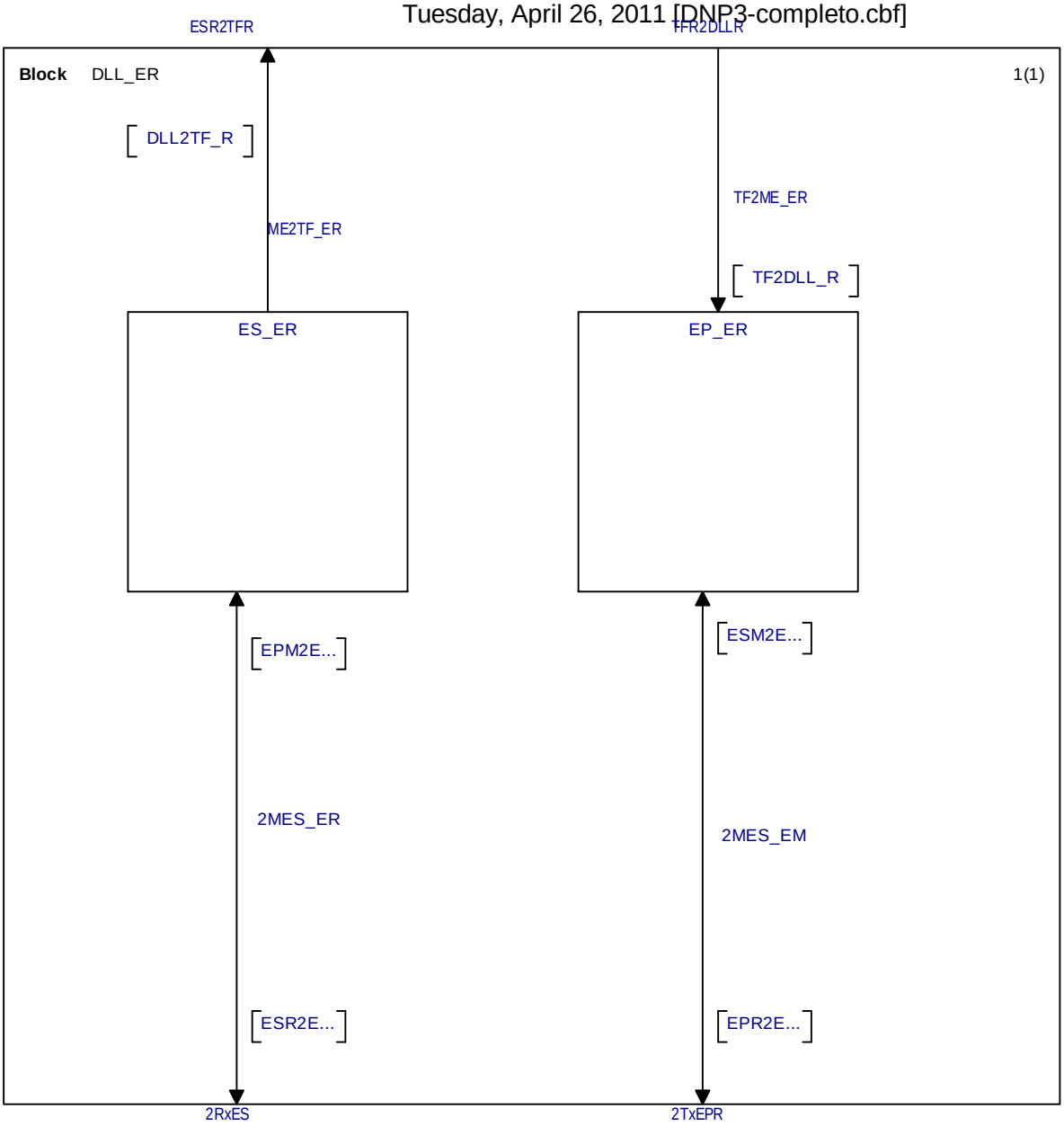


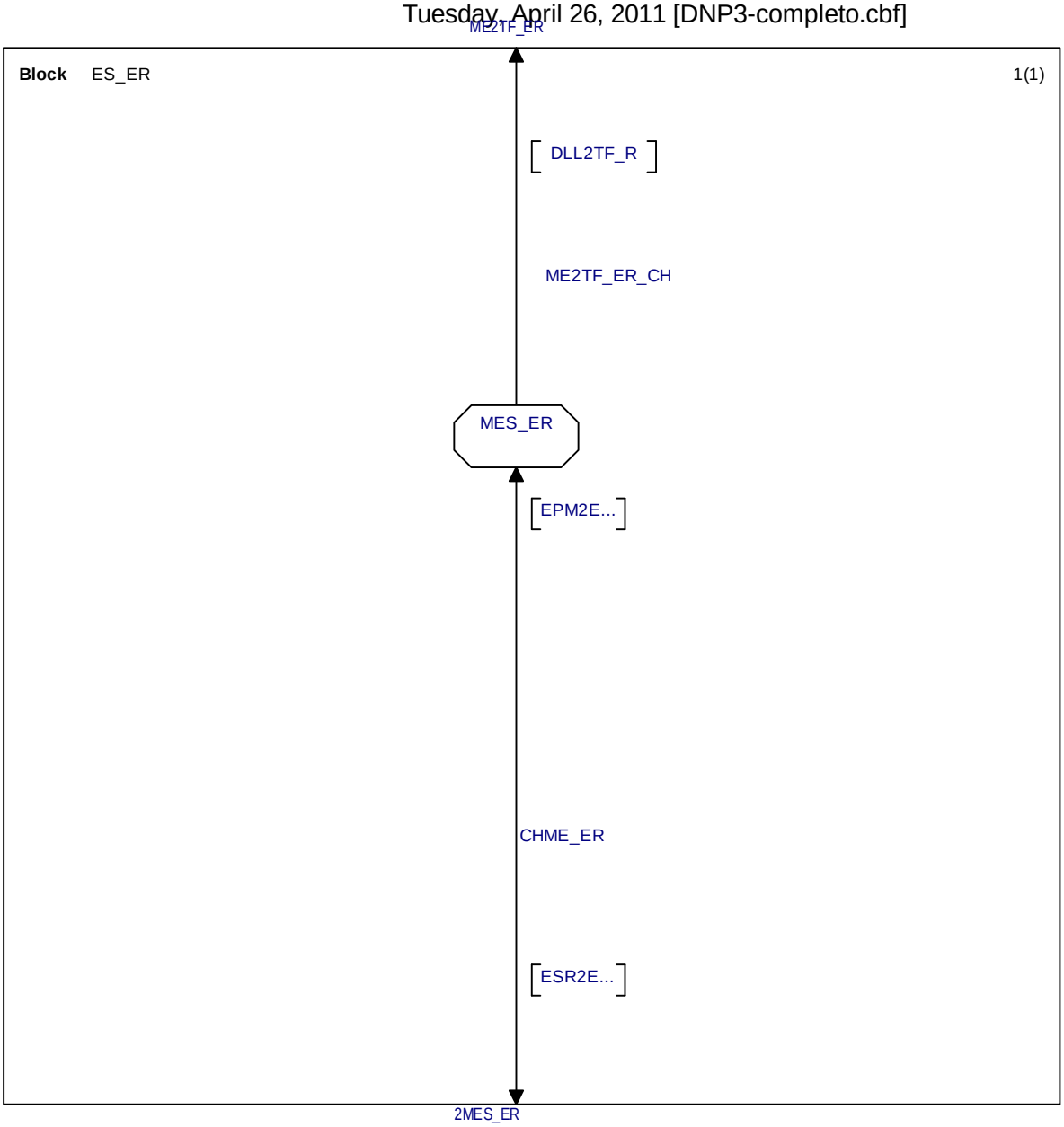
Process Ens_ES 2(2)



Process Frag_ER 1(1)

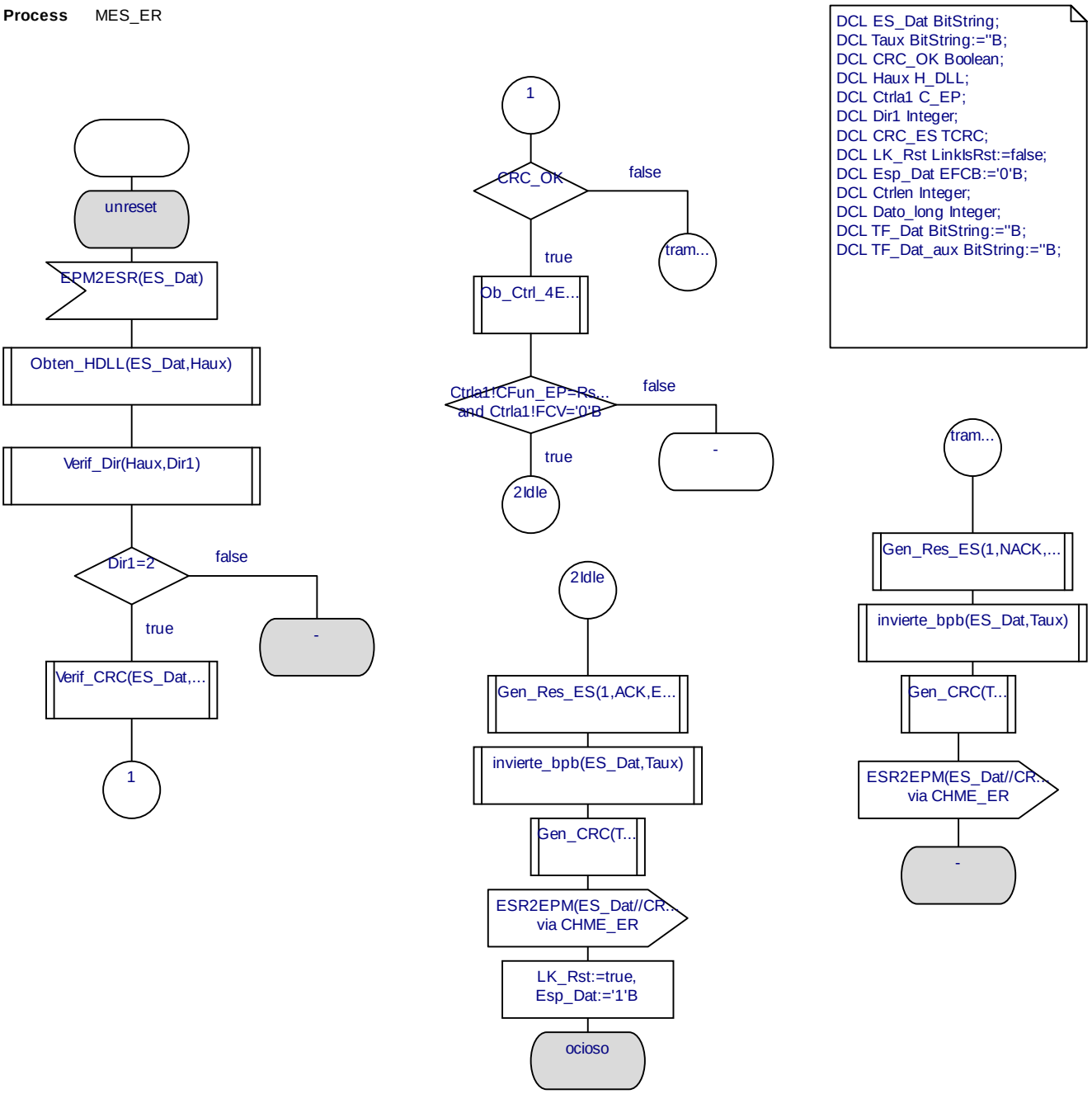



ESR2TFR
TFR2DLLR
Block DLL_ER
1(1)
DLL2TF_R
ME2TF_ER
TF2ME_ER
TF2DLL_R
ES_ER
EP_ER
EPM2E...
ESM2E...
2MES_ER
2MES_EM
ESR2E...
EPR2E...
2RxES
2TxEPR

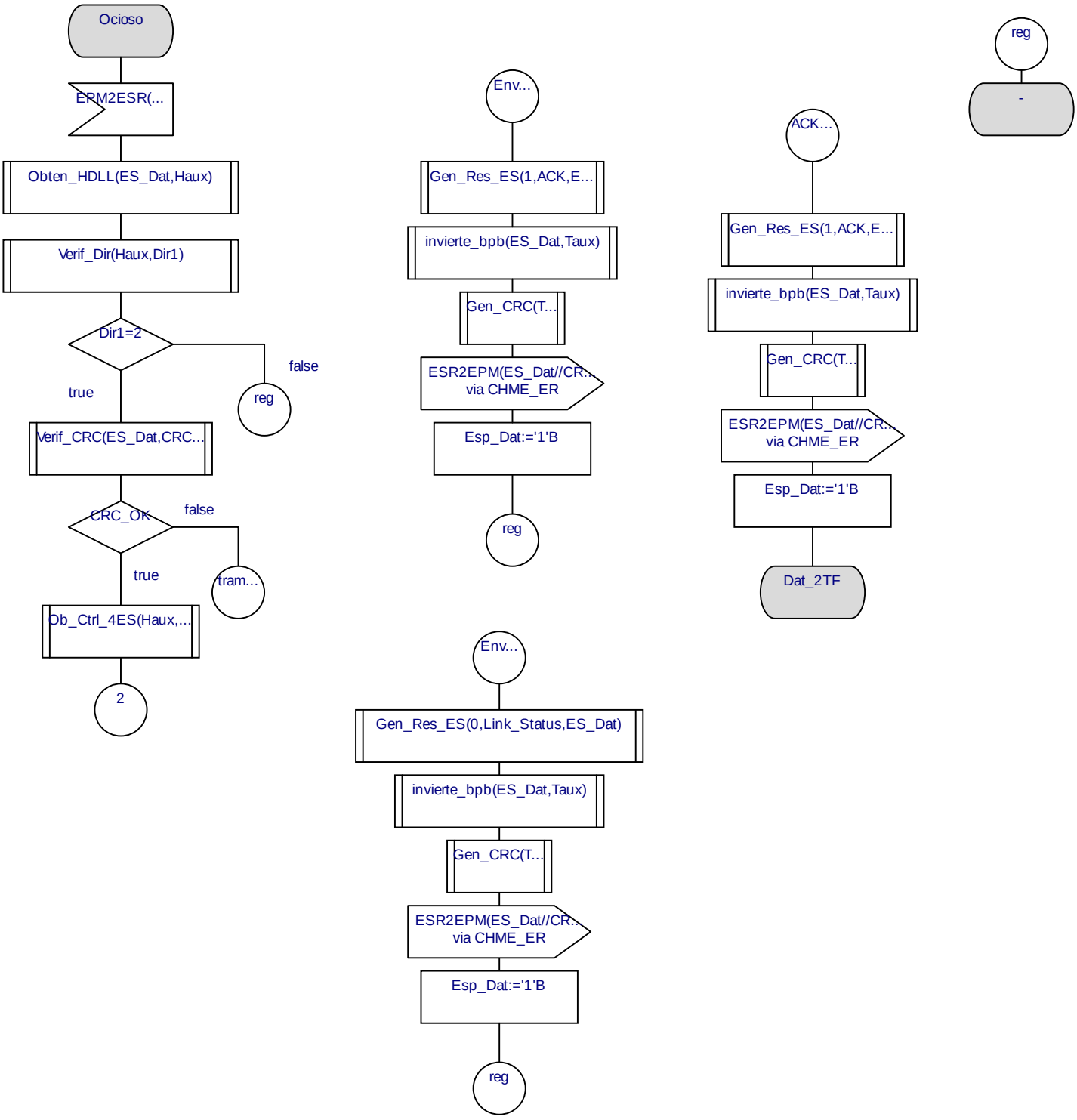

ME2TF_ER
Block ES_ER
1(1)
DLL2TF_R
ME2TF_ER_CH
MES_ER
EPM2E...
CHME_ER
ESR2E...
2MES_ER

Process MES_ER

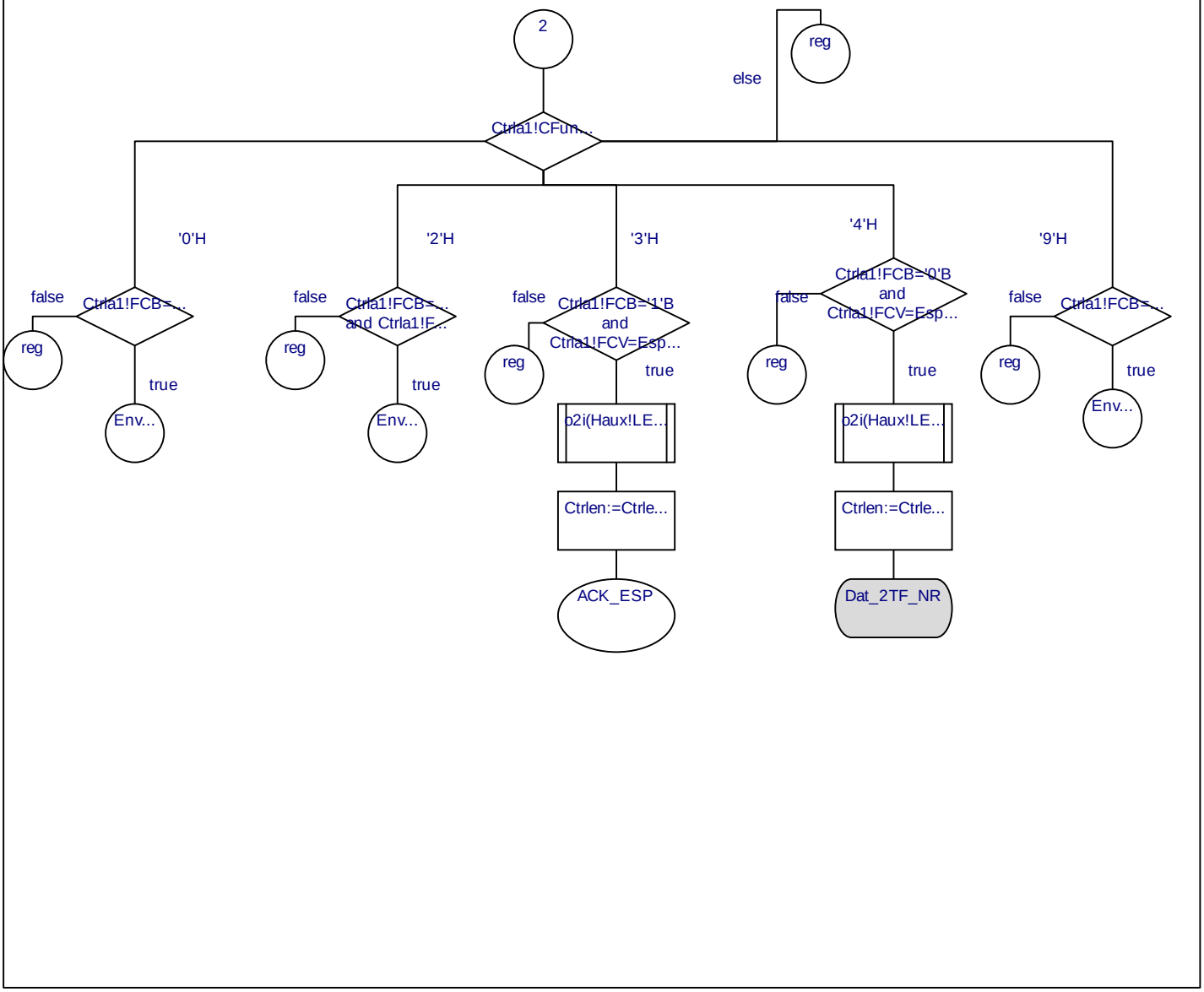
1(4)



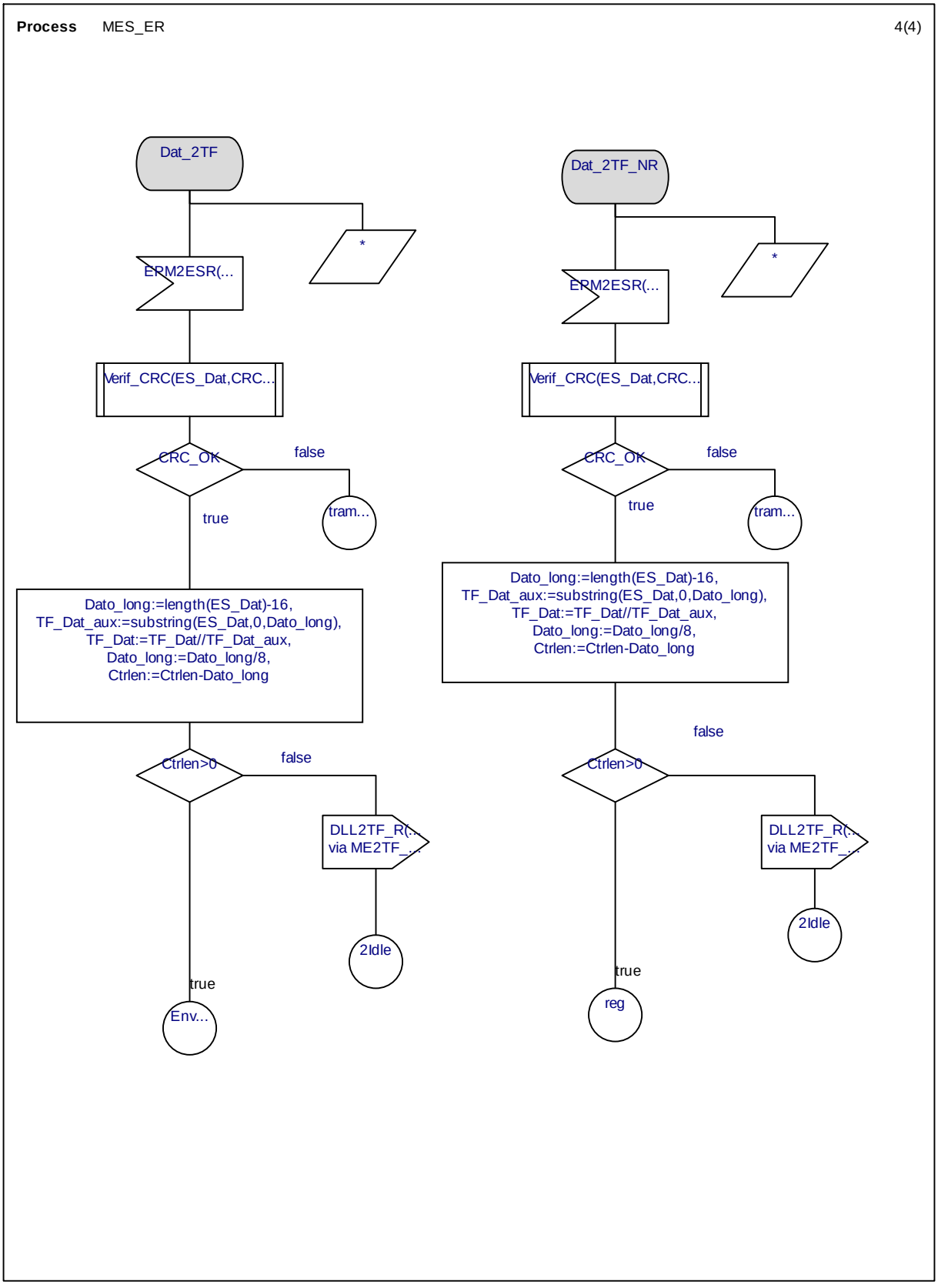
Process MES_ER 2(4)

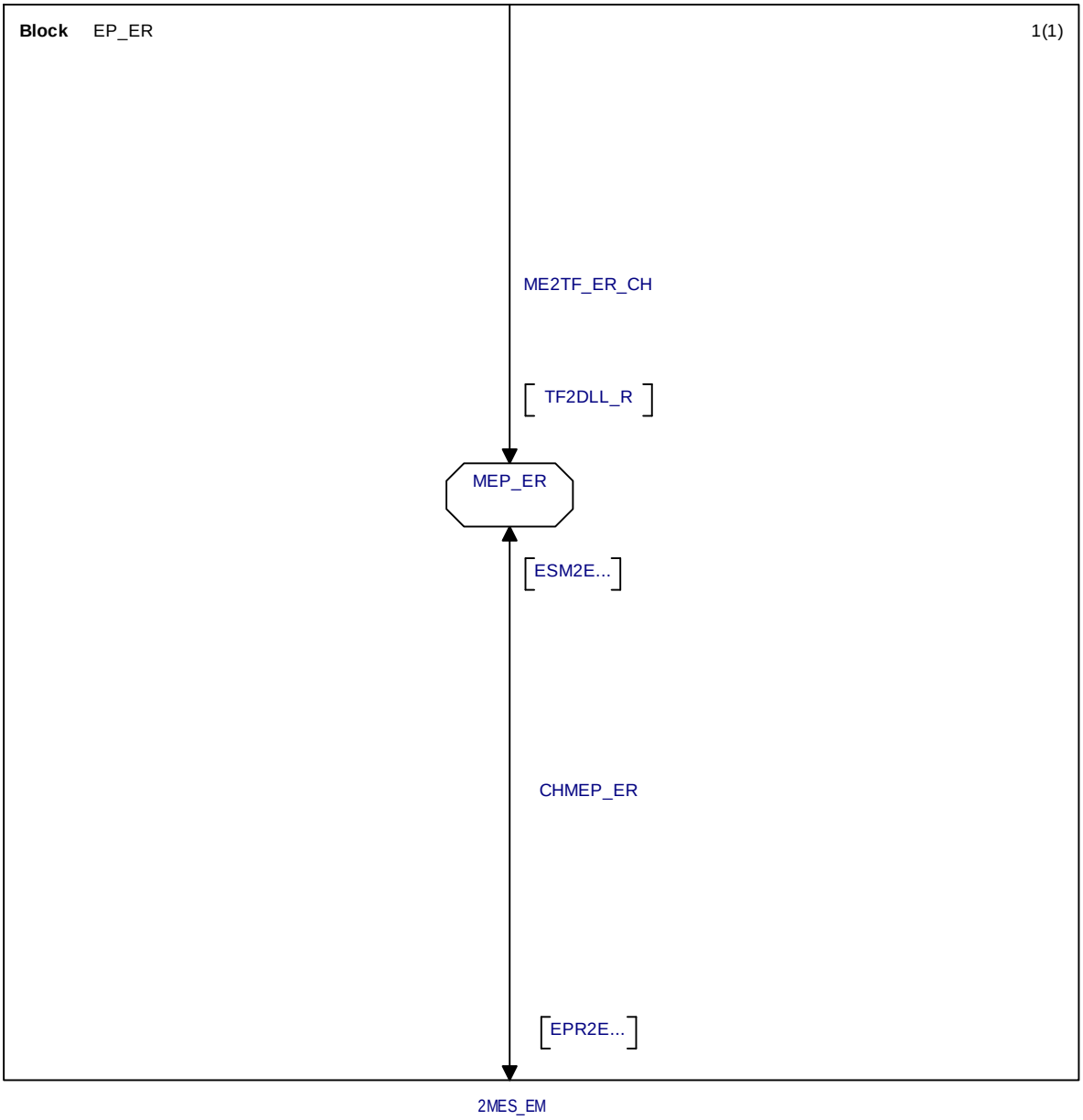


Process MES_ER 3(4)

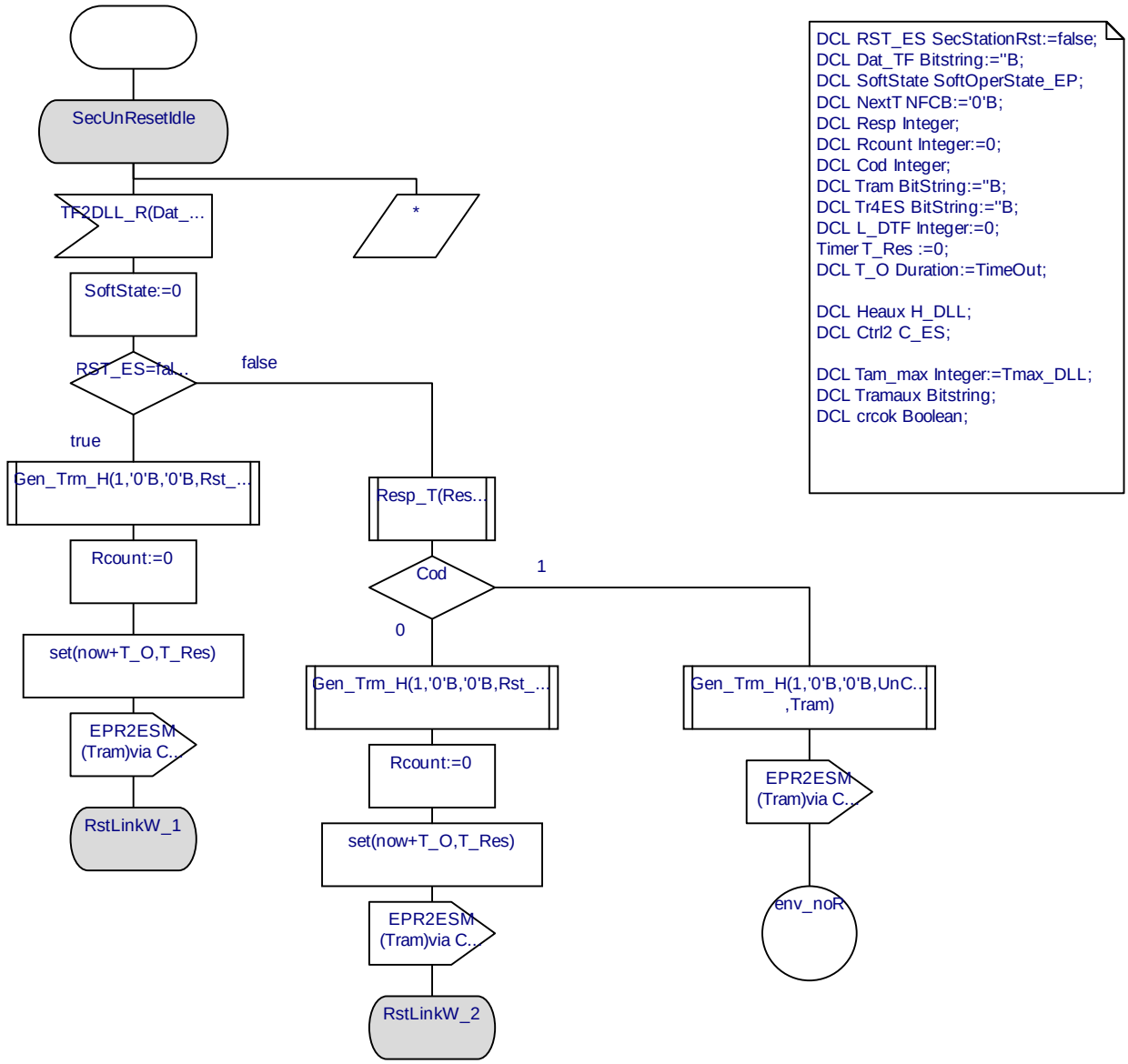


Process MES_ER 4(4)

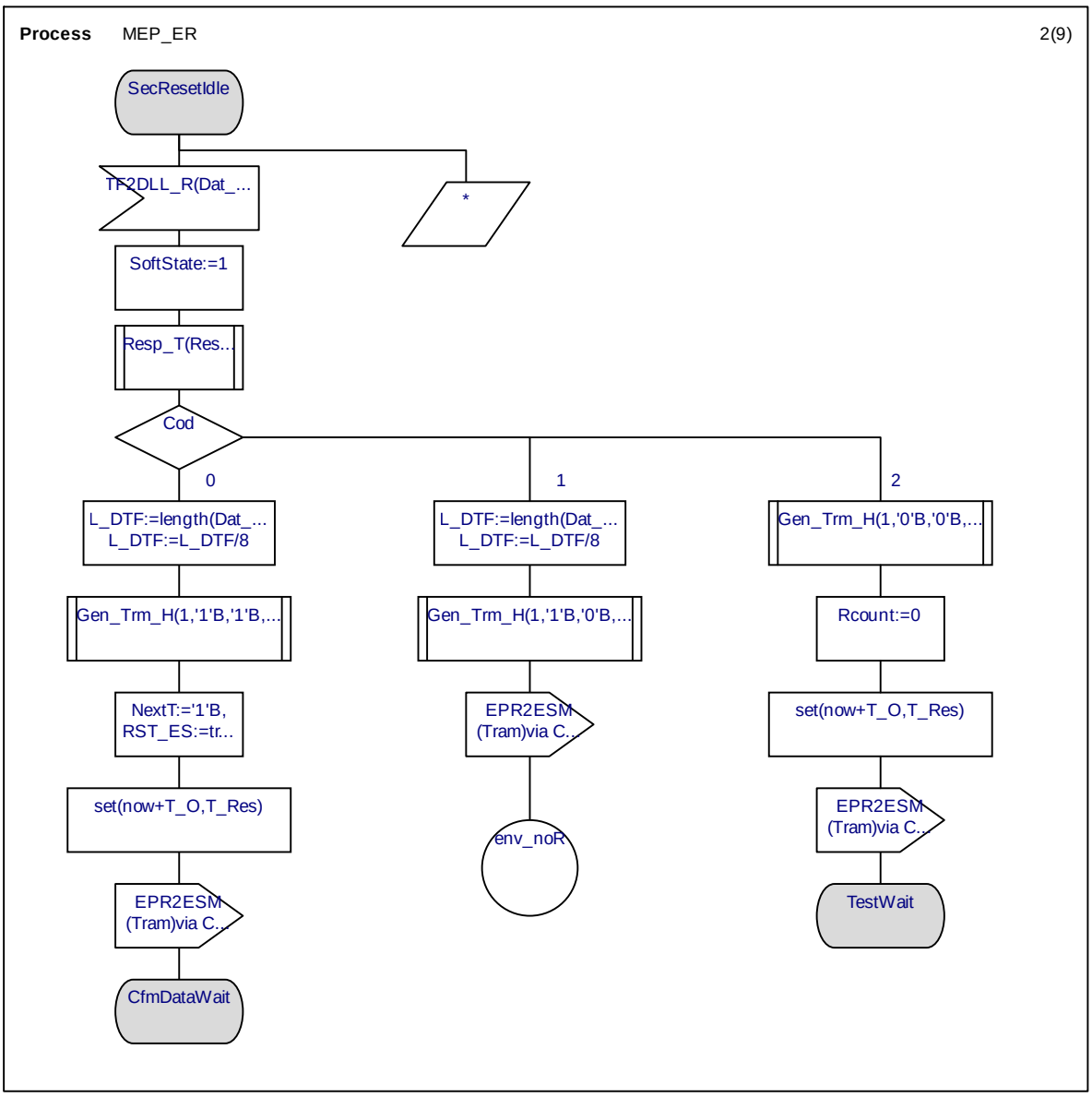



TF2ME_ER
Block EP_ER
1(1)
ME2TF_ER_CH
TF2DLL_R
MEP_ER
ESM2E...
CHMEP_ER
EPR2E...
2MES_EM

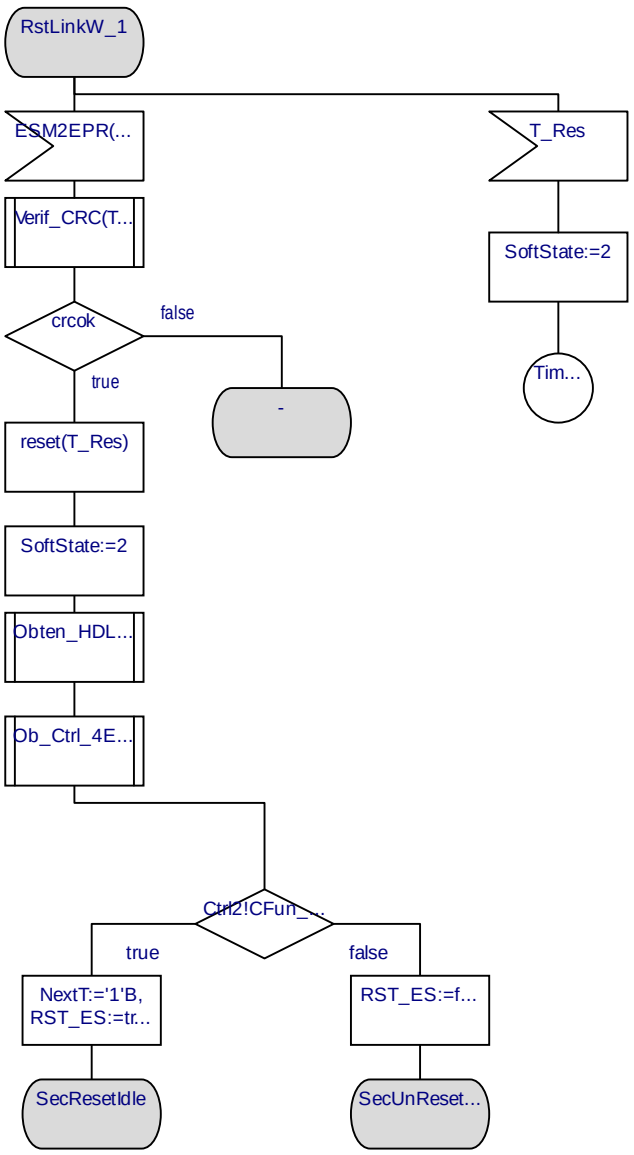
Process MEP_ER 1(9)



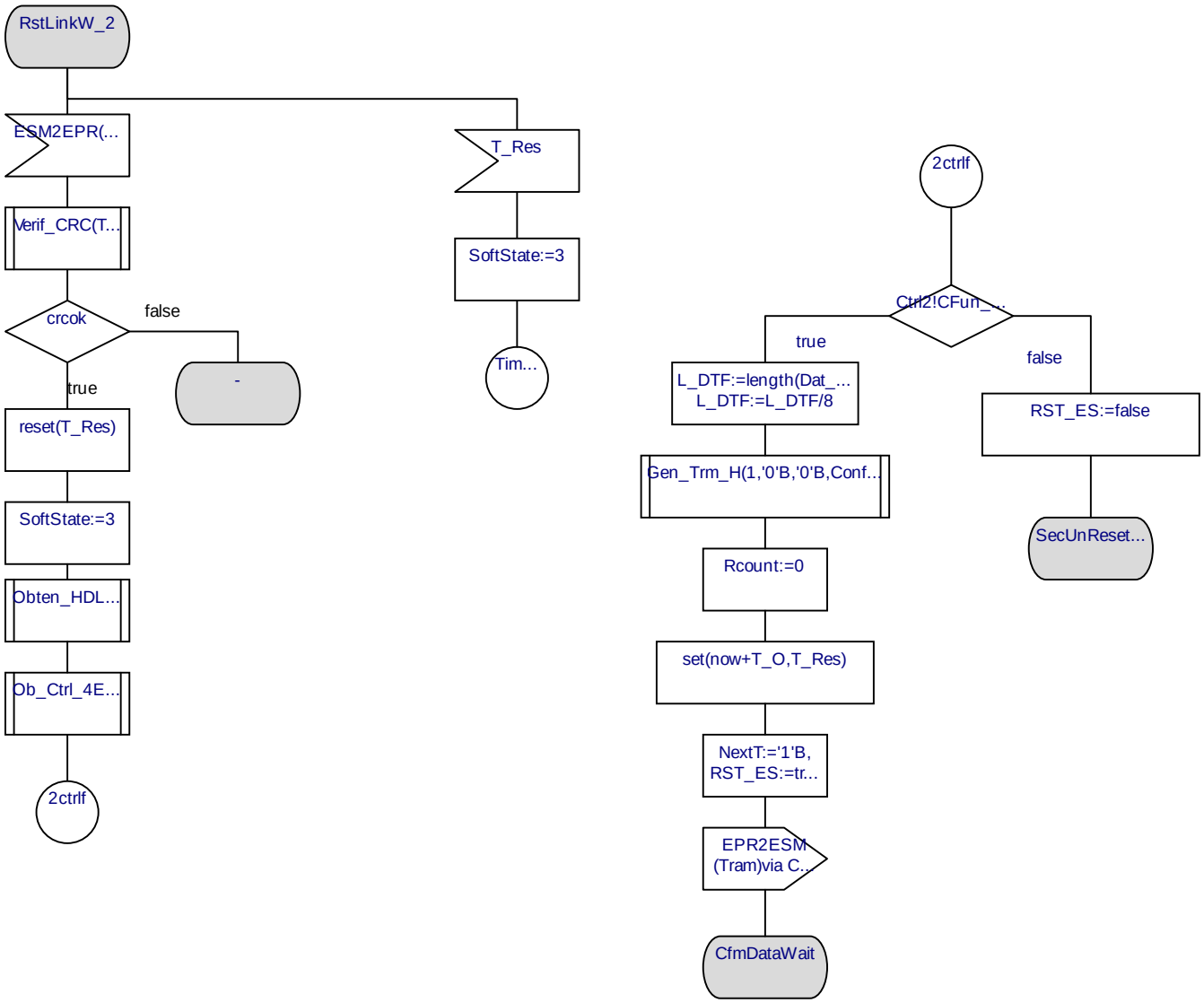
Process MEP_ER 2(9)



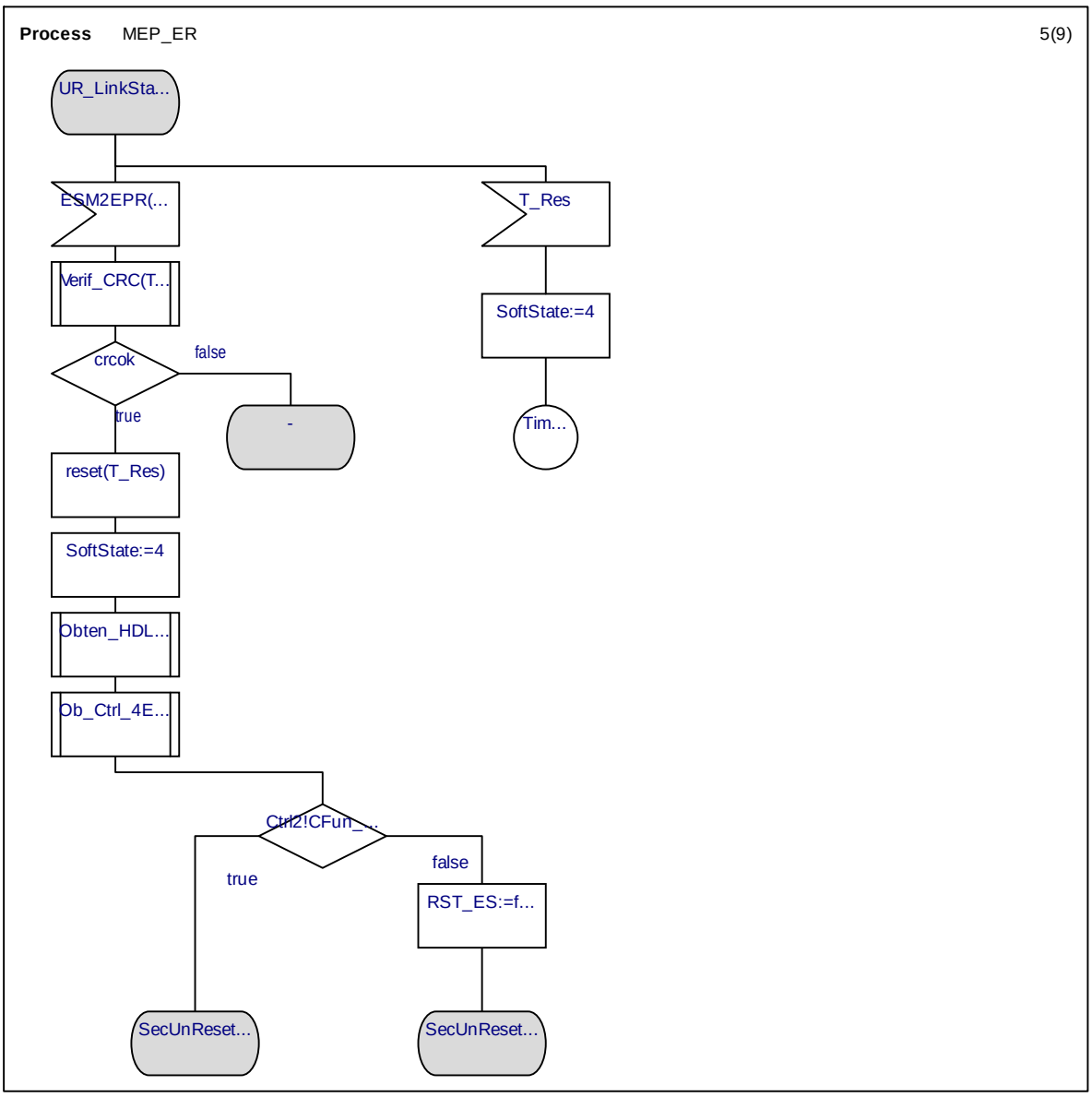
Process MEP_ER 3(9)



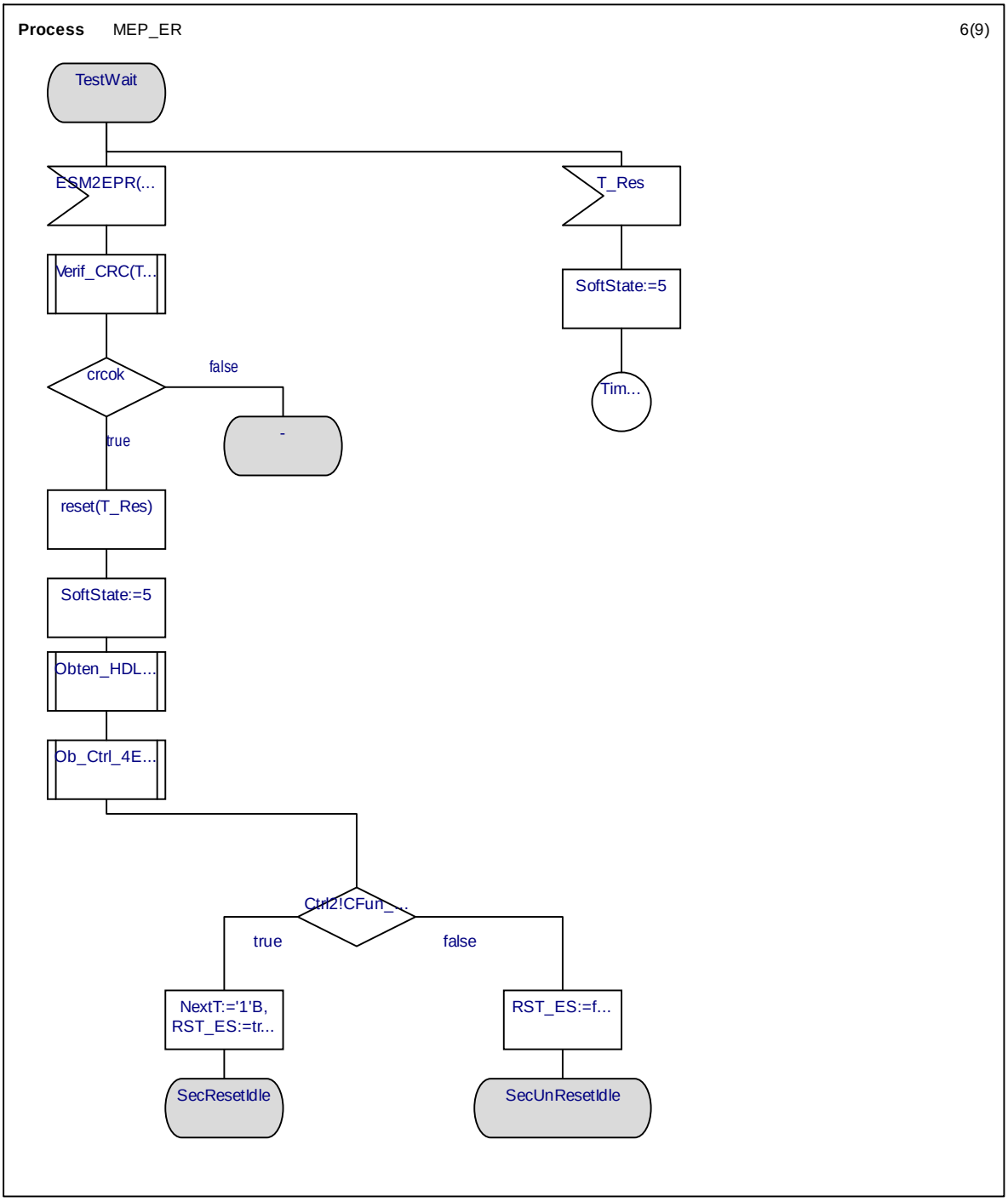
Process MEP_ER 4(9)



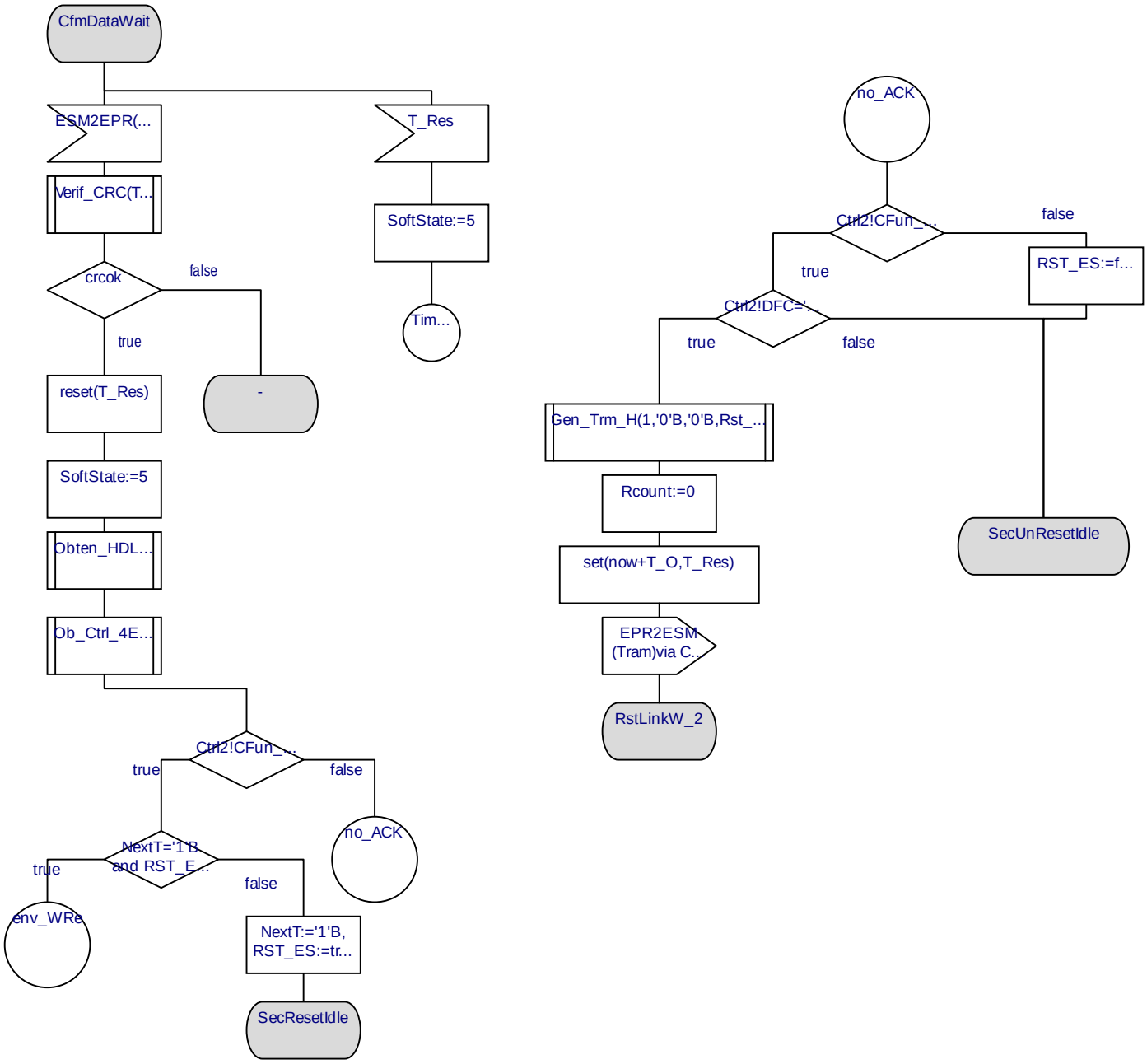
Process MEP_ER 5(9)



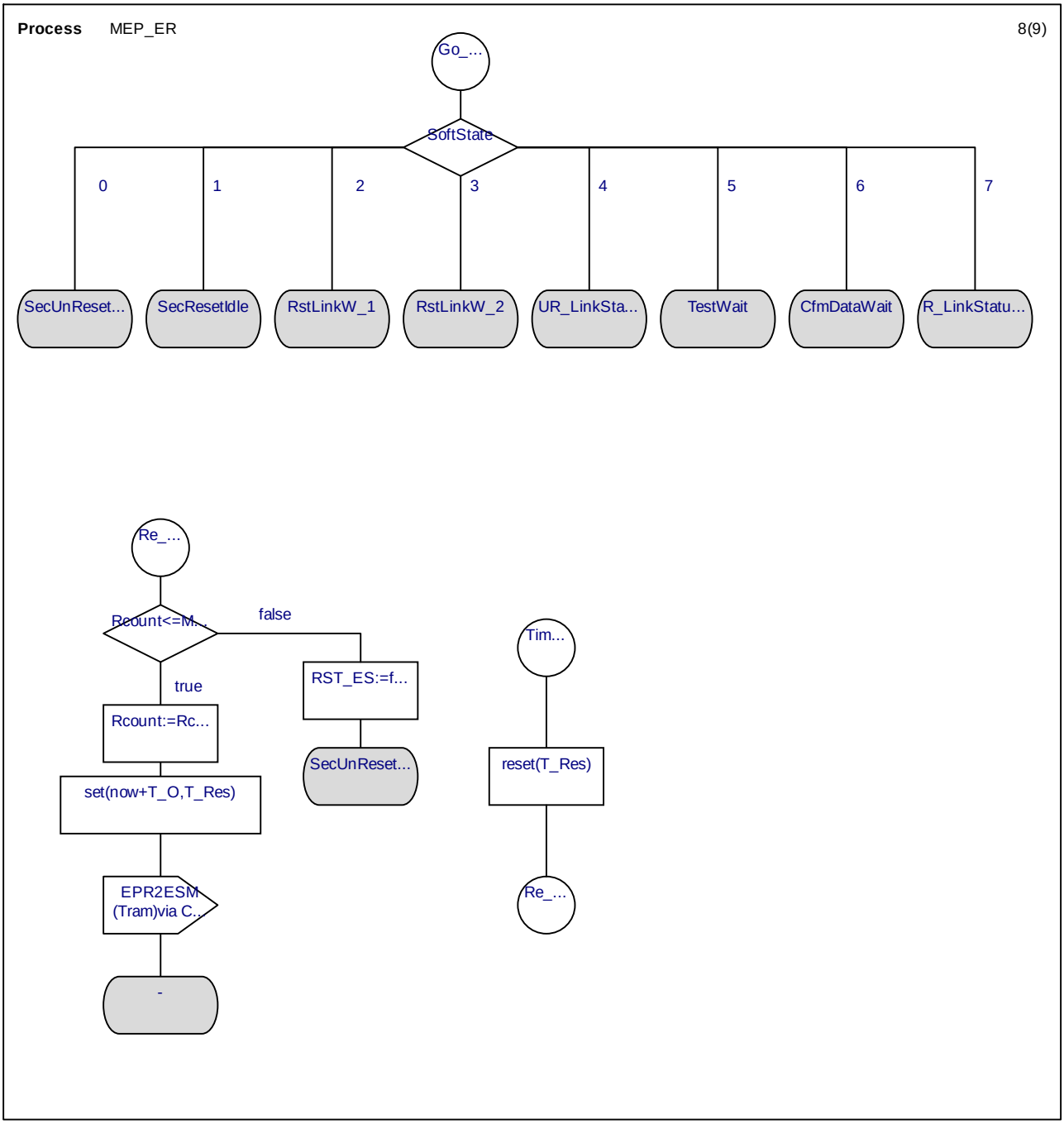
Process MEP_ER 6(9)



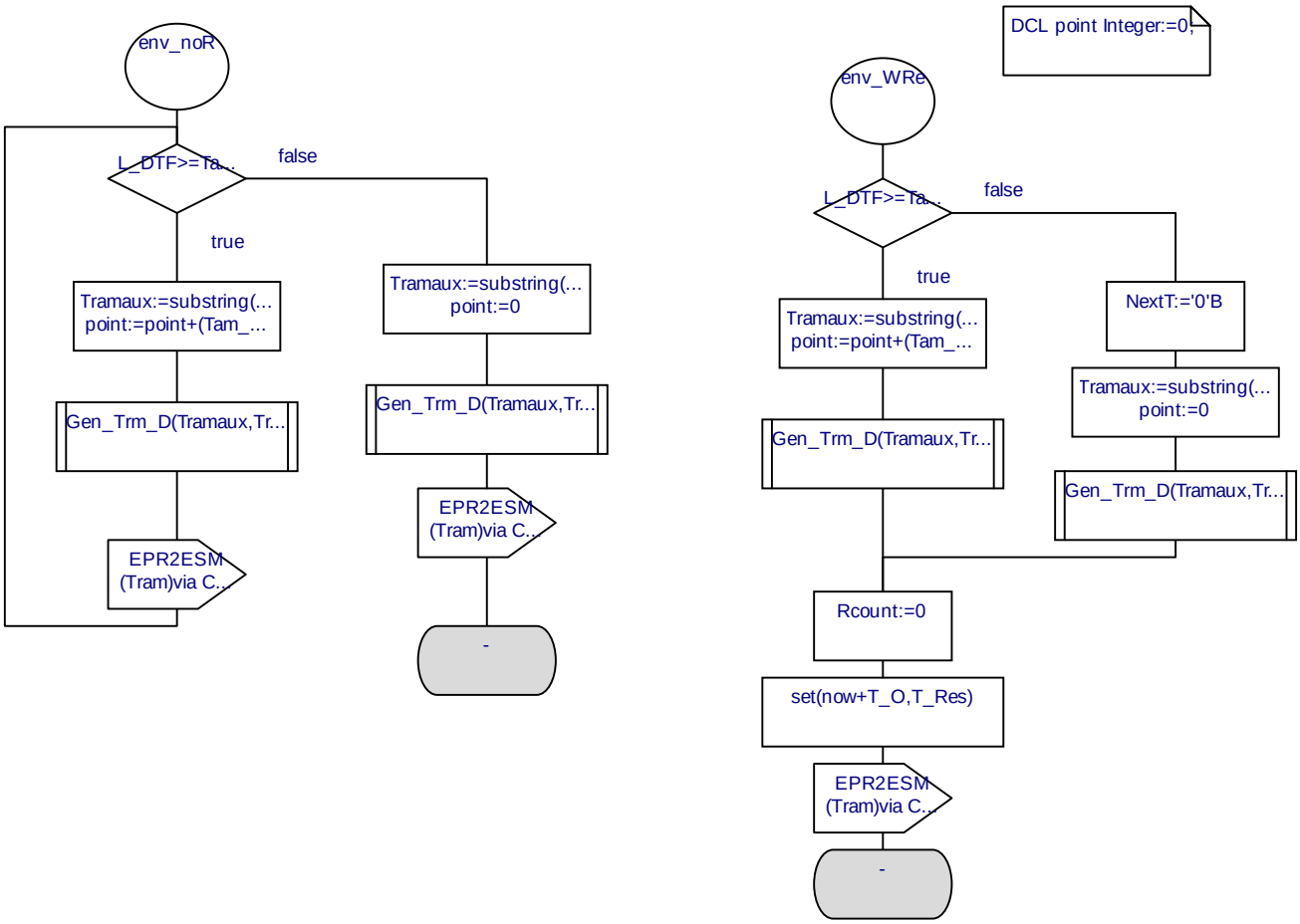
Process MEP_ER 7(9)

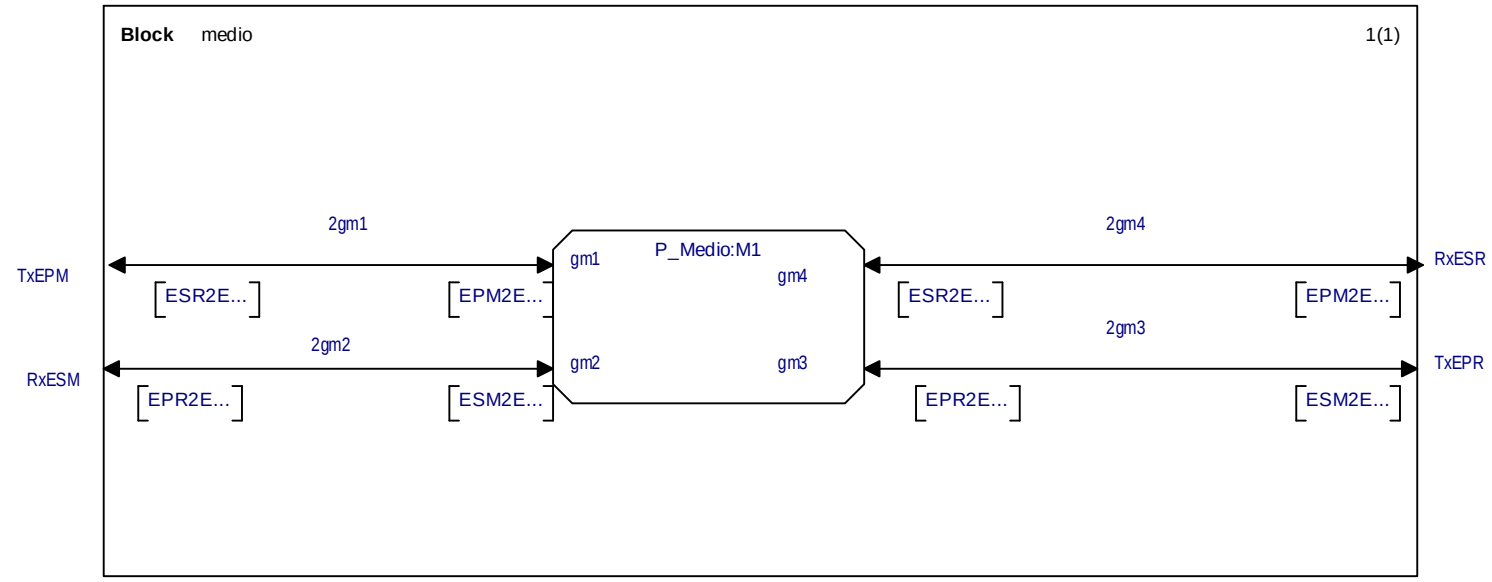


Process MEP_ER 8(9)



Process MEP_ER 9(9)




Block medio
1(1)
2gm1
2gm4
TxEPM
RxESR
P_Medio:M1
gm1
gm4
ESR2E...
EPM2E...
ESR2E...
EPM2E...
2gm3
2gm2
RxESM
TxEPR
gm2
gm3
EPR2E...
ESM2E...
EPR2E...
ESM2E...

Package DNP3_Procedimientos 1(1)

use DNP3_Datos;

/*Procesimientos que se emplean para calcular y rectificar el CRC*/

invierte_bpb | Gen_CRC | mod2 | invierte_CRC | Verif_CRC | Chek_0

/*Procesimientos que se emplean para convertir de entero a bitstring y de String a a entero*/

i2o | o2i | pow | i2B6 | B62i

/*Procesimientos que se emplean para corroborar el destino y generar la trama de la ES*/

Obten_HDLL | Verif_Dir | Ob_Ctrl_4ES | Gen_Res_ES

/*Procesimientos que se emplean para crear la trama de la EP*/

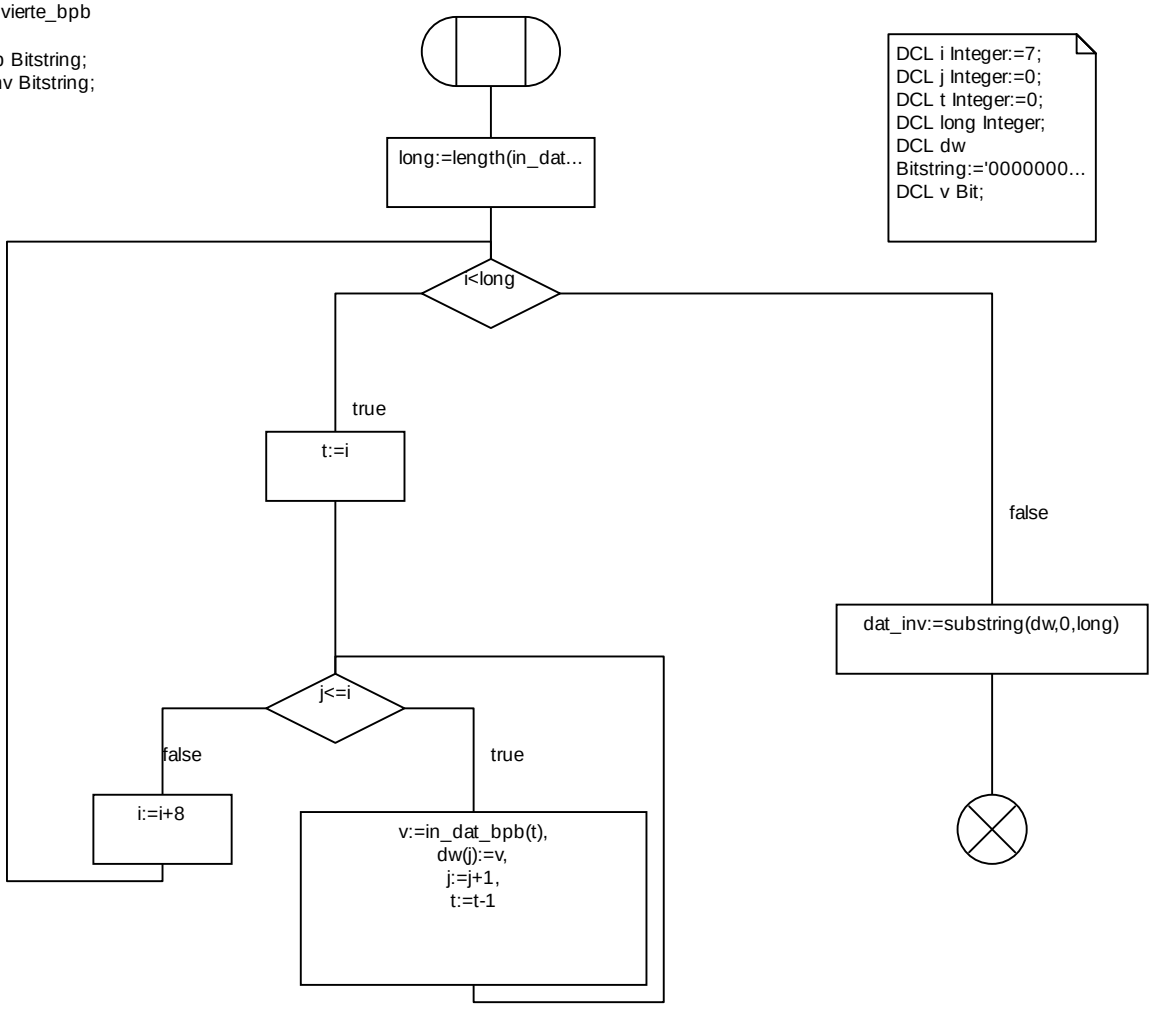
Gen_Pet_EP | Ob_Ctrl_4EP | Gen_CtrlDLL | Resp_T | Gen_Trm_H | Gen_Trm_D

/*Procesimientos que se emplean para crear la obtener la cabecera de TF*/

Ob_HTF

Procedure invierte_bpb 1(1)

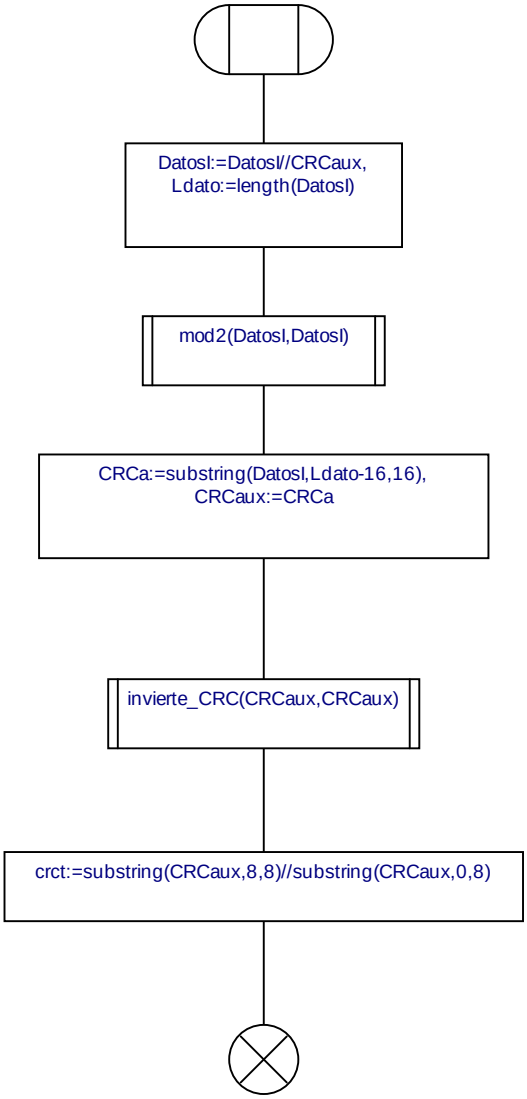
FPAR in_dat_bpb Bitstring;
RETURNS dat_inv Bitstring;



Procedure Gen_CRC 1(1)

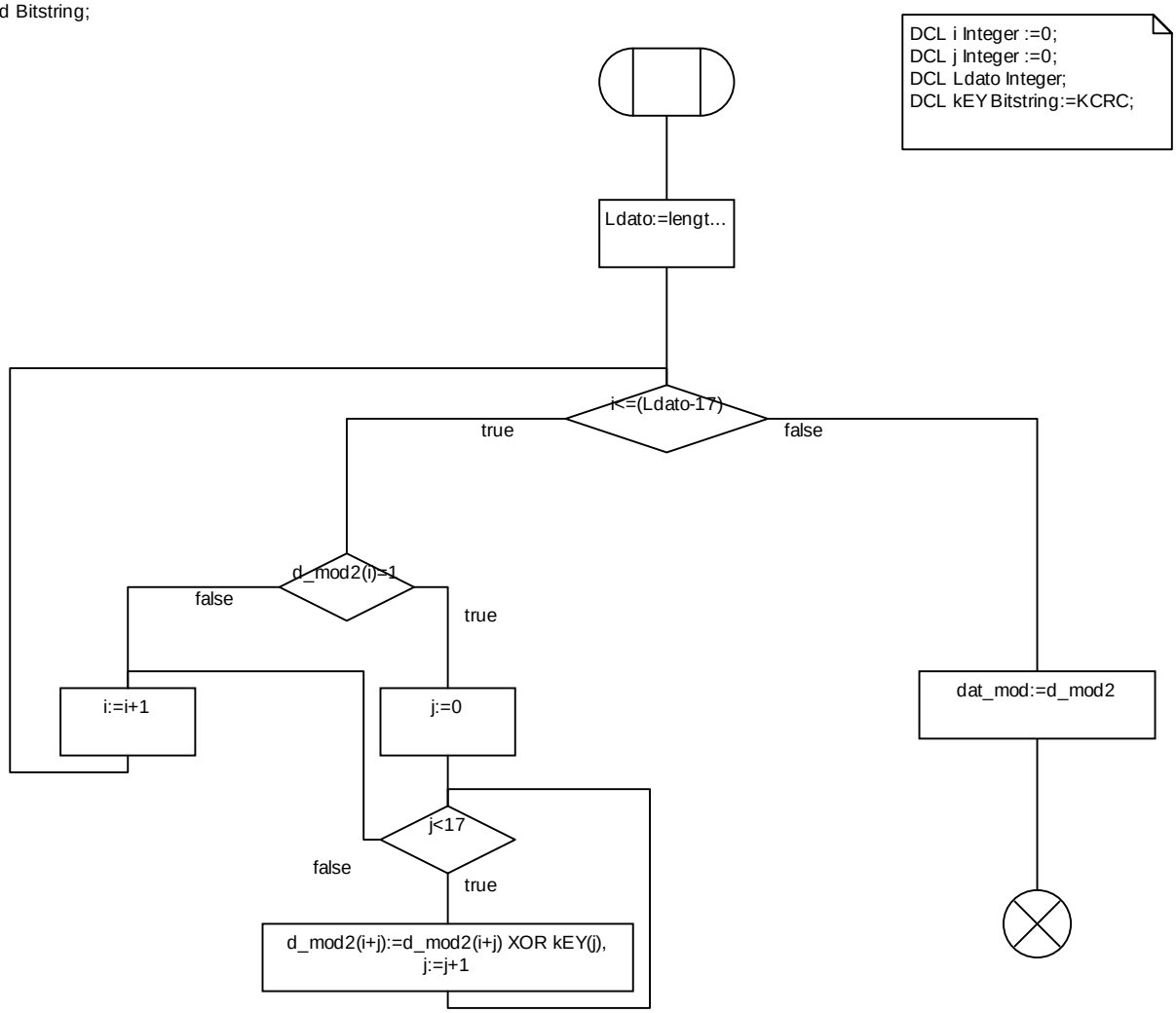
FPAR
Datosl Bitstring
Returns
crct TCRC

DCL CRCaux TCRC:='0000'H;
DCL Ldato Integer;
DCL CRCa Bitstring;



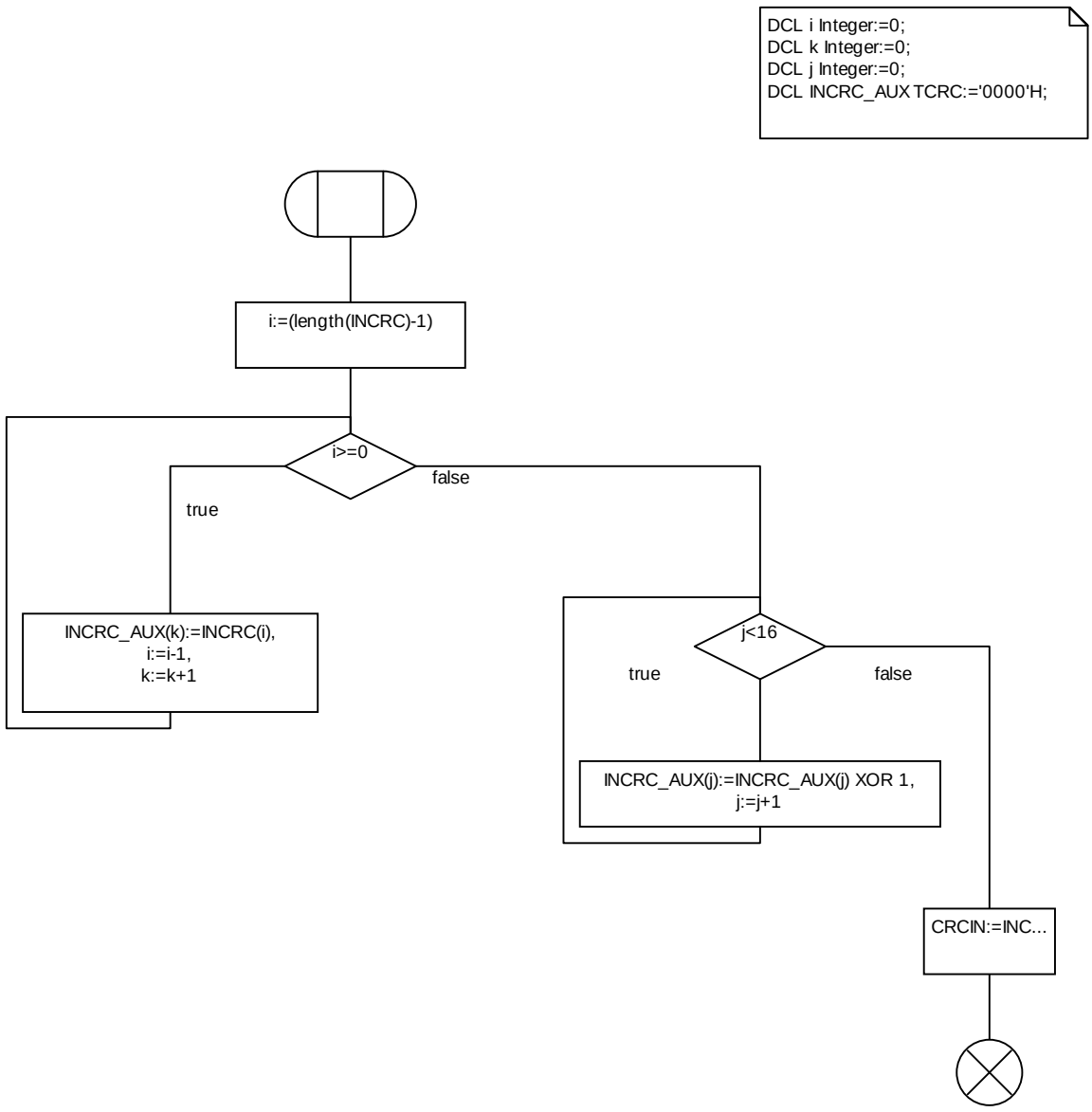
Procedure mod2 1(1)

FPAR d_mod2 Bitstring;
RETURNS dat_mod Bitstring;



Procedure invierte_CRC 1(1)

FPAR
INCRC TCRC
Returns
CRCIN TCRC

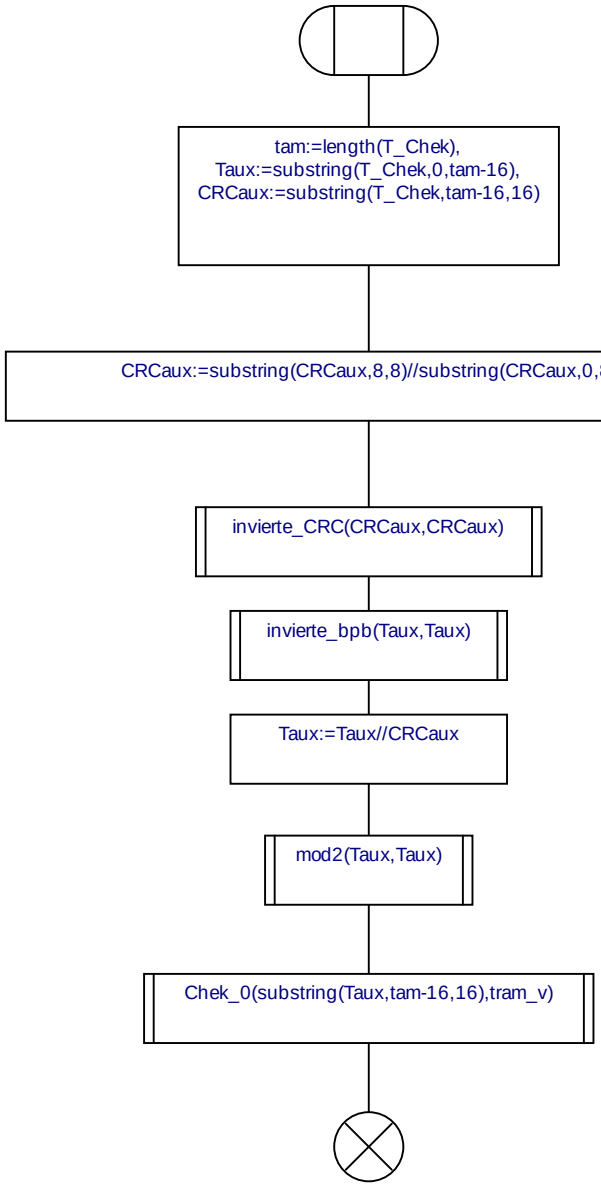


Procedure Verif_CRC

FPAR T_Chek Bitstring;
RETURNS tram_v Boolean;

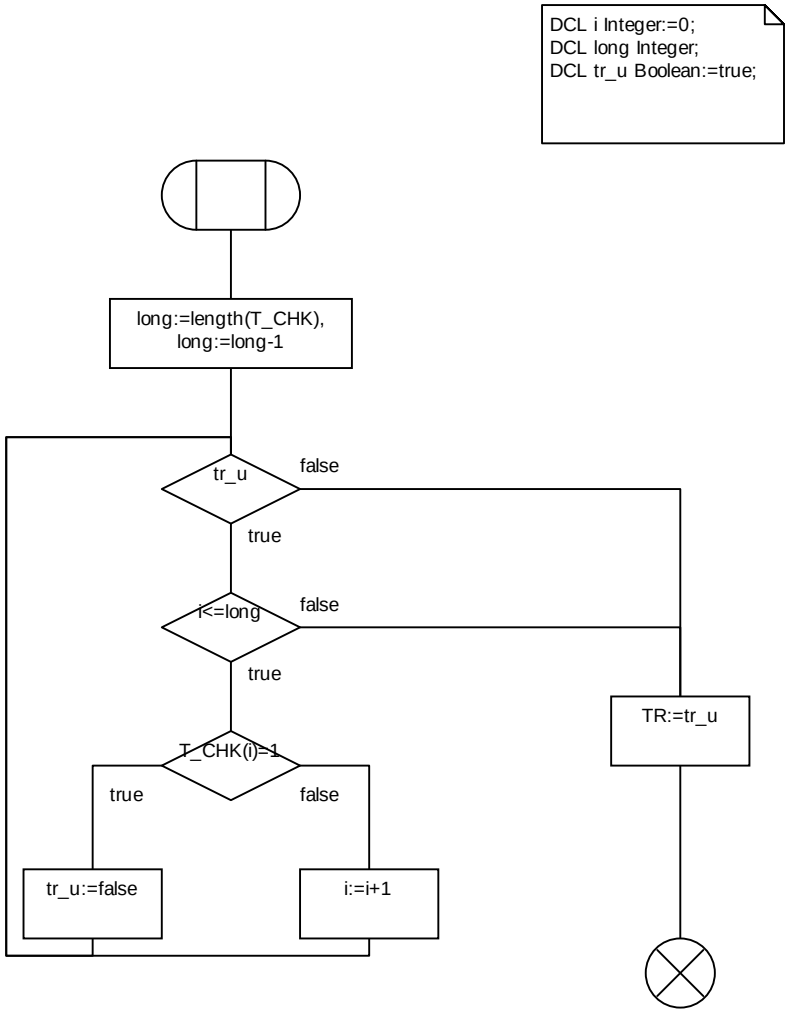
1(1)

DCL tam Integer:=0;
DCL Taux Bitstring;
DCL CRCaux TCRC;



Procedure Chek_0 1(1)

FPAR T_CHK Bitstring;
RETURNS TR Boolean;

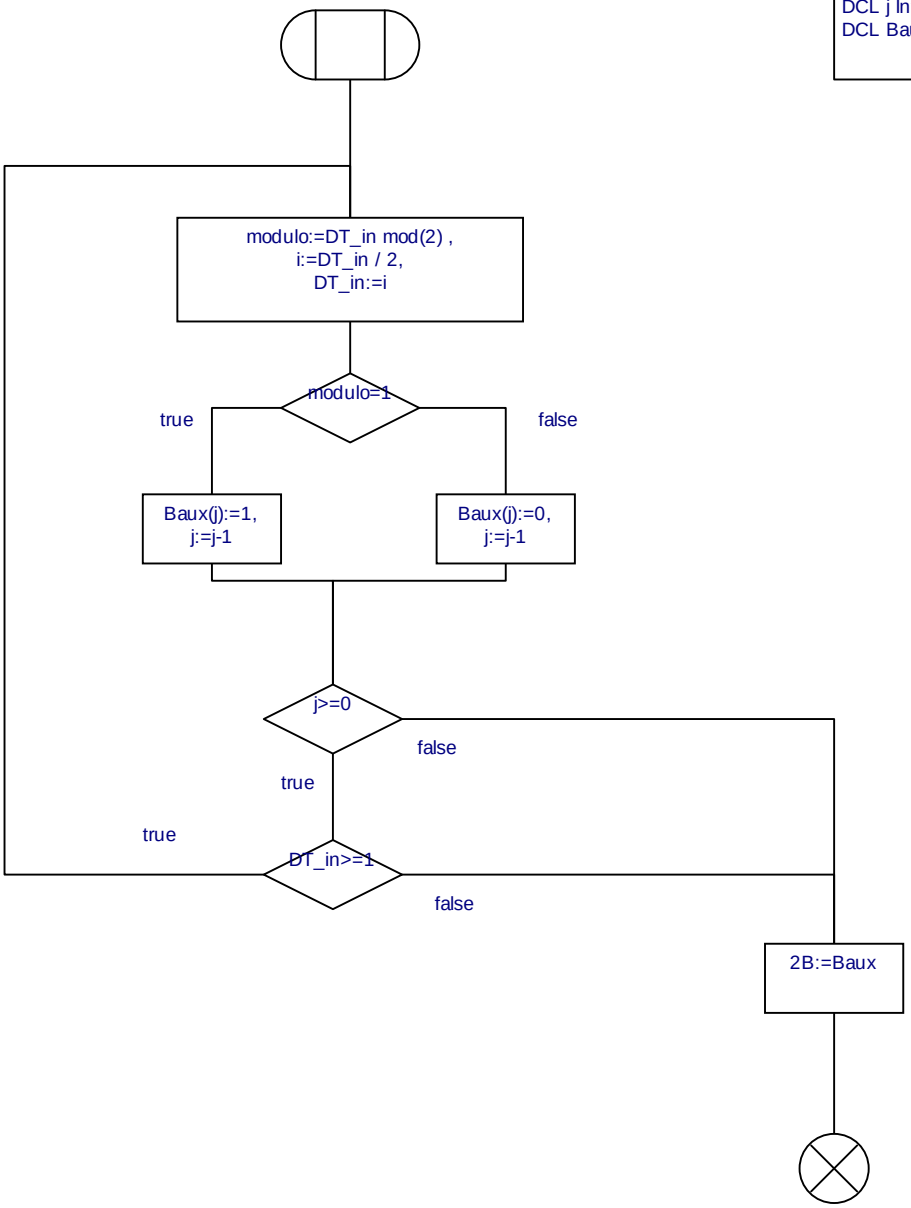


Procedure i2o

1(1)

FPAR DT_in Integer;
RETURNS 2B Bitstring;

DCL modulo Integer;
DCL i Integer;
DCL j Integer:=7;
DCL Baux Bitstring:='00'H;

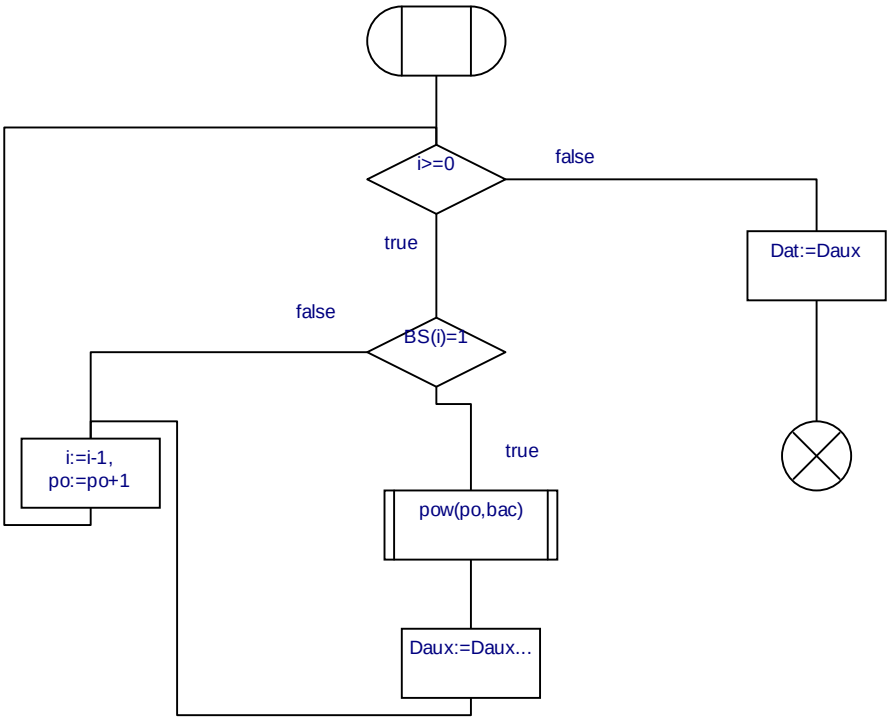


Procedure o2i

1(1)

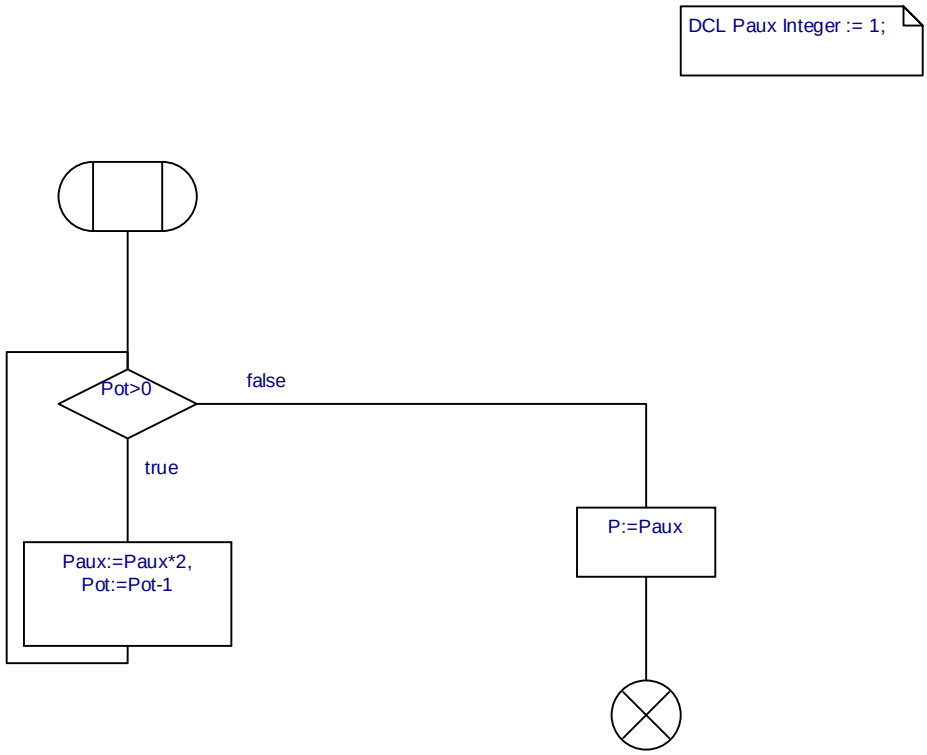
FPAR BS Bitstring;
RETURNS Dat Integer;

DCL i Integer:=7;
DCL bac Integer;
DCL po Integer:=0;
DCL Daux Integer:=0;



Procedure pow 1(1)

FPAR Pot Integer;
RETURNS P Integer;

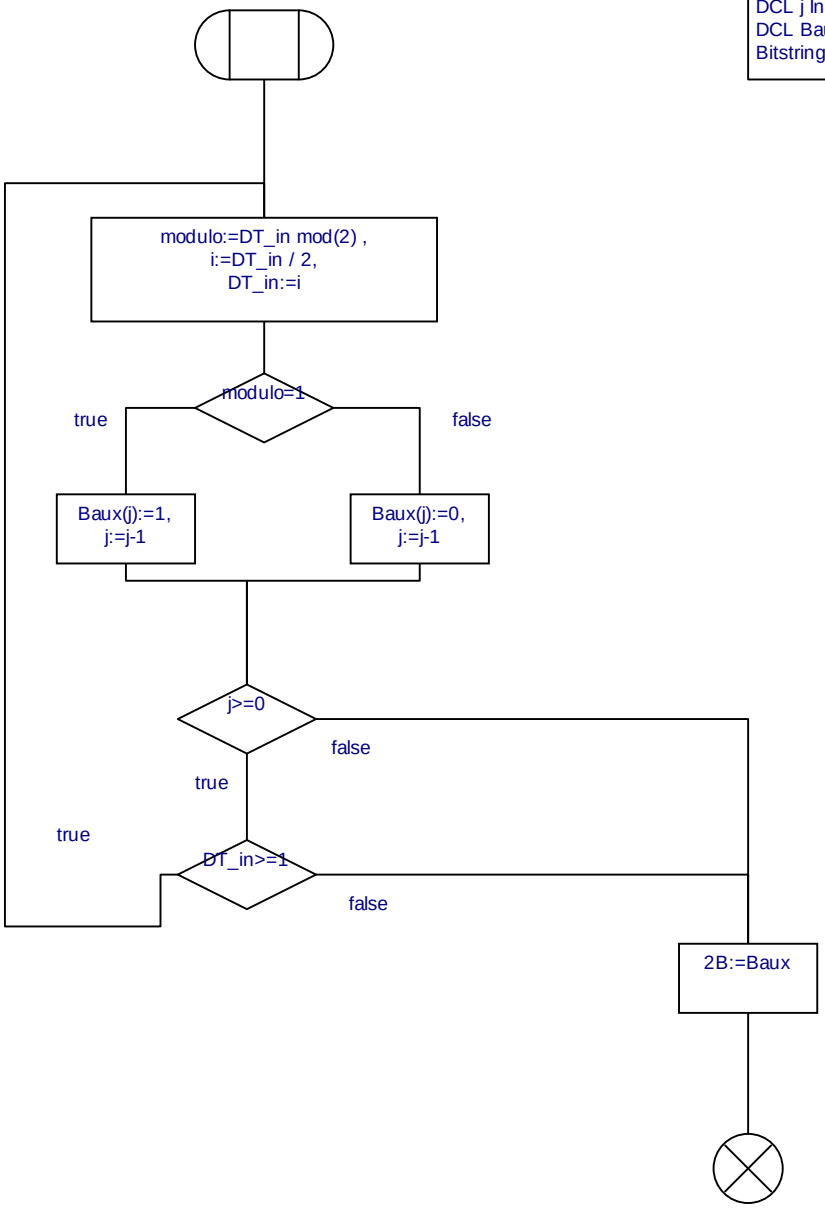


Procedure i2B6

1(1)

FPAR DT_in Integer;
RETURNS 2B Bitstring;

DCL modulo Integer;
DCL i Integer:=0;
DCL j Integer:=5;
DCL Baux
Bitstring:='000000'B;

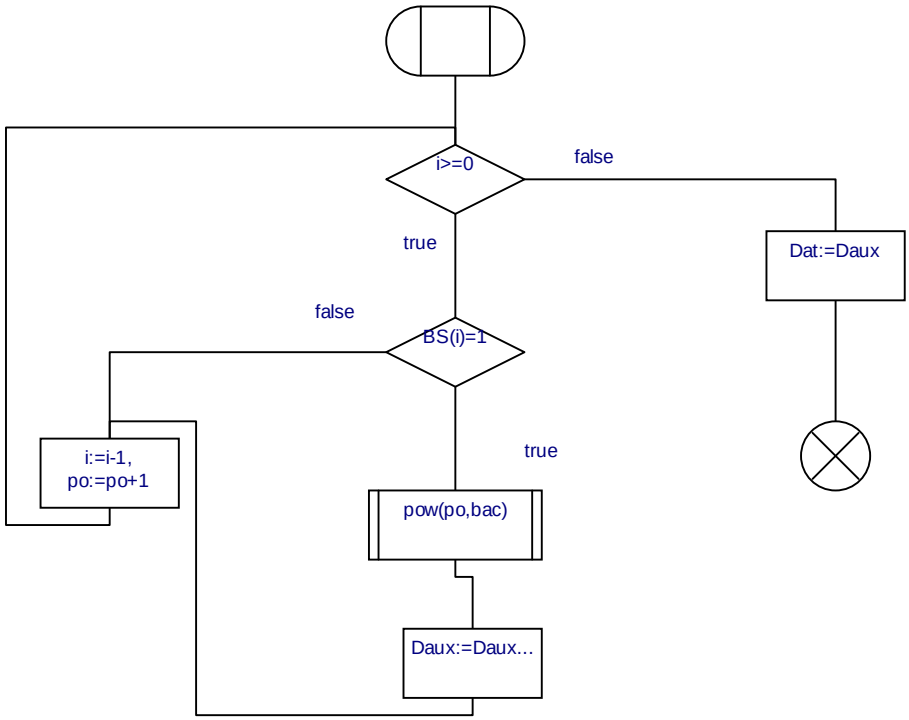


Procedure B62i

1(1)

FPAR BS Bitstring;
RETURNS Dat Integer;

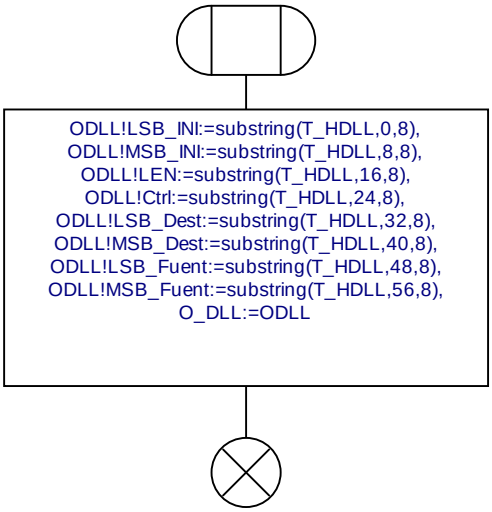
DCL i Integer:=5;
DCL bac Integer:=2;
DCL po Integer:=0;
DCL Daux Integer:=0;



Procedure Obten_HDLL 1(1)

FPAR T_HDLL Bitstring;
RETURNS O_DLL H_DLL;

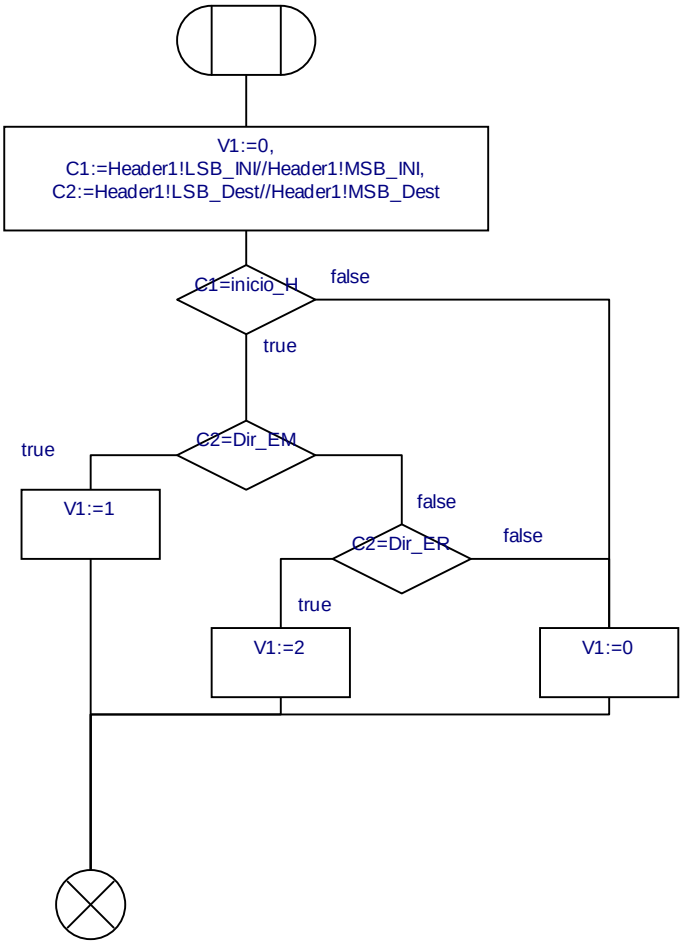
DCL ODLL H_DLL;



Procedure Verif_Dir 1(1)

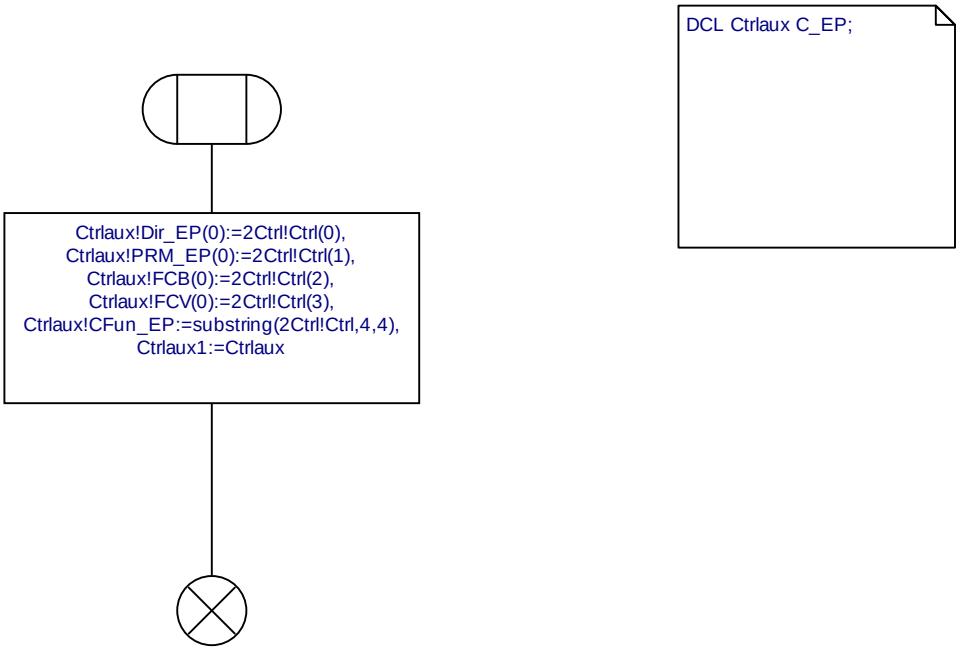
FPAR Header1 H_DLL;
RETURNS V1 Integer;

DCL C1 Bitstring;
DCL C2 Bitstring;



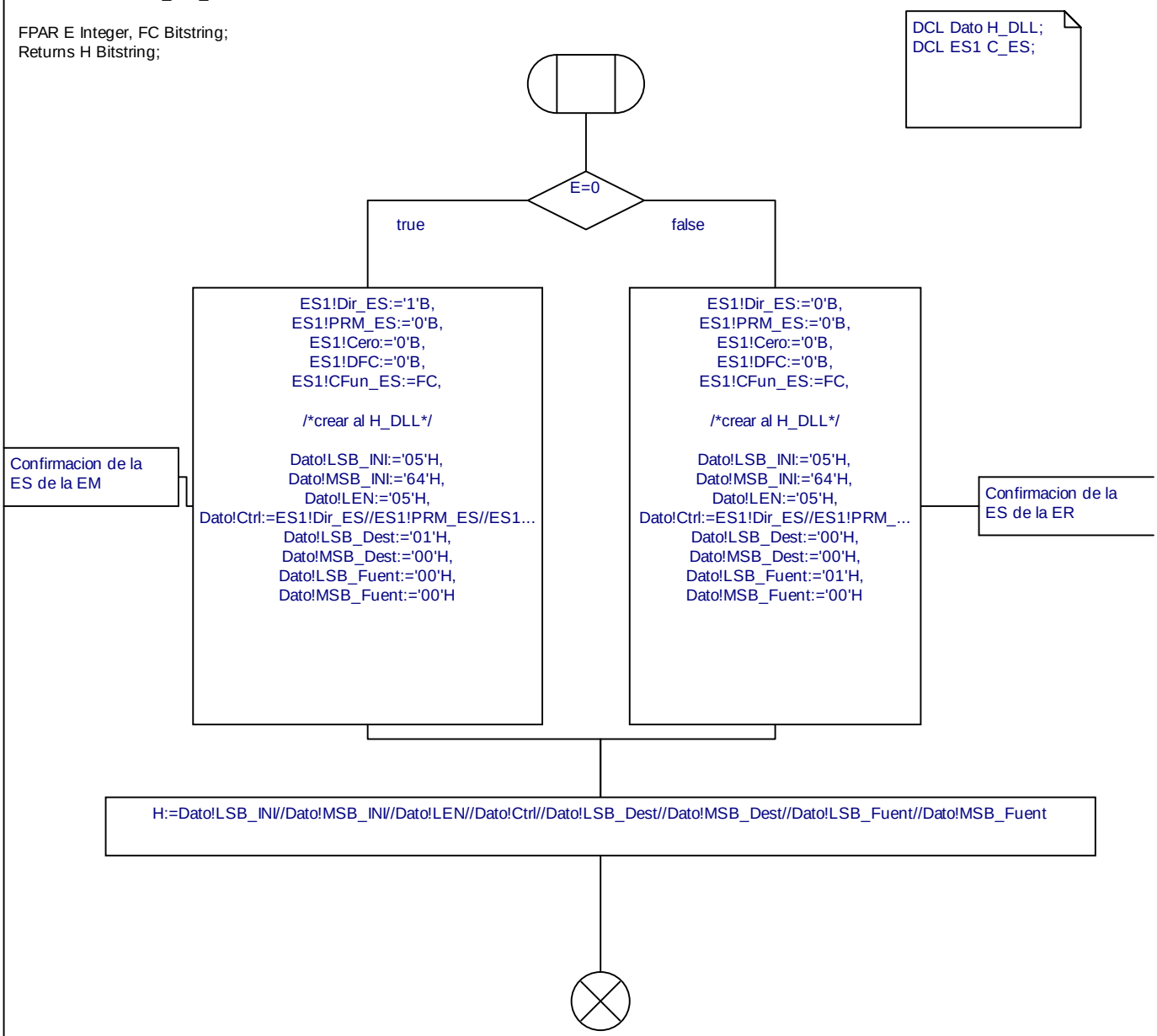
Procedure Ob_Ctrl_4ES 1(1)

FPAR 2Ctrl H_DLL;
Returns Ctrlaux1 C_EP;



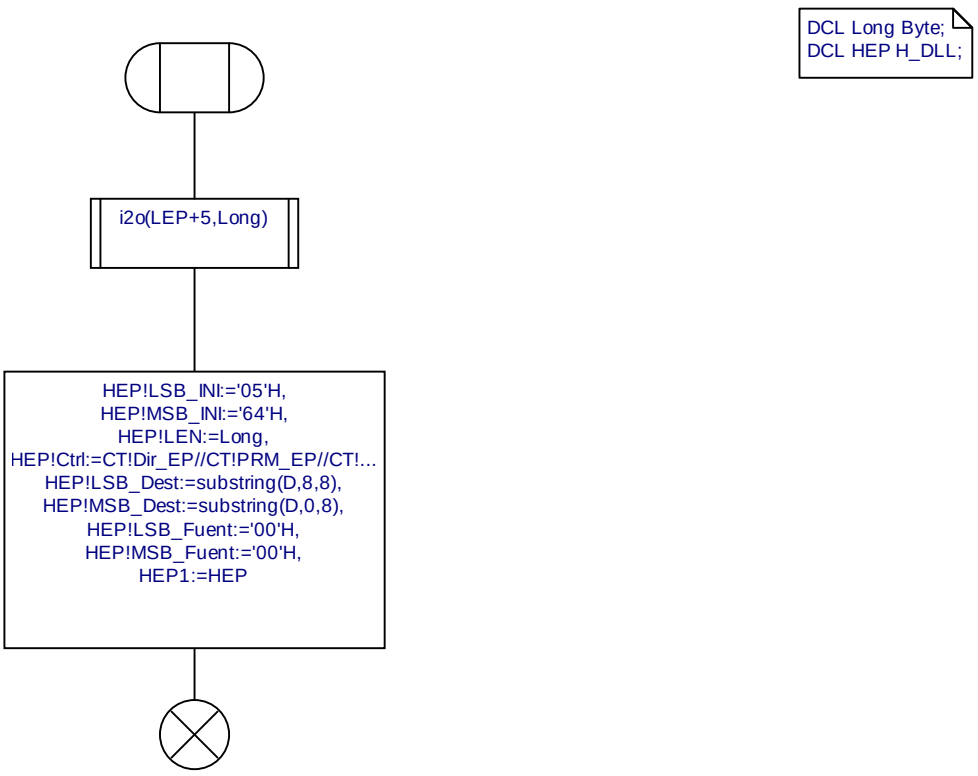
Procedure Gen_Res_ES 1(1)

FPAR E Integer, FC Bitstring;
Returns H Bitstring;



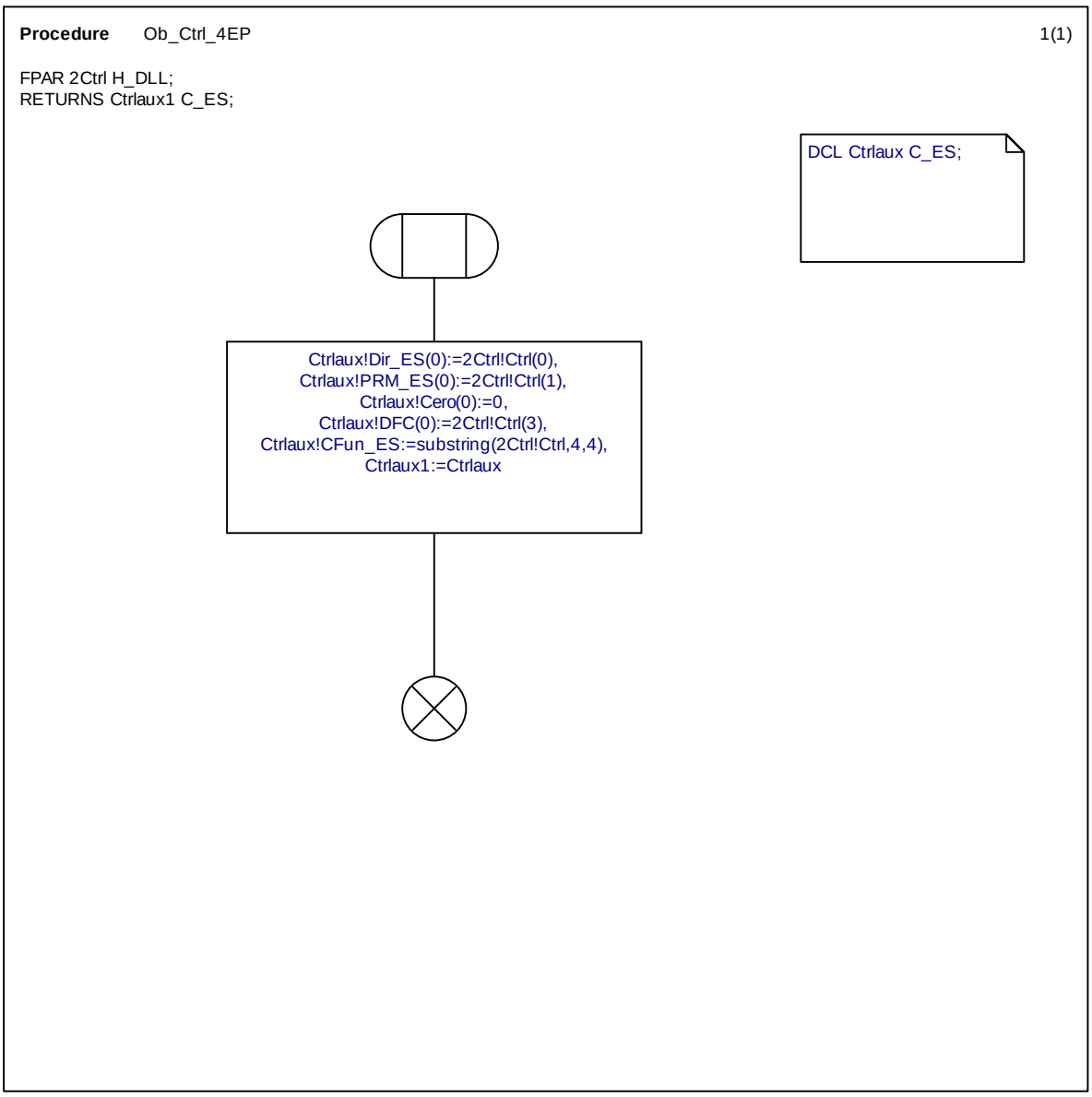
Procedure Gen_Pet_EP 1(1)

FPAR LEP Integer,CT C_EP,D Bitstring;
RETURNS HEP1 H_DLL;



Procedure Ob_Ctrl_4EP 1(1)

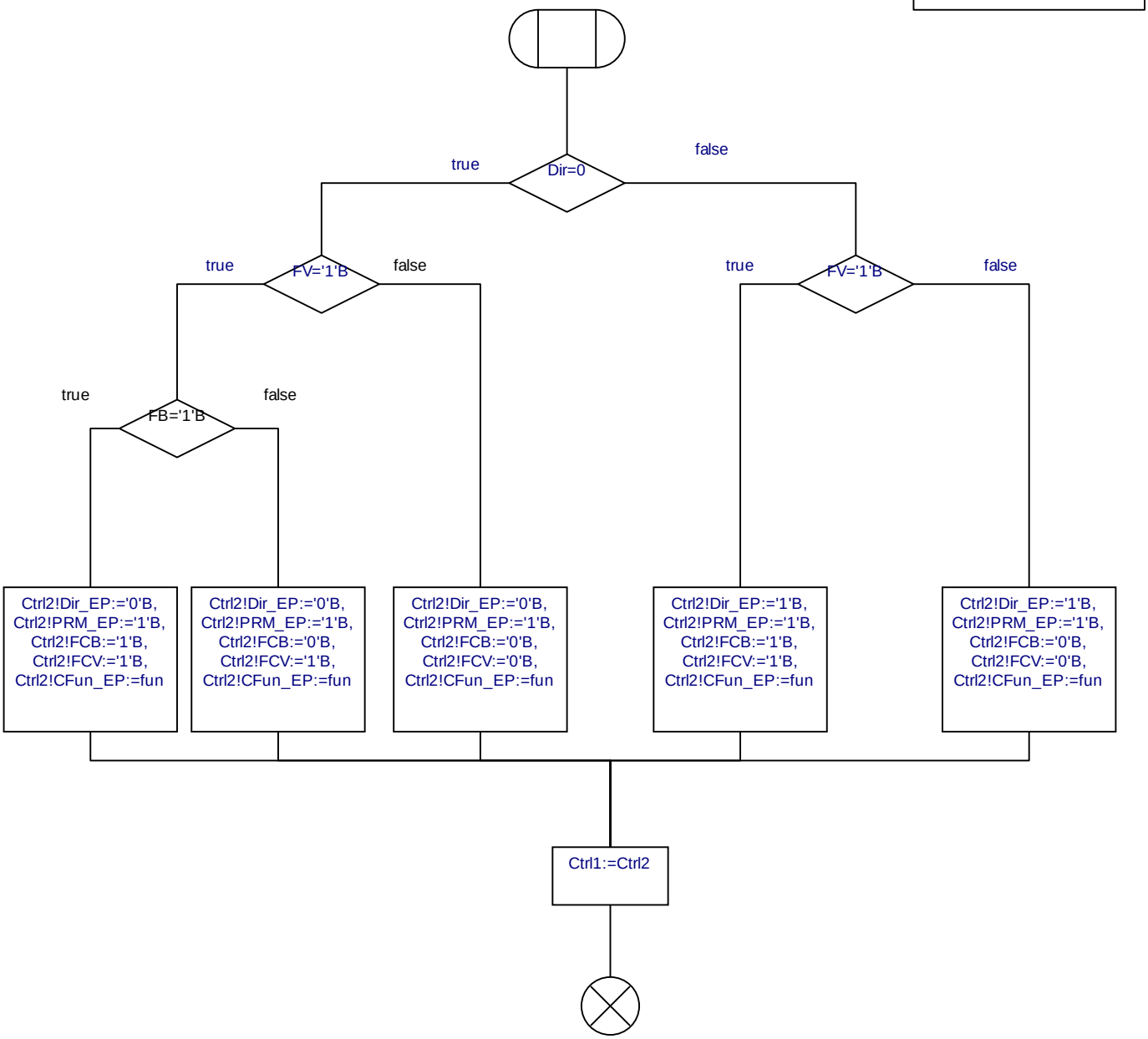
FPAR 2Ctrl H_DLL;
RETURNS Ctrlaux1 C_ES;



Procedure Gen_CtrlDLL 1(1)

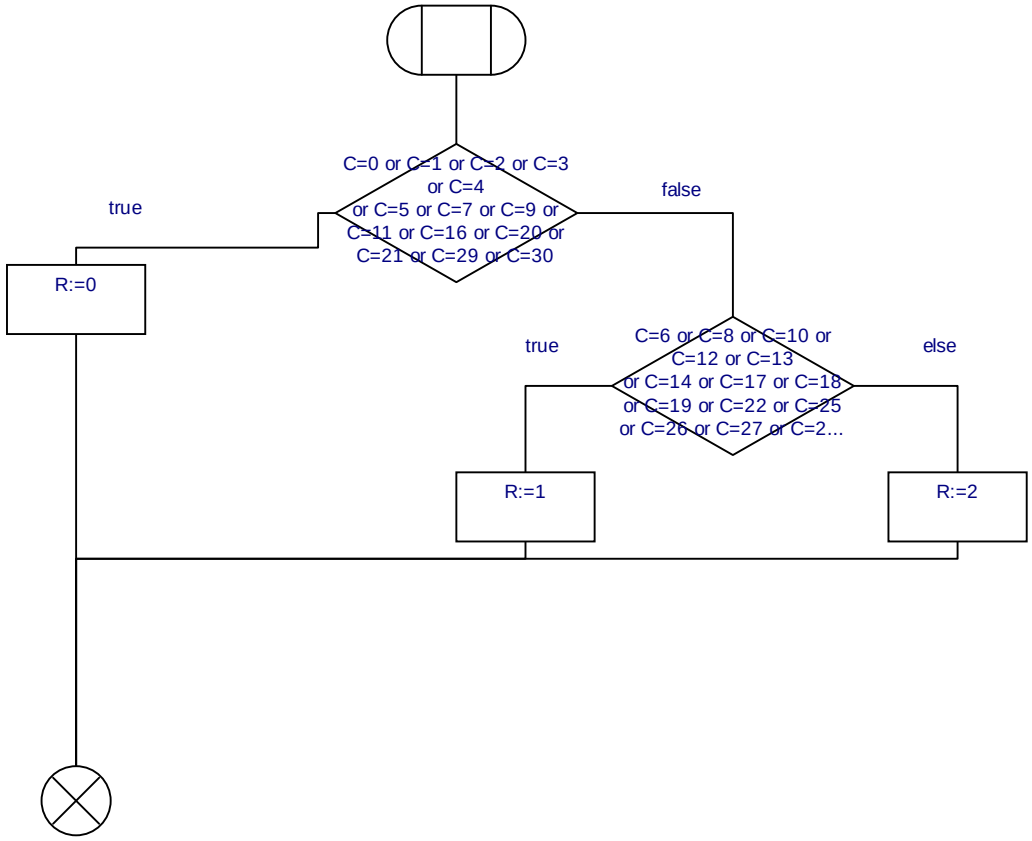
FPAR Dir Integer, FV Bit_t, FB Bit_t, fun Bit_4;
Returns Ctrl1 C_EP;

DCL Ctrl2 C_EP;



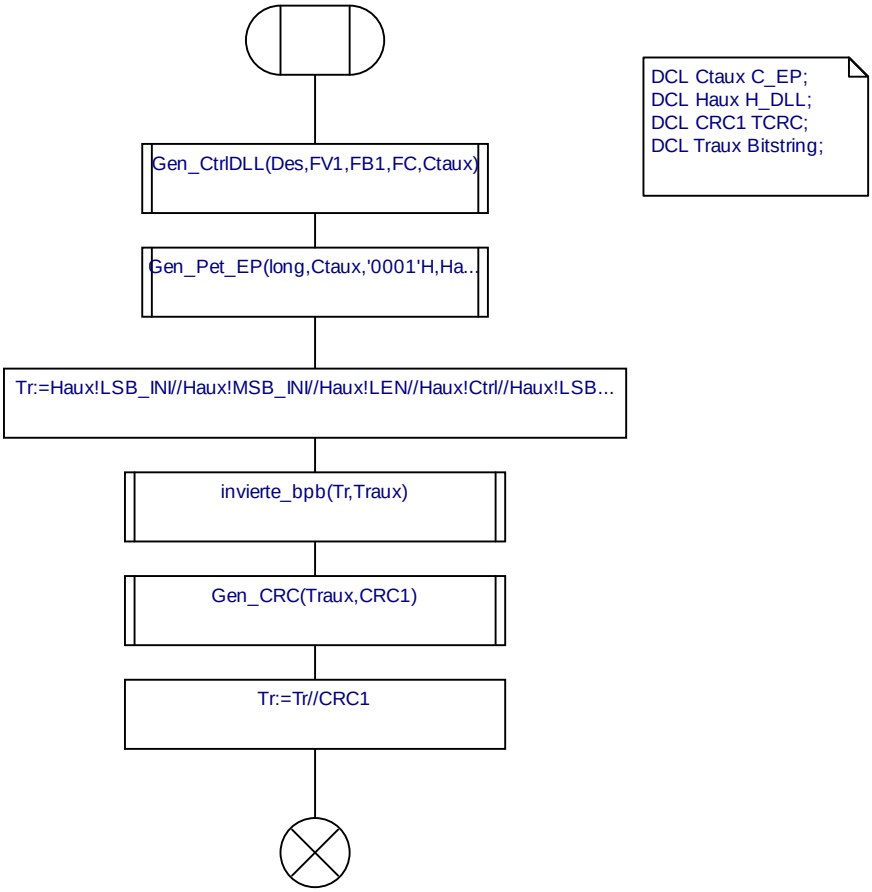
Procedure Resp_T 1(1)

FPAR C Integer;
Returns R Integer;



Procedure Gen_Trm_H 1(1)

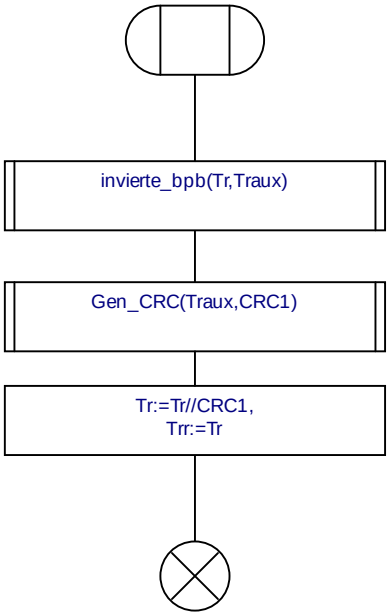
FPAR Des Integer, FV1 Bit_t, FB1 Bit_t,FC Bit_4,long Integer;
Returns Tr Bitstring;



Procedure Gen_Trm_D 1(1)

FPAR Tr BitString;
Returns Trr BitString;

DCL Traux BitString;
DCL CRC1 TCRC;



Procedure Ob_HTF 1(1)

FPAR D2 Bitstring;
Returns H2 H_TF;