UNIVERSITAT JAUME I

ESCOLA SUPERIOR DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

GRADO EN INGENIERÍA ELÉCTRICA

METODOLOGÍA Y HERRAMIENTAS PARA LA SIMULACIÓN EN MATLAB/SIMULINK DE SISTEMAS DE CONTROL BASADOS EN LA NORMA IEC-61499

TRABAJO FIN DE GRADO

AUTOR/A

Héctor Sánchez Bou

DIRECTOR/A

Julio Ariel Romero Pérez

Castellón, Noviembre de 2017

ÍNDICE

1. INTRODUCCIÓN

En los últimos años, ha crecido la demanda de una mayor flexibilidad y reconfigurabilidad del sistema de control en la industria de la automatización. Dando lugar a la propuesta de utilizar un sistema de control distribuido en lugar de disponer de un control centralizado como en los sistemas tradicionales, lo cual introduce un nuevo concepto de estructura descentralizada, con múltiples controladores que manejan dispositivos o subsistemas individuales y más específicos. Estos controladores pueden comunicarse entre sí a través de algún canal de comunicación, enviándole así ordenes e información. Sin embargo, los sistemas descentralizados tienen un diseño más complejo, lo cual hace difícil asegurar un funcionamiento correcto y robusto del controlador, de esta manera, se debe enfatizar la importancia de la verificación y validación el diseño mediante un simulador.

Para abordar esta cuestión, la Comisión Electrotécnica Internacional (IEC) definió un nuevo concepto de diseño de sistemas de control a base de bloques de función (FB), cada uno de los cuales tiene un función específica. De esta manera se potencia la reutilización de software y la interoperabilidad, logrando así una alta reconfigurabilidad. Esta idea fue posteriormente ilustrada y establecida como la norma IEC-61499. Esta cuestión conduce a una idea de vincular herramientas existentes y lenguajes con bloques de función. Esto es exactamente donde un proceso de transformación entra en juego críticamente.

En cuanto a la industria de la automatización, los diseñadores están familiarizados con herramientas tales como MATLAB o LABVIEW, de manera que si se puede realizar una transformación automatizada entre los modelos de estas herramientas existentes a los modelos de bloques de funciones, esto mejoraría seriamente la aceptación industrial de los bloques de funciones. Además, puesto que no existe una herramienta fácilmente utilizable para la validación y verificación de sistemas de bloques de funciones, incluir tal procedimiento de transformación en nuestro entorno de software integrado propuesto puede reunir todos los requisitos para la verificación del sistema antes de su utilización. Ya que, los bloques de función son buenos para diseñar controladores para sistemas distribuidos con capacidad de despliegue directo, mientras que las herramientas de modelado como MATLAB/Simulink son buenas para simulación y análisis de sistemas de control.

En dicho contexto, este proyecto presenta un nuevo método de modelado de sistemas de automatización basándose en los Bloques de función o « Function Blocks » de la norma IEC-61499, mediante el programa 4DIAC, y el bloque « Matlab function » de la herramienta Simulink del programa MATLAB, los cuales serán explicados más adelante. El objetivo final de este proyecto es el de obtener de manera totalmente automatizada una traducción entre ambas plataformas con las que se trabaja (Matlab/Simulink y 4DIAC), diseñando la manera de simular los bloques de la norma IEC-61499 en Matlab/Simulink, de manera que al pasar dicho código por un programa de Matlab, se obtenga un archivo que se pueda ejecutar en el programa 4DIAC (de extensión « .fbt »). Dicha traducción surge de la necesidad de un entorno de simulación para la validación adecuada de los modelos de bloques de función.

De la misma manera, en éste proyecto también se incluye una demostración del correcto funcionamiento de los programas de traducción, tanto de Simulink a 4DIAC como viceversa mediante ejemplos prácticos y una explicación de todos los conceptos que han sido necesarios para la realización del programa de traducción. Así mismo, también incluye una discusión sobre los métodos utilizados para ambas transformaciones y las reglas como la estructura se deberán seguir para una correcta transformación en el programa, muy importante para el correcto funcionamiento del programa en la plataforma a la que se haya traducido.

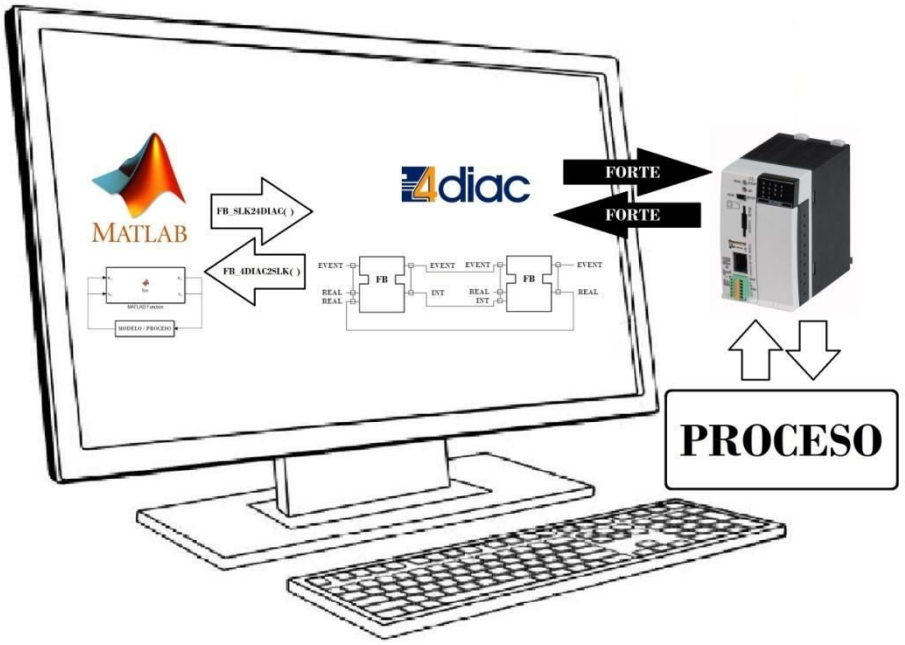

Imagen 1. Esquema general de proceso para la traducción y ejecución de un programa.

2. MARCO TEÓRICO
2.1. IEC-61499

De todo el contenido de la norma, éste proyecto se centra sobre todo en los capítulos 2 y 3, los cuales definen todos los aspectos del bloque de función que se necesitan para entender su funcionamiento.

2.1.1. Definición del Bloque de Función.

El Bloque de Función o « Function Block », es un elemento el cual incorpora un código que realiza una función específica como encender un motor, accionar un pistón o calentar un depósito de agua. Dicho bloque será activado y desactivado mediante una o varias señales externas a las cuales se les denomina « Eventos », de manera que otro bloque de función puede llamar a este bloque y ejecutarlo, por esta razón se dice que son elementos pasivos. A la unión de varios bloques en un mismo programa se le denomina « Aplicación », los cuales pueden o bien llamar o bien ser llamados por otras aplicaciones o por otros bloques de función, formando así el programa. La norma define la ejecución de un Bloque de Función como algorítmica, lo cual quiere decir que acaba en un tiempo finito.

2.1.2. Funcionamiento del Bloque de Función según la IEC-61499

El Bloque de Función se divide en 4 partes claramente diferenciadas (Eventos y Variables, Control de ejecución, Funcionalidad y Recursos), cada una de las cuales tiene una función específica tal y como se puede observar en la *«* Imagen 2 *»*.

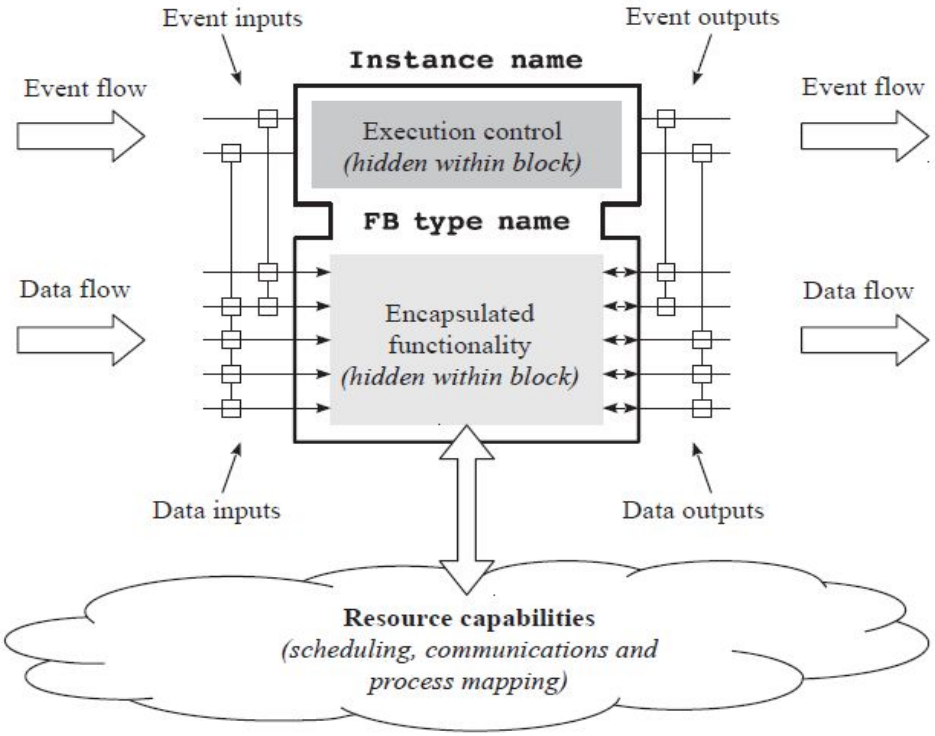

Imagen 2. Estructura del Bloque de función o « FB ».

Para empezar, el Bloque de Función divide tanto las señales de entrada como las de salida del mismo en « Eventos » y « Variables » :

Los Eventos son señales lógicas, por lo cual solamente diferencian entre los estados activado o desactivado. Dichos Eventos se dividen en señales de entrada y de salida. Los Eventos de entrada son señales que indican al Bloque de Función que cierta parte de su código debe ser ejecutado, mientras que los Eventos de salida son señales que activan Eventos de entrada de otros Bloques de Función.

Las Variables son señales que portan datos como la temperatura o el estado de un sensor, al igual que pasa con los Eventos, existen tanto de entrada como de salida. Las Variables de entrada portan datos que proceden o bien de otros Bloques de Función o bien del entorno que se está siendo controlado, mientras que las Variables de salida portan datos que provienen de la

ejecución del mismo Bloque de Función. Además, las Variables se vinculan con uno o más Eventos mediante el cualificador « WITH » (que en su representación gráfica se muestra como un pequeño conector cuadrado que conecta Eventos con sus Variables asociadas), de manera que los valores de las Variables con las que trabaja el Bloque de Función solamente se actualizarán al recibir el Bloque de Función un Evento al cual dicha Variable está vinculada. Es decir, al llegar un Evento, se actualizan todas las Variables que están vinculadas a éste.

Una de las partes más importantes del Bloque de Función es el « Control de ejecución », el cual contiene la información que enlaza la activación de un Evento con una función específica de la « Funcionalidad », haciendo que se ejecute dicho trozo de código. A su vez, también se encarga de gestionar cuando se envía la información de los Eventos de salida o « Outputs ». Se suele representar mediante un diagrama denominado « ECC » por sus siglas en inglés (Execution Control Chart). En dicho diagrama se representa la evolución de los estados del ECC y se indica en cuáles estados se ejecutan ciertos algoritmos o se activan ciertos Eventos de salida. Dicha activación de Eventos de salida y algoritmos, son los trozos de código que forman la « Funcionalidad ».

Un Bloque de Función contiene generalmente uno o más Algoritmos. Mientras éste se ejecuta, La « Funcionalidad » tiene acceso a las Variables de entrada y de salida, al igual que a los « Recursos », los cuales están formados por los datos de las Variables internas e información que el Bloque de Función necesita almacenar para poder acceder a ella en otro momento. La « Funcionalidad » puede modificar los valores de los « Recursos ». Sin embargo, no puede modificar los valores de las Variables de entrada o de salida y no tiene acceso a ningún tipo de datos externos al Bloque de Función. Así mismo, Una vez se ha ejecutado el algoritmo, el « Control de ejecución » ya pude enviar los Eventos de salida para señalar que las Variables de salida han sido actualizadas.

La IEC-61499 no define ningún tipo de lenguaje específico para definir algoritmos. Sin embargo, el lenguaje que generalmente se utiliza es « xml », el cual se utiliza en el programa « 4DIAC » que se explica en el siguiente punto.

El Diagrama de Control de Ejecución o « ECC », el cual tiene una estructura como la que se muestra en la « Imagen 3 », tiene una estructura similar a la del Sequential Function Chart o « SFC » de la norma IEC-61131-3 (predecesora de la norma IEC-61499). Sin embargo, el propósito y el comportamiento del ECC es muy distinto al del SFC, por lo cual no deben ser confundidos.

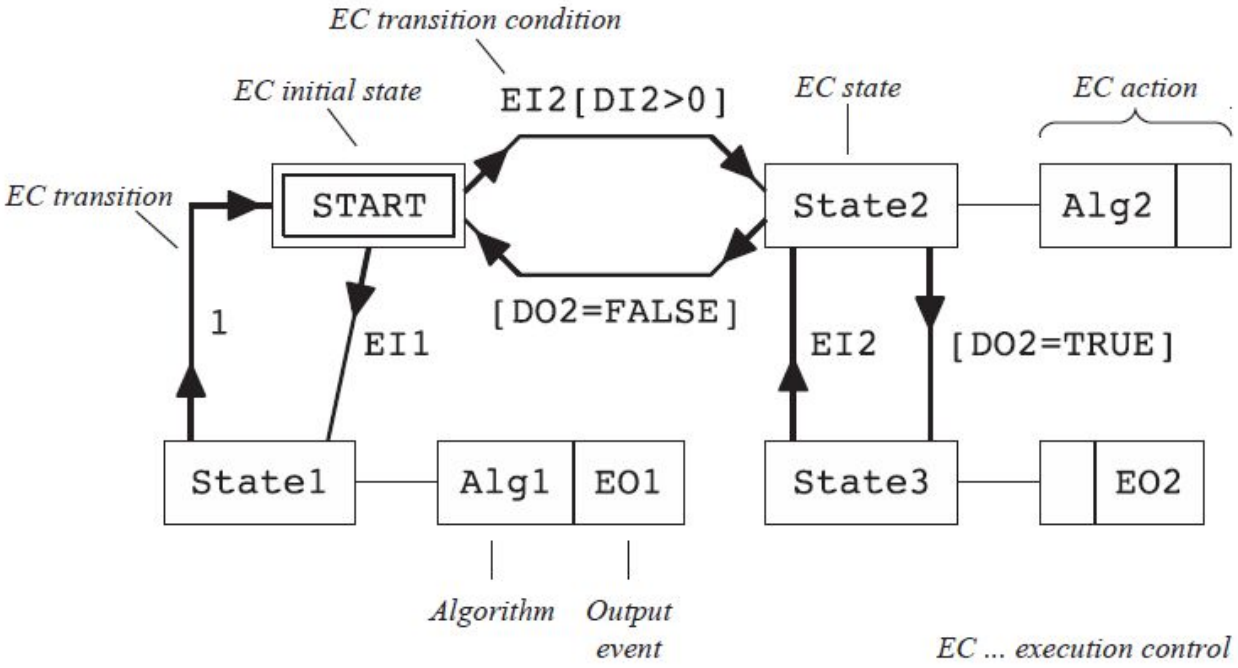

Imagen 3. Ejemplo de diagrama de control de ejecución o « ECC ».

La norma define a su vez características y reglas que se deben aplicar al ECC y que por lo tanto se deben tener en cuenta a la hora de realizar el diagrama, las cuales son :

- Cada Bloque de Función debe contener exclusivamente un ECC
- El ECC puede utilizar pero no modificar Variables declaradas en el Bloque de Función
- Una vez las Variables de entrada han sido muestreadas, estas no vuelven a cambiar su valor en toda la ejecución del Bloque de Función y se haya activado algún Evento de entrada de nuevo. Esto asegura estabilidad durante la ejecución del Bloque de Función.
- El estado START representa el estado inicial del ECC y se representa con una doble línea
- Las transiciones definen los posibles cambios de estado y están asociadas a condiciones Booleanas
- Las condiciones de las transiciones pueden ser o bien eventos, expresiones Booleanas o expresión incondicional (que siempre sea verdadero).
- Al igual que varias transiciones pueden acabar en un mismo Estado, también pueden salir varias transiciones de un mismo Estado. Sin embargo, el hecho de que haya varias posibilidades de salir de un Estado puede dar problemas en el caso que se cumplan dos o más transiciones a la vez. Por lo cual, se hace necesario asignarles un nivel de prioridad a las transiciones que salen de un mismo Estado, evitando así posibles errores. Dicha prioridad va marcada por el orden en el cual se declaran dichas transiciones.
- El modelo de la norma IEC-6499 asume que no hay colas de eventos de entrada asociadas a las entradas del bloque de función. Sin embargo, también define que el entorno de ejecución debe asegurar que solamente un Evento es enviado en un mismo instante de tiempo.

2.2. 4DIAC

Para modelar un sistema de control distribuido según la norma IEC-61499 existen varias herramientas de software tanto libres como de pago, pero sin duda una de las conocidas y usadas en el desarrollo de estudios experimentales es 4DIAC. que es un programa de código abierto basado en la plataforma Eclipse. Mediante éste programa se puede obtener un entorno de automatización y control en el cual se desarrollan nuevos Bloques de función que junto con los ya existentes en el estándar, conformarán la aplicación de control deseada que posteriormente se mapeará en los distintos recursos del mismo dispositivo formando un sistema de control distribuido.

La ejecución de la aplicación modelada se realiza mediante el run-time de « 4DIAC », llamado « FORTE », el cual es una implementación conforme a la IEC-61499, enfocado a pequeños dispositivos de control empotrados. Se encuentra implementado en C++ y puede ser aplicado sobre múltiples plataformas.

El objetivo de « 4DIAC » es proporcionar herramientas que conforme al estándar permiten establecer un entorno de automatización basado en los objetivos de portabilidad, configurabilidad e interoperabilidad, mencionados en la IEC-61499. De esta manera, « 4DIAC » trata de proporcionar un incentivo para el uso en la industria de la IEC-61499 incluyendo Bloques de función de comunicación (Client/Server y Publish/Subscribe para Ethernet).

2.3. Otras definiciones importantes.

- Durante todo el proyecto se habla de variables. Sin embargo, hay una diferencia importante entre las « Variables » y las « variables ». Las primeras se refieren a las entradas y salidas del Bloque de función que no son Eventos, mientras que las segundas son las se se utilizan en los programas de Matlab (los traductores) para almacenar ciertos datos. Algo parecido sucede con los « Estados » y los « estados », ya que los primeros hacen referencia a un componente del « ECC », mientras que los segundos se refieren a si los primeros se encuentran activados o desactivados.

- Cuando en el proyecto se habla de un « array », se refiere a un vector, bien sea de números, de caracteres o de cadenas de caracteres. De la misma forma, los « Outputs » son los Eventos de salida.

- Un Estado inestable es un Estado cuya transición posterior a esta pone en riesgo que dicho Estado no se llegue a ejecutar. Esto sucede cuando la condición de dicha transición es « 1 », « TRUE » o igual a la transición que precede a dicho Estado. En éste caso, la norma obliga a evitar que dicho Estado no se ejecute, sino que siempre se ejecuten como mínimo una vez antes de cambiar de Estado.

2.4. Normas para la introducción de datos en el Bloque de función.

Al estar trabajando con dos lenguajes de programación distintos como son MATLAB y el « xml », puede darse que existan algunas incompatibilidades. Por lo cual, se hace necesario definir unas normas acerca de la manera en la cual se deben introducir ciertos datos en el Bloque de Función.

Para empezar, existen ciertos caracteres que no pueden contener ni los nombres de los Eventos, ni los de las Variables. Dichos caracteres son; « ' », « " » y « # ». Además, también existen ciertas palabras por las que no pueden comenzar los nombres de los Eventos o de las Variables; « if », « IF », « while », « WHILE », « for », « FOR », « else », « ELSE », « ELSIF », « switch », « CASE », « otherwise », « end », « END », « e_ » y « v_ ». Éstos dos últimos, aunque se deban introducir precediendo los nombres de los Eventos o las Variables respectivamente, donde no se deben introducir es en el mismo nombre. De la misma manera, tanto las Variables como los Eventos pueden contener números en su nombre, pero deben comenzar por una letra y no pueden incluir por incompatibilidad en los programas los siguientes elementos: « ¡ », « ! », « ” », « # », « $ », « % », « & », « ` », « (», «) », « - », « = », « ^ », « ~ », « \ », « / », « | », « @ », « [», «] », « { », « } », « ; », « : », « + », « * », « , », « . », « < », « > », « ¿ » y « ? ».

Otro dato importante, es que en 4DIAC es posible declarar una Variable dentro de un algoritmo, de manera que dicha Variable solamente se puede utilizar en dicho algoritmo. Sin embargo, en el Bloque de Función de Simulink no se diferencian las Variables que pertenecen a un algoritmo, por lo que dos Variables con el mismo nombre se entenderían como la misma, de manera que dichas Variables deben ser declaradas en el Bloque de Función como Variables Internas.

En cuanto a los algoritmos y las condiciones de las transiciones entre los estados del « ECC » en Simulink, solamente podrán incluir elementos compatibles con ambos programas, los cuales son:

- Nombres de variables que hayan sido ya declaradas con anterioridad y números.
- Los operadores matemáticos « + », « - », « * », « / » y « ^ ».
- Asignaciones « = » y paréntesis « ( ) ».
- Los operadores lógicos « || », « && », « >= », « <= », « > », « < », « == », « ~= » y « ~ ».
- Las funciones « abs() », « sqrt() », « log() », « log10() », « exp() », « sin() », « cos() », « tan() », « asin() », « acos() », « atan() », « not() » y « length() ».

Además de ésto, los algoritmos (y no las transiciones) pueden incluir ciertas instrucciones

- Bucles « while », « for », « switch » y condiciones « if ». Los cuales pueden estar anidados (ej: un bucle « for » que incluya una condición « if »).

Todos los operadores y funciones que se han mostrado aparecen con la forma de introducirlos en Matlab. Sin embargo, algunos de ellos deben introducirse en « xml » de una farma distinta tal y como se indica en la *«* Tabla 1 *»*.

Tipo de Variable	En Matlab	En « xml »		Tipo de Variable	En Matlab	En « xml »
Exponencial	^	**		Valor absoluto	abs()	ABS()
Operador de asignación	=	:=		Raíz cuadrada	sqrt()	SQRT()
Operador lógico OR	||	OR		Logaritmo natural	log()	LN()
Operador lógico AND	&&	AND		Logaritmo de base 10	log10()	LOG()
Operador mayor o igual que	>=	≥		Exponencial natural	exp()	EXP()
Operador menor o igual que	<=	≤		Función seno	sin()	SIN()
Operador mayor que	>	>		Función coseno	cos()	COS()
Operador menor que	<	<		Función tangente	tan()	TAN()
Operador de igualdad	==	=		Función arcoseno	asin()	ASIN()
Operador de desigualdad	~=	<>		Función arcocoseno	acos()	ACOS()
Operador/ función de negación	~, not()	NOT()		Función arcotangente	atan()	ATAN()
				Extraer la longitud de una cadena	length()	LEN()

Tabla 1. Formas de introducir operadores y funciones en « Matalb » y en « xml ».

Por último, el traductor necesita saber el tipo de las Variables del programa para poder declararlas correctamente. El tipo de dato de una Variable se introduce cuando ésta declara su valor inicial e indica el rango de datos que se pueden introducir en dicha Variable. Al declarar los valores iniciales se debe, por tanto, declarar el tipo de Variable, el cual irá definido junto a su estado inicial según la *«* Tabla 2 *»*. La *«* Imagen 18 *»* del apartado *«* **3.2. Simulación del Bloque de Función en Matlab/Simulink** *»*.

Tipo de Variable	En Matlab	En « xml »
Número entero	0	InitialValue="0" Type="INT"
Número real	0.0	InitialValue="0" Type="REAL"
Booleano	logical(0)	InitialValue="0" Type="BOOL"
Número Real de doble longitud	double(0)	InitialValue="0" Type="LREAL"
Número Entero corto	int8(0)	InitialValue="0" Type="SINT"
Número Doble entero	int32(0)	InitialValue="0" Type="DINT"
Número Entero largo	int64(0)	InitialValue="0" Type="LINT"
Número Entero corto sin signo	uint8(0)	InitialValue="0" Type="USINT"
Número Entero sin signo	uint16(0)	InitialValue="0" Type="UINT"
Número Doble entero sin signo	uint32(0)	InitialValue="0" Type="UDINT"
Número Entero largo sin signo	uint64(0)	InitialValue="0" Type="ULINT"
Cadena de caracteres	'0'	InitialValue="0" Type="STRING"

Tabla 2. Formas de introducir « 0 » con los distintos tipos de Variables.

Un dato importante de las cadenas de caracteres es que no deben incluir espacios, en su lugar se pueden utilizar caracteres « _ ». Así mismo, para introducir su valor inicial como una cadena vacía, se debe introducir únicamente « _ ».

2.5. Funciones de Matlab utilizadas en los programas.

A la hora de traducir el código a la norma IEC-61499 a Simulink y viceversa, se han estudiado distintas funciones que incorpora la librería de Matlab para tratar el código y poder traducirlo, de esta manera se ha utilizado:

Para trabajar con archivos hacen falta funciones que abran, lean o escriban y cierren el archivo al acabar. Para ello se han utilizado las siguientes funciones :

- Se han utilizado las funciones « fopen() » y « fclose() » para abrir y cerrar tanto el archivo del que se traduce (fuente) como del archivo al cual se traduce (destino), creado al ejecutar el programa. De esta forma, se puede trabajar con la información del archivo fuente para generar el archivo destino.

- La función « fgetl() » se utiliza una vez abierto el archivo fuente para extraer y analizar línea por línea todo el archivo. De esta manera, se puede buscar más fácilmente la información necesaria en el cada parte de la estructura del Bloque de Función en el archivo destino. Otras funciones similares que surgieron como alternativa fueron « fgets() » y « fread() », pero resultaban problemáticas ya que « fread() » extrae toda la información del archivo de golpe en una cadena de caracteres, mientras que « fgets() », aún pareciéndose a « fgetl() » ya que extrae la información línea por línea, mantiene los caracteres de salto de línea y por tanto daba problemas al trabajar con ella. Por otra parte, la función utilizada lee una línea cada vez que la función es llamada y solo la información escrita en dicha línea, de manera que se puede ir leyendo el archivo hasta que se encuentre lo deseado, simplificando el programa.

- La función « fprintf() » se utiliza, una vez ya se ha extraído la información necesaria, para escribir en el archivo destino una cadena de caracteres. De esta forma, se puede crear la estructura deseada en el archivo destino. Otra función similar que surgió como alternativa fue « fwrite() », pero no era útil ya que ésta función escribe los elementos de un « array » en código binario en vez de utilizar caracteres, por lo cual aún haciéndose bien la traducción, el programa 4DIAC no podría leerlo.

- La función « disp() » se utiliza únicamente en los traductores para mostrar por pantalla al ejecutar el traductor que ha habido algún error o bien leyendo el archivo o bien con cierta información que se ha leído en el archivo fuente.

- La función « strfind() » se utiliza para buscar uno o varios caracteres en una cadena de caracteres, devolviendo las posiciones de éstos en dicha cadena dentro de un « array », que estará vacío si no se ha encontrado. De esta manera, comprobamos si la información que hemos extraído del documento es la que estamos buscando o no. Como alternativa existen dos funciones « regexp() » y « regexpi() », pero ambas devuelven solamente la primera y la última posición, por lo que se pierde información que en algunas ocasiones va a ser útil.

- La función « isempty() » se utiliza para saber si algún « array » o alguna cadena de caracteres está vacía, devolviendo « 1 » si se encuentra vacío o « 0 » en caso contrario. En el proyecto se suele utilizar junto con la anterior función cuando solamente se quiere comprobar si existe cierta información en una cadena de caracteres o no.

- La función « strrep() » se utiliza para reemplazar parte de una cadena de caracteres por otro carácter o cadena de caracteres. De esta manera, se puede modificar la información del archivo y traducirla al lenguaje del programa de destino. Como alternativa existe la función « regexprep() », la cual tiene la misma función que « strrep() » pero con más opciones a la hora de reemplazar (como buscar un carácter en concreto que vaya seguido de algo y otro carácter en concreto) que no van a ser necesarias para la realización de este proyecto, haciéndola más complicada de utilizar que « strrep() ».

- La función « strtok() » se utiliza para dividir la cadena donde se encuentra un carácter en concreto (delimitador), devolviendo ambas partes de la división. De esta manera, se puede tratar la información por trozos facilitando así la traducción. Como alternativa existe la función « strsplit() », la cual divide una cadena de caracteres en trozos por un delimitador y devuelve un « array » de cadenas de caracteres con los trozos de la cadena original. Al final, las dos funciones son válidas para este proyecto, pero la función « strtok » resulta más cómoda, ya que se suele utilizar dentro de bucles o condiciones y a medida que se va tratando cada trozo de información, se va recorriendo la misma cadena sin necesidad de utilizar más variables.

- La función « textscan() » se utiliza para dividir cadenas de caracteres en celdas que contienen un « array » de cadenas de caracteres. De esta manera, incluyendo la estructura que debe seguir dicha cadena de cráteres a la hora de llamar a la función (ej : A = textscan(cadena, '%s %s') divide la « cadena » en tres celdas, introduciendo la primera palabra en la primera celda, la segunda palabra en la segunda celda, la tercera palabra en la primera celda de nuevo y así sucesivamente), se puede dividir y ordenar la información en celdas según lo que contengan. En el proyecto se suele

utilizar para extraer de manera ordenada el nombre de una variable, su valor inicial y su tipo, de manera que no se mezclen y se sepa qué datos corresponden con cuales.

- La función « char() » se utiliza para transformar a cadena de caracteres cierta información. En el proyecto es muy útil a la hora de tratar con la información que obtenemos de la función « textscan() » ya que el resultado de ésta son celdas y transformándose a cadenas de caracteres es más fácil tratar con esta información.

- La función « length() » se utiliza para saber la longitud de una cadena de caracteres. En el proyecto es muy útil a la hora de recorrer dichas cadenas carácter a carácter. Sobre todo en algunos casos que la cadena se debe recorrer de final a inicio para evitar posibles problemas.

- Las funciones « strcmp() » y « isequal() » se utilizan para comprobar si dos cadenas de caracteres contienen la misma información. En el proyecto su función más importante es indicar cuándo se ha llegado a uno de los apartados en concreto del archivo del cual se quiere traducir o cuando ya se ha acabado dicho apartado. Se usan indistintamente ambas funciones ya que realizan la misma acción.

- La función « strcat() » se utiliza para unir (concatenar) cadenas de caracteres. Sin embargo, hay que tener cuidado con los espacios al inicio y al final de las cadenas, ya que la función los elimina al concatenar cadenas. En el proyecto es útil para introducir en alguna variable trozos de información que tienen que ser tratados por partes antes de introducirlos en el nuevo archivo. En este caso se utiliza el carácter « " » para indicar los espacios y posteriormente se reemplazan por estos.

- La función « isletter() » se utiliza para comprobar si un carácter es una letra o no, en el caso de que si sea una letra el resultado será « 1 » y en caso contrario « 0 » (para una cadena de caracteres se obtiene un « array » de resultados). En el proyecto se utiliza sobre todo para saber donde acaba el nombre de una variable que esté introducida por ejemplo en una transición o un algoritmo.

- La función « sort() » se utiliza para ordenar los elementos de un « array » en orden ascendente. En el proyecto se utiliza sobre todo para ordenar los posibles bucles (o condiciones « if ») de un algoritmo que estén dentro de otro bucle (o de una condición « if ») e identificar cual está dentro de cual.

De la misma forma, a la hora de trabajar en Simulink se ha escogido el bloque « MATLAB function » para simular el Bloque de Función ya que gráficamente utiliza una estructura muy similar a este e internamente también puede ser programado mediante código. Además, la interfaz gráfica facilita la utilización de diversas herramientas para la simulación que matlab posee como los « Scope ».

3. DESARROLLO

3.1. Estructura del Bloque de Función en « xml » para « 4DIAC »

A la hora de incluir un Bloque de Función en 4DIAC, éste debe estar escrito con una estructura específica en el lenguaje « xml ». Para mostrar dicha estructura nos basaremos en el Bloque de Función “Event_driven_PID” cuya interfaz gráfica se muestra en la « Imagen 4 ».

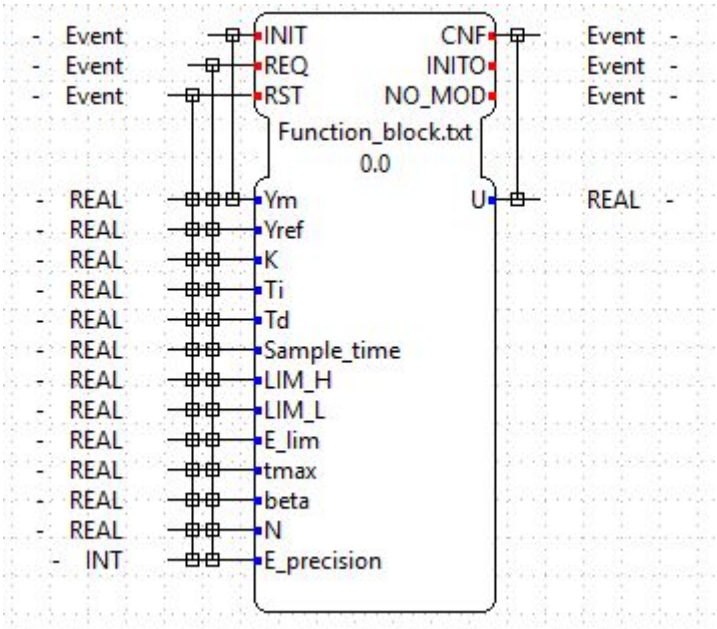

Imagen 4. Estructura del « FB » en « 4DIAC », interfaz gráfica del Bloque de Función.

La estructura, se divide en dos partes claramente diferenciadas por una serie de instrucciones : « <InterfaceList> » y « <BasicFB> » antes de las cuales se introducen ciertas características predeterminadas. De manera que :

Lo primero que debe tener son las características del bloque. Con lo cual, todos los bloques deberán contener la misma información que se muestra en la *«* Imagen 5 *»*.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">
3  <FBType Comment="Tamplate for a simple Basic Function Block Type" Name="Event_driven_PID">
```

Imagen 5. Estructura del « FB » en « 4DIAC », declaración del tipo de versión.

Una vez hecho esto, ya se puede escribir información en el bloque de función « FBType », se deberá introducir un doble espacio antes de cada una de las siguientes instrucciones para indicar que lo que se introduce pertenece al « FBType ».

De esta manera, lo siguiente que se debe introducir es el tipo de versión del bloque y distintos identificadores, de manera que todos los Bloques de Función también deberán contener la misma información que se muestra en la *«* Imagen 6 *»*.

```
4      <Identification Standard="61499-2"/>
5      <VersionInfo Author="4DIAC-IDE" Date="2016-11-02" Organization="4DIAC-Consortium" Version="0.0"/>
6      <VersionInfo Author="AZ" Date="2016-05-26" Organization="fortiss GmbH" Version="1.0"/>
```

Imagen 6. Estructura del « FB » en « 4DIAC », declaración del tipo de versión (2).

Después de todo esto, se introduce la instrucción « <InterfaceList> », dentro de la cual se declararán tanto los Eventos como las Variables de entrada y de salida. Por lo tanto, de la misma manera que con el « FBType », se deberá introducir un doble espacio adicional antes de cada una de las siguientes instrucciones para indicar que pertenecen a la « <InterfaceList> ».

Lo primero que se debe declarar ahora son los Eventos de entrada, para ello se introduce la instrucción « <EventInputs> » y en las siguientes líneas (introduciendo otro doble espacio antes de ellas) se utilizará la instrucción « <Event Name="" Type="Event"> » para declarar un Evento de entrada, escribiendo el nombre del Evento en cuestión entre los « "" » que van seguidos de « Name= ».

Una vez hecho esto, se indican las Variables asociadas con dicho Evento, introduciendo por cada Variable asociada al Evento en cuestión otro doble espacio más primero (ya que pertenecen al Evento en cuestión) y usando la instrucción « <With Var=""/> » con el nombre de la Variable en cuestión entre « "" ». Al acabar, se introducirá en la siguiente línea « </Event> », esta vez sin el último doble espacio, ya que esta instrucción indica que ya se ha acabado de declarar dicho Evento. Se repite el mismo procedimiento para todos los Eventos de entrada y se introduce la instrucción « </EventInputs> » para indicar que se han acabado de declarar todos los Eventos de entrada (por lo tanto tampoco llevará el doble espacio correspondiente a « <EventInputs> ») tal y como se muestra en las *«* Imágenes 7 y 8 *»*.

```
<InterfaceList>
  <EventInputs>
    <Event Comment="Initialization Request" Name="INIT" Type="Event">
      <With Var="Ym"/>
    </Event>
```

Imagen 7. Estructura del « FB » en « 4DIAC », declaración de los Eventos de entrada.

```
    <Event Comment="Normal Execution Request" Name="REQ" Type="Event">
      <With Var="Ym"/>
      <With Var="Yref"/>
      <With Var="K"/>
      <With Var="Ti"/>
      <With Var="Td"/>
      <With Var="Sample_time"/>
      <With Var="LIM_H"/>
      <With Var="LIM_L"/>
      <With Var="E_lim"/>
      <With Var="tmax"/>
      <With Var="beta"/>
      <With Var="N"/>
      <With Var="E_precision"/>
    </Event>
    <Event Name="RST" Type="Event">
      <With Var="Ym"/>
      <With Var="Yref"/>
      <With Var="K"/>
      <With Var="Ti"/>
      <With Var="Td"/>
      <With Var="Sample_time"/>
      <With Var="LIM_H"/>
      <With Var="LIM_L"/>
      <With Var="E_lim"/>
      <With Var="tmax"/>
      <With Var="beta"/>
      <With Var="N"/>
      <With Var="E_precision"/>
    </Event>
  </EventInputs>
```

Imagen 8. Estructura del « FB » en « 4DIAC », declaración de los Eventos de entrada (2).

Una vez hecho esto, se declaran los Eventos de salida o « Outputs », de la misma forma que los Eventos de entrada, pero utilizando las instrucciones « <EventOutputs> » y « </EventOutputs> » para indicar que los que se introduce entre estas instrucciones son los Eventos de salida tal y como se muestra en la *«* Imagen 9 *»*.

```
  <EventOutputs>
    <Event Comment="Initialization Confirm" Name="INITO" Type="Event"/>
    <Event Comment="Execution Confirmation" Name="CNF" Type="Event">
      <With Var="U"/>
    </Event>
    <Event Name="NO_MOD" Type="Event"/>
  </EventOutputs>
```

Imagen 9. Estructura del « FB » en « 4DIAC », declaración de los Eventos de salida.

A continuación, se declaran las Variables de entrada, para ello se utilizan las instrucciones « <InputVars> » y « </InputVars> » para delimitar el trozo de código en el que son declaradas. Entre estas dos instrucciones, se deben declarar (con un doble espacio adicional para indicar que pertenecen a « <InputVars> ») las Variables de entrada con la siguiente instrucción « <VarDeclaration Name="" Type=""/> » poniendo el nombre de la Variable en cuestión entre las « "" » que van seguidas de « Name= » y el tipo de la variable entre las « "" » que van seguidas de « Type= » tal y como se muestra en la *«* Imagen 10 *»*. Los posibles tipos de las Variables son los indicados en la *«* Tabla 2 *»* del apartado *«* **2.4. Normas para la introducción de datos en el Bloque de función** *»*.

```
50    <InputVars>
51      <VarDeclaration Comment="y measured" Name="Ym" Type="REAL"/>
52      <VarDeclaration Comment="reference" Name="Yref" Type="REAL"/>
53      <VarDeclaration Comment="Gain" Name="K" Type="REAL"/>
54      <VarDeclaration Comment="Integral time" Name="Ti" Type="REAL"/>
55      <VarDeclaration Comment="Derivetive time" Name="Td" Type="REAL"/>
56      <VarDeclaration Comment="beta" Name="beta" Type="REAL"/>
57      <VarDeclaration Comment="N" Name="N" Type="REAL"/>
58      <VarDeclaration Comment="elapsed time" Name="Sample_time" Type="REAL"/>
59      <VarDeclaration Comment="maximum elapsed time" Name="tmax" Type="REAL"/>
60      <VarDeclaration Comment="limit error" Name="E_lim" Type="REAL"/>
61      <VarDeclaration Comment="Error decimal precision" Name="E_precision" Type="INT"/>
62      <VarDeclaration Comment="Hight limit of the output" Name="LIM_H" Type="REAL"/>
63      <VarDeclaration Comment="Low limit of the output" Name="LIM_L" Type="REAL"/>
64    </InputVars>
```

Imagen 10. Estructura del « FB » en « 4DIAC », declaración de las Variables de entrada.

Así mismo, las Variables de salida se deben declarar de la misma forma que las Variables de entrada. Sin embargo, en este caso se utilizarán las instrucciones « <OutputVars> » y « </OutputVars> » tal y como se muestra en la *«* Imagen 11 *»*.

```
65    <OutputVars>
66      <VarDeclaration Comment="Control signal" Name="U" Type="REAL"/>
67    </OutputVars>
```

Imagen 11. Estructura del « FB » en « 4DIAC », declaración de los Variables de salida.

Una vez hecho esto, ya se ha terminado de declarar la « InterfaceList », por lo tanto se debe añadir la instrucción « </InterfaceList> » seguida de un único doble espacio (ya que solamente pertenece a « FBType »). Seguidamente se debe abrir la segunda parte del programa mediante la instrucción « <BasicFB> » en la siguiente línea, dentro de la cual se deben declarar antes que nada las Variables internas. Para indicar esto, se deben utilizar las instrucciones « <InternalVars> » y « </InternalVars> » para declarar dichas Variables entre estas instrucciones de la siguiente manera. Se introduce la instrucción « <VarDeclaration InitialValue="" Name="" Type=""/> » por cada Variable interna que se vaya a declarar, poniendo el valor inicial de la Variable interna en cuestión entre las « "" » que van seguidas de « InitialValue= », el nombre de la misma entre las « "" » que van seguidas de « Name= » y el tipo de la variable entre las « "" » que van seguidas de « Type= » tal y como se muestra en la *«* Imagen 12 *»*. Los posibles tipos de las Variables son los indicados en la *«* Tabla 2 *»* del apartado *«* **2.4. Normas para la introducción de datos en el Bloque de función** *»*.

```
68  </InterfaceList>
69  <BasicFB>
70    <InternalVars>
71      <VarDeclaration Comment="Internal Variable" InitialValue="0.0" Name="ES" Type="REAL"/>
72      <VarDeclaration Comment="Internal Variable" InitialValue="0.0" Name="YOLD" Type="REAL"/>
73      <VarDeclaration Comment="Internal Variable" InitialValue="0.0" Name="UD" Type="REAL"/>
74      <VarDeclaration Comment="Internal Variable" InitialValue="0.0" Name="UI" Type="REAL"/>
75      <VarDeclaration Comment="Internal Variable" InitialValue="0.0" Name="Elapsed_time" Type="REAL"/>
76    </InternalVars>
```

Imagen 12. Estructura del « FB » en « 4DIAC », declaración de las Variables internas.

A continuación, se empieza a declarar el esquema de control de ejecución o « ECC » mediante la instrucción « <ECC> » seguida de un doble espacio adicional. Aquí se deben declarar por orden de prioridad los estados y las transiciones que componen el esquema, siendo el primer estado declarado el estado inicial, cuyo nombre siempre será « START ». El « ECC » del FB "Event_driven_PID" que estamos usando como ejemplo en esta sección, se muestra en la « Imagen 13 ».

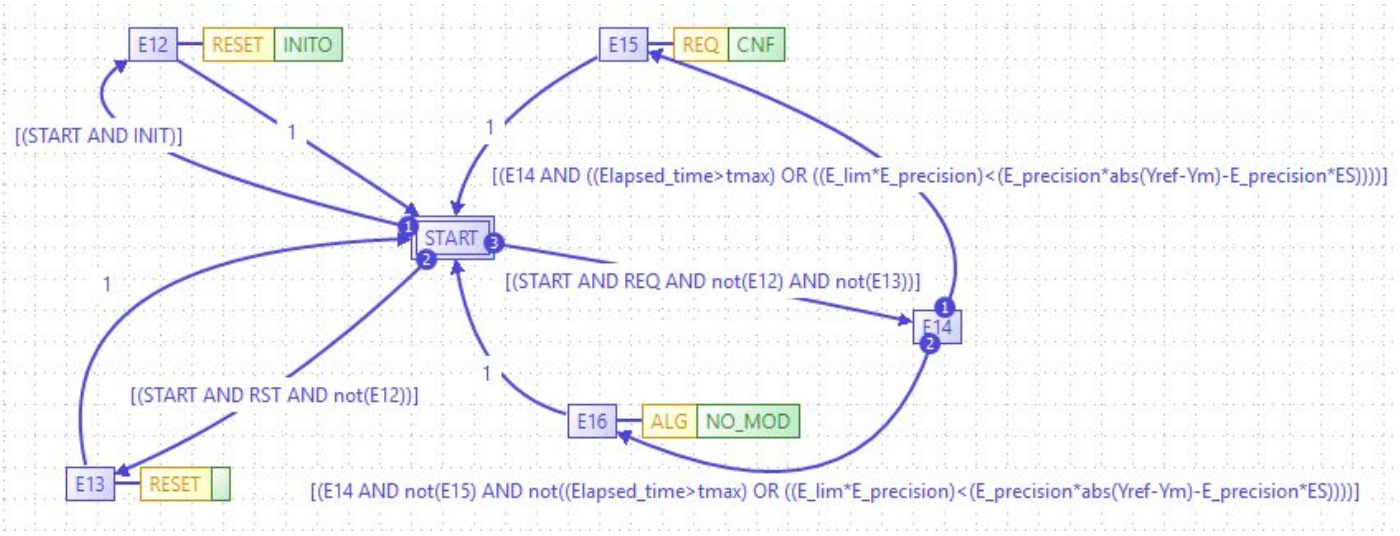

Imagen 13. Estructura del « FB » en « 4DIAC », estructura del « ECC ».

Para declarar los estados, se utiliza la instrucción « <ECState Name="" x="" y=""/> » si el estado no tiene ningún algoritmo asociado u « Output », introduciendo el nombre del estado entre los « "" » que van seguidos de « Name= », los datos de « x » e « y » solamente indican la posición en el dibujo esquema del programa, por lo cual tienen poca relevancia. En el caso de que si que tuviera algún « Output » o algoritmo asociado, se utilizará primero la instrucción « <ECState Name="" x="" y=""> », en la siguiente línea y con un doble espacio adicional, se indicarán dichos « Outputs » o algoritmos mediante « <ECAction Algorithm="" Output=""/> », donde se debe poner entre « "" » el nombre del algoritmo u « Output » en cuestión. En el caso que haya más de uno se introducirán en instrucciones separadas. Finalmente, se añade « </ECState> » para indicar el fin de la declaración del estado tal y como se muestra en la *«* Imagen 14-1 *»*.

```
77    <ECC>
78      <ECState Comment="Initial State" Name="START" x="1805.0" y="1045.0"/>
79      <ECState Comment="Initialization" Name="INIT" x="2565.0" y="285.0">
80        <ECAction Algorithm="RESET" Output="INITO"/>
81      </ECState>
82      <ECState Name="RST" x="2945.0" y="1425.0">
83        <ECAction Algorithm="RESET"/>
84      </ECState>
85      <ECState Name="REQ" x="95.0" y="1805.0">
86        <ECAction/>
87      </ECState>
88      <ECState Name="SENDCNF" x="285.0" y="380.0">
89        <ECAction Algorithm="REQ" Output="CNF"/>
90      </ECState>
91      <ECState Name="SENDNOMOD" x="475.0" y="1045.0">
92        <ECAction Algorithm="ALG" Output="NO_MOD"/>
93      </ECState>
```

Imagen 14-1. Estructura del « FB » en « 4DIAC », declaración del « ECC ».

Para declarar las transiciones, se utiliza la instrucción « <ECTransition Condition="" Destination="" Source="" x="" y=""/> », en la cual se debe introducir la condición de la transición en Texto Estructurado entre los « "" » que van seguidos de « Condition= », el nombre del Estado que tiene como destino dicha transición entre los « "" » que van seguidos de « Destination= » y el nombre del Estado que precede a dicha transición entre los « "" » que van seguidos de « Source= », los datos de « x » e « y » solamente indican la posición en el dibujo esquema del programa, por lo cual tienen poca relevancia. Una vez hecho esto, ya se puede cerrar el « ECC » mediante la instrucción « </ECC> » tal y como se muestra en la *«* Imagen 14-2 *»*.

```
94      <ECTransition Comment="" Condition="INIT" Destination="INIT" Source="START" x="2850.0" y="765.0"/>
95      <ECTransition Comment="" Condition="1" Destination="START" Source="INIT" x="2010.0" y="280.0"/>
96      <ECTransition Comment="" Condition="REQ" Destination="REQ" Source="START" x="1195.0" y="2310.0"/>
97      <ECTransition Comment="" Condition="RST" Destination="RST" Source="START" x="2320.0" y="1720.0"/>
98      <ECTransition Comment="" Condition="1" Destination="START" Source="RST" x="2835.0" y="995.0"/>
99      <ECTransition Comment="" Condition="((abs((Yref-Ym)*E_precision-ES*E_precision)&gt;(E_lim*E_precision)) OR...
100     <ECTransition Comment="" Condition="1" Destination="START" Source="SENDCNF" x="1150.0" y="300.0"/>
101     <ECTransition Comment="" Condition="not((abs((Yref-Ym)*E_precision-ES*E_precision)&gt;(E_lim*E_precision))...
102     <ECTransition Comment="" Condition="1" Destination="START" Source="SENDNOMOD" x="1255.0" y="1485.0"/>
103   </ECC>
```

Imagen 14-2. Estructura del « FB » en « 4DIAC », declaración del « ECC ».

Por último, se deben declarar los algoritmos asociados a cada uno de las etapas de la EEC del FB. Los algoritmos del FB que estamos analizando en esta sección, visualizados mediante 4DIAC, se muestran en la « Imagen 15-1 ». Para declarar los algoritmos se deben utilizar las instrucciones « <Algorithm Name=""> » y « </Algorithm> », introduciendo entre estas « <ST Text=""/> » para introducir el código del algoritmo en lenguaje de Texto Estructurado entre los « "" » tal y como se muestra en la « Imagen 15-2 ».

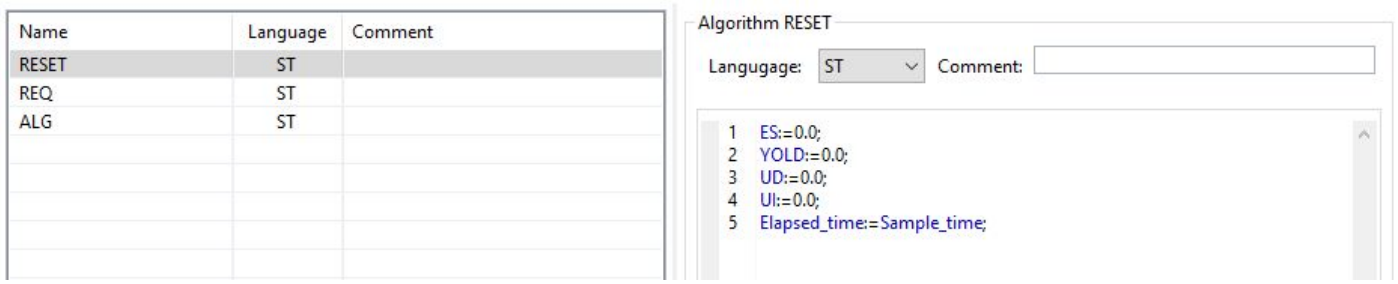

Imagen 15-1. Estructura del « FB » en « 4DIAC », Interfaz para declarar algoritmos.

```
104    <Algorithm Comment="new algorithm" Name="RESET">
105      <ST Text="ES:=0.0;&#13;&#10;YOLD:=0.0;&#13;&#10;UD:=0.0;&#13;&#10;UI:=0.0;&#13;&#10;Elapsed_time:=Sample_time;"/>
106    </Algorithm>
107    <Algorithm Comment="new algorithm" Name="REQ">
108      <ST Text="VAR&#10;&#9;up: REAL;&#10;END_VAR&#10;&#10;&#9;ES:=Yref-Ym;&#10;&#9;up:=K*(beta*Yref-Ym);&#10;&#9;UD:=T...
109    </Algorithm>
110    <Algorithm Comment="new algorithm" Name="ALG">
111      <ST Text="Elapsed_time:=Elapsed_time+Sample_time;"/>
112    </Algorithm>
```

Imagen 15-2. Estructura del « FB » en « 4DIAC », declaración de los algoritmos.

Una vez, declarado todo lo anterior, ya se puede cerrar esta parte del programa mediante la instrucción « </BasicFB> » seguida de un único doble espacio. En la línea siguiente se introduce la instrucción « </FBType> » concluyendo así el programa tal y como se muestra en la *«* Imagen 16 *»*.

```
113    </BasicFB>
114  </FBType>
```

Imagen 16. Estructura del « FB » en « 4DIAC », cierre del Bloque de Función.

3.2. Simulación del Bloque de Función en Matlab/Simulink

Como se comentó en la sección *«* **2.1. IEC-61499** *»*, la norma « IEC-61499 » describe de forma bastante clara los aspectos relativos a la ejecución de los « FB ». Evidentemente, para poder realizar la simulación de un « FB » dentro de un modelo de Simulink, se deben de tener en cuenta dichos aspectos definidos en la norma. Para cumplir con este requisito básico, lo primero que se debe hacer es elegir cuál de los distintos bloques de « Simulink » puede ser utilizado para implementar el bloque que simula un « FB ». La decisión en este punto fue elegir el bloque « Matlab function » dada la gran flexibilidad del mismo al tratarse de un bloque que básicamente llama a una función de « Matlab ». Además, ya que los « FB » en « 4DIAC » se definen mediante ficheros de texto escritos en « xml », y una función de Matlab también es un fichero de texto, la traducción automática de « FB » entre « 4DIAC » y « Simulink » se facilita pues se trataría básicamente de modificar la estructura de un fichero de texto en otro.

Una vez decidido esto, queda definir la estructura que debe tener la función de « Matlab » que será llamada por el bloque « Matlab Function » de « Simulink » para que simule de forma correcta el comportamiento de un « FB » según lo descrito en la « IEC-61499 ».

La estructura que se propone se divide en diez partes claramente diferenciadas por una serie de comentarios:

El bloque « MATLAB function » debe comenzar con el comentario « %1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE », a continuación del cual se deben declarar tanto los Eventos como las Variables del Bloque de Función ya sean de entrada como de salida. La manera de hacerlo es introduciendo la instrucción « function [] = fcn () » en la línea siguiente al comentario, escribiendo entre los « [] », que preceden al signo « = », primero los nombres de los Eventos de salida, seguidos de los nombres de las Variables de salida. A continuación, entre los « () », que siguen a la instrucción « fcn », se declaran primero los Eventos de entrada, seguidos de los nombres de las Variables de entrada. Así mismo, cada Evento debe declararse con una « e_ » precediendo el nombre y cada Variable con una « v_ » precediendo el nombre. Además, cada Evento o Variable declarada se debe separar por una « , » de otra declaración tal y como se muestra en la *«* Imagen 17 *»*.

```
1  %1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE
2  function [e_INITO, e_CNF, e_NO_MOD, v_U]  = fcn (e_INIT, e_REQ, e_RST, v_Ym, v_Yref, v_K,
   v_Ti, v_Td, v_Sample_time, v_LIM_H, v_LIM_L, v_E_lim, v_tmax, v_beta, v_N, v_E_precision)
```

Imagen 17. Estructura del « FB » en Simulink, declaración de los Eventos y las Variables del « FB ».

El siguiente apartado es « %2-DECLARACION DE VARIABLES INTERNAS », a continuación del cual se deben declarar las Variables internas del Bloque de Función. La manera de hacerlo es introduciendo la instrucción « persistent » seguido de un espacio y el nombre de las Variables internas, separados también por un espacio. Una vez hecho esto, se introduce un « ; » tal y como se muestra en la *«* Imagen 18 *»*.

```
4  %2-DECLARACION DE VARIABLES INTERNAS
5  persistent ES YOLD UD UI Elapsed_time up;
```

Imagen 18. Estructura del « FB » en Simulink, declaración de las Variables internas.

El siguiente apartado es « %3-DECLARACION DEL ESTADO INICIAL », a continuación del cual se deben declarar preferiblemente en este orden : una copia de los Eventos de salida, una copia de las Variables de salida, una segunda copia de las Variables de salida, una copia de los Eventos de entrada, una copia de las Variables de entrada y las variables « E » y « CE ». La manera de hacerlo es introduciendo la instrucción « persistent » seguido de un espacio y el nombre de los Eventos o Variables, en el caso de las copias, los nombres se escribirán sin « e_ » o « v_ », en cuanto a las segundas copias, se deben declarar con un « prev_ » precediendo el nombre. Una vez hecho esto, se introduce un « ; » tal y como se muestra en la *«* Imagen 19 *»*.

```
7  %3-DECLARACION DEL ESTADO INICIAL
8  persistent Ym Yref K Ti Td Sample_time LIM_H LIM_L E_lim tmax
   beta N E_precision E CE U INITO CNF NO_MOD prev_U INIT REQ RST;
```

Imagen 19. Estructura del « FB » en Simulink, declaración del Estado inicial.

En la siguiente línea, se debe introducir la instrucción « if isempty(E) » de manera que se inicia una condición con la instrucción « if », el cual solamente se ejecutará una vez al iniciar el programa. Dentro de éste se deben declarar tanto el estado inicial de los Estados (por ejemplo : « E =[1 0 0 0 0 0]; » en el caso de que hayan 6 Estados, representando cada número a un Estado en valor ascendiente, siendo el primero el Estado inicial ya que es la única que está inicialmente activa « 1 »), como los valores iniciales de las Variables internas y de todo lo declarado al principio de éste apartado. Así mismo, las copias de los Eventos se deben inicializar a « 0 » tal y como se muestra en la *«* Imagen 20 *»*.

```
10  if isempty(E)
11      E =[1 0 0 0 0 0]; %Estado de las etapas
12      %Variables internas
13      ES=0.0;
14      YOLD=0.0;
15      UD=0.0;
16      UI=0.0;
17      Elapsed_time=0.0;
18      %Variables de entrada
19      Ym=0.0;
20      Yref=0.0;
21      K=0.0;
22      Ti=0.0;
23      Td=0.0;
24      Sample_time=0.0;
25      LIM_H=0.0;
26      LIM_L=0.0;
27      E_lim=0.0;
28      tmax=0.0;
29      beta=0.0;
30      N=0.0;
31      E_precision=0;
32      %Variables de salida
33      U=0.0;
34      %Eventos de entrada
35      INIT=0;
36      REQ=0;
37      RST=0;
38      %Eventos de salida
39      INITO=0;
40      CNF=0;
41      NO_MOD=0;
42      prev_U=0;
43  end
```

Imagen 20. Estructura del « FB » en Simulink, declaración del Estado inicial (2).

Una vez declarado todo esto, se cierra la instrucción « if » introduciendo un « end » al final.

El siguiente apartado es « %4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH) », a continuación del cual se deben introducir los vínculos entre Eventos de entrada y Variables de entrada. La manera de hacerlo es introduciendo la instrucción « if » seguido de un espacio, un « e_ » y el nombre del Evento, de esta manera solamente se ejecutará el código de su interior al recibir el Bloque de Función una señal activación de dicho Evento. Una vez hecho esto, se introduce en la siguiente línea el nombre del mismo evento seguido de « =1; », de esta manera la copia del evento se deja activa. A continuación, se igualan las copias de las variables a las variables tal y como se muestra en la *«* Imagen 21 *»*.

```
45  %4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)
46  if (e_INIT)
47      INIT = 1;
48      Ym = v_Ym;
49  end
```

Imagen 21. Estructura del « FB » en Simulink, actualización de las Variables de entrada asociadas a Eventos de entrada.

Una vez declarado todo esto, se cierra la instrucción« if » introduciendo un « end » al final y se repite el mismo procedimiento con todos los Eventos de entrada. *«* Imagen 22 *»*.

```
if (e_REQ)
    REQ = 1;
    Ym = v_Ym;
    Yref = v_Yref;
    K = v_K;
    Ti = v_Ti;
    Td = v_Td;
    Sample_time = v_Sample_time;
    LIM_H = v_LIM_H;
    LIM_L = v_LIM_L;
    E_lim = v_E_lim;
    tmax = v_tmax;
    beta = v_beta;
    N = v_N;
    E_precision = v_E_precision;
end

if (e_RST)
    RST = 1;
    Ym = v_Ym;
    Yref = v_Yref;
    K = v_K;
    Ti = v_Ti;
    Td = v_Td;
    Sample_time = v_Sample_time;
    LIM_H = v_LIM_H;
    LIM_L = v_LIM_L;
    E_lim = v_E_lim;
    tmax = v_tmax;
    beta = v_beta;
    N = v_N;
    E_precision = v_E_precision;
end
```

Imagen 22. Estructura del « FB » en Simulink, actualización de las Variables de entrada asociadas a Eventos de entrada (2).

El siguiente apartado es « %5-DIAGRAMA DE CONTROL DE EJECUCION (ECC) », a continuación del cual se debe inicializar antes de nada unas segundas copias de los Eventos de salida (lo cual implica introducir « prev _ » antes del nombre del Evento) y « CE » a cero (por ejemplo : « CE =[0 0 0 0 0 0]; » en el caso de que hayan 6 Estados, representando cada « 0 » a un Estado). tal y como se muestra en la *«* Imagen 23 *»*.

```
%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)
prev_INITO=0;
prev_CNF=0;
prev_NO_MOD=0;
CE=[0 0 0 0 0 0]; %Copia de etapas
```

Imagen 23. Estructura del « FB » en Simulink, declaración del « ECC ».

Una vez hecho esto, se introduce la instrucción « while ~isequal(CE,E) », la cual inicia un bucle « while » que se encarga de que el Bloque de Función no envíe ninguna señal hasta que el ECC se encuentre en un Estado estable. Por lo tanto, lo que se tendrá que declarar dentro es el ECC. En la siguiente línea, se debe introducir la instrucción « CE = E ; » para actualizar los estados de los Estados. A continuación, se introduce por cada transición que haya en el ECC una instrucción « if » para que cuando se cumpla la condición de la transición se desactive el Estado anterior y se active la siguiente, de manera que lo que se debe introducir es la instrucción « if » un espacio, el Estado que va ser desactivada pero utilizando « CE » (por ejemplo : « CE(1) » para el Estado « 1 »), un « && », para indicar que ambas condiciones se deben cumplir, y la condición de la transición tal y como se muestra en las *«* Imágenes 24 y 25 *»*.

```
90  while ~isequal(CE,E)
91      CE=E;
92
93      if (CE(1,1) && INIT) %Transición de la Etapa0 a la Etapa1 (Prioridad 1)
94          E(1,1)=0;
95          E(1,2)=1;
96      end
97
98      if (CE(1,1) && RST && ~CE(1,2)) %Transición de la Etapa0 a la Etapa2 (Prioridad 2)
99          E(1,1)=0;
100         E(1,3)=1;
101     end
102
103     if (CE(1,1) && REQ && ~CE(1,2) && ~CE(1,3)) %Transición de la Etapa0 a la Etapa3 (Prioridad 3)
104         E(1,1)=0;
105         E(1,4)=1;
106     end
107
108     if (CE(1,4) && ((Elapsed_time>tmax)||((E_lim*E_precision)<(E_precision*abs(Yref-Ym)-E_precision*ES))))
109         E(1,4)=0;
110         E(1,5)=1;
111     end
112
113     if (CE(1,4) && ~CE(1,5) && not((Elapsed_time>tmax)||((E_lim*E_precision)<(E_precision*abs(Yref-Ym)-E_precision*ES))))
114         E(1,4)=0;
115         E(1,6)=1;
116     end
```

Imagen 24. Estructura del « FB » en Simulink, declaración del « ECC » (2).

```
118     if (CE(1,2))
119         E(1,2)=0;
120         E(1,1)=1;
121     end
122
123     if (CE(1,3))
124         E(1,3)=0;
125         E(1,1)=1;
126     end
127
128     if (CE(1,5))
129         E(1,5)=0;
130         E(1,1)=1;
131     end
132
133     if (CE(1,6))
134         E(1,6)=0;
135         E(1,1)=1;
136     end
```

Imagen 25. Estructura del « FB » en Simulink, declaración del « ECC » (3).

Un aspecto importante es que en la primera transición, el Estado que se desactiva debe ser el Estado inicial. Para indicar el orden de prioridad de las transiciones que salen de una misma etapa, se deben ordenar de mayor a menor prioridad. En Simulink, la prioridad se introduce añadiendo la instrucción « && ~CE() » a la condición y poniendo entre « () » el número del Estado que tiene mayor prioridad, tal y como se muestra en las líneas 98 y 103.

El siguiente apartado es « %6-EJECUCION DE ALGORITMOS », a continuación del cual se debe definir cuándo se ejecutarán los algoritmos. La manera de hacerlo es introduciendo el nombre del algoritmo, con un « Alg_ »precediendo el nombre, seguido de un « = », añadiendo « E() && ~CE()) », lo cual indica que solo se activará cuando el Estado se acaba de activar ya que la copia de Estado indica el estado anterior, por cada Estado al cual esté asociado el algoritmo, separándolas mediante un « || » (ya que el algoritmo se debe ejecutar cuando cualquiera de los Estados a las cuales está asociado se ejecute) y poniendo entre « () » el número del Estado. Una vez hecho esto, se añade « ; » al final y se repite el mismo procedimiento con todos los algoritmos existentes tal y como se muestra en la *«* Imagen 26 *»*.

```
138     %6-EJECUCION DE ALGORITMOS
139     Alg_RESET = (E(1,2) && ~CE(1,2))||(E(1,3) && ~CE(1,3));
140     Alg_REQ   = (E(1,5) && ~CE(1,5));
141     Alg_ALG   = (E(1,6) && ~CE(1,6));
```

Imagen 26. Estructura del « FB » en Simulink, ejecución de algoritmos.

El siguiente apartado es « %7-DECLARACION DE ALGORITMOS », a continuación del cual se deben definir las instrucciones de los algoritmos. La manera de hacerlo es introduciendo la instrucción « if » para que el algoritmo solamente se ejecute cuando su Variable asociada (compuesta por « Alg_ » y el nombre de la Variable) esté activa. Para ello, se añade la instrucción « if » seguida de la variable asociada al algoritmo. Una vez hecho esto, se introduce todo el código del algoritmo e introduciendo un « end » al final de éste tal y como se muestra en la *«* Imagen 27 *»*.

```
%7-DECLARACION DE ALGORITMOS
if (Alg_RESET) %Algoritmo de RESET
    ES = 0.0;
    YOLD = 0.0;
    UD = 0.0;
    UI = 0.0;
    Elapsed_time = Sample_time;
end

if (Alg_REQ) %Algoritmo REQ
    ES = Yref-Ym;
    up = K*(beta*Yref-Ym);
    UD = Td/(Td+N*Elapsed_time)*UD - Td/(Td+N*Elapsed_time)*K*N*(Ym - YOLD);
    U = up + UD + UI;
    UI = UI + K/Ti * Elapsed_time*(Yref - Ym);
    YOLD = Ym;

    if (U > LIM_H)
        U = 100.0;
    end

    if (U < LIM_L)
        U = 0.0;
    end

    Elapsed_time = Sample_time;
end

if (Alg_ALG) %Algoritmo ALG
    Elapsed_time = Elapsed_time + Sample_time;
end
```

Imagen 27. Estructura del « FB » en Simulink, declaración de algoritmos.

El siguiente apartado es « %8-EJECUCION DE EVENTOS DE SALIDA », a continuación del cual se debe definir cuándo se ejecutarán las señales de los Eventos de salida. La manera de hacerlo es introduciendo el nombre del evento seguido de un « = » y y « E() » por cada Estado que tenga asociada dicho evento de salida, separándolas con un « || » (ya que el Evento de salida se debe ejecutar cuando cualquiera de los Estados a las cuales está asociado se ejecute) y poniendo entre « () » el número del Estado. Una vez hecho esto, se añade « ; » al final y se repite el mismo procedimiento con todos los Eventos de salida existentes tal y como se muestra en la *«* Imagen 28 *»*.

```
%8-EJECUCION DE EVENTOS DE SALIDA
INITO = E(1,2);
CNF = E(1,5);
NO_MOD = E(1,6);
```

Imagen 28. Estructura del « FB » en Simulink, ejecución de Eventos de salida.

El siguiente apartado es « %9-EJECUCION DE VARIABLES DE SALIDA (WITH) », a continuación del cual se deben actualizar las Variables de salida a los valores de sus copias dentro del Bloque de Función. La manera de hacerlo es introduciendo una condición « if » para que la Variable de salida solamente se actualiza al valor de su copia en el Bloque de Función cuando uno de los Eventos de salida al que esté asociada se encuentre activo. Para ello, se añade la instrucción « if » seguida de los eventos a los cuales esté asociada la Variable de salida en cuestión, en el caso de que haya más de uno se deben separar con un « || » (ya que la Variable de salida se debe ejecutar cuando cualquiera de los Eventos de salida a los cuales está asociada se ejecuta). En la siguiente línea se añade « v_ » y el nombre de la Variable en cuestión, un « = » y el nombre de la Variable una vez más seguido de « ; », de esta manera se actualizará el valor de la Variable en cuestión. Una vez hecho esto y para que el valor de la Variable de salida no sea « 0 » mientras no se llame al Evento que la actualice, se introduce un « else » en la siguiente línea y otra línea más abajo se introduce « v_ » seguido del nombre de la Variable en cuestión, un « = prev_ », el nombre de la misma Variable una vez más y « ; ». De esta manera, mientras ningún Evento que actualice dicha Variable se encuentre activo, la Variable de salida siempre tendrá su anterior valor. Después de esto, se debe actualizar el valor de la segunda copia de la Variable de salida (la que incluye « prev_ » antes del nombre) ya que éste será el anterior valor de la Variable en cuestión la próxima vez que se ejecute el Bloque de Función. Para ello, se introduce « prev_ » seguido del nombre de la Variable en cuestión, un « = v_ », el nombre de la

misma Variable una vez más y « ; ». Una vez hecho esto, se repite el mismo procedimiento para todas las Variables de salida tal y como se muestra en la *«* Imagen 29 *»*.

```
180     %9-EJECUCION DE VARIABLES DE SALIDA (WITH)
181     if (CNF)
182         v_U = U;
183     else
184         v_U = prev_U;
185     end
186     prev_U = v_U;
```

Imagen 29. Estructura del « FB » en Simulink, actualización de las Variables de salida asociadas a Eventos de salida.

Por último, tenemos el apartado « %10-FINAL », a continuación del cual se deben igualar a « 0 » todas las copias de los Eventos de entrada introduciendo el nombre del Evento de entrada y un « = 0; ». Una vez hecho esto, si se acaba de activar algún Evento de salida durante la ejecución del Bloque de Función, éste debe permanecer activo aunque haya un Estado inestable para enviar dicha señal al acabar la ejecución del Bloque de Función. La manera de hacer esto es introducir la instrucción « if », el nombre del Evento de salida, un « && ~prev_ » y el nombre del Evento una vez más. En la siguiente línea se introduce de nuevo « prev_ » seguido del nombre del Evento en cuestión y un « = 1; », activando así la segunda copia del Evento de salida en cuestión. Una vez hecho esto, se introduce un « end » y se repite el mismo procedimiento para todos los Eventos de salida. Justo después de todo esto, se introduce otro « end » que cierra el bucle « while » que se abrió en el quinto apartado. De esta forma, desde dicho apartado hasta aquí se irá ejecutando el código hasta llegar a un Estado estable. Para acabar, se deben actualizar los valores de los Eventos de salida y de las Variables de salida (en éste orden), para que en el caso de que se ejecute el Bloque de Función y no haya habido ningún cambio de Estado en el ECC, los valores de estas sigan siendo los anteriores. Para ello, se debe introducir o bien « e_ » (si es un Evento) o bien « v_ » (si es una Variable) seguido del nombre de dicho/a Evento o Variable, un « = prev_ » y el nombre del/de la mismo/a Evento o Variable seguido de un « ; » tal y como se muestra en la *«* Imagen 30 *»*.

```
188     %10-FINAL
189     INIT = 0;
190     REQ  = 0;
191     RST  = 0;
192
193     if INITO && ~prev_INITO
194         prev_INITO = 1;
195     end
196
197     if CNF && ~prev_CNF
198         prev_CNF = 1;
199     end
200
201     if NO_MOD && ~prev_NO_MOD
202         prev_NO_MOD = 1;
203     end
204 end
205
206 e_INITO = prev_INITO;
207 e_CNF = prev_CNF;
208 e_NO_MOD = prev_NO_MOD;
209 v_U = prev_U;
```

Imagen 30. Estructura del « FB » en Simulink, actualización de los valores de los Eventos y las Variables de salida.

3.3. Metodología para la traducción de Matlab/Simulink a 4DIAC

Para realizar la traducción a 4DIAC se utilizará la función « FB_SLK24DIAC( ) », la cual recibe el nombre del archivo que se desea traducir « Archivo_destino » y devuelve el nombre de otro archivo « Archivo_destino » que se genera con el resultado de la traducción del anterior tal y como se muestra en la « Imagen 31 ».

Así mismo, en ésta sección se explicará cómo tratar la información que se quiere traducir. A la hora de introducir el código del archivo de 4DIAC « Archivo_destino », se precisa que el programa siga una estructura similar a éste para facilitar así la traducción. De ésta manera, se divide la traducción en diversos puntos en los cuales se repetirá un mismo procedimiento para obtener la traducción:

1. Lo primero es buscar en la estructura de « Matlab/Simulink » el punto o los puntos donde se encuentra la información del « Archivo_fuente » que necesitamos para realizar la traducción de cierto punto de la estructura de « 4DIAC » al « Archivo_destino ».
2. Una vez tenemos localizada la información, hay que extraerla del « Archivo_fuente » y guardarla de una forma ordenada en una o varias variables de « Matlab ».
3. Con la información extraída, se traduce y/o prepara ésta para el código y la estructura de « 4DIAC ».
4. Una vez tenemos ésto, se vuelca la nueva información con el tipo de estructura de « 4DIAC » en el « Archivo_destino ».

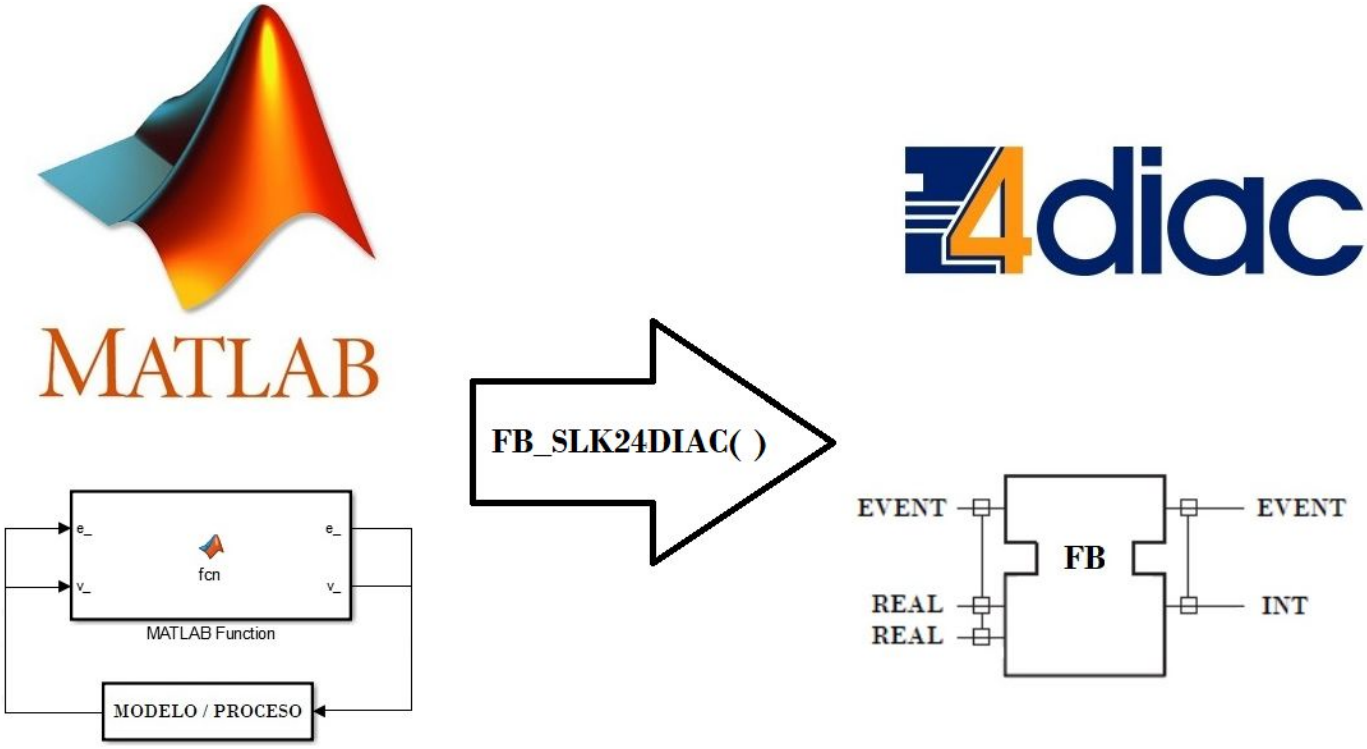

Imagen 31. Esquema de traducción a « 4DIAC ».

Así mismo, el archivo de Simulink « Archivo_fuente » será necesario abrirlo varias veces por cada punto de la estructura de 4DIAC en la mayoría de los casos, con el fin de extraer toda la información necesaria tal y como se muestra en la « Tabla 3 ».

Apartados de la traducción Matlab/Simulink - 4DIAC	**Puntos de la estructura de « Simulink » utilizados**	**Puntos de la estructura de « 4DIAC » que se declaran**
1. Preparación del « Archivo_destino »	-	<Identification> <VersionInfo>
2. Declaración de los Eventos de entrada	%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE %4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)	<EventInputs>
3. Declaración de Eventos de salida	%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE %9-EJECUCION DE VARIABLES DE SALIDA (WITH)	<EventOutputs>
4. Obtención de datos para la declaración de Variables	%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE %3-DECLARACION DEL ESTADO INICIAL	-
5. Declaración de Variables de entrada y de salida	-	<InputVars> <OutputVars>
6. Declaración de Variables internas	%2-DECLARACION DE VARIABLES INTERNAS	<InternalVars>
7. Obtención de los datos de los Estados y las transiciones del ECC	%6-EJECUCION DE ALGORITMOS %8-EJECUCION DE EVENTOS DE SALIDA %5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)	-
8. Declaración de los Estados del ECC	-	<ECState> <ECAction>
9. Declaración de las transiciones del ECC	-	<ECTransition>
10. Obtención y declaración de los algoritmos	%7-DECLARACION DE ALGORITMOS	<Algorithm>

Tabla 3. Puntos de ambas estructuras utilizados para la traducción a « 4DIAC » de cada apartado.

3.3.1. Preparación del « Archivo_destino ».

Las primeras líneas de código sirven para declarar el código que se va a escribir como una función y obtener los nombres de los dos archivos que se can a utilizar, los cuales estarán guardados en las variables « Archivo_fuente » y « Archivo_destino » tal y como se muestra en la « Imagen 32 ».

```
2     function [Archivo_destino] = FB_SLK24DIAC (Archivo_fuente)
3 -     Archivo_destino = strcat('IEC_', Archivo_fuente(1:length(Archivo_fuente)-4),'.fbt');
```

Imagen 32. Traducción a « 4DIAC », obtención de los nombres de los archivos.

El siguiente paso es abrir el « Archivo_destino » para poder escribir en el cuando sea necesario. Al abrirlo, se genera una dirección del archivo, la cual se guarda en « FID_d ». En el caso que haya algún error al abrir el archivo, el valor de « FID_d » será « -1 », por lo cual se introduce la condición de la línea 6, que al cumplirse enviará un mensaje por pantalla con el error.

```
5 -     [FID_d,MESSAGE]= fopen(Archivo_destino, 'wt');
6 -     if (FID_d==-1)
7 -         disp(strcat('Error al intentar leer el archivo: ', Archivo_destino, ',', MESSAGE))
```

Imagen 33. Traducción a « 4DIAC », apertura del « Archivo_destino ».

En el caso en que no se encuentre ningún error, el programa seguirá ejecutándose. Para empezar, se introducirán todos los datos de versión, fecha etc, que son necesarios antes de comenzar con el programa. Cada « \n » que se introduce marca un salto de línea en el « Archivo_destino ». De la misma manera, cada doble espacio introducido después de « \n » marca la pertenencia a « FBType ». Seguidamente se declara el inicio de la « InterfaceList » en la línea 26.

```
8 -   else
9         %Escribir la version y todo lo que precede a la declaración del bloque
10 -      fprintf(FID_d,'<?xml version="1.0" encoding="UTF-8" standalone="no"?>');
11 -      fprintf(FID_d,'\n');
12 -      fprintf(FID_d,'<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">');
13 -      fprintf(FID_d,'\n');
14 -      Comentario = strtok(Archivo_fuente,'.');
15 -      fprintf(FID_d,strcat('<FBType Comment="',Comentario,'" Name="',Archivo_fuente,'">'));
16 -      fprintf(FID_d,'\n');
17 -      fprintf(FID_d,'  ');
18 -      fprintf(FID_d,'<Identification Standard="61499-2"/>');
19 -      fprintf(FID_d,'\n');
20 -      fprintf(FID_d,'  ');
21 -      [Y,M,D]=ymd(datetime('now'));
22 -      DMY = strcat(int2str(D),'-',int2str(M),'-',int2str(Y));
23 -      fprintf(FID_d,strcat('<VersionInfo Author="FB_SLK24DIAC" Date="',DMY,'" Organization="UJI" Version="0.0"/>'));
24 -      fprintf(FID_d,'\n');
25 -      fprintf(FID_d,'  ');
26 -      fprintf(FID_d,'<InterfaceList>');
27 -      fprintf(FID_d,'\n');
```

Imagen 34. Traducción a « 4DIAC », introducción de los datos de versión.

3.3.2. Declaración de los Eventos de entrada.

Una vez hecho esto, abrimos el « Archivo_fuente » para comenzar a extraer los datos que necesitamos para declarar los Eventos de entrada.

```
28        %Escribir los EVENTOS DE ENTRADA con los puntos 1 y 4
29 -      [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r');
30 -      if (FID_f==-1)
31 -          disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
```

Imagen 35. Traducción a « 4DIAC », apertura del « Archivo_fuente ».

Para evitar errores, se inicia la variable « leer » como una cadena vacía de caracteres. Ya que lo primero con lo que nos encontramos en el archivo es el primer punto « %1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE », se introduce la condición de que hasta que no se encuentre el siguiente punto, la variable « leer »irá recorriendo el archivo por la función de la línea 46. Mientras no se haya llegado al segundo punto, la condición de las líneas de la 35 a la 37 se encarga de

separar del código los posibles comentarios introducidos. Mientras que la condición de las líneas de la 38 a la 45 se encarga de guardar en « E » los nombres de los Eventos de entrada (que comienzan por « e_ ») y las Variables de entrada (que comienzan por « v_ »).

```
32 -    else
33 -        leer= '';
34 -        while isempty(strfind(leer,'%2-DECLARACION DE VARIABLES INTERNAS'))
35 -            if ~isempty(strfind(leer,'%'))
36 -                leer = strtok(leer, '%');
37 -            end
38 -            if ~isempty(strfind(leer,'function'))
39 -                leer = strrep(leer,' ','');
40 -                [~, leer] = strtok(leer, '=');
41 -                leer = strrep(leer,'=fcn(','');
42 -                leer = strrep(leer,')','');
43 -                leer = strrep(leer,',',' ');
44 -                E    = textscan(leer, '%s');
45 -            end
46 -            leer = fgetl(FID_f);
47 -        end
```

Imagen 36. Traducción a « 4DIAC », extracción de los Eventos y las Variables de entrada.

Antes de comenzar a declarar los Eventos de entrada en el « Archivo_destino » hay que indicarlo en este, para ello están las líneas de la 49 a la 52. Seguidamente, se utiliza el bucle de las líneas de la 54 a la 56 para llegar al punto 4 del « Archivo_fuente »

```
49 -    fprintf(FID_d,'  ');
50 -    fprintf(FID_d,'  ');
51 -    fprintf(FID_d,'<EventInputs>');
52 -    fprintf(FID_d,'\n');
53
54 -    while isempty(strfind(leer, '%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)'))
55 -        leer = fgetl(FID_f);
56 -    end
```

Imagen 37. Traducción a « 4DIAC », apertura de la declaración de Eventos de entrada.

Una vez en el punto 4, declaramos la variable « Aux » como una celda que contiene « vacio » para poder marcar en « E » los Eventos que se van introduciendo en el « Archivo_destino ». La condición de la línea 62, marca cuándo se ha encontrado un Evento de entrada que contiene Variables asociadas con « WHITH », en este caso se separa toda la información que no sea el nombre del Evento y se busca en « E » para quitarlo.

```
57 -    Aux=textscan('vacio','%s');
58 -    while isempty(strfind(leer, '%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)'))
59 -        if ~isempty(strfind(leer,'%'))
60 -            leer = strtok(leer, '%');
61 -        end
62 -        if ~isempty(strfind(leer,'if'))
63 -            leer = strrep(leer,' ','');
64 -            leer = strrep(leer,'if','');
65 -            leer = strrep(leer,'(','');
66 -            leer = strrep(leer,')','');
67 -            H = length(E{1});
68 -            for i=1:H
69 -                if strcmp(char(E{1}(i)),leer)
70 -                    E{1}(i)=Aux{1};
71 -                end
72 -            end
```

Imagen 38. Traducción a « 4DIAC », extracción de las Variables de entrada asociadas a Eventos de entrada.

Una vez tenemos esto, se le quita la extensión « e_ » y se declara en el « Archivo_destino » tal y como se indica en la línea 77, se almacena el nombre del Evento en otra variable « C » y se continúa leyendo el archivo.

```
73 -                    leer = leer(3:length(leer));
74 -                    fprintf(FID_d,'  ');
75 -                    fprintf(FID_d,'  ');
76 -                    fprintf(FID_d,'  ');
77 -                    fprintf(FID_d, strcat('<Event Name="', leer, '" Type="Event">'));
78 -                    fprintf(FID_d,'\n');
79 -                    C=leer;
80 -                    leer = fgetl(FID_f);
```

Imagen 39. Traducción a « 4DIAC », declaración de los Eventos de entrada.

En la estructura de Simulink, los enlaces entre Eventos y Variables mediante « WHITH » se declaran en un bucle. Por lo tanto, hasta que se encuentre el final de este bucle « end » se irá recorriendo el archivo. Cuando se encuentre una Variable que no tenga el mismo nombre que el Evento, se declarará en el « Archivo_destino » tal y como se muestra en la línea 92. Una vez se llega al « end » del bucle, se cierra la declaración de dicho Evento (línea 100) y se repite el mismo procedimiento desde la línea 58 para todos os Eventos que tengan Variables asociadas.

```
81 -        while isempty(strfind(leer,'end'))
82 -            if ~isempty(strfind(leer,'%'))
83 -                leer = strtok(leer, '%');
84 -            end
85 -            leer = strtok(leer,'=');
86 -            leer = strrep(leer,' ','');
87 -            if ~isequal(C,leer)
88 -                fprintf(FID_d,'  ');
89 -                fprintf(FID_d,'  ');
90 -                fprintf(FID_d,'  ');
91 -                fprintf(FID_d,'  ');
92 -                fprintf(FID_d,strcat('<With Var="',leer,'"/>'));
93 -                fprintf(FID_d,'\n');
94 -            end
95 -            leer = fgetl(FID_f);
96 -        end
97 -        fprintf(FID_d,'  ');
98 -        fprintf(FID_d,'  ');
99 -        fprintf(FID_d,'  ');
100 -       fprintf(FID_d, strcat('</Event>'));
101 -       fprintf(FID_d,'\n');
102 -   end
103 -   leer = fgetl(FID_f);
104 - end
```

Imagen 40. Traducción a « 4DIAC », declaración de los Eventos de entrada (2).

A continuación, se declaran el resto de Eventos que no tengan Variables asociadas, cuyos nombres se encontrarán en los elementos de la variable « E » que comiencen por « e_ ». Una vez ya se han declarado todos los Eventos de entrada, se cierra la declaración de éstos (línea 120).

```
105 -           H = length(E{1});
106 -           for i=1:H
107 -               C = char(E{1}(i));
108 -               if (C(1)=='e')&&(C(2)=='_')
109 -                   C = C(3:length(C));
110 -                   fprintf(FID_d,'  ');
111 -                   fprintf(FID_d,'  ');
112 -                   fprintf(FID_d,'  ');
113 -                   fprintf(FID_d, strcat('<Event Name="',C,'" Type="Event"/>'));
114 -                   fprintf(FID_d,'\n');
115 -                   E{1}(i) = Aux{1};
116 -               end
117 -           end
118 -           fprintf(FID_d,'  ');
119 -           fprintf(FID_d,'  ');
120 -           fprintf(FID_d,'</EventInputs>');
121 -           fprintf(FID_d,'\n');
122 -       end
123 -       fclose(FID_f);
```

Imagen 41. Traducción a « 4DIAC », declaración de los Eventos de entrada (3).

3.3.3. Declaración de Eventos de salida.

La declaración de Eventos de salida se estructura de la misma manera que la de Eventos de entrada. Se vuelve a abrir el « Archivo_fuente », ya que nos vuelve a hacer falta información del punto 1, y se extraen los nombres de los Eventos de salida (que comienzan por « e_ ») y las Variables de salida (que comienzan por « v_ »), dichos nombres se almacenan en la variable « S ».

```
125     %Escribir los EVENTOS DE SALIDA con los puntos 1 y 9
126 -   [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r'); %open file for reading
127 -   if (FID_f==-1)
128 -       disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
129 -   else
130 -       while isempty(strfind(leer,'%2-DECLARACION DE VARIABLES INTERNAS'))
131 -           if ~isempty(strfind(leer,'%'))
132 -               leer = strtok(leer, '%');
133 -           end
134 -           if ~isempty(strfind(leer,'function'))
135 -               leer = strrep(leer,' ','');
136 -               [leer] = strtok(leer, '=');
137 -               leer = strrep(leer,'function[','');
138 -               leer = strrep(leer,']','');
139 -               leer = strrep(leer,',',' ');
140 -               S    = textscan(leer, '%s');
141 -           end
142 -           leer = fgetl(FID_f);
143 -       end
```

Imagen 42. Traducción a « 4DIAC », extracción de los Eventos y las Variables de salida.

Una vez se tienen los nombres, se abre la declaración de Eventos de salida (línea 147) y se busca el punto 9 (líneas 150-152). Al igual que en el apartado anterior, se buscan Eventos de salida que tengan Variables asociadas con « WHITH », en este caso se separa toda la información que no sea el nombre del Evento y se busca en « S » para quitarlo.

```
145 -           fprintf(FID_d,'  ');
146 -           fprintf(FID_d,'  ');
147 -           fprintf(FID_d,'<EventOutputs>');
148 -           fprintf(FID_d,'\n');
149
150 -           while isempty(strfind(leer, '%9-EJECUCION DE VARIABLES DE SALIDA (WITH)'))
151 -               leer = fgetl(FID_f);
152 -           end
153 -           while isempty(strfind(leer, '%10-FINAL'))
154 -               if ~isempty(strfind(leer,'%'))
155 -                   leer = strtok(leer, '%');
156 -               end
157 -               if ~isempty(strfind(leer,'if'))
158 -                   leer = strrep(leer,' ','');
159 -                   leer = strrep(leer,'if','');
160 -                   leer = strrep(leer,'(','');
161 -                   leer = strrep(leer,')','');
162 -                   leer = strcat('e_',leer);
163 -                   H = length(S{1});
164 -                   for i=1:H
165 -                       if strcmp(char(S{1}(i)),leer)
166 -                           S{1}(i)=Aux{1};
167 -                       end
168 -                   end
```

Imagen 43. Traducción a « 4DIAC », extracción de las Variables de entrada asociadas a Eventos de entrada.

A continuación, se quita el « e_ » mediante la instrucción de la línea 169 y se declara dicho Evento. En este caso, la estructura de Simulink declara los enlaces entre Eventos y Variables mediante « WHITH » entre el inicio de un bucle y la condición « else ». Por lo tanto, hasta que se encuentre dicha condición se irá recorriendo el archivo. Cuando se encuentre una Variable , se declarará en el « Archivo_destino » tal y como se muestra en la línea 187.

```
169 -                   leer = leer(3:length(leer));
170 -                   fprintf(FID_d,'  ');
171 -                   fprintf(FID_d,'  ');
172 -                   fprintf(FID_d,'  ');
173 -                   fprintf(FID_d, strcat('<Event Name="', leer, '" Type="Event">'));
174 -                   fprintf(FID_d,'\n');
175 -                   leer = fgetl(FID_f);
176 -                   while isempty(strfind(leer,'else'))
177 -                       if ~isempty(strfind(leer,'%'))
178 -                           leer = strtok(leer, '%');
179 -                       end
180 -                       leer = strtok(leer,'=');
181 -                       leer = strrep(leer,' ','');
182 -                       leer = leer(3:length(leer));
183 -                       fprintf(FID_d,'  ');
184 -                       fprintf(FID_d,'  ');
185 -                       fprintf(FID_d,'  ');
186 -                       fprintf(FID_d,'  ');
187 -                       fprintf(FID_d,strcat('<With Var="',leer,'"/>'));
188 -                       fprintf(FID_d,'\n');
189 -                       leer = fgetl(FID_f);
190 -                   end
```

Imagen 44. Traducción a « 4DIAC », declaración de Eventos de salida.

Una vez se llega al « else » de la condición, se cierra la declaración de dicho Evento (línea 194) y se repite el mismo procedimiento desde la línea 153 para todos os Eventos que tengan Variables asociadas. Seguidamente, se declaran el resto de Eventos que no tengan Variables asociadas, cuyos nombres se encontrarán en los elementos de la variable « S » que comienzan por « e_ ». Una vez ya se han declarado todos los Eventos de salida se cierra la declaración de éstos (líne 214).

```
191 -                   fprintf(FID_d,'  ');
192 -                   fprintf(FID_d,'  ');
193 -                   fprintf(FID_d,'  ');
194 -                   fprintf(FID_d, strcat('</Event>'));
195 -                   fprintf(FID_d,'\n');
196 -               end
197 -               leer = fgetl(FID_f);
198 -           end
199 -           H = length(S{1});
200 -           for i=1:H
201 -               C = char(S{1}(i));
202 -               if (C(1)=='e')&&(C(2)=='_')
203 -                   C = C(3:length(C));
204 -                   fprintf(FID_d,'  ');
205 -                   fprintf(FID_d,'  ');
206 -                   fprintf(FID_d,'  ');
207 -                   fprintf(FID_d, strcat('<Event Name="',C,'" Type="Event"/>'));
208 -                   fprintf(FID_d,'\n');
209 -                   S{1}(i) = Aux{1};
210 -               end
211 -           end
212 -           fprintf(FID_d,'  ');
213 -           fprintf(FID_d,'  ');
214 -           fprintf(FID_d,'</EventOutputs>');
215 -           fprintf(FID_d,'\n');
216 -       end
217 -       fclose(FID_f);
```

Imagen 45. Traducción a « 4DIAC », declaración de Eventos de salida (2).

3.3.4. Obtención de datos para la declaración de Variables.

Una vez se han declarado los Eventos, se necesita saber el tipo de las Variables para poder declararlas. Dicha información se encuentra en se encuentra en el punto 3 del « Archivo_fuente ». Por lo cual, se vuelve a abrir dicho archivo y se busca el punto en cuestión. Seguidamente, se busca en dicho punto la instrucción « isempty », ya que es la función donde se comienzan a declarar todos los valores iniciales. Dicha declaración finaliza al cerrar la condición en el que se encuentra la función « isempty », por lo tanto se revisarán todas las Variables hasta encontrar un « end » (línea 235), se evitan líneas vacías y declaraciones de « array » mediante la línea 240 y se separa el nombre de la Variable y su valor inicial (líneas 241-243).

```
219         %Escribir los VARIABLES DE ENTRADA con los puntos 1 y 3
220 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r'); %open file for reading
221 -       if (FID_f==-1)
222 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
223 -       else
224 -           while isempty(strfind(leer, '%3-DECLARACION DEL ESTADO INICIAL'))
225 -               leer = fgetl(FID_f);
226 -           end
227             %extraemos los valores iniciales y su tipo.
228 -           while isempty(strfind(leer,'%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)'))
229 -               if ~isempty(strfind(leer,'%'))
230 -                   leer = strtok(leer, '%');
231 -               end
232 -               if ~isempty(strfind(leer,'isempty'))
233 -                   leer = fgetl(FID_f);
234 -                   Vars='';
235 -                   while isempty(strfind(leer,'end'))
236 -                       if ~isempty(strfind(leer,'%'))
237 -                           leer = strtok(leer, '%');
238 -                       end
239 -                       leer=strrep(leer,' ','');
240 -                       if isempty(strfind(leer,'['))&&~isempty(leer)
241 -                           leer=strrep(leer,'=',' ');
242 -                           [~,C]=strtok(leer);
243 -                           C=strrep(C,';','');
```

Imagen 46. Traducción a « 4DIAC », obtención del tipo de variable de las Variables.

Mediante los valores iniciales obtenidos, se comprueba el tipo de Variable siguiendo la *«* Tabla 2 *»* del apartado *«* **2.4. Normas para la introducción de datos en el Bloque de función** *»*. De esta manera, si se introduce un « . » implica que la Variable será un « REAL » y en el caso que se encuentre un « (» implicará que se ha utilizado alguna función.

```
244 -                           if ~isempty(strfind(C,'.'))
245 -                               Vars=strcat(Vars,leer,'REAL"');
246 -                           elseif ~isempty(strfind(C,'('))
247 -                               [C,C1]=strtok(C,'(');
248 -                               C=strrep(C,' ','');
249 -                               leer=strtok(leer);
250 -                               C1=strrep(C1,'(','');
251 -                               C1=strrep(C1,')','');
252 -                               leer = strcat(leer,'"',C1,';');
253 -                               leer = strrep(leer,'"',' ');
```

Imagen 47. Traducción a « 4DIAC », obtención del tipo de variable de las Variables (2).

En este caso, la función indica el tipo de Variable, mientras que el valor entre « () » indicará el valor inicial. En el caso de que ninguna función coincidiera, aparecería un error por pantalla (línea 273). En el caso de no ser una función, se tratará o bien de una cadena de caracteres « STRING » (en el caso que se encuentren « ' ») o bien un número entero « INT », el cual está formado únicamente por números.

```
254 -                                   if ~isempty(strfind(C,'logical'))
255 -                                       Vars=strcat(Vars,leer,'BOOL"');
256 -                                   elseif ~isempty(strfind(C,'double'))
257 -                                       Vars=strcat(Vars,leer,'LREAL"');
258 -                                   elseif ~isempty(strfind(C,'int8'))
259 -                                       Vars=strcat(Vars,leer,'SINT"');
260 -                                   elseif ~isempty(strfind(C,'int32'))
261 -                                       Vars=strcat(Vars,leer,'DINT"');
262 -                                   elseif ~isempty(strfind(C,'int64'))
263 -                                       Vars=strcat(Vars,leer,'LINT"');
264 -                                   elseif ~isempty(strfind(C,'uint8'))
265 -                                       Vars=strcat(Vars,leer,'USINT"');
266 -                                   elseif ~isempty(strfind(C,'uint16'))
267 -                                       Vars=strcat(Vars,leer,'UINT"');
268 -                                   elseif ~isempty(strfind(C,'uint32'))
269 -                                       Vars=strcat(Vars,leer,'UDINT"');
270 -                                   elseif ~isempty(strfind(C,'uint64'))
271 -                                       Vars=strcat(Vars,leer,'ULINT"');
272 -                                   else
273 -                                       disp(strcat('no se ha podido leer bien: ',C));
274 -                                   end
275 -                               elseif ~isempty(strfind(C,''''))
276 -                                   leer=strrep(leer,'''',' ');
277 -                                   Vars=strcat(Vars,leer,'STRING"');
278 -                               else
279 -                                   Vars=strcat(Vars,leer,'INT"');
280 -                               end
```

Imagen 48. Traducción a « 4DIAC », obtención del tipo de variable de las Variables (3).

Una vez obtenidas todas las Variables con sus valores iniciales y su tipo en la variable « Vars ». Se separan mediante función « textscan » (línea 286),de manera que se obtiene una variable « Vals », la cual contiene 3 celdas con todos los datos nombrados de forma ordenada, y se cierra el archivo (línea 291). Con los datos de las Variables ya organizados, se pueden comenzar a declarar éstas.

```
281 -                       end
282 -                       leer = fgetl(FID_f);
283 -                   end
284 -                   Vars=strrep(Vars,';','"');
285 -                   Vars=strrep(Vars,'"',' ');
286 -                   Vals=textscan(Vars, '%s %s %s');
287 -               end
288 -               leer = fgetl(FID_f);
289 -           end
290 -       end
291 -       fclose(FID_f);
```

Imagen 49. Traducción a « 4DIAC », obtención del tipo de variable de las Variables de entrada (4).

3.3.5. Declaración de Variables de entrada y de salida.

A la hora de declarar las Variables tanto de entrada como de salida, se sigue la misma estructura. Primero se abre la declaración de Variables (líneas 295 y 324). Seguidamente, se abre un bucle « for » que recorrerá la variable « E » o « S » (Extraídas en las Declaraciones de Eventos) dependiendo de si se trata de Variables de entrada o de salida respectivamente. Mientras se recorre la variable en cuestión (ya que esta es un « array »), si se encuentra alguna Variable, lo cual implica que comienza por « v_ », se extraerá el nombre de las Variable y se buscará en la celda « Vals{1} » (líneas 304 y 333), ya que es la que contiene los nombres de las Variables. Una vez encontrada, se introducirá la declaración de la Variable en cuestión en el « Archivo_destino » con el tipo de Variable « Vals{3} » (líneas 308 y 337) y se quitará dicho nombre de « E » o « S » (líneas 310 y 339). Una vez declaradas las Variables, se cierra la declaración de éstas (líneas 318 y 346).

```
293 -       fprintf(FID_d,'  ');
294 -       fprintf(FID_d,'  ');
295 -       fprintf(FID_d,'<InputVars>');
296 -       fprintf(FID_d,'\n');
297 -       H = length(E{1});
298 -       for i=1:H
299 -           C = char(E{1}(i));
300 -           if (C(1)=='v')&&(C(2)=='_')
301 -               C = C(3:length(C));
302 -               L=length(Vals{1});
303 -               for j=1:L
304 -                   if strcmp(char(Vals{1}(j)),C)
305 -                       fprintf(FID_d,'  ');
306 -                       fprintf(FID_d,'  ');
307 -                       fprintf(FID_d,'  ');
308 -                       fprintf(FID_d,strcat('<VarDeclaration Name="',C,'" Type="',char(Vals{3}(j)),'"/>'));
309 -                       fprintf(FID_d,'\n');
310 -                       E{1}(i) = Aux{1};
311 -                   end
312 -               end
313 -           end
314 -       end
315
316 -       fprintf(FID_d,'  ');
317 -       fprintf(FID_d,'  ');
318 -       fprintf(FID_d,'</InputVars>');
319 -       fprintf(FID_d,'\n');
```

Imagen 50. Traducción a « 4DIAC », declaración de las Variables de entrada.

```
322 -       fprintf(FID_d,'  ');
323 -       fprintf(FID_d,'  ');
324 -       fprintf(FID_d,'<OutputVars>');
325 -       fprintf(FID_d,'\n');
326 -       H = length(S{1});
327 -       for i=1:H
328 -           C = char(S{1}(i));
329 -           if (C(1)=='v')&&(C(2)=='_')
330 -               C = C(3:length(C));
331 -               L=length(Vals{1});
332 -               for j=1:L
333 -                   if strcmp(char(Vals{1}(j)),C)
334 -                       fprintf(FID_d,'  ');
335 -                       fprintf(FID_d,'  ');
336 -                       fprintf(FID_d,'  ');
337 -                       fprintf(FID_d,strcat('<VarDeclaration Name="',C,'" Type="',char(Vals{3}(j)),'"/>'));
338 -                       fprintf(FID_d,'\n');
339 -                       S{1}(i) = Aux{1};
340 -                   end
341 -               end
342 -           end
343 -       end
344 -       fprintf(FID_d,'  ');
345 -       fprintf(FID_d,'  ');
346 -       fprintf(FID_d,'</OutputVars>');
347 -       fprintf(FID_d,'\n');
348 -       fprintf(FID_d,'  ');
```

Imagen 51. Traducción a « 4DIAC », declaración de las Variables de salida.

3.3.6. Declaración de Variables internas.

Las Variables internas se declaran dentro del « BasicFB ». Por lo tanto, antes que nada hay que cerrar la « InterfaceList » (línea 349) y abrir el « BasicFB » (línea 354) antes que nada. Una vez hecho ésto, se vuelve a abrir el « Archivo_fuente » para extraer el nombre de las Variables internas, lo cual se encuentra en el punto 2, y se abre la declaración de Variables internas (línea 366).

```
348 -       fprintf(FID_d,'  ');
349 -       fprintf(FID_d,'</InterfaceList>');
350 -       fprintf(FID_d,'\n');
351
352         %Escribir las VARIABLES INTERNAS con el punto 2
353 -       fprintf(FID_d,'  ');
354 -       fprintf(FID_d,'<BasicFB>');
355 -       fprintf(FID_d,'\n');
356 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r'); %open file for reading
357 -       if (FID_f==-1)
358 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
359 -       else
360 -           while isempty(strfind(leer,'%2-DECLARACION DE VARIABLES INTERNAS'))
361 -               leer = fgetl(FID_f);
362 -           end
363 -           leer = fgetl(FID_f);
364 -           fprintf(FID_d,'  ');
365 -           fprintf(FID_d,'  ');
366 -           fprintf(FID_d,'<InternalVars>');
367 -           fprintf(FID_d,'\n');
```

Imagen 52. Traducción a « 4DIAC », declaración de las Variables internas.

Los nombres de las Variables internas se encuentran seguidos de la instrucción « presistent », por lo tanto se busca dicha palabra y se separa del resto de información (líneas 372-373). Los nombres se encuentran separados por espacios, con lo cual, mientras se encuentren espacios habrá Variables internas por declarar (línea 374). Dichas Variables se comparan con la lista que ya tenemos de « Vals » y se declaran en el « Archivo_destino ». Hay que tener en cuenta que no se pueden introducir « STRINGS » vacíos

como valor inicial, en su lugar se debe introducir « _ ». Por lo tanto, si el valor inicial « Vals{2} » es « _ » (línea 384) no se deberá introducir ningún carácter, tal y como se muestra en la línea 385.

```
368 -         while isempty(strfind(leer,'%3-DECLARACION DEL ESTADO INICIAL'))
369 -             if ~isempty(strfind(leer,'%'))
370 -                 leer = strtok(leer, '%');
371 -             end
372 -             if ~isempty(strfind(leer,'persistent'))
373 -                 leer = strrep(leer,'persistent','');
374 -                 while ~isempty(strfind(leer,' '))
375 -                     [C, leer] = strtok(leer);
376 -                     C = strrep(C,' ','');
377 -                     C = strrep(C,';','');
378 -                     L=length(Vals{1});
379 -                     for j=1:L
380 -                         if strcmp(char(Vals{1}(j)),C)
381 -                             fprintf(FID_d,'  ');
382 -                             fprintf(FID_d,'  ');
383 -                             fprintf(FID_d,'  ');
384 -                             if isequal(char(Vals{2}(j)),'_')
385 -                                 fprintf(FID_d,'<VarDeclaration InitialValue="" Name="');
386 -                                 fprintf(FID_d,C);
387 -                                 fprintf(FID_d,'" Type="');
388 -                                 C1=char(Vals{3}(j));
389 -                                 fprintf(FID_d,C1);
390 -                                 fprintf(FID_d,'"/>');
```

Imagen 53. Traducción a « 4DIAC », declaración de las Variables internas (2).

En el caso de que no se trate de un « STRING » vacío, la declaración se realiza como se muestra en las líneas de la 392 a la 400. Se repite el mismo procedimiento para todas las Variables internas y se cierra dicha declaración al terminar (línea 411).

```
391 -                             else
392 -                                 fprintf(FID_d,'<VarDeclaration InitialValue="');
393 -                                 C1=char(Vals{2}(j));
394 -                                 fprintf(FID_d,C1);
395 -                                 fprintf(FID_d,'" Name="');
396 -                                 fprintf(FID_d,C);
397 -                                 fprintf(FID_d,'" Type="');
398 -                                 C1=char(Vals{3}(j));
399 -                                 fprintf(FID_d,C1);
400 -                                 fprintf(FID_d,'"/>');
401 -                             end
402 -                             fprintf(FID_d,'\n');
403 -                         end
404 -                     end
405 -                 end
406 -             end
407 -             leer = fgetl(FID_f);
408 -         end
409 -         fprintf(FID_d,'  ');
410 -         fprintf(FID_d,'  ');
411 -         fprintf(FID_d,'</InternalVars>');
412 -         fprintf(FID_d,'\n');
```

Imagen 54. Traducción a « 4DIAC », declaración de las Variables internas (3).

3.3.7. Obtención de los datos de los Estados y las transiciones del ECC.

Para poder declarar los Estados del « ECC », se necesita saber los nombres de éstos y los algoritmos y « Outputs » asociados a cada uno de los mismos. Para ello, se debe buscar en el « Archivo_fuente » el punto 6 (líneas 415-417), donde se encuentran los

algoritmos, con los caracteres « Alg_ » precediendo el nombre, y los Estados asociados a estos. Sabiendo esto, se quitan los posibles espacios y se busca que los 4 primeros caracteres sean « Alg_ » (línea 425) para ir juntando estas declaraciones en « Ejec_Alg ». El hecho de quitar los posibles espacios es muy importante, ya que posteriormente se utiliza la función « textscan » para separar en celdas los algoritmos e introducirlos en la variable « Ejec_Alg », por lo cual se precisa que solamente se encuentren separados por espacios los distintos algoritmos.

```
415 -          while isempty(strfind(leer,'%6-EJECUCION DE ALGORITMOS'))
416 -              leer = fgetl(FID_f);
417 -          end
418 -          leer = fgetl(FID_f);
419 -          Ejec_Alg='';
420 -          while isempty(strfind(leer,'%7-DECLARACION DE ALGORITMOS'))
421 -              if ~isempty(strfind(leer,'%'))
422 -                  leer = strtok(leer, '%');
423 -              end
424 -              leer = strrep(leer,' ','');
425 -              if ~isempty(leer)&&strcmp(leer(1:4),'Alg_')
426 -                  leer = strrep(leer,'Alg_','');
427 -                  leer = strrep(leer,'(','');
428 -                  leer = strrep(leer,',','');
429 -                  leer = strrep(leer,')','');
430 -                  Ejec_Alg=strcat(Ejec_Alg,'"',leer);
431 -              end
432 -              leer = fgetl(FID_f);
433 -          end
434 -          Ejec_Alg=strrep(Ejec_Alg,'"',' ');
435 -          Algor = textscan(Ejec_Alg,'%s');
```

Imagen 55. Traducción a « 4DIAC », obtención de los algoritmos asociados a un Evento de entrada concreto.

Lo siguiente que se precisa para poder declarar los Estados son los « Outputs ». Para ello, se busca en el « Archivo_fuente » el punto 8 (líneas 437-439), donde se encuentran los « Outputs » y los Estados asociados a estos. Se quitan los posibles espacios para evitar el mismo problema que con los algoritmos y, mientras no se encuentre una línea vacía, se irán juntando las declaraciones en « Ejec_Outputs » para posteriormente tenerlos organizados en la variable « Outp ».

```
437 -          while isempty(strfind(leer,'%8-EJECUCION DE EVENTOS DE SALIDA'))
438 -              leer = fgetl(FID_f);
439 -          end
440 -          leer = fgetl(FID_f);
441 -          Ejec_Outputs='';
442 -          while isempty(strfind(leer,'%9-EJECUCION DE VARIABLES DE SALIDA (WITH)'))
443 -              if ~isempty(strfind(leer,'%'))
444 -                  leer = strtok(leer, '%');
445 -              end
446 -              leer = strrep(leer,' ','');
447 -              if ~isempty(leer);
448 -                  leer = strrep(leer,'(','');
449 -                  leer = strrep(leer,',','');
450 -                  leer = strrep(leer,')','');
451 -                  Ejec_Outputs=strcat(Ejec_Outputs,leer);
452 -              end
453 -              leer = fgetl(FID_f);
454 -          end
455 -          Ejec_Outputs=strrep(Ejec_Outputs,';',' ');
456 -          Outp = textscan(Ejec_Outputs,'%s');
457 -      end
458 -      fclose(FID_f);
```

Imagen 56. Traducción a « 4DIAC », obtención de los Eventos de salida asociados a un Evento de entrada concreto.

Una vez tenemos estos datos, solamente nos queda saber los nombres de los Estados. Sin embargo, el punto en el que encontramos ésta información es el mismo que necesitaremos más adelante para declarar las transiciones, por lo cual se preparará la información para las transiciones a la vez que los nombres de los Estados.

Se abre el « ECC » y se busca en el « Archivo_fuente » el punto 5 (líneas 471-473), donde se encuentran los Estados y las condiciones para el cambio de Estado (transiciones). Si se encuentra algún « if » implicará que un cambio de Estado se encuentra declarado en ese punto, por lo cual se introduce la condición de la línea 482 y se guarda la información en la variable « Tra » sin el « if » para comenzar a tratarla (línea 484).

```
462 -       fprintf(FID_d,'  ');
463 -       fprintf(FID_d,'  ');
464 -       fprintf(FID_d,'<ECC>');
465 -       fprintf(FID_d,'\n');
466 -       Transiciones = '';
467 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r'); %open file for reading
468 -       if (FID_f==-1)
469 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
470 -       else
471 -           while isempty(strfind(leer,'%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)'))
472 -               leer = fgetl(FID_f);
473 -           end
474 -           Tra = '';
475 -           Etps='';
476 -           y=300;
477 -           Etapa_inicial = '';
478 -           while isempty(strfind(leer,'%6-EJECUCION DE ALGORITMOS'))
479 -               if ~isempty(strfind(leer,'%'))
480 -                   leer = strtok(leer, '%');
481 -               end
482 -               if ~isempty(strfind(leer,'if'))
483 -                   leer = strrep(leer,' ','');
484 -                   Tra=leer(3:length(leer));
```

Imagen 57. Traducción a « 4DIAC », preparación de la obtención de datos para la declaración del « ECC ».

El primer paso ahora, es traducir el nombre de los Estados, de manera que el formato sea « EXX » en vez de « CE(X,X) » siendo « X » un número que depende del Estado. Para ello, se buscan las posiciones en las que aparezca « E » y se comprueba que el siguiente carácter sea un « (» para asegurarnos que es un Estado y no parte de alguna Variable de la transición. De la misma manera, se comprueba que le preceda el carácter « C » y en caso contrario aparecerá un error ya que los Estados deben aparecer con la copia de Estados en este trozo de código. El nombre del Estado terminará con el carácter «) », por lo tanto se separa el nombre del Estado (líneas 494 y 500) y se eliminan los elementos prescindibles para dejar un nombre más claro.

```
487 - Tra = strrep(Tra,' ','');
488 - Pos = strfind(Tra,'E');
489 - if ~isempty(Pos)
490 -     H=length(Pos);
491 -     for i=H:-1:1
492 -         if (Tra(Pos(i)+1) == '(')
493 -             if (Tra(Pos(i)-1) == 'C')
494 -                 C = strtok(Tra(Pos(i):length(Tra)),')');
495 -                 L = length(C);
496 -                 C = strrep(C,'(','');
497 -                 C = strrep(C,',','');
498 -                 Tra = strcat(Tra(1:(Pos(i)-2)),C,Tra((Pos(i)+L+1):length(Tra)));
499 -             else
500 -                 disp(strcat('El Estado de la transición: "', Tra, '" no se enc...
501 -             end
502 -         end
503 -     end
504 - end
```

Imagen 58. Traducción a « 4DIAC », traducción de los nombres de los Estados.

El siguiente paso es encontrar los posibles Evento o Variables en las transiciones. Para ello, se sabe que sus nombres van precedidos de « e_ » en el caso de los Eventos y de « v_ » en caso de las Variables, pero al buscar dichos caracteres puede darse el caso de que formen parte del nombre de alguna Variable. Por lo tanto, para asegurar que se trata de un Evento o una Variable, se condiciona a que el carácter que precede a « e_ » o « v_ » no sea ni una letra ni un número (líneas 512 y 523), ya que solamente pueden ir precedidos por un operador matemático. Una vez se tiene el Evento o Variable, se le quita la extensión « e_ » o « v_ » (líneas 513 y 524).

```
506 -   Pos = strfind(Tra,'e_');
507 -   if ~isempty(Pos)
508 -       H = length(Pos);
509 -       Letra = isletter(Tra);
510 -       for i=H:-1:1
511 -           j = Pos(i)-1;
512 -           if ~Letra(j) && (Tra(j)~='0')&&(Tra(j)~='1')&&(Tra(j)~='2')&&(Tra(j)~='3')&&
513 -               Tra = strcat(Tra(1:(Pos(i)-1)),Tra((Pos(i)+2):length(Tra)));
514 -           end
515 -       end
516 -   end
517 -   Pos = strfind(Tra,'v_');
518 -   if ~isempty(Pos)
519 -       H = length(Pos);
520 -       Letra = isletter(Tra);
521 -       for i=H:-1:1
522 -           j = Pos(i)-1;
523 -           if ~Letra(j) && (Tra(j)~='0')&&(Tra(j)~='1')&&(Tra(j)~='2')&&(Tra(j)~='3')&&
524 -               Tra = strcat(Tra(1:(Pos(i)-1)),Tra((Pos(i)+2):length(Tra)));
525 -           end
526 -       end
527 -   end
```

Imagen 59. Traducción a « 4DIAC », traducción de los nombres de los Eventos y Variables.

Algunas instrucciones o funciones son muy parecidas pero necesitan ser traducidas. Para ello, se sustituyen los nombres de dichas funciones por las de « xml » tal y como se muestra en la « Imagen 60 ».

```
528 -   Tra = strrep(Tra,'abs(','"ABS(');
529 -   Tra = strrep(Tra,'sqrt(','"SQRT(');
530 -   Tra = strrep(Tra,'log(','"LN(');
531 -   Tra = strrep(Tra,'log10(','"LOG(');
532 -   Tra = strrep(Tra,'exp(','"EXP(');
533 -   Tra = strrep(Tra,'sin(','"SIN(');
534 -   Tra = strrep(Tra,'cos(','"COS(');
535 -   Tra = strrep(Tra,'tan(','"TAN(');
536 -   Tra = strrep(Tra,'asin(','"ASIN(');
537 -   Tra = strrep(Tra,'acos(','"ACOS(');
538 -   Tra = strrep(Tra,'atan(','"ATAN(');
539 -   Tra = strrep(Tra,'length(','"LEN(');
540 -   Tra = strrep(Tra,'not(','"NOT(');
```

Imagen 60. Traducción a « 4DIAC », traducción de las funciones matemáticas.

De la misma manera, para traducir las instrucciones de comparación, simplemente hay que sustituirlas por su traducción. Sin embargo, el orden es muy importante ya que, por ejemplo, si el símbolo « ~= », el cual resultaría « <> », se traduce antes de « < » o « > », a la hora de traducir éstos 2 últimos se traduciría de manera errónea. Por ésta razón se traduce en el orden que aparece en las líneas de la 541 a la 547. La dificultad en el símbolo « = » se encuentra en diferenciarla de « == », ya que el primero se traduce como «:= » mientras que el segundo se traduce como « = ». Para solucionar ésto, se buscan las posiciones en las que haya un « = » y se condiciona a que el carácter siguiente también sea un « = » o que el carácter que le precede no sea un « = » para asegurar que es simplemente dicho símbolo (líneas 140-144). Al introducir caracteres en la cadena, las posiciones obtenidas por la función « strfind » no serían correctas, por esta razón se recorre el « array » de posiciones de final a inicio.

```
541 -   Tra = strrep(Tra,'||','"OR"');
542 -   Tra = strrep(Tra,'&&','"AND"');
543 -   Tra = strrep(Tra,'>=','&ge;');
544 -   Tra = strrep(Tra,'<=','&le;');
545 -   Tra = strrep(Tra,'>','&gt;');
546 -   Tra = strrep(Tra,'<','&lt;');
547 -   Tra = strrep(Tra,'~=','<>');
548 -   Pos = strfind(Tra,'=');
549 -   if ~isempty(Pos)
550 -       H = length(Pos);
551 -       for i = H:-1:1
552 -           if Tra(Pos(i)+1) == '='
553 -               Tra = strcat(Tra(1:Pos(i)-1),Tra(Pos(i)+1:length(Tra)));
554 -           elseif Tra(Pos(i)-1) ~= '='
555 -               Tra = strcat(Tra(1:Pos(i)-1),':',Tra(Pos(i):length(Tra)));
556 -           end
557 -       end
558 -   end
```

Imagen 61. Traducción a « 4DIAC », traducción de los operadores lógicos.

Una vez ya se ha traducido el símbolo « ~= », ya se puede traducir « ~ » como la función « not() ». Al ser una función, si « ~ » precede a un « (», simplemente se traducirá « ~ » por « not ». En caso contrario, se tendrá que buscar el final del Estado o Variable negado e introducir el nombre entre « () » (línea 577). Se busca el siguiente carácter después de « ~ » que no sea ni una letra ni un número (línea 570), en caso de no encontrarse (línea 574) significa que el Evento o Variable es lo último declarado en la transición, por lo cual se introduciría «) » al final (línea 575).

```
559 -   Pos = strfind(Tra,'~');
560 -   if ~isempty(Pos)
561 -       H = length(Pos);
562 -       for i=H:-1:1
563 -           if Tra(i+1)=='('
564 -               Tra = strcat(Tra(1:(Pos(i)-1)),'NOT',Tra((Pos(i)+1):(length(Tra))));
565 -           else
566 -               L=Pos(i)+1;
567 -               Letra = isletter(Tra);
568 -               k=0;
569 -               for j=length(Letra):-1:L
570 -                   if ~Letra(j) && (Tra(j)~='0')&&(Tra(j)~='1')&&(Tra(j)~='2')&&(Tra(j)~='3')&&(Tra(j)~=
571 -                       k=j;
572 -                   end
573 -               end
574 -               if k==0
575 -                   Tra = strcat(Tra(1:(Pos(i)-1)),'NOT(',Tra((Pos(i)+1):(length(Tra))),')');
576 -               else
577 -                   Tra = strcat(Tra(1:(Pos(i)-1)),'NOT(',Tra((Pos(i)+1):(k-1)),')',Tra(k:length(Tra)));
578 -               end
579 -           end
580 -       end
581 -   end
```

Imagen 62. Traducción a « 4DIAC », traducción de la función de negación.

Por último antes de empezar con la declaración de los Estados, se deben traducir los « ^ » (línea 582) e introducir los saltos de línea «

 ». Los cuales se deben introducir seguido de los « ; », por lo cual se buscan las posiciones de éstos y se incluye dicha instrucción en el código (línea 588), teniendo en cuenta que los « ; » que pertenezcan a las instrucciones « > », « < », « ≥ » o « ≤ » no deben tenerse en cuenta (línea 587).

```
582 -  Tra = strrep(Tra,'^','**');
583 -  Pos = strfind(Tra,';');
584 -  if ~isempty(Pos)
585 -      H = length(Pos);
586 -      for i = H:-1:1
587 -          if ~strcmp(Tra(Pos(i)-3:Pos(i)),'&gt;') && ~strcmp(Tra(Pos(i)-3:Pos(i)),'&lt;') &&
588 -              Tra = strcat(Tra(1:Pos(i)),'&#10;&#13;',Tra(Pos(i)+1:length(Tra)));
589 -          end
590 -      end
591 -  end
```

Imagen 63. Traducción a « 4DIAC », introducción de los saltos de línea en el código.

3.3.8. Declaración de los Estados del ECC.

Una vez tenemos una transición, se debe obtener el Estado que precede a ésta y el que lo sigue. Para ello se sabe que el Estado que se desactiva será el que esté igualado a « 0 », mientras que el que esté igualado a « 1 » será el que se active. Por lo tanto, se extrae la información a la derecha del « = » (línea 599) y se comprueba (línea 606). Otra cosa que hay que tener en cuenta, es que el Estado inicial debe declararse como « START », por lo tanto, el primer Estado que se desactive (línea 607), el cual será el inicial, será declarado de esta manera (línea 609).

```
592 -  leer = fgetl(FID_f);
593 -  ESou='';
594 -  EDes='';
595 -  while isempty(strfind(leer,'end'))
596 -      if ~isempty(strfind(leer,'%'))
597 -          leer = strtok(leer, '%');
598 -      end
599 -      [Etp,C]=strtok(leer,'=');
600 -      Etp=strrep(Etp,',','');
601 -      Etp=strrep(Etp,' ','');
602 -      Etp=strrep(Etp,'(','');
603 -      Etp=strrep(Etp,')','');
604 -      C=strrep(C,'=','');
605 -      C=strrep(C,';','');
606 -      if C=='0'
607 -          if isempty(Etapa_inicial)
608 -              Etapa_inicial = Etp;
609 -              ESou = 'START';
```

Imagen 64. Traducción a « 4DIAC », obtención del Estado fuente « Source ».

Una vez tenemos el Estado inicial, comenzamos a declararlo tal y como se muestra en las líneas 614, 616, 620 y 622, dejando la declaración sin acabar, ya que aún no se sabe si dicho Estado tiene algoritmos o « Outputs » asociados. La variable « y » marca la posición horizontal del Estado en el « ECC », por lo tanto se debe ir incrementando cada vez que se declare un Estado, de manera que el esquema quede más o menos ordenado. Una vez declarado ésto, se buscan los algoritmos asociados a éste (línea 626) y se guardan en la variable « Ejec_Alg ».

```
610 -                Etps=strcat(Etps,Etp);
611 -                fprintf(FID_d,'  ');
612 -                fprintf(FID_d,'  ');
613 -                fprintf(FID_d,'  ');
614 -                fprintf(FID_d,'<ECState Name="');
615 -                if isequal(Etapa_inicial,Etp)
616 -                    fprintf(FID_d,'START');
617 -                else
618 -                    fprintf(FID_d,Etp);
619 -                end
620 -                fprintf(FID_d,'" x="1000.0" y="');
621 -                y=y+100;
622 -                fprintf(FID_d,int2str(y));
623 -                Ejec_Alg='';
624 -                H=length(Algor{1});
625 -                for i=1:H
626 -                    if strfind(char(Algor{1}(i)),Etp)
627 -                        if isempty(Ejec_Alg)
628 -                            Ejec_Alg=strtok(char(Algor{1}(i)),'=');
629 -                        else
630 -                            Ejec_Alg=strcat(Ejec_Alg,',',strtok(char(Algor{1}(i)),'='));
631 -                            Ejec_Alg = strrep(Ejec_Alg,',',' ');
632 -                        end
633 -                    end
634 -                end
```

Imagen 65. Traducción a « 4DIAC », declaración del Estado inicial.

Se repite el mismo procedimiento para los « Output » asociados a los Estados (línea 639) y se guardan en la variable « Ejec_Outputs ».

Seguidamente, se comprueba si solamente hay uno o más algoritmos y/o « Outputs » asociados (línea 647) ya que la declaración será distinta. Si solamente hay un algoritmo y un « Output », primero se cierra la anterior instrucción (línea 649) y luego se declara en una única instrucción y se cierra el estado (líneas 655 y 660).

```
635 -  Ejec_Outputs='';
636 -  H=length(Outp{1});
637 -  for i=1:H
638 -      if strfind(char(Outp{1}(i)),Etp)
639 -          if isempty(Ejec_Outputs)
640 -              Ejec_Outputs=strtok(char(Outp{1}(i)),'=');
641 -          else
642 -              Ejec_Outputs=strcat(Ejec_Outputs,',',strtok(char(Outp{1}(i)),'='));
643 -              Ejec_Outputs = strrep(Ejec_Outputs,',',' ');
644 -          end
645 -      end
646 -  end
647 -  if isempty(strfind(Ejec_Alg,' ')) && isempty(strfind(Ejec_Outputs,' '))
648 -      if ~isempty(Ejec_Alg)&&~isempty(Ejec_Outputs)
649 -          fprintf(FID_d,'">');
650 -          fprintf(FID_d,'\n');
651 -          fprintf(FID_d,'  ');
652 -          fprintf(FID_d,'  ');
653 -          fprintf(FID_d,'  ');
654 -          fprintf(FID_d,'  ');
655 -          fprintf(FID_d,strcat('<ECAction Algorithm="',Ejec_Alg,'" Output="',Ejec_Outputs,'"/>'));
656 -          fprintf(FID_d,'\n');
657 -          fprintf(FID_d,'  ');
658 -          fprintf(FID_d,'  ');
659 -          fprintf(FID_d,'  ');
660 -          fprintf(FID_d,'</ECState>');
```

Imagen 66. Traducción a « 4DIAC », declaración del Estado inicial (2).

En los casos en los que solamente hay o bien un algoritmo o bien un « Output », se repite el mismo procedimiento, pero solamente aparecerá en la declaración de lo que haya (líneas 668 y 681).

```
661 -       elseif ~isempty(Ejec_Alg)&&isempty(Ejec_Outputs)
662 -           fprintf(FID_d,'">');
663 -           fprintf(FID_d,'\n');
664 -           fprintf(FID_d,'  ');
665 -           fprintf(FID_d,'  ');
666 -           fprintf(FID_d,'  ');
667 -           fprintf(FID_d,'  ');
668 -           fprintf(FID_d,strcat('<ECAction Algorithm="',Ejec_Alg,'"/>'));
669 -           fprintf(FID_d,'\n');
670 -           fprintf(FID_d,'  ');
671 -           fprintf(FID_d,'  ');
672 -           fprintf(FID_d,'  ');
673 -           fprintf(FID_d,'</ECState>');
674 -       elseif isempty(Ejec_Alg)&&~isempty(Ejec_Outputs)
675 -           fprintf(FID_d,'">');
676 -           fprintf(FID_d,'\n');
677 -           fprintf(FID_d,'  ');
678 -           fprintf(FID_d,'  ');
679 -           fprintf(FID_d,'  ');
680 -           fprintf(FID_d,'  ');
681 -           fprintf(FID_d,strcat('<ECAction Output="',Ejec_Outputs,'"/>'));
682 -           fprintf(FID_d,'\n');
683 -           fprintf(FID_d,'  ');
684 -           fprintf(FID_d,'  ');
685 -           fprintf(FID_d,'  ');
686 -           fprintf(FID_d,'</ECState>');
```

Imagen 67. Traducción a « 4DIAC », declaración del Estado inicial (3).

Si no se encuentra ningún algoritmo ni « Output » asociado, simplemente se acaba la anterior instrucción (línea 688). En el caso que haya más de un algoritmo asociado o más de un « Output » asociado (línea 691), se organizan los algoritmos y los « Outputs » en celdas y se comprueba de que hay más cantidad (línea 694).

```
687 -       else
688 -           fprintf(FID_d,'"/>');
689 -       end
690 -       fprintf(FID_d,'\n');
691 -   else
692 -       Algor = textscan(Ejec_Alg,'%s');
693 -       Outp  = textscan(Ejec_Outputs,'%s');
694 -       if length(Algor{1}) > length(Outp{1})
695 -           H = length(Algor{1});
696 -       else
697 -           H = length(Outp{1});
698 -       end
```

Imagen 68. Traducción a « 4DIAC », declaración del Estado inicial (4).

Para declarar todos los algoritmos y « Outputs » se abre un bucle en el cual ; mientras haya tanto algoritmos como « Outputs » por declarar se declarará uno de cada (línea 707), en el caso que solamente queden algoritmos por declarar se declararán como en la línea 720.

```
699 -  for i=1:H
700 -      if (i<=length(Algor{1}))&&(i<=length(Outp{1}))
701 -          fprintf(FID_d,'">');
702 -          fprintf(FID_d,'\n');
703 -          fprintf(FID_d,'  ');
704 -          fprintf(FID_d,'  ');
705 -          fprintf(FID_d,'  ');
706 -          fprintf(FID_d,'  ');
707 -          fprintf(FID_d,strcat('<ECAction Algorithm="',char(Algor{1}(i)),'" Output="',char(Outp{1}(i)),'"/>'));
708 -          fprintf(FID_d,'\n');
709 -          fprintf(FID_d,'  ');
710 -          fprintf(FID_d,'  ');
711 -          fprintf(FID_d,'  ');
712 -          fprintf(FID_d,'</ECState>');
713 -      elseif (i<=length(Algor{1}))&&(i>length(Outp{1}))
714 -          fprintf(FID_d,'">');
715 -          fprintf(FID_d,'\n');
716 -          fprintf(FID_d,'  ');
717 -          fprintf(FID_d,'  ');
718 -          fprintf(FID_d,'  ');
719 -          fprintf(FID_d,'  ');
720 -          fprintf(FID_d,strcat('<ECAction Algorithm="',char(Algor{1}(i)),'"/>'));
721 -          fprintf(FID_d,'\n');
722 -          fprintf(FID_d,'  ');
723 -          fprintf(FID_d,'  ');
724 -          fprintf(FID_d,'  ');
725 -          fprintf(FID_d,'</ECState>');
```

Imagen 69. Traducción a « 4DIAC », declaración del Estado inicial (5).

Por el contrario, si solamente quedan por declarar « Outputs » se declararán como en la línea 733.

El « else » de la línea 743 indica el caso en el que el Estado en cuestión no corresponda al inicial. En éste caso y para no duplicar información, ya que todos los Estados se activarán y se desactivarán, se guardará el nombre del Estado como Estado « Source » en « ESou » (línea 747). La línea 750 cierra el bucle del Estado que se desactiva.

```
726 -                  elseif (i>length(Algor{1}))&&(i<=length(Outp{1}))
727 -                      fprintf(FID_d,'">');
728 -                      fprintf(FID_d,'\n');
729 -                      fprintf(FID_d,'  ');
730 -                      fprintf(FID_d,'  ');
731 -                      fprintf(FID_d,'  ');
732 -                      fprintf(FID_d,'  ');
733 -                      fprintf(FID_d,strcat('<ECAction Output="',char(Outp{1}(i)),'"/>'));
734 -                      fprintf(FID_d,'\n');
735 -                      fprintf(FID_d,'  ');
736 -                      fprintf(FID_d,'  ');
737 -                      fprintf(FID_d,'  ');
738 -                      fprintf(FID_d,'</ECState>');
739 -                  end
740 -              end
741 -              fprintf(FID_d,'\n');
742 -          end
743 -      else
744 -          if isequal(Etp,Etapa_inicial)
745 -              ESou = 'START';
746 -          else
747 -              ESou = Etp;
748 -          end
749 -      end
750 -  end
```

Imagen 70. Traducción a « 4DIAC », declaración del Estado inicial (6).

Si por el contrario nos encontramos con un Estado que se activa, el procedimiento será similar que para el Estado inicial, pero en éste caso para el resto de Estados cuyo nombre se guardará en « EDes ». Se declara el Estado (línea 765) de la misma manera que anteriormente con el Estado inicial (línea 614).

```
751 -           if C=='1'
752 -               if isempty(EDes)
753 -                   if isequal(Etapa_inicial,Etp)
754 -                       EDes = 'START';
755 -                   else
756 -                       EDes = Etp;
757 -                   end
758 -               end
759 
760 -               if isempty(strfind(Etps,Etp)) && ~isequal(Etapa_inicial,Etp)
761 -                   Etps=strcat(Etps,Etp);
762 -                   fprintf(FID_d,'  ');
763 -                   fprintf(FID_d,'  ');
764 -                   fprintf(FID_d,'  ');
765 -                   fprintf(FID_d,'<ECState Name="');
766 -                   if isequal(Etapa_inicial,Etp)
767 -                       fprintf(FID_d,'START');
768 -                   else
769 -                       fprintf(FID_d,Etp);
770 -                   end
771 -                   fprintf(FID_d,'" x="1000.0" y="');
772 -                   y=y+100;
773 -                   fprintf(FID_d,int2str(y));
774 -                   Ejec_Alg='';
775 -                   H=length(Algor{1});
```

Imagen 71. Traducción a « 4DIAC », obtención del Estado destino « Destination » y declaración de Estados.

Se repite el mismo procedimiento que en la línea 624 para extraer los algoritmos (línea 775) y los « Outputs » (línea 787) asociados. En resumen, el código de la línea 607 a la 742 es el mismo que de la 759 a la 893.

Una vez declarados ambos Estados (« Source » y « Destination ») se cierra el bucle para seguir con el resto de transiciones (la línea 897 acaba el bucle que se comenzó en la línea 595).

```
892 -                               fprintf(FID_d,'\n');
893 -                           end
894 -                       end
895 -                   end
896 -                   leer = fgetl(FID_f);
897 -               end
```

Imagen 72. Traducción a « 4DIAC », declaración de Estados.

3.3.9. Declaración de las transiciones del ECC.

Antes de empezar a declarar las transiciones, se modifica los trozos de dicha transición en los cuales aparezca el Estado inicial por « START » (línea 900) y se comprueba si la transición es únicamente el Estado « Source » (línea 905), ya que en dicho caso el Estado es inestable y la transición siempre será « TRUE », es decir, « 1 ».

```
898 -                   i = strfind(Tra,Etapa_inicial);
899 -                   if ~isempty(i)
900 -                       Tra = strcat(Tra(1:i-1), 'START',Tra(i+length(Etapa_inicial):length(Tra)));
901 -                   end
902 -                   C = strrep(Tra,'(','');
903 -                   C = strrep(C,')','');
904 -                   C = strrep(C,'"','');
905 -                   if isequal(ESou,C)
906 -                       Tra = '1';
907 -                   end
```

Imagen 73. Traducción a « 4DIAC », corrección del nombre del Estado inicial y de las transiciones de un Estado inestable.

Las transiciones que se van guardando en la variable « Tra » se van juntando con sus respectivos Estados « Source » y « Destination » en la variable « Transiciones », separados por un « # » ya que las transiciones pueden contener « " » para indicar

los espacios. El carácter « # » solamente se utiliza en las transiciones precedido por « & ». Por lo cual, se introduce la condición de la línea 920 a la hora de cambiar los « # » por espacios una vez ya se haya cerrado el bucle que recorre todo el punto 5 del « Archivo_fuente » (línea 915). Una vez hecho ésto, se ordenan las transiciones en la variable « TRA » (línea 924), de manera que ; « TRA{1} » corresponderá a un « array » de celdas con las condiciones de la transiciones, « TRA{2} » a otro « array » de celdas con los Estados « Destination » y « TRA{3} » a otro « array » de celdas con los Estados « Source ».

```
908 -                   if isempty(Transiciones)
909 -                       Transiciones = strcat(Tra,'#',EDes,'#',ESou);
910 -                   else
911 -                       Transiciones = strcat(Transiciones,'#',Tra,'#',EDes,'#',ESou);
912 -                   end
913 -               end
914 -               leer = fgetl(FID_f);
915 -           end
916 -       end
917 -       Pos = strfind(Transiciones,'#');
918 -       H = length(Pos);
919 -       for i = H:-1:1
920 -           if Transiciones(Pos(i)-1) ~= '&'
921 -               Transiciones(Pos(i)) = ' ';
922 -           end
923 -       end
924 -       TRA = textscan(Transiciones,'%s %s %s');
925 -       fclose(FID_f);
```

Imagen 74. Traducción a « 4DIAC », organización de los datos de las transiciones para su declaración.

Con toda ésta información ya organizada, se declaran las transiciones como se muestra en el bucle de la línea 927 y se cierra posteriormente la declaración del « ECC » (línea 944).

```
926 -       H = length(TRA{1});
927 -       for i = 1:H
928 -           fprintf(FID_d,'  ');
929 -           fprintf(FID_d,'  ');
930 -           fprintf(FID_d,'  ');
931 -           fprintf(FID_d,'<ECTransition Condition="');
932 -           Tra = char(TRA{1}(i));
933 -           Tra = strrep(Tra,'"',' ');
934 -           fprintf(FID_d,Tra);
935 -           fprintf(FID_d,'" Destination="');
936 -           fprintf(FID_d,char(TRA{2}(i)));
937 -           fprintf(FID_d,'" Source="');
938 -           fprintf(FID_d,char(TRA{3}(i)));
939 -           fprintf(FID_d,strcat('" x="1200.0" y="',int2str(y+150),'"/>'));
940 -           fprintf(FID_d,'\n');
941 -       end
942 -       fprintf(FID_d,'  ');
943 -       fprintf(FID_d,'  ');
944 -       fprintf(FID_d,'</ECC>');
945 -       fprintf(FID_d,'\n');
```

Imagen 75. Traducción a « 4DIAC », declaración de transiciones.

3.3.10. Obtención y declaración de los algoritmos.

Para obtener la información necesaria para la declaración de los algoritmos, es necesario abrir el « Archivo_fuente » de nuevo y buscar el punto 7. De una forma similar a las transiciones de antes, el código de los algoritmos también se debe traducir. La diferencia más significativa con respecto a las transiciones es que aquí pueden aparecer bucles y condiciones.

Para empezar, los algoritmos van declarados dentro de la instrucción « if ». Por lo cual, se deberá buscar « if » para encontrar el inicio de los algoritmos (línea 957). Una vez se encuentra, se saca el nombre del algoritmo y se declara como se indica en la línea 964. Seguidamente se abre la declaración del código de dicho algoritmo, aunque primero se tendrá que traducir dicho código.

```
946 -    [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r');
947 -    if (FID_f==-1)
948 -        disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
949 -    else
950 -        while isempty(strfind(leer,'%7-DECLARACION DE ALGORITMOS'))
951 -            leer = fgetl(FID_f);
952 -        end
953 -        while isempty(strfind(leer,'%8-EJECUCION DE EVENTOS DE SALIDA'))
954 -            if ~isempty(strfind(leer,'%'))
955 -                leer = strtok(leer, '%');
956 -            end
957 -            if ~isempty(strfind(leer,'if'))
958 -                leer=strrep(leer,' ','');
959 -                leer=strrep(leer,'(','');
960 -                leer=strrep(leer,')','');
961 -                leer=leer(7:length(leer));
962 -                fprintf(FID_d,'  ');
963 -                fprintf(FID_d,'  ');
964 -                fprintf(FID_d,strcat('<Algorithm Name="',leer,'">'));
965 -                fprintf(FID_d,'\n');
966 -                fprintf(FID_d,'  ');
967 -                fprintf(FID_d,'  ');
968 -                fprintf(FID_d,'  ');
969 -                fprintf(FID_d,'<ST Text="');
```

Imagen 76. Traducción a « 4DIAC », declaración de algoritmos.

A la hora de trabajar con bucles y condiciones dentro del algoritmo, hay que tener cuidado ya que es necesario saber a qué bucle pertenece cada instrucción « end », además de que el indicador de fin del código del algoritmo también es un « end ». Por lo cual se inician dos variables ; « Tra » que será donde se irá guardando el código y « bucles » que irá contando dentro de cuántos bucles nos encontramos en el « Archivo_fuente ». Si se encuentra un bucle se sumará « 1 » a « bucles », mientras que si se encuentra un « end » se le restará. De esta forma, iniciando « bucles » a « 1 » sabremos que ha terminado de leer el algoritmo cuando éste sea « 0 » (línea 973).

Otro problema que puede haber con los bucles es al buscarlos por su nombre, ya que puede existir alguna Variable cuyo nombre incluya alguna de estas instrucciones como « if ». Para diferenciarlo, se incluye una segunda condición al encontrar el bucle, de manera que solamente se considerará bucle si la línea de código empieza con su nombre (líneas 981, 991, 996, 1002, 1008 y 1015).

La traducción de la mayoría de los bucles, condiciones, o partes de éstos como « elseif » o « otherwise », es sencilla ya que se basa en cambiar el nombre del bucle y añadir algo al final tal y como se muestra en las líneas 987, 993, 999, 1005, 1010, 1013, 1027 y 1031. Aunque éste último, que pertenece a los « end », no esté del todo acabado de declarar ya que se incluirá el bucle o condición al que pertenece más adelante.

```
971 -     Tra = '';
972 -     bucles=1;
973 -   while isempty(Tra) || bucles~=0
974 -         leer = fgetl(FID_f);
975 -         if ~isempty(strfind(leer,'%'))
976 -             leer = strtok(leer, '%');
977 -         end
978 -         leer = strrep(leer,' ','');
979
980 -         if ~isempty(strfind(leer,'if')) && isempty(strfind(leer,'elseif'))
981 -             if ~isempty(strcmp(leer(1:2),'if'))
982 -                 bucles = bucles+1;
983 -                 leer = strrep(leer,'if','');
984 -                 if isempty(Tra) && ~isempty(leer)
985 -                     Tra = strcat('IF"',leer,'"THEN;');
986 -                 elseif ~isempty(leer)
987 -                     Tra = strcat(Tra,'IF"',leer,'"THEN;');
988 -                 end
989 -             end
990 -         elseif ~isempty(strfind(leer,'elseif'))
991 -             if ~isempty(strcmp(leer(1:6),'elseif'))
992 -                 leer = strrep(leer,'elseif','');
993 -                 Tra = strcat(Tra,'ELSIF"',leer,'"THEN;');
994 -             end
```

Imagen 77. Traducción a « 4DIAC », traducción de bucles y condiciones en algoritmos.

```
995 -         elseif ~isempty(strfind(leer,'while'))
996 -             if ~isempty(strcmp(leer(1:5),'while'))
997 -                 bucles = bucles+1;
998 -                 leer = strrep(leer,'while','');
999 -                 Tra = strcat(Tra,'WHILE"',leer,'"DO;');
1000 -            end
1001 -        elseif ~isempty(strfind(leer,'switch'))
1002 -            if ~isempty(strcmp(leer(1:6),'switch'))
1003 -                bucles = bucles+1;
1004 -                leer = strrep(leer,'switch','');
1005 -                Tra = strcat(Tra,'CASE"',leer,'"OF;');
1006 -            end
1007 -        elseif ~isempty(strfind(leer,'case'))
1008 -            if ~isempty(strcmp(leer(1:4),'case'))
1009 -                leer = strrep(leer,'case','');
1010 -                Tra = strcat(Tra,leer,':"');
1011 -            end
1012 -        elseif ~isempty(strfind(leer,'otherwise'))
1013 -            Tra = strcat(Tra,'ELSE;');
```

Imagen 78. Traducción a « 4DIAC », traducción de bucles y condiciones en algoritmos (2).

Sin embargo, para el bucle « for » es algo distinto, ya que depende de cómo se declare la variable puede contener uno o dos caracteres « : ». De esta manera, si la variable recorre números consecutivos estará formado por un único carácter « : » (declarándose como en la línea 1021), mientras que si recorre números no consecutivos tendrá dos caracteres « : » (declarándose como en la línea 1023).

```
1014 -      elseif ~isempty(strfind(leer,'for'))
1015 -          if ~isempty(strcmp(leer(1:3),'for'))
1016 -              bucles = bucles+1;
1017 -              leer = strrep(leer,'for','');
1018 -              Pos=strfind(leer,':');
1019 -              tam=size(Pos);
1020 -              if tam(2)==1
1021 -                  Tra = strcat(Tra,'FOR"',leer(1:Pos-1),'"TO"',leer(Pos+1:length(leer)),'"BY"1"DO;');
1022 -              elseif tam(2)==2
1023 -                  Tra = strcat(Tra,'FOR"',leer(1:Pos(1)-1),'"TO"',leer(Pos(2)+1:length(leer)),'"BY"',
1024 -              end
1025 -          end
1026 -      elseif ~isempty(strfind(leer,'else'))
1027 -          Tra = strcat(Tra,'ELSE;');
1028 -      elseif isequal(leer,'end')
1029 -          bucles = bucles-1;
1030 -          if bucles ~= 0
1031 -              Tra = strcat(Tra,'END_;');
1032 -          end
1033 -      else
1034 -          Tra = strcat(Tra,leer);
1035 -      end
1036 - end
```

Imagen 79. Traducción a « 4DIAC », traducción de bucles y condiciones en algoritmos (3).

Una vez traducidos los bucles y condiciones, se repiten bastantes instrucciones que para la traducción de las transiciones. De manera que el código de la línea 488 a la 581 es el mismo que de la línea 1037 a la 1130. El resto de código ya no es el mismo, ya que al introducir los saltos de línea, tenemos que tener en cuenta ésta vez que las instrucciones ; « THEN », « OF », « DO » y « ELSE » de los bucles o condiciones, van seguidas de « ; » y también deben incluir un salto de línea (líneas 1135-1136).

Seguidamente, se extraen las posiciones de los bucles, las condiciones y los « END_ » para ordenarlos mediante la función de la línea 1148 « sort ». De ésta manera, se podrá deducir el tipo de bucle o condición de cada « END_ ».

```
1131 - Pos = strfind(Tra,';');
1132 - if ~isempty(Pos)
1133 -     H = length(Pos);
1134 -     for i = H:-1:1
1135 -         if strcmp(Tra(Pos(i)-4:Pos(i)-1),'THEN') || strcmp(Tra(Pos(i)-2:Pos(i)-1),'OF') || str
1136 -             Tra = strcat(Tra(1:Pos(i)-1),'&#13;&#10;',Tra(Pos(i)+1:length(Tra)));
1137 -         elseif ~strcmp(Tra(Pos(i)-3:Pos(i)),'&gt;') && ~strcmp(Tra(Pos(i)-3:Pos(i)),'&lt;') &&
1138 -             Tra = strcat(Tra(1:Pos(i)),'&#13;&#10;',Tra(Pos(i)+1:length(Tra)));
1139 -         end
1140 -     end
1141 - end
1142 - Tra = strcat('"',Tra);
1143 - Pos_IF = strfind(Tra,'IF');
1144 - Pos_WH = strfind(Tra,'WHILE');
1145 - Pos_FO = strfind(Tra,'FOR');
1146 - Pos_CA = strfind(Tra,'CASE');
1147 - Pos_END = strfind(Tra,'END_');
1148 - Pos = sort([Pos_IF Pos_WH Pos_FO Pos_CA Pos_END]);
1149 - H = 0;
1150 - L = 0;
```

Imagen 80. Traducción a « 4DIAC », introducción de saltos de línea en los algoritmos.

Para saber qué « END_ » pertenece a qué bucle o condición, al comenzar cada uno por una letra distinta, se utiliza ésto para diferenciarlos. La variable « j » indica el « END_ » del que se está intentando saber el bucle o condición, mientras que « H » indica la cantidad de « END_ » que ya hemos recorrido hasta el punto en cuestión del « array » « Pos », de manera que se cumple la ecuación de la línea 1155. En el caso de la condición « IF », por ejemplo, se busca la posición del « END_ » de ésta, la cual será « Pos_END(j) » (línea 1157).

Cuando « Pos_END(j) ==0 », el bucle o condición contiene otro bucle o condición, haciendo que no se detecte bien la posición de END_, sino que detecta la del bucle o condición. Por lo cual hay que buscar la siguiente en la que no se haya escrito. Para ello, Pos_END(k) no debe ser « 0 ». Una vez se ha encontrado, la variable « L » tendrá el valor « 1 » por lo cual no se volverá a entrar

en dicha condición (líneas 1159-1165). Seguidamente, se cambia la posición del « END_ » que ya ha sido completado por un « 0 » y se actualizan las posiciones de los « END_ » para evitar errores (líneas 1168-1175).

Dicho código se repite para todos los bucles y condiciones tal y como se muestra en las imágenes de la 81 a la 84.

```
1151 -  for i = length(Pos):-1:1
1152 -      if Tra(Pos(i))=='E'
1153 -          H = H + 1;
1154 -      elseif Tra(Pos(i))=='I' && Tra(Pos(i)-1)~='S'
1155 -          j = length(Pos_END)+1-H;
1156 -          if Pos_END(j)~=0
1157 -              Tra = strcat(Tra(1:Pos_END(j)+3),'IF',Tra(Pos_END(j)+4:length(Tra)));
1158 -          else
1159 -              for k = j:length(Pos_END)
1160 -                  if Pos_END(k)~=0 && L==0
1161 -                      Tra = strcat(Tra(1:Pos_END(k)+3),'IF',Tra(Pos_END(k)+4:length(Tra)));
1162 -                      j = k;
1163 -                      L = 1;
1164 -                  end
1165 -              end
1166 -          end
1167 -          L = 0;
1168 -          Pos_END(j)=0;
1169 -          Pos_END2 = strfind(Tra,'END_');
1170 -          for k = 1:length(Pos_END)
1171 -              if Pos_END(k)==0
1172 -                  Pos_END2(k) = 0;
1173 -              end
1174 -          end
1175 -          Pos_END = Pos_END2;
```

Imagen 81. Traducción a « 4DIAC », obtención de la pertenencia de los « END_ » a su bucle o condición.

```
1176 -      elseif Tra(Pos(i))=='W'
1177 -          j = length(Pos_END)+1-H;
1178 -          if Pos_END(j)~=0
1179 -              Tra = strcat(Tra(1:Pos_END(j)+3),'WHILE',Tra(Pos_END(j)+4:length(Tra)));
1180 -          else
1181 -              for k = j:length(Pos_END)
1182 -                  if Pos_END(k)~=0 && L==0
1183 -                      Tra = strcat(Tra(1:Pos_END(k)+3),'WHILE',Tra(Pos_END(k)+4:length(Tra)));
1184 -                      j = k;
1185 -                      L = 1;
1186 -                  end
1187 -              end
1188 -          end
1189 -          L = 0;
1190 -          Pos_END(j)=0;
1191 -          Pos_END2 = strfind(Tra,'END_');
1192 -          for k = 1:length(Pos_END)
1193 -              if Pos_END(k)==0
1194 -                  Pos_END2(k) = 0;
1195 -              end
1196 -          end
1197 -          Pos_END = Pos_END2;
```

Imagen 82. Traducción a « 4DIAC », obtención de la pertenencia de los « END_ » a su bucle o condición (2).

```
1198 -      elseif Tra(Pos(i))=='F'
1199 -          j = length(Pos_END)+1-H;
1200 -          if Pos_END(j)~=0
1201 -              Tra = strcat(Tra(1:Pos_END(j)+3),'FOR',Tra(Pos_END(j)+4:length(Tra)));
1202 -          else
1203 -              for k = j:length(Pos_END)
1204 -                  if Pos_END(k)~=0 && L==0
1205 -                      Tra = strcat(Tra(1:Pos_END(k)+3),'FOR',Tra(Pos_END(k)+4:length(Tra)));
1206 -                      j = k;
1207 -                      L = 1;
1208 -                  end
1209 -              end
1210 -          end
1211 -          L = 0;
1212 -          Pos_END(j)=0;
1213 -          Pos_END2 = strfind(Tra,'END_');
1214 -          for k = 1:length(Pos_END)
1215 -              if Pos_END(k)==0
1216 -                  Pos_END2(k) = 0;
1217 -              end
1218 -          end
1219 -          Pos_END = Pos_END2;
```

Imagen 83. Traducción a « 4DIAC », obtención de la pertenencia de los « END_ » a su bucle o condición (3).

```
1220 -      elseif Tra(Pos(i))=='C'
1221 -          j = length(Pos_END)+1-H;
1222 -          if Pos_END(j)~=0
1223 -              Tra = strcat(Tra(1:Pos_END(j)+3),'CASE',Tra(Pos_END(j)+4:length(Tra)));
1224 -          else
1225 -              for k = j:length(Pos_END)
1226 -                  if Pos_END(k)~=0 && L==0
1227 -                      Tra = strcat(Tra(1:Pos_END(k)+3),'CASE',Tra(Pos_END(k)+4:length(Tra)));
1228 -                      j = k;
1229 -                      L = 1;
1230 -                  end
1231 -              end
1232 -          end
1233 -          L = 0;
1234 -          Pos_END(j)=0;
1235 -          Pos_END2 = strfind(Tra,'END_');
1236 -          for k = 1:length(Pos_END)
1237 -              if Pos_END(k)==0
1238 -                  Pos_END2(k) = 0;
1239 -              end
1240 -          end
1241 -          Pos_END = Pos_END2;
1242 -      end
1243 - end
```

Imagen 84. Traducción a « 4DIAC », obtención de la pertenencia de los « END_ » a su bucle o condición (4).

Una vez tenemos los « END_ » con el nombre de su bucle o condición, se vuelven a guardar las posiciones de los bucles o condiciones. Ésta vez, para encontrar los puntos dentro de éstos donde se debe incluir tabuladores « %#9 ; », incluyendo también las posiciones de los saltos de línea (línea 1249). El bucle de la línea 1252, recorre el « array » de posiciones « Pos » y va calculando la cantidad de bucles y condicciones en las que se encuentra la variable « Pos », cuyo valor se guarda en la variable « H ». Por lo cual, « H » indicará la cantidad de veces que ha de introducirse « %#9 ; ». En caso de encontrar una posición que corresponda a un « END_ » (línea 1253), se incrementará « H » en uno. Mientras que si se encuentra una posición que corresponda a algún bucle o condición, se decrementará « H » en uno. Lo cual quiere decir que se deben evitar las posiciones de; los « ELSIF », los saltos de línea «
 » y los « END_ » (línea 1255). Seguidamente, se introducen los tabuladores necesarios para cada posición (líneas 1259-1263) y se corrigen los que no deben tener (líneas 1266-1267).

```
1244 -    Pos_IF = strfind(Tra,'IF');
1245 -    Pos_WH = strfind(Tra,'WHILE');
1246 -    Pos_FO = strfind(Tra,'FOR');
1247 -    Pos_CA = strfind(Tra,'CASE');
1248 -    Pos_END = strfind(Tra,'END_');
1249 -    Pos_SL = strfind(Tra,'&#10;');
1250 -    Pos = sort([Pos_IF Pos_WH Pos_FO Pos_CA Pos_END Pos_SL]);
1251 -    H = 0;
1252 -    for i = length(Pos):-1:1
1253 -        if ~isempty(strfind(Pos_END,Pos(i)))
1254 -            H = H + 1;
1255 -        elseif Tra(Pos(i)-1)~='S' && Tra(Pos(i)-1)~='_' && ~isequal(Tra(Pos(i):Pos(i)+4),'&#10;')
1256 -            H = H - 1;
1257 -        end
1258
1259 -        if isequal(Tra(Pos(i):Pos(i)+4),'&#10;')
1260 -            for j = 1:H
1261 -                Tra = strcat(Tra(1:Pos(i)+4),'&#9;',Tra(Pos(i)+5:length(Tra)));
1262 -            end
1263 -        end
1264 -    end
1265 -    Tra = Tra(2:length(Tra));
1266 -    Tra = strrep(Tra,'&#9;ELS','ELS');
1267 -    Tra = strrep(Tra,'&#9;END_','END_');
```

Imagen 85. Traducción a « 4DIAC », introducción de los tabuladores en el algoritmo.

Una vez tenemos el algoritmo traducido, se comprueba si los últimos caracteres corresponden a un salto de línea o a un salto de línea con tabulador. En cuyo caso se elimina ésta última instrucción y se procede a introducir el algoritmo en el « Archivo_destino » (línea 1275). Seguidamente, se cierra la declaración (línea 1280) y se repite el mismo código que empieza en la línea 957 para todos los algoritmos del programa.

Una vez está todo declarado en el « Archivo_destino », se cierran tanto el « BasicFB » como el « FBType » finalizando así el programa.

```
1269 -                    if isequal(Tra(length(Tra)-9:length(Tra)),'&#13;&#10;')
1270 -                        Tra = Tra(1:length(Tra)-10);
1271 -                    elseif isequal(Tra(length(Tra)-13:length(Tra)),'&#13;&#10;&#9;')
1272 -                        Tra = Tra(1:length(Tra)-14);
1273 -                    end
1274 -                    Tra = strrep(Tra,'"',' ');
1275 -                    fprintf(FID_d,Tra);
1276 -                    fprintf(FID_d,'"/>');
1277 -                    fprintf(FID_d,'\n');
1278 -                    fprintf(FID_d,'  ');
1279 -                    fprintf(FID_d,'  ');
1280 -                    fprintf(FID_d,'</Algorithm>');
1281 -                    fprintf(FID_d,'\n');
1282 -                end
1283 -                leer = fgetl(FID_f);
1284 -            end
1285 -        end
1286 -        fclose(FID_f);
1287
1288 -        fprintf(FID_d,'  ');
1289 -        fprintf(FID_d,'</BasicFB>');
1290 -        fprintf(FID_d,'\n');
1291 -        fprintf(FID_d,'</FBType>');
1292 -    end
1293 -    fclose(FID_d);
```

Imagen 86. Traducción a « 4DIAC », declaración de algoritmos y cierre del programa.

3.4. Metodología para la traducción de 4DIAC a Matlab/Simulink

Para realizar la traducción a Matlab/Simulink se utilizará la función « FB_4DIAC2SLK( ) », la cual recibe el nombre del archivo que se desea traducir « Archivo_destino » y devuelve el nombre de otro archivo « Archivo_destino » que se genera con el resultado de la traducción del anterior tal y como se muestra en la « Imagen 87 ».

Así mismo, en ésta sección se explicará cómo tratar la información que se quiere traducir. Al igual que en el apartado anterior, para introducir el código del archivo de Simulink « Archivo_destino », se precisa que el programa siga una estructura similar a éste para facilitar así la traducción. De ésta manera, se divide la traducción en diversos puntos en los cuales se repetirá un mismo procedimiento para obtener la traducción:

1. Lo primero es buscar en la estructura de « 4DIAC » el punto o los puntos donde se encuentra la información del « Archivo_fuente » que necesitamos para realizar la traducción de cierto punto de la estructura de « Matlab/Simulink » al « Archivo_destino ».
2. Una vez tenemos localizada la información, hay que extraerla del « Archivo_fuente » y guardarla de una forma ordenada en una o varias variables de « Matlab ».
3. Con la información extraída, se traduce y/o prepara ésta para el código y la estructura de « Matlab/Simulink ».
4. Una vez tenemos ésto, se vuelca la nueva información con el tipo de estructura de « Matlab/Simulink » en el « Archivo_destino »

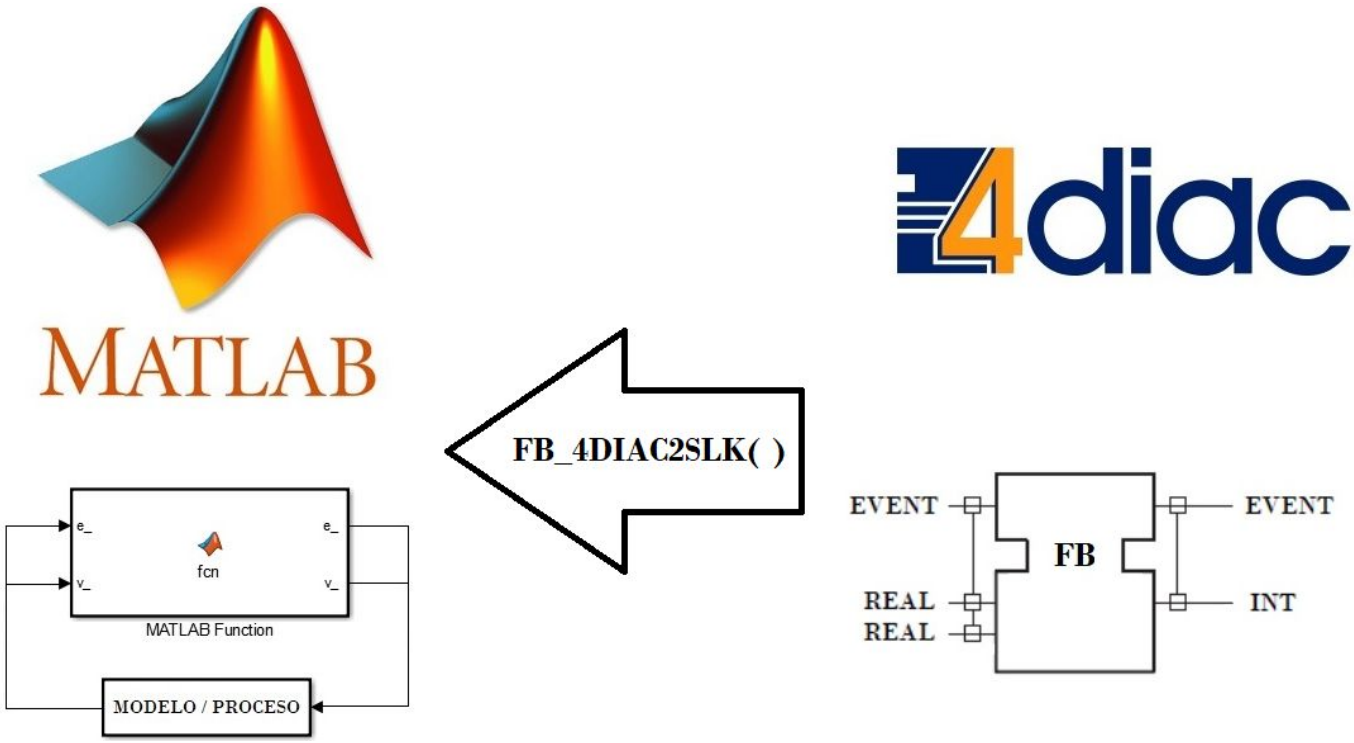

Imagen 87. Esquema de traducción a « Simulink ».

Así mismo, el archivo de 4DIAC « Archivo_fuente » será necesario abrirlo varias veces por cada punto de la estructura de Simulink en la mayoría de los casos, con el fin de extraer toda la información necesaria tal y como se muestra en la « Tabla 4 ».

Apartados de la traducción Matlab/Simulink - 4DIAC	**Puntos de la estructura de « 4DIAC » utilizados**	**Puntos de la estructura de « Simulink » que se declaran**
1. Declaración de Eventos y Variables del « FB »	<EventInputs> <EventOutputs> <InputVars> <OutputVars>	%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE
2. Declaración de Variables internas	<InternalVars>	%2-DECLARACION DE VARIABLES INTERNAS
3. Declaración del estado inicial de todas las variables	<ECState>	%3-DECLARACION DEL ESTADO INICIAL
4. Actualización de las Variables de entrada asociadas a Eventos « WITH »	<EventInputs>	%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)
5. Diagrama de control de ejecución « ECC »	<ECTransition>	%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)
6. Ejecución de algoritmos	<ECState> <ECAction>	%6-EJECUCION DE ALGORITMOS
7. Declaración de algoritmos	<Algorithm>	%7-DECLARACION DE ALGORITMOS
8. Ejecución de Eventos de salida	<ECState> <ECAction>	%8-EJECUCION DE EVENTOS DE SALIDA
9. Actualización de Variables de salida asociadas a Eventos « WITH »	<EventOutputs>	%9-EJECUCION DE VARIABLES DE SALIDA (WITH)
10. Preparación de Eventos de salida y Variables de salida antes de enviarse	<EventInputs> <EventOutputs> <OutputVars>	%10-FINAL

Tabla 4. Puntos de ambas estructuras utilizados para la traducción a « Simulink » de cada apartado.

3.4.1. Declaración de Eventos y Variables del « FB »

Las primeras líneas de código sirven, al igual que con el anterior código, para declarar el código que se va a escribir como una función y obtener los nombres de los dos archivos que se can a utilizar, los cuales estarán guardados en las variables « Archivo_fuente » y « Archivo_destino » tal y como se muestra en la « Imagen 88 ».

```
2    function [Archivo_destino] = FB_4DIAC2SLK (Archivo_fuente)
3 -  Archivo_destino = strcat('Slk_', Archivo_fuente(5:length(Archivo_fuente)-4),'.txt');
```

Imagen 88. Traducción a Simulink, obtención de los nombres de los archivos.

Una vez obtenidos los nombres de los archivos, se abre el « Archivo_destino » y se introduce el nombre del primer punto, tal y como se muestra en la línea 12.

```
 6 -    [FID_d,MESSAGE]= fopen(Archivo_destino, 'wt');
 7 -    if (FID_d==-1)
 8 -        disp(strcat('Error al intentar leer el archivo: ', Archivo_destino, ',', MESSAGE))
 9 -    else
10 -        leer = '';
11
12 -        fprintf(FID_d,'%%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE');
13 -        fprintf(FID_d,'\n');
14 -        EI = '';
15 -        EO = '';
16 -        IV = '';
17 -        OV = '';
18 -        IV_Type = '';
19 -        OV_Type = '';
20 -        C = '';
```

Imagen 89. Traducción a Simulink, declaración de Eventos y Variables en el « FB ».

Para poder declarar los Eventos y las Variables como entradas y salidas del Bloque de función de Simulink, se precisa saber los nombres de éstos. Por lo cual, se abre el « Archivo_fuente » y se buscan las instrucciones « <EventInputs> », « <EventOutputs> », « <InputVars> » y « <OutputVars> ». Después de las cuales se encuentra la información que buscamos. Por cada línea antes de llegar al final de la declaración (marcado por un « / » como por ejemplo: « </EventInputs> »), hay un Evento o Variable con el nombre declarado seguido de la sentencia « Name=" ». De manera que buscando el código desde dicha sentencia a el siguiente « " » se encontrará el nombre. Una vez hecho ésto, se añade « e_ » o « v_ » según corresponda y se guarda en las variables « EI », « EO », « IV » o « OV » según corresponda.

Una diferencia a la hora de declarar los Eventos y las Variables, es que las Variables tienen un tipo de variable, mientras que los Eventos siempre son del tipo « Event ». Por lo cual, en el caso de las Variables también se debe extraer dicha información, la cual se encuentra seguida de la sentencia « Type=" ». Sin embargo, dicha información no es necesaria para éste apartado, pero si para apartados posteriores.

```
[FID_f,MESSAGE] = fopen(Archivo_fuente, 'r');
if (FID_f==-1)
    disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
else
    while isempty(strfind(leer,'<EventInputs>'))
        leer = fgetl(FID_f);
    end
    leer = fgetl(FID_f);

    while isempty(strfind(leer,'</EventInputs>'))
        if ~isempty(strfind(leer,'<Event'))

            C = leer(strfind(leer,'Name="'):length(leer));
            C = C(length('Name="'):length(C));
            C = strtok(C,'"');

            if isempty(EI)
                EI = strcat('e_',C);
            else
                EI = strcat(EI,',e_',C);
            end
        end
        leer = fgetl(FID_f);
    end
```

Imagen 90. Traducción a Simulink, obtención de los Eventos de entrada.

```
    while isempty(strfind(leer,'<EventOutputs>'))
        leer = fgetl(FID_f);
    end
    leer = fgetl(FID_f);

    while isempty(strfind(leer,'</EventOutputs>'))
        if ~isempty(strfind(leer,'<Event'))

            C = leer(strfind(leer,'Name="'):length(leer));
            C = C(length('Name="'):length(C));
            C = strtok(C,'"');

            if isempty(EO)
                EO = strcat('e_',C);
            else
                EO = strcat(EO,',e_',C);
            end
        end
        leer = fgetl(FID_f);
    end
```

Imagen 91. Traducción a Simulink, obtención de los Eventos de salida.

```
67 -        while isempty(strfind(leer,'</InputVars>'))
68 -            if ~isempty(strfind(leer,'<VarDeclaration'))
69 
70 -                C = leer(strfind(leer,'Name="'):length(leer));
71 -                C = C(length('Name="'):length(C));
72 -                C = strtok(C,'"');
73 
74 -                Cl = leer(strfind(leer,'Type="'):length(leer));
75 -                Cl = Cl(length('Type="'):length(Cl));
76 -                Cl = strtok(Cl,'"');
77 
78 -                IV_Type = strcat(IV_Type,'"',Cl);
79 -                if isempty(IV)
80 -                    IV = strcat('v_',C);
81 -                else
82 -                    IV = strcat(IV,',v_',C);
83 -                end
84 -            end
85 -            leer = fgetl(FID_f);
86 -        end
87 -        IV_Type = strrep(IV_Type,'"',' ');
```

Imagen 92. Traducción a Simulink, obtención de los Variables de entrada.

```
89 -        while isempty(strfind(leer,'</OutputVars>'))
90 -            if ~isempty(strfind(leer,'<VarDeclaration'))
91 
92 -                C = leer(strfind(leer,'Name="'):length(leer));
93 -                C = C(length('Name="'):length(C));
94 -                C = strtok(C,'"');
95 
96 -                Cl = leer(strfind(leer,'Type="'):length(leer));
97 -                Cl = Cl(length('Type="'):length(Cl));
98 -                Cl = strtok(Cl,'"');
99 
00 -                OV_Type = strcat(OV_Type,'"',Cl);
01 -                if isempty(OV)
02 -                    OV = strcat('v_',C);
03 -                else
04 -                    OV = strcat(OV,',v_',C);
05 -                end
06 -            end
07 -            leer = fgetl(FID_f);
08 -        end
09 -        OV_Type = strrep(OV_Type,'"',' ');
```

Imagen 93. Traducción a Simulink, obtención de los Variables de salida.

Con los nombres de los Estados y las Variables ya guardados, solamente queda introducirlos en el programa tal y como se muestra en la línea 111.

```
111 -        leer = strcat('function"[',EO,',',OV,']"="fcn"(',EI,',',IV,')');
112 -        leer = strrep(leer,'"',' ');
113 -        fprintf(FID_d,leer);
114 -        fprintf(FID_d,'\n');
```

Imagen 94. Traducción a Simulink, obtención de los Eventos de salida.

3.4.2. Declaración de Variables internas

Al igual que en el apartado anterior, la declaración de las Variables internas se precisa el nombre de éstas. Sin embargo, se obtendrá también la información del tipo de variable y el valor inicial para posteriores apartados. Para ello, se utilizan las mismas instrucciones que en el apartado anterior.

```
117 -           fprintf(FID_d,'\n');
118 -           fprintf(FID_d,'%%2-DECLARACION DE VARIABLES INTERNAS');
119 -           fprintf(FID_d,'\n');
120 -           VI = '';
121 -           VI_Type = '';
122 -           VI_Inic = '';
123 -           Etp = '';
124
125 -           while isempty(strfind(leer,'<InternalVars>'))
126 -               leer = fgetl(FID_f);
127 -           end
128 -           leer = fgetl(FID_f);
129
130 -           while isempty(strfind(leer,'</InternalVars>'))
131 -               if ~isempty(strfind(leer,'<VarDeclaration'))
132
133 -                   C = leer(strfind(leer,'Name="'):length(leer));
134 -                   C = C(length('Name="'):length(C));
135 -                   C = strtok(C,'"');
136
137 -                   C1 = leer(strfind(leer,'Type="'):length(leer));
138 -                   C1 = C1(length('Type="'):length(C1));
139 -                   C1 = strtok(C1,'"');
```

Imagen 95. Traducción a Simulink, obtención de las Variables internas.

Además, ésta vez se utilizará la sentencia « InitialValue=" » para obtener el valor inicial de las Variables. Una vez se ha obtenido la información necesaria, se guarda en las variables « VI », « VI_Type » y « VI_Inic » dependiendo de si se refiere al nombre, al tipo o al valor inicial de la Variable respectivamente. Por último, se declaran éstas Variables tal y como se indica en la línea 159.

```
141 -                   C2 = leer(strfind(leer,'InitialValue="'):length(leer));
142 -                   C2 = C2(length('InitialValue="'):length(C2));
143 -                   if isempty(strfind(C2,'""'))
144 -                       C2 = strtok(C2,'"');
145 -                       C2 = strrep(C2,' ','_');
146 -                   else
147 -                       C2 = '_';
148 -                   end
149
150 -                   VI_Inic = strcat(VI_Inic,'"',C2);
151 -                   VI_Type = strcat(VI_Type,'"',C1);
152 -                   VI = strcat(VI,'"',C);
153 -               end
154 -               leer = fgetl(FID_f);
155 -           end
156 -           VI_Inic = strrep(VI_Inic,'"',' ');
157 -           VI_Type = strrep(VI_Type,'"',' ');
158
159 -           leer = strcat('persistent',VI,';');
160 -           leer = strrep(leer,'"',' ');
161 -           fprintf(FID_d,leer);
162 -           fprintf(FID_d,'\n');
```

Imagen 96. Traducción a Simulink, obtención y declaración de las Variables internas.

3.4.3. Declaración del estado inicial de todas las variables.

Éste apartado se compone de dos partes. En la primera se declaran todas las variables que se van a utilizar en el programa y aún no se han declarado en los apartados anteriores, como son las copias de los Estados y las Variables o los « array » donde se guardan los estados de los Estados (« E » y « CE »). La mayoría de ésta información ya la tenemos de los apartados anteriores, pero es necesario saber la cantidad de Estados que tiene el « ECC » para poder declarar « E » y « CE ». Sin embargo, ya que se van a necesitar los nombres de los Estados más adelante, se extraen éstos, se guardan y se ordenan en la variable « E ».

```
165 -           while isempty(strfind(leer,'<ECC>'))
166 -               leer = fgetl(FID_f);
167 -           end
168 -           while isempty(strfind(leer,'</ECC>'))
169 -               if ~isempty(strfind(leer,'<ECState'))
170 -
171 -                   C = leer(strfind(leer,'Name="'):length(leer));
172 -                   C = C(length('Name="'):length(C));
173 -                   C = strtok(C,'"');
174 -
175 -                   Etp = strcat(Etp,'"',C);
176 -
177 -               end
178 -               leer = fgetl(FID_f);
179 -           end
180 -           Etp = strrep(Etp,'"',' ');
181 -           E = textscan(Etp,'%s');
182 -       end
```

Imagen 97. Traducción a Simulink, obtención de los nombres de los Estados.

Seguidamente, se adaptan los nombres a la manera en la que se deben declarar y se declaran tal y como se muestra en la línea 200. Una vez declarada la primera parte de éste apartado, se empieza a declarar la segunda dentro de la condición « if » (líneas 205-208).

```
185 -       fprintf(FID_d,'\n');
186 -       fprintf(FID_d,'%%%3-DECLARACION DEL ESTADO INICIAL');
187 -       fprintf(FID_d,'\n');
188 -       Etp = '';
189 -       C1 = '';
190 -       C2 = '';
191
192 -       EI = strrep(EI,',e_','"');
193 -       EI = EI(3:length(EI));
194 -       EO = strrep(EO,',e_','"');
195 -       EO = EO(3:length(EO));
196 -       IV = strrep(IV,',v_','"');
197 -       IV = strcat('"',IV(3:length(IV)));
198 -       OV = strrep(OV,',v_','"');
199 -       OV = strcat('"',OV(3:length(OV)));
200 -       leer = strcat('persistent"',EO,OV,'"',EI,IV,'"E"CE"',strrep(OV,'"','"prev_'),';');
201 -       leer = strrep(leer,'"',' ');
202 -       fprintf(FID_d,leer);
203 -       fprintf(FID_d,'\n');
204 -       fprintf(FID_d,'\n');
205 -       fprintf(FID_d,'if isempty(E)');
206 -       fprintf(FID_d,'\n');
207 -       fprintf(FID_d,'\t');
208 -       fprintf(FID_d,'E=[');
```

Imagen 98. Traducción a Simulink, declaración de las copias de Estados y Variables.

Lo primero en declarar son los estados iniciales de los Estados, los cuales son todos desactivados excepto el Estado inicial « START » (líneas 209-216). A ésto le sigue la declaración de los estados iniciales de los Eventos de salida, los cuales siempre se inician a « 0 » tal y como se muestra en la línea 224.

```
209 -       for i=1:length(E{1})
210 -           if isequal(char(E{1}(i)),'START')
211 -               fprintf(FID_d,' 1');
212 -           else
213 -               fprintf(FID_d,' 0');
214 -           end
215 -       end
216 -       fprintf(FID_d,'];');
217 -
218 -
219 -       C2 = strrep(EO,'"',' ');
220 -       while ~isempty(C2)
221 -           [C1,C2] = strtok(C2);
222 -           fprintf(FID_d,'\n');
223 -           fprintf(FID_d,'\t');
224 -           fprintf(FID_d,strcat(C1,'=0;'));
225 -           if isempty(strrep(C2,' ',''))
226 -               C2 = '';
227 -           end
228 -       end
```

Imagen 99. Traducción a Simulink, declaración de los estados iniciales de los Estados y de los Eventos de salida.

Al contrario que con los Eventos de salida, las Variables de salida deben ser declaradas según su tipo, por lo cual se busca el tipo de variable y se introduce en la sentencia de la declaración.

```
229 -       C2 = strrep(OV,'"',' ');
230 -       while ~isempty(C2)
231 -           [C,C2] = strtok(C2);
232 -           fprintf(FID_d,'\n');
233 -           fprintf(FID_d,'\t');
234 -           [C1,OV_Type] = strtok(OV_Type);
235 -           C1 = strrep(C1,' ','');
236 -           if isequal(C1,'REAL')
237 -               fprintf(FID_d,strcat(C,'=0.0;'));
238 -               fprintf(FID_d,'\n');
239 -               fprintf(FID_d,'\t');
240 -               fprintf(FID_d,strcat('prev_',C,'=0.0;'));
241 -           elseif isequal(C1,'INT') || isequal(C1,'BOOL')
242 -               fprintf(FID_d,strcat(C,'=0;'));
243 -               fprintf(FID_d,'\n');
244 -               fprintf(FID_d,'\t');
245 -               fprintf(FID_d,strcat('prev_',C,'=0;'));
246 -           elseif isequal(C1,'STRING')
247 -               fprintf(FID_d,strcat(C,''''';'));
248 -               fprintf(FID_d,'\n');
249 -               fprintf(FID_d,'\t');
250 -               fprintf(FID_d,strcat('prev_',C,''''';'));
```

Imagen 100. Traducción a Simulink, declaración de los valores iniciales de las Variables de salida.

```
251 -           elseif isequal(Cl,'LREAL')
252 -               fprintf(FID_d,strcat(C,'=double(0);'));
253 -               fprintf(FID_d,'\n');
254 -               fprintf(FID_d,'\t');
255 -               fprintf(FID_d,strcat('prev_',C,'=double(0);'));
256 -           elseif isequal(Cl,'SINT')
257 -               fprintf(FID_d,strcat(C,'=int8(0);'));
258 -               fprintf(FID_d,'\n');
259 -               fprintf(FID_d,'\t');
260 -               fprintf(FID_d,strcat('prev_',C,'=int8(0);'));
261 -           elseif isequal(Cl,'DINT')
262 -               fprintf(FID_d,strcat(C,'=int32(0);'));
263 -               fprintf(FID_d,'\n');
264 -               fprintf(FID_d,'\t');
265 -               fprintf(FID_d,strcat('prev_',C,'=int32(0);'));
266 -           elseif isequal(Cl,'LINT')
267 -               fprintf(FID_d,strcat(C,'=int64(0);'));
268 -               fprintf(FID_d,'\n');
269 -               fprintf(FID_d,'\t');
270 -               fprintf(FID_d,strcat('prev_',C,'=int64(0);'));
271 -           elseif isequal(Cl,'USINT')
272 -               fprintf(FID_d,strcat(C,'=uint8(0);'));
273 -               fprintf(FID_d,'\n');
274 -               fprintf(FID_d,'\t');
275 -               fprintf(FID_d,strcat('prev_',C,'=uint8(0);'));
```

Imagen 101. Traducción a Simulink, declaración de los valores iniciales de las Variables de salida (2).

```
276 -           elseif isequal(Cl,'UINT')
277 -               fprintf(FID_d,strcat(C,'=uint16(0);'));
278 -               fprintf(FID_d,'\n');
279 -               fprintf(FID_d,'\t');
280 -               fprintf(FID_d,strcat('prev_',C,'=uint16(0);'));
281 -           elseif isequal(Cl,'UDINT')
282 -               fprintf(FID_d,strcat(C,'=uint32(0);'));
283 -               fprintf(FID_d,'\n');
284 -               fprintf(FID_d,'\t');
285 -               fprintf(FID_d,strcat('prev_',C,'=uint32(0);'));
286 -           elseif isequal(Cl,'ULINT')
287 -               fprintf(FID_d,strcat(C,'=uint64(0);'));
288 -               fprintf(FID_d,'\n');
289 -               fprintf(FID_d,'\t');
290 -               fprintf(FID_d,strcat('prev_',C,'=uint64(0);'));
291 -           else
292 -               disp(strcat('no se ha podido leer bien: ',Cl,' como tipo'));
293 -           end
294 -           if isempty(strrep(C2,' ',''))
295 -               C2 = '';
296 -           end
297 -       end
```

Imagen 102. Traducción a Simulink, declaración de los valores iniciales de las Variables de salida (3).

Seguidamente se declaran los Eventos de entrada, de la misma manera que los Eventos de salida, y las Variables de entrada, de la misma manera que las Variables de salida.

```
298 -       C2 = strrep(EI,'"',' ');
299 -       while ~isempty(C2)
300 -           [C1,C2] = strtok(C2);
301 -           fprintf(FID_d,'\n');
302 -           fprintf(FID_d,'\t');
303 -           fprintf(FID_d,strcat(C1,'=0;'));
304 -           if isempty(strrep(C2,' ',''))
305 -               C2 = '';
306 -           end
307 -       end
```

Imagen 103. Traducción a Simulink, declaración de los estados iniciales de los Eventos de entrada.

```
308 -       C2 = strrep(IV,'"',' ');
309 -       while ~isempty(C2)
310 -           [C1,C2] = strtok(C2);
311 -           fprintf(FID_d,'\n');
312 -           fprintf(FID_d,'\t');
313 -           fprintf(FID_d,strcat(C1,'='));
314 -           [C1,IV_Type] = strtok(IV_Type);
315 -           C1 = strrep(C1,' ','');
316 -           if isequal(C1,'REAL')
317 -               fprintf(FID_d,'0.0;');
318 -           elseif isequal(C1,'INT') || isequal(C1,'BOOL')
319 -               fprintf(FID_d,'0;');
320 -           elseif isequal(C1,'STRING')
321 -               fprintf(FID_d,''''';');
322 -           elseif isequal(C1,'REAL')
323 -               fprintf(FID_d,'double(0);');
324 -           elseif isequal(C1,'SINT')
325 -               fprintf(FID_d,'int8(0);');
326 -           elseif isequal(C1,'DINT')
327 -               fprintf(FID_d,'int32(0);');
328 -           elseif isequal(C1,'LINT')
329 -               fprintf(FID d,'int64(0);');
```

Imagen 104. Traducción a Simulink, declaración de los valores iniciales de las Variables de entrada.

```
330 -           elseif isequal(C1,'USINT')
331 -               fprintf(FID_d,'uint8(0);');
332 -           elseif isequal(C1,'UINT')
333 -               fprintf(FID_d,'uint16(0);');
334 -           elseif isequal(C1,'UDINT')
335 -               fprintf(FID_d,'uint32(0);');
336 -           elseif isequal(C1,'ULINT')
337 -               fprintf(FID_d,'uint64(0);');
338 -           else
339 -               disp(strcat('no se ha podido leer bien: ',C1,' como tipo'));
340 -           end
341 -           if isempty(strrep(C2,' ',''))
342 -               C2 = '';
343 -           end
344 -       end
```

Imagen 105. Traducción a Simulink, declaración de los valores iniciales de las Variables de entrada (2).

Por último, se extraen los valores de las Variables internas y se declaran dependiendo de su tipo tal y como se muestra en las imágenes 106 y 107. Una vez hecho ésto, se cierra el bucle y se finaliza con la declaración de estados iniciales.

```
345 -       C2 = strrep(VI,'"',' ');
346 -       while ~isempty(C2)
347 -           [C1,C2] = strtok(C2);
348 -           fprintf(FID_d,'\n');
349 -           fprintf(FID_d,'\t');
350 -           fprintf(FID_d,strcat(C1,'='));
351 -           [C1,VI_Type] = strtok(VI_Type);
352 -           [C,VI_Inic] = strtok(VI_Inic);
353 -           C1 = strrep(C1,' ','');
354 -           C = strrep(C,' ','');
355 -           if isequal(C1,'REAL')
356 -               fprintf(FID_d,strcat(C,';'));
357 -           elseif isequal(C1,'INT') || isequal(C1,'BOOL')
358 -               fprintf(FID_d,strcat(C,';'));
359 -           elseif isequal(C1,'STRING')
360 -               fprintf(FID_d,strcat('''',C,''';'));
361 -           elseif isequal(C1,'REAL')
362 -               fprintf(FID_d,strcat('double(',C,');'));
363 -           elseif isequal(C1,'SINT')
364 -               fprintf(FID_d,strcat('int8(',C,');'));
```

Imagen 106. Traducción a Simulink, declaración de los estados iniciales de las Variables internas.

```
365 -           elseif isequal(C1,'DINT')
366 -               fprintf(FID_d,strcat('int32(',C,');'));
367 -           elseif isequal(C1,'LINT')
368 -               fprintf(FID_d,strcat('int64(',C,');'));
369 -           elseif isequal(C1,'USINT')
370 -               fprintf(FID_d,strcat('uint8(',C,');'));
371 -           elseif isequal(C1,'UINT')
372 -               fprintf(FID_d,strcat('uint16(',C,');'));
373 -           elseif isequal(C1,'UDINT')
374 -               fprintf(FID_d,strcat('uint32(',C,');'));
375 -           elseif isequal(C1,'ULINT')
376 -               fprintf(FID_d,strcat('uint64(',C,');'));
377 -           else
378 -               disp(strcat('no se ha podido leer bien: ',C1,' como tipo'));
379 -           end
380 -           if isempty(strrep(C2,' ',''))
381 -               C2 = '';
382 -           end
383 -       end
384 -       fprintf(FID_d,'\n');
385 -       fprintf(FID_d,'end');
386 -       fprintf(FID_d,'\n');
```

Imagen 107. Traducción a Simulink, declaración de los estados iniciales de las Variables internas (2).

3.4.4. Actualización de las Variables de entrada asociadas a Eventos « WITH »

La actualización de Variables se realiza para cada Evento. Lo cual está declarado de una manera similar en el « Archivo_fuente », por lo cual se busca el punto donde se declaran (línea 397), se obtiene el nombre del Evento de entrada (líneas 405-408) se comienza a declarar la condición en la que se actualizarán las Variables de entrada. A parte de actualizar los Valores de las Variables de entrada, también se marca igualando el valor de la copia del Evento de entrada en cuestión a « 1 » que éste se encuentra activo (línea 415), ya que el « ECC », el cual se declarará más adelante, solamente puede modificar variables internas del bloque. Por último, se declaran todas las Variables de entrada asociadas (líneas 420-423 y 426), se cierra la condición y se repite el mismo procedimiento para todos los Eventos de entrada.

```
389 -       fprintf(FID_d,'\n');
390 -       fprintf(FID_d,'%%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)');
391 -       fprintf(FID_d,'\n');
392
393 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r'); %open file for reading
394 -       if (FID_f==-1)
395 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
396 -       else
397 -           while isempty(strfind(leer,'<EventInputs>'))
398 -               leer = fgetl(FID_f);
399 -           end
400 -           leer = fgetl(FID_f);
401
402 -           while isempty(strfind(leer,'</EventInputs>'))
403 -               if ~isempty(strfind(leer,'<Event')) && isempty(strfind(leer,'/>'))
404
405 -                   C = leer(strfind(leer,'Name="'):length(leer));
406 -                   C = C(length('Name="'):length(C));
407 -                   C = strtok(C,'"');
408 -                   leer = C;
409
410 -                   C = strcat('if"e_',leer);
411 -                   C = strrep(C,'"',' ');
412 -                   fprintf(FID_d,C);
413 -                   fprintf(FID_d,'\n');
414 -                   fprintf(FID_d,'\t');
415 -                   fprintf(FID_d,strcat(leer,'=1;'));
416 -                   fprintf(FID_d,'\n');
```

Imagen 108. Traducción a Simulink, declaración de los « WITH » de los Eventos de entrada.

```
417 -                   while isempty(strfind(leer,'</Event>'))
418 -                       if ~isempty(strfind(leer,'<With'))
419
420 -                           C = leer(strfind(leer,'Var="'):length(leer));
421 -                           C = C(length('Var="'):length(C));
422 -                           C = strtok(C,'"');
423 -                           leer = C;
424
425 -                           fprintf(FID_d,'\t');
426 -                           fprintf(FID_d,strcat(leer,'=v_',leer,';'));
427 -                           fprintf(FID_d,'\n');
428 -                       end
429 -                       leer = fgetl(FID_f);
430 -                   end
431 -                   fprintf(FID_d,'end');
432 -                   fprintf(FID_d,'\n');
433 -               end
434 -               leer = fgetl(FID_f);
435 -           end
```

Imagen 109. Traducción a Simulink, declaración de los « WITH » de los Eventos de entrada (2).

3.4.5. Diagrama de control de ejecución « ECC »

Antes de comenzar con la declaración del « ECC », se inicializan a « 0 » los valores de las segundas copias de los Eventos de salida (línea 446) y de los estados de la variable « CE » (líneas 452-456). Seguidamente se abre el bucle donde se va a declarar el « ECC », el cual comienza igualando la variable copia de Estados « CE » a « E » (línea 461).

```
438 -            fprintf(FID_d,'\n');
439 -            fprintf(FID_d,'%%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)');
440 -            fprintf(FID_d,'\n');
441 -            k = 0;
442
443 -            C2 = strrep(EO,'"',' ');
444 -            while ~isempty(C2)
445 -                [C,C2] = strtok(C2);
446 -                fprintf(FID_d,strcat('prev_',C,'=0;'));
447 -                fprintf(FID_d,'\n');
448 -                if isempty(strrep(C2,' ',''))
449 -                    C2 = '';
450 -                end
451 -            end
452 -            fprintf(FID_d,'CE=[');
453 -            for i=1:length(E{1})
454 -                fprintf(FID_d,' 0');
455 -            end
456 -            fprintf(FID_d,'];');
457 -            fprintf(FID_d,'\n');
458 -            fprintf(FID_d,'while ~isequal(CE,E)');
459 -            fprintf(FID_d,'\n');
460 -            fprintf(FID_d,'\t');
461 -            fprintf(FID_d,'CE=E;');
462 -            fprintf(FID_d,'\n');
463 -            fprintf(FID_d,'\t');
```

Imagen 110. Traducción a Simulink, inicialización de variables para el « ECC ».

Al igual que con la traducción a « 4DIAC », para poder declarar el « ECC » las transiciones deben ser traducidas, ésta vez a lenguaje de Matlab. Por lo cual, lo primero que se debe hacer es extraer el código de la transición (líneas 472-474) y lo siguiente es ir traduciendo ordenadamente para evitar problemas, al igual que en el apartado « **3.3.7. Obtención de los datos de los Estados y las transiciones del ECC.** » (imágenes de la 60 a la 63) de la traducción a « 4DIAC ». Una vez tenemos la transición traducida, se declara como condición de transición en una condición « if » (línea 512).

```
464 -         while isempty(strfind(leer,'<ECC>'))
465 -             leer = fgetl(FID_f);
466 -         end
467 -         leer = fgetl(FID_f);
468 
469 -         while isempty(strfind(leer,'</ECC>'))
470 -             if ~isempty(strfind(leer,'<ECTransition'))
471 
472 -                 C = leer(strfind(leer,'Condition="'):length(leer));
473 -                 C = C(length('Condition="'):length(C));
474 -                 C = strtok(C,'"');
475 
476 -                 Pos = strfind(C,'=');
477 -                 H = length(Pos);
478 -                 if ~isempty(Pos)
479 -                     for i=H:-1:1
480 -                         if C(Pos(i)-1) ~= ':'
481 -                             C = strcat(C(1:Pos(i)-1),'==',C(Pos(i)+1:length(C)));
482 -                         end
483 -                     end
484 -                 end
```

Imagen 111. Traducción a Simulink, obtención del código de las transiciones del « ECC ».

```
485 -                 C = strrep(C,':=','=');
486 -                 C = strrep(C,'<>','~=');
487 -                 C = strrep(C,'&gt;','>');
488 -                 C = strrep(C,'&ge;','>=');
489 -                 C = strrep(C,'&lt;','<');
490 -                 C = strrep(C,'&le;','<=');
491 -                 C = strrep(C,'**','^');
492 -                 C = strrep(C,'OR','||');
493 -                 C = strrep(C,'AND','&&');
494 -                 C = strrep(C,'&#9;','');
495 -                 C = strrep(C,'&#10;','');
496 -                 C = strrep(C,'&#13;','');
```

Imagen 112. Traducción a Simulink, traducción del código de las transiciones del « ECC » (1).

```
498 -                 C = strrep(C,'ABS(','"abs(');
499 -                 C = strrep(C,'SQRT(','"sqrt(');
500 -                 C = strrep(C,'LN(','"log(');
501 -                 C = strrep(C,'LOG(','"log10(');
502 -                 C = strrep(C,'EXP(','"exp(');
503 -                 C = strrep(C,'SIN(','"sin(');
504 -                 C = strrep(C,'COS(','"cos(');
505 -                 C = strrep(C,'TAN(','"tan(');
506 -                 C = strrep(C,'ASIN(','"asin(');
507 -                 C = strrep(C,'ACOS(','"acos(');
508 -                 C = strrep(C,'ATAN(','"atan(');
509 -                 C = strrep(C,'LEN(','"length(');
510 -                 C = strrep(C,'NOT(','"not(');
511 
512 -                 C = strcat('if"',C);
513 -                 C = strrep(C,'"',' ');
514 -                 H = length(E{1});
515 -                 for i=H:-1:1
516 -                     Pos = strfind(C,char(E{1}(i)));
517 -                     if ~isempty(Pos)
518 -                         for j=length(Pos):-1:1
519 -                             k = length(char(E{1}(i)));
520 -                             C = strcat(C(1:Pos(j)-1),'CE(',int2str(i),')',C(Pos(j)+k:length(C)));
521 -                         end
522 -                     end
523 -                 end
```

Imagen 113. Traducción a Simulink, traducción del código de las transiciones del « ECC » (2).

Por último, se obtienen los nombres del Estado que se activa y el que se desactiva para cada transición, se igualan a « 1 » y « 0 » respectivamente para indicarlo y se cierra la condición. Una vez hecho ésto, se repite el mismo procedimiento para todas las transiciones.

```
                if isequal(C,'if 1')
                    k = 1;
                else
                    k = 0;
                end

                C1 = leer(strfind(leer,'Source="'):length(leer));
                C1 = C1(length('Source="'):length(C1));
                C1 = strtok(C1,'"');

                for i=1:length(E{1})
                    if isequal(char(E{1}(i)),C1)
                        if k == 0
                            fprintf(FID_d,C);
                            fprintf(FID_d,'\n');
                            fprintf(FID_d,'\t');
                            fprintf(FID_d,'\t');
                            fprintf(FID_d,strcat('E(',int2str(i),')=0;'));
                        else
                            C = strcat('if"CE(',int2str(i),')');
                            C = strrep(C,'"',' ');
                            fprintf(FID_d,C);
                            fprintf(FID_d,'\n');
                            fprintf(FID_d,'\t');
                            fprintf(FID_d,'\t');
                            fprintf(FID_d,strcat('E(',int2str(i),')=0;'));
                        end
                    end
```

Imagen 114. Traducción a Simulink, obtención y declaración del Estado fuente « Source ».

```
                end
                fprintf(FID_d,'\n');
                fprintf(FID_d,'\t');
                fprintf(FID_d,'\t');

                C2 = leer(strfind(leer,'Destination="'):length(leer));
                C2 = C2(length('Destination="'):length(C2));
                C2 = strtok(C2,'"');

                for i=1:length(E{1})
                    if isequal(char(E{1}(i)),C2)
                        fprintf(FID_d,strcat('E(',int2str(i),')=1;'));
                    end
                end
                fprintf(FID_d,'\n');
                fprintf(FID_d,'\t');
                fprintf(FID_d,'end');
                fprintf(FID_d,'\n');
                fprintf(FID_d,'\t');
            end
            leer = fgetl(FID_f);
        end
    end
```

Imagen 115. Traducción a Simulink, obtención y declaración del Estado destino « Destination ».

3.4.6. Ejecución de algoritmos

Para éste apartado se necesitan saber tanto los nombres de los algoritmos como los Estados en los que se ejecutan dichos algoritmos. Para ello, se debe abrir el « Archivo_fuente » y buscar en la declaración del « ECC » los Estados.

```
578 -       fprintf(FID_d,'\n');
579 -       fprintf(FID_d,'\t');
580 -       fprintf(FID_d,'%%6-EJECUCION DE ALGORITMOS');
581 -       fprintf(FID_d,'\n');
582 -       fprintf(FID_d,'\t');
583 -       Ejec_Alg = '';
584
585 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r');
586 -       if (FID_f==-1)
587 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
588 -       else
589 -           while isempty(strfind(leer,'<ECC>'))
590 -               leer = fgetl(FID_f);
591 -           end
592 -           leer = fgetl(FID_f);
```

Imagen 116. Traducción a Simulink, apertura del « Archivo_fuente » para la obtención de los algoritmos.

Una vez se encuentra un Estado, se guarda su nombre y se busca en las acciones asociadas si existe algún algoritmo (línea 603), en tal caso se guarda el nombre de dicho algoritmo con el nombre del Estado al que está asociado en la variable « Ejec_Alg ».

```
594 -       while isempty(strfind(leer,'</ECC>'))
595 -           if ~isempty(strfind(leer,'<ECState')) && isempty(strfind(leer,'/>'))
596 -               k = strfind(leer,'Name="');
597 -               C2 = leer(k:length(leer));
598 -               C2 = C2(length('Name="'):length(C2));
599 -               C2 = strtok(C2,'"');
600 -               while isempty(strfind(leer,'</ECState>'))
601 -                   for i=1:length(E{1})
602 -                       if isequal(char(E{1}(i)),C2)
603 -                           if ~isempty(strfind(leer,'<ECAction')) && ~isempty(strfind(leer,'Algorithm="'))
604 -                               C = leer(strfind(leer,'Algorithm="'):length(leer));
605 -                               C = C(length('Algorithm="'):length(C));
606 -                               C = strtok(C,'"');
607 -                               if isempty(Ejec_Alg)
608 -                                   Ejec_Alg = strcat('Alg_',C,'"E(',int2str(i),')');
609 -                               else
610 -                                   Ejec_Alg = strcat(Ejec_Alg,'"Alg_',C,'"E(',int2str(i),')');
611 -                               end
612 -                           end
613 -                       end
614 -                   end
615 -                   leer = fgetl(FID_f);
616 -               end
617 -           end
618 -           leer = fgetl(FID_f);
619 -       end
```

Imagen 117. Traducción a Simulink, obtención de los algoritmos.

Con todos los algoritmos y los Estados asociados en « Ejec_Alg », se organizan en la variable « Alg » cuyo « array » se recorren para formar las instrucciones que se introducirán en el programa.

```
620 -           C2 = '';
621 -           Ejec_Alg = strrep(Ejec_Alg,'"',' ');
622 -           Alg = textscan(Ejec_Alg,'%s %s');
623 -           H = length(Alg{1});
624 -           for i=1:H
625 -               C = char(Alg{1}(i));
626 -               if isempty(strfind(C2,C))
627 -                   k = 0;
628 -                   for j=1:H
629 -                       if isequal(C,char(Alg{1}(j))) && k==0
630 -                           Cl = strcat(C,'"=="(',char(Alg{2}(j)),'"&&"~C',char(Alg{2}(j)),')');
631 -                           Cl = strrep(Cl,'"',' ');
632 -                           fprintf(FID_d,Cl);
633 -                           k = 1;
634 -                       elseif isequal(C,char(Alg{1}(j))) && k==1
635 -                           Cl = strcat('"||"(',char(Alg{2}(j)),'"&&"~C',char(Alg{2}(j)),')');
636 -                           Cl = strrep(Cl,'"',' ');
637 -                           fprintf(FID_d,Cl);
638 -                       end
639 -                   end
640 -                   fprintf(FID_d,';');
641 -                   fprintf(FID_d,'\n');
642 -                   fprintf(FID_d,'\t');
643 -                   C2 = strcat(C2,'"',C);
644 -                   C2 = strrep(C2,'"',' ');
645 -               end
646 -           end
```

Imagen 118. Traducción a Simulink, declaración de algoritmos y los Estados a los que están asociados.

3.4.7. Declaración de algoritmos

Para poder declarar los algoritmos se necesita obtener y traducir el código de éstos. Para ello, se sigue recorriendo el « Archivo_fuente » buscando las instrucciones « <Algorithm », ya que en cuya línea de código se encuentra el nombre del algoritmo y en la siguiente línea se encuentra declarado el código del algoritmo. Por lo tanto, se obtiene el nombre del algoritmo y se comienza a declarar en una condición « if ».

```
649 -           fprintf(FID_d,'\n');
650 -           fprintf(FID_d,'\t');
651 -           fprintf(FID_d,'%%7-DECLARACION DE ALGORITMOS');
652 -           fprintf(FID_d,'\n');
653 -           fprintf(FID_d,'\t');
654
655 -           while isempty(strfind(leer,'</BasicFB>'))
656 -               if ~isempty(strfind(leer,'<Algorithm'))
657 -                   C = leer(strfind(leer,'Name="'):length(leer));
658 -                   C = C(length('Name="'):length(C));
659 -                   C = strtok(C,'"');
660 -                   C = strcat('if"Alg_',C);
661 -                   C = strrep(C,'"',' ');
662 -                   fprintf(FID_d,C);
663 -                   fprintf(FID_d,'\n');
664 -                   fprintf(FID_d,'\t');
```

Imagen 119. Traducción a Simulink, obtención y declaración del nombre del algoritmo.

Seguidamente, se obtiene el código del algoritmo y si comienza a traducir de una manera similar a las transiciones en el apartado anterior, pero ésta vez con bucles y condiciones. Primero traduciendo los « END_ » como « end » (imagen 122).

```
666 -           while isempty(strfind(leer,'</Algorithm>'))
667 -               if ~isempty(strfind(leer,'<ST'))
668 -                   fprintf(FID_d,'\t');
669 -                   C = leer(strfind(leer,'Text="'):length(leer));
670 -                   C = C(length('Text="'):length(C));
671 -                   C = strtok(C,'"');
672
673 -                   Pos = strfind(C,'=');
674 -                   H = length(Pos);
675 -                   if ~isempty(Pos)
676 -                       for i=H:-1:1
677 -                           if C(Pos(i)-1) ~= ':'
678 -                               C = strcat(C(1:Pos(i)-1),'==',C(Pos(i)+1:length(C)));
679 -                           end
680 -                       end
681 -                   end
```

Imagen 120. Traducción a Simulink, obtención y traducción del código del algoritmo.

```
682 -                   C = strrep(C,':=','=');
683 -                   C = strrep(C,'<>','~=');
684 -                   C = strrep(C,'&gt;','>');
685 -                   C = strrep(C,'&ge;','>=');
686 -                   C = strrep(C,'&lt;','<');
687 -                   C = strrep(C,'&le;','<=');
688 -                   C = strrep(C,'**','^');
689 -                   C = strrep(C,'OR','||');
690 -                   C = strrep(C,'AND','&&');
691
692 -                   C = strrep(C,'ABS(','"abs(');
693 -                   C = strrep(C,'SQRT(','"sqrt(');
694 -                   C = strrep(C,'LN(','"log(');
695 -                   C = strrep(C,'LOG(','"log10(');
696 -                   C = strrep(C,'EXP(','"exp(');
697 -                   C = strrep(C,'SIN(','"sin(');
698 -                   C = strrep(C,'COS(','"cos(');
699 -                   C = strrep(C,'TAN(','"tan(');
700 -                   C = strrep(C,'ASIN(','"asin(');
701 -                   C = strrep(C,'ACOS(','"acos(');
702 -                   C = strrep(C,'ATAN(','"atan(');
703 -                   C = strrep(C,'LEN(','"length(');
704 -                   C = strrep(C,'NOT(','"not(');
```

Imagen 121. Traducción a Simulink, traducción del código del algoritmo.

```
707 -   for i=H:-1:1
708 -       Pos = strfind(C,char(E{1}(i)));
709 -       if ~isempty(Pos)
710 -           for j=length(Pos):-1:1
711 -               k = length(char(E{1}(i)));
712 -               C = strcat(C(1:Pos(j)-1),'CE(',int2str(i),')',C(Pos(j)+k:length(C)));
713 -           end
714 -       end
715 -   end
716 -   Pos = strfind(C,'END_');
717 -   H = length(Pos);
718 -   if ~isempty(Pos)
719 -       for i=H:-1:1
720 -           if C(Pos(i)+4)=='I'
721 -               C = strcat(C(1:Pos(i)-1),'end',C(Pos(i)+7:length(C)));
722 -           elseif C(Pos(i)+4)=='W'
723 -               C = strcat(C(1:Pos(i)-1),'end',C(Pos(i)+10:length(C)));
724 -           elseif C(Pos(i)+4)=='F'
725 -               C = strcat(C(1:Pos(i)-1),'end',C(Pos(i)+8:length(C)));
726 -           elseif C(Pos(i)+4)=='C'
727 -               C = strcat(C(1:Pos(i)-1),'end',C(Pos(i)+9:length(C)));
728 -           end
729 -       end
730 -   end
```

Imagen 122. Traducción a Simulink, traducción del código del algoritmo.

Seguidamente, se traducen los nombres de los bucles y condiciones, de manera que no hay confusión con los que aparecen seguidos de los « END_ » en el « Archivo_fuente ». Se traducen los saltos de línea y los tabuladores (líneas 741-743). Una vez hecho ésto, se declara el código del algoritmo, se cierra el bucle o condición y se repite el mismo procedimiento para todos los algoritmos del programa.

```
731 -                           C = strrep(C,'IF','if');
732 -                           C = strrep(C,'WHILE','while');
733 -                           C = strrep(C,'FOR','for');
734 -                           C = strrep(C,'CASE','switch');
735 -                           C = strrep(C,'THEN','');
736
737 -                           C = strrep(C,'"',' ');
738 -                           if isequal(C(length(C)-9:length(C)),'&#13;&#10;')
739 -                               C = C(1:length(C)-10);
740 -                           end
741 -                           C = strrep(C,'&#9;','\t');
742 -                           C = strrep(C,'&#10;','\n\t\t');
743 -                           C = strrep(C,'&#13;','');
744
745 -                           fprintf(FID_d,C);
746 -                           fprintf(FID_d,'\n');
747 -                           fprintf(FID_d,'\t');
748 -                       end
749 -                       leer = fgetl(FID_f);
750 -                   end
751 -                   fprintf(FID_d,'end');
752 -                   fprintf(FID_d,'\n');
753 -                   fprintf(FID_d,'\t');
754 -               end
755 -               leer = fgetl(FID_f);
756 -           end
757 -       end
```

Imagen 123. Traducción a Simulink, traducción y declaración del algoritmo.

3.4.8. Ejecución de Eventos de salida

Para este apartado, simplemente se necesitan los nombres de los « Outputs » y de los Estados a los que éstos están asociados. Para ello, se vuelve a abrir el « Archivo_fuente » para volver a leer el « ECC », ya que aquí se muestran los Estados en los que se ejecutan dichos « Outputs ».

```
760 -       fprintf(FID_d,'\n');
761 -       fprintf(FID_d,'\t');
762 -       fprintf(FID_d,'%%8-EJECUCION DE EVENTOS DE SALIDA');
763 -       fprintf(FID_d,'\n');
764 -       fprintf(FID_d,'\t');
765 -       Ejec_Outputs = '';
766 -       C='';
767 -       C2='';
768
769 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r'); %open file for reading
770 -       if (FID_f==-1)
771 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
772 -       else
773 -           while isempty(strfind(leer,'<ECC>'))
774 -               leer = fgetl(FID_f);
775 -           end
776 -           leer = fgetl(FID_f);
```

Imagen 124. Traducción a Simulink, declaración del inicio de la « Ejecución de Eventos de salida ».

Por lo tanto, se extrae el nombre del Estado y se busca si tiene asociado algún Evento de salida, en tal caso, se guarda ésta información en la variable « Ejec_Outputs » y se continúa buscando Estados.

```
778 -       while isempty(strfind(leer,'</ECC>'))
779 -           C = leer(strfind(leer,'Name="'):length(leer));
780 -           C = C(length('Name="'):length(C));
781 -           C = strtok(C,'"');
782 -           if ~isempty(strfind(leer,'<ECState')) && isempty(strfind(leer,'/>'))
783 -               while isempty(strfind(leer,'</ECState>'))
784 -                   for i=1:length(E{1})
785 -                       if isequal(char(E{1}(i)),C)
786 -                           if ~isempty(strfind(leer,'<ECAction')) && ~isempty(strfind(leer,'Output="'))
787 -                               C2 = leer(strfind(leer,'Output="'):length(leer));
788 -                               C2 = C2(length('Output="'):length(C2));
789 -                               C2 = strtok(C2,'"');
790 -                               if isempty(Ejec_Outputs)
791 -                                   Ejec_Outputs = strcat(C2,'"E(',int2str(i),')');
792 -                               else
793 -                                   Ejec_Outputs = strcat(Ejec_Outputs,'"',C2,'"E(',int2str(i),')');
794 -                               end
795 -                           end
796 -                       end
797 -                   end
798 -                   leer = fgetl(FID_f);
799 -               end
800 -           end
801 -           leer = fgetl(FID_f);
802 -       end
```

Imagen 125. Traducción a Simulink, obtención de los « Outputs » y los Estados a los cuales están asociados.

Una vez se tienen todos los « Outputs », se organizan en la variable « Outp » y se declaran de la misma manera que se declaró la « **3.4.6. Ejecución de algoritmos** » (imagen 119).

```
803 -           C2 = '';
804 -           Ejec_Outputs = strrep(Ejec_Outputs,'"',' ');
805 -           Outp = textscan(Ejec_Outputs,'%s %s');
806 -           H = length(Outp{1});
807 -           for i=1:H
808 -               C = char(Outp{1}(i));
809 -               if isempty(strfind(C2,C))
810 -                   k = 0;
811 -                   for j=1:H
812 -                       if isequal(C,char(Outp{1}(j))) && k==0
813 -                           C1 = strcat(C,'"=("',char(Outp{2}(j)),'"&&"~C',char(Outp{2}(j)),')');
814 -                           C1 = strrep(C1,'"',' ');
815 -                           fprintf(FID_d,C1);
816 -                           k = 1;
817 -                       elseif isequal(C,char(Outp{1}(j))) && k==1
818 -                           C1 = strcat('"||"(',char(Outp{2}(j)),'"&&"~C',char(Outp{2}(j)),')');
819 -                           C1 = strrep(C1,'"',' ');
820 -                           fprintf(FID_d,C1);
821 -                       end
822 -                   end
823 -                   fprintf(FID_d,';');
824 -                   fprintf(FID_d,'\n');
825 -                   fprintf(FID_d,'\t');
826 -                   C2 = strcat(C2,'"',C);
827 -                   C2 = strrep(C2,'"',' ');
828 -               end
829 -           end
830 -       end
```

Imagen 126. Traducción a Simulink, declaración de « Outputs ».

3.4.9. Actualización de Variables de salida asociadas a Eventos « WITH »

Para declarar este apartado, solamente es necesaria saber que Variables de salida están asociadas a qué « Outputs », lo cual se encuentra en la declaración de Eventos de salida del « Archivo_fuente ». Por lo cual, se busca la instrucción « EventOutputs », se extrae el nombre del Evento en cuestión y se declara en una condición « if ».

```
833 -       fprintf(FID_d,'\n');
834 -       fprintf(FID_d,'\t');
835 -       fprintf(FID_d,'%%9-EJECUCION DE VARIABLES DE SALIDA (WITH)');
836 -       fprintf(FID_d,'\n');
837 -       fprintf(FID_d,'\t');
838 -       C2='';
839
840 -       [FID_f,MESSAGE] = fopen(Archivo_fuente, 'r');
841 -       if (FID_f==-1)
842 -           disp(strcat('Error al intentar leer el archivo: ', Archivo_fuente, ',', MESSAGE))
843 -       else
844 -           while isempty(strfind(leer,'<EventOutputs>'))
845 -               leer = fgetl(FID_f);
846 -           end
847 -           leer = fgetl(FID_f);
848
849 -           while isempty(strfind(leer,'</EventOutputs>'))
850 -               if ~isempty(strfind(leer,'<Event')) && isempty(strfind(leer,'/>'))
851 -                   C = leer(strfind(leer,'Name="'):length(leer));
852 -                   C = C(length('Name="'):length(C));
853 -                   C = strtok(C,'"');
854 -                   C = strcat('if"e_',C);
855 -                   C = strrep(C,'"',' ');
856 -                   fprintf(FID_d,C);
857 -                   fprintf(FID_d,'\n');
858 -                   fprintf(FID_d,'\t');
```

Imagen 127. Traducción a Simulink, obtención y declaración de las Variables de salida asociadas a Eventos de salida.

La declaración de éstas Variables de salida asociadas, es más compleja que la de las Variables de entrada, ya que al encontrarse dentro del bucle del « ECC » necesitará usar una segunda copia de dicha Variable para conservar el valor de la primera ejecución de éste en caso de que se ejecute más de una vez. Para ello, se extrae y se declara el nombre de las Variables asociadas igualadas a la Variable del Bloque de función (línea 865), la cual contiene « v_ ». Una vez declaradas, se introduce la condición « else » y se declararán las segundas copias de las Variables asociadas igualadas a la misma Variable del Bloque de función (línea 881). Se cierra el bucle y se introduce una última instrucción, en la cual se actualiza el valor de la segunda copia de la Variable con el valor de la Variable del Bloque de función (línea 890).

```
859 -                   while isempty(strfind(leer,'</Event>'))
860 -                       if ~isempty(strfind(leer,'<With'))
861 -                           C = leer(strfind(leer,'Var="'):length(leer));
862 -                           C = C(length('Var="'):length(C));
863 -                           C = strtok(C,'"');
864 -                           fprintf(FID_d,'\t');
865 -                           fprintf(FID_d,strcat('v_',C,'=',C,';'));
866 -                           fprintf(FID_d,'\n');
867 -                           fprintf(FID_d,'\t');
868 -                           C2 = strcat(C2,'"',C);
869 -                           C2 = strrep(C2,'"',' ');
870 -                       end
871 -                       leer = fgetl(FID_f);
872 -                   end
873 -                   fprintf(FID_d,'else');
874 -                   fprintf(FID_d,'\n');
875 -                   fprintf(FID_d,'\t');
876 -                   Outp = textscan(C2,'%s');
877 -                   H = length(Outp{1});
878 -                   for i=1:H
879 -                       C = char(Outp{1}(i));
880 -                       fprintf(FID_d,'\t');
881 -                       fprintf(FID_d,strcat('v_',C,'=prev_',C,';'));
882 -                       fprintf(FID_d,'\n');
883 -                       fprintf(FID_d,'\t');
884 -                   end
885 -                   fprintf(FID_d,'end');
```

Imagen 128. Traducción a Simulink, declaración de las Variables de salida asociadas a Eventos de salida.

```
886 -                   fprintf(FID_d,'\n');
887 -                   for i=1:H
888 -                       C = char(Outp{1}(i));
889 -                       fprintf(FID_d,'\t');
890 -                       fprintf(FID_d,strcat('prev_',C,'=v_',C,';'));
891 -                       fprintf(FID_d,'\n');
892 -                       fprintf(FID_d,'\t');
893 -                   end
894 -               end
895 -               leer = fgetl(FID_f);
896 -           end
897 -       end
```

Imagen 129. Traducción a Simulink, declaración de las Variables de salida asociadas a Eventos de salida (2).

3.4.10. Preparación de Eventos de salida y Variables de salida antes de enviarse

Al final del código, hay unas cuantas variables cuyo valor se debe cambiar para evitar errores en el bucle del « ECC ». Lo primero es resetear las variables de los Eventos de entrada a « 0 » para que no se repita la misma condición en dicho bucle (línea 911).

```
900 -       fprintf(FID_d,'\n');
901 -       fprintf(FID_d,'\t');
902 -       fprintf(FID_d,'%%10-FINAL');
903 -       fprintf(FID_d,'\n');
904 -       fprintf(FID_d,'\t');
905
906 -       EI = strrep(EI,'"',' ');
907 -       Inp = textscan(EI,'%s');
908 -       H = length(Inp{1});
909 -       for i=1:H
910 -           C = char(Inp{1}(i));
911 -           C2 = strcat(C,'"=0;');
912 -           C2 = strrep(C2,'"',' ');
913 -           fprintf(FID_d,C2);
914 -           fprintf(FID_d,'\n');
915 -           fprintf(FID_d,'\t');
916 -       end
```

Imagen 130. Traducción a Simulink, reseteo de los Eventos de entrada.

Seguidamente, se declara una condición « if » por cada Evento de salida. En el cual, solamente si se detecta un flanco en dicho Evento (línea 925), la segunda copia de éste se igualará a « 1 » (línea 931) guardando así el estado de éste aunque se vuelva a modificar el valor de la variable principal en el bucle.

```
918 -       EO = strrep(EO,'"',' ');
919 -       Outp = textscan(EO,'%s');
920 -       H = length(Outp{1});
921 -       for i=1:H
922 -           C = char(Outp{1}(i));
923 -           fprintf(FID_d,'\n');
924 -           fprintf(FID_d,'\t');
925 -           C2 = strcat('if"',C,'"&&"~prev_',C);
926 -           C2 = strrep(C2,'"',' ');
927 -           fprintf(FID_d,C2);
928 -           fprintf(FID_d,'\n');
929 -           fprintf(FID_d,'\t');
930 -           fprintf(FID_d,'\t');
931 -           C2 = strcat('prev_',C,'"=1;');
932 -           C2 = strrep(C2,'"',' ');
933 -           fprintf(FID_d,C2);
934 -           fprintf(FID_d,'\n');
935 -           fprintf(FID_d,'\t');
936 -           fprintf(FID_d,'end');
937 -       end
938 -       fprintf(FID_d,'\n');
939 -       fprintf(FID_d,'end');
```

Imagen 131. Traducción a Simulink, detección de flancos en los Eventos de salida.

Por último, se vuelven a actualizar los Eventos y las Variables de salida para verificar que la información que se va a enviar sea la correcta. Para ello, se igualan las variables de los Eventos y Variables de salida del Bloque de función, las que empiezan por « e_ » o « v_ », a las segundas copias de los mismos Eventos o Variables tal y como se muestra en las líneas 943 y 954.

```
941 -       for i=1:H
942 -           C = char(Outp{1}(i));
943 -           C2 = strcat('e_',C,'"="prev_',C);
944 -           C2 = strrep(C2,'"',' ');
945 -           fprintf(FID_d,'\n');
946 -           fprintf(FID_d,C2);
947 -       end
948
949 -       OV = strrep(OV,'"',' ');
950 -       Outp = textscan(OV,'%s');
951 -       H = length(Outp{1});
952 -       for i=1:H
953 -           C = char(Outp{1}(i));
954 -           C2 = strcat('v_',C,'"="prev_',C);
955 -           C2 = strrep(C2,'"',' ');
956 -           fprintf(FID_d,'\n');
957 -           fprintf(FID_d,C2);
958 -       end
959 -   end
```

Imagen 132. Traducción a Simulink, Actualización de Eventos de salida y Variables de salida.

3.5. Ejemplos prácticos

3.5.1. Ejemplo 1: Control por histéresis del nivel de un depósito

En éste ejemplo se considera un Bloque de Función que aplica un algoritmo de control de un depósito por banda de histéresis, lo cual quiere decir que si se detecta que el « Nivel_actual » es mayor que el « Nivel » demandado más la banda de histéresis, la bomba debe ir a mínima potencia « Pot_min ». Mientras que si el « Nivel_actual » es menor que el « Nivel » demandado menos la banda de histéresis, la bomba debe ir a máxima potencia « Pot_max ». Quedando el diagrama de control que se muestra en la « Imagen 133 ».

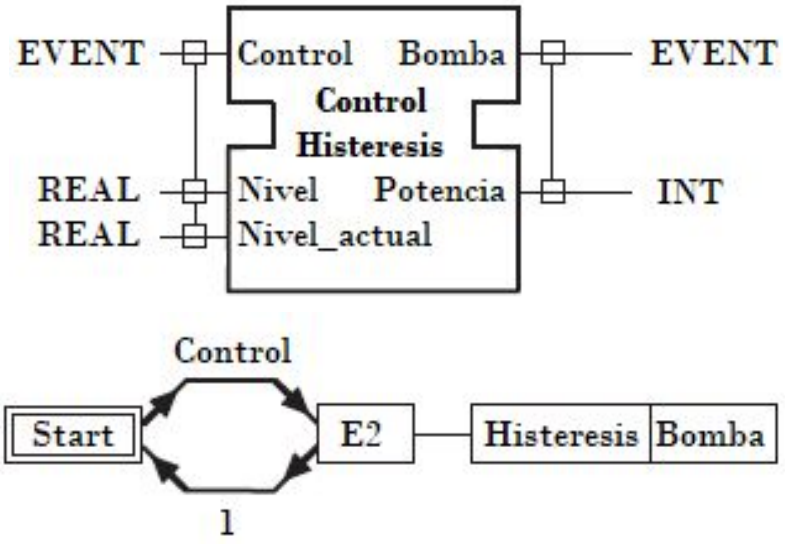

Imagen 133. Ejemplo 1, esquema del Bloque de función y el « ECC ».

Una vez tenemos claro el esquema, se genera dicho Bloque de función en Simulink, obteniendo así el siguiente código:

```
%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE
function [e_Bomba, v_Potencia]  = fcn (e_Control, v_Nivel_actual, v_Nivel)

%2-DECLARACION DE VARIABLES INTERNAS
persistent Pot_min Pot_max H;

%3-DECLARACION DEL ESTADO INICIAL
persistent Bomba Potencia Control Nivel_actual Nivel E CE prev_Potencia;

if isempty(E)
   E =[1 0]; %Estado de las etapas
   %Variables internas
   H=0.05;
   Pot_min=10;
   Pot_max=90;
   %Variables de entrada
   Nivel=0.0;
   Nivel_actual=0.0;
   %Variables de salida
   Potencia=0.0;
   prev_Potencia=0.0;
   %Eventos de entrada
   Control=0;
   %Eventos de salida
   Bomba=0;
end

%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)
```

```
if (e_Control)
   Control = 1;
   Nivel = v_Nivel;
   Nivel_actual = v_Nivel_actual;
end

%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)
prev_Bomba=0;
CE=[0 0]; %Copia de etapas
while ~isequal(CE,E)
   CE=E;

   if (CE(1,1) && Control)
      E(1,1)=0;
      E(1,2)=1;
   end

   if CE(1,2)
      E(1,2)=0;
      E(1,1)=1;
   end

   %6-EJECUCION DE ALGORITMOS
   Alg_Histeresis = E(1,2) && ~CE(1,2);

   %7-DECLARACION DE ALGORITMOS
   if (Alg_Histeresis) %Algoritmo de control por Histéresis
      if (Nivel + Nivel*H) < Nivel_actual
         Potencia = Pot_min;
      end

      if (Nivel - Nivel*H) > Nivel_actual
         Potencia = Pot_max;
      end
   end

   %8-EJECUCION DE EVENTOS DE SALIDA
   Bomba = E(1,2);

   %9-EJECUCION DE VARIABLES DE SALIDA (WITH)
   if (Bomba)
      v_Potencia = Potencia;
   else
      v_Potencia = prev_Potencia;
   end
   prev_Potencia = v_Potencia;

   %10-FINAL
   Control = 0;

   if Bomba && ~prev_Bomba
      prev_Bomba = 1;
   end
end

e_Bomba = prev_Bomba;
v_Potencia = prev_Potencia;
```

Con el código ya introducido en el programa, se comprueba su funcionamiento mediante su simulación, de manera que se obtienen los resultados de las « imágenes 134, 135 y 136 ».

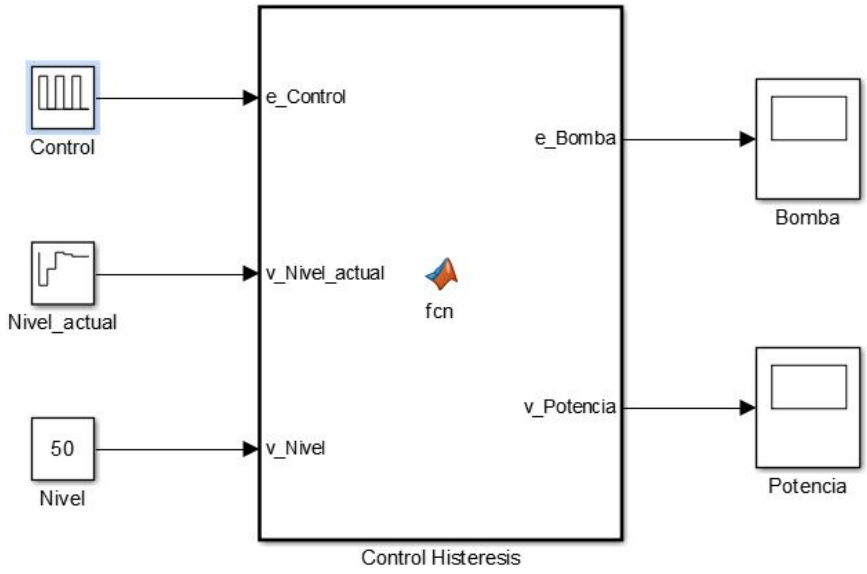

Imagen 134. Ejemplo 1, Bloque de función en Simulink.

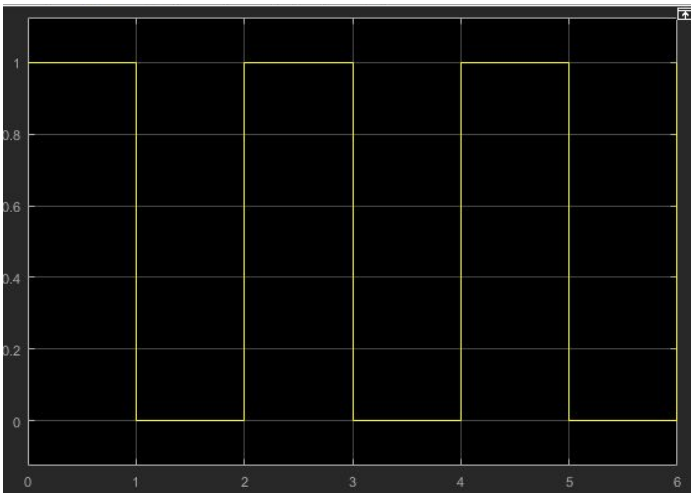

Imagen 135. Ejemplo 1, Evento de salida « Bomba ».

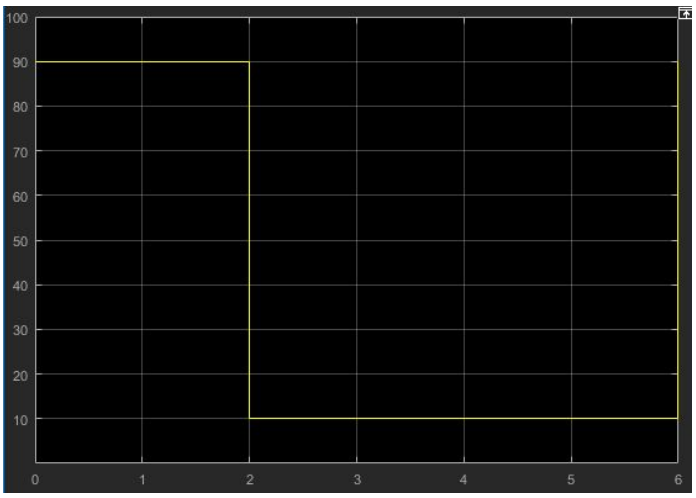

Imagen 136. Ejemplo 1, Evento de salida « Potencia ».

Una vez comprobado el correcto funcionamiento del Bloque de función, se pasa dicho código por el traductor y se obtiene el siguiente código del programa para « 4DIAC »:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">
<FBType Comment="Tamplate for a simple Basic Function Block Type" Name="Event_driven_PID">
 <Identification Standard="61499-2"/>
 <VersionInfo Author="4DIAC-IDE" Date="2016-11-02" Organization="4DIAC-Consortium" Version="0.0"/>
 <VersionInfo Author="AZ" Date="2016-05-26" Organization="fortiss GmbH" Version="1.0"/>
 <InterfaceList>
  <EventInputs>
   <Event Name="Control" Type="Event">
    <With Var="Nivel"/>
    <With Var="Nivel_actual"/>
   </Event>
  </EventInputs>
  <EventOutputs>
   <Event Name="Bomba" Type="Event">
    <With Var="Potencia"/>
   </Event>
  </EventOutputs>
```

```
  <InputVars>
    <VarDeclaration Name="Nivel_actual" Type="REAL"/>
    <VarDeclaration Name="Nivel" Type="REAL"/>
  </InputVars>
  <OutputVars>
    <VarDeclaration Name="Potencia" Type="REAL"/>
  </OutputVars>
 </InterfaceList>
 <BasicFB>
  <InternalVars>
    <VarDeclaration InitialValue="10" Name="Pot_min" Type="INT"/>
    <VarDeclaration InitialValue="90" Name="Pot_max" Type="INT"/>
    <VarDeclaration InitialValue="0.05" Name="H" Type="REAL"/>
  </InternalVars>
  <ECC>
    <ECState Name="START" x="1000.0" y="400"/>
    <ECState Name="E12" x="1000.0" y="500">
      <ECAction Algorithm="Histeresis" Output="Bomba"/>
    </ECState>
    <ECTransition Condition="(START AND Control)" Destination="E12" Source="START" x="1200.0" y="650"/>
    <ECTransition Condition="1" Destination="START" Source="E12" x="1200.0" y="650"/>
  </ECC>
  <Algorithm Name="Histeresis">
    <ST Text="IF (Nivel+Nivel*H)&lt;Nivel_actual
THEN&#13;&#10;&#9;Potencia:=Pot_min;&#13;&#10;END_IF;&#13;&#10;IF (Nivel-Nivel*H)&gt;Nivel_actual
THEN&#13;&#10;&#9;Potencia:=Pot_max;&#13;&#10;END_IF;"/>
  </Algorithm>
 </BasicFB>
</FBType>
```


Por último, se comprueba que el programa « 4DIAC » entiende el código sin ningún problema para verificar que el funcionamiento de la traducción ha sido el correcto, obteniendo las « Imágenes 137 y 138 ».

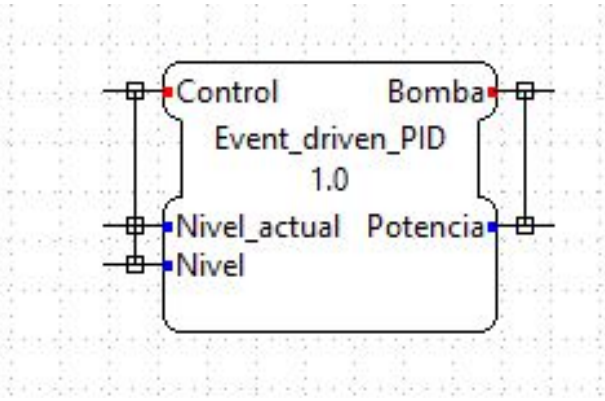

Imagen 137. Ejemplo 1, Bloque de función en « 4DIAC ».

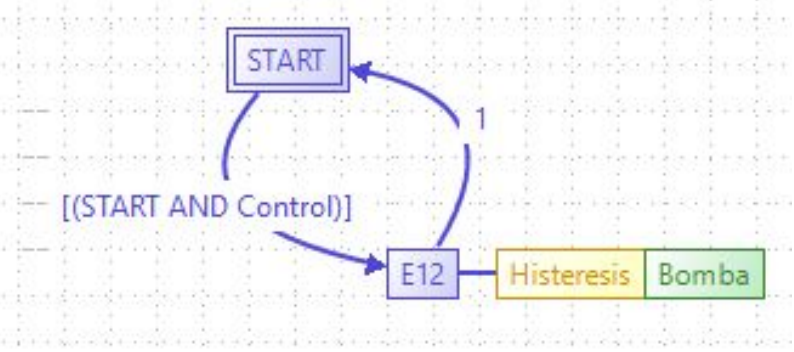

Imagen 138. Ejemplo 1, diagrama de control de ejecución en « 4DIAC ».

3.5.2. Ejemplo 2: Voter

En éste ejemplo se considera un Bloque de Función que aplica un algoritmo de '2 de 3 votaciones' en tres entradas « A », « B » y « C ». La interfaz de este Bloque de Función se puede modelar como se muestra en la « Imagen 139 ». El bloque de funciones tiene dos Eventos de entrada, « Vote » y « Restablecer ». El evento « Vote » se usa para activar el proceso de votación que verifica el estado de las tres entradas booleanas « A », « B » y « C ». Si dos o más entradas son « TRUE », el estado de salida se establece en « TRUE » y permanece en dicho estado hasta que se active el Evento « Reset ». Cuando se completa la votación, se produce un Evento de salida en « Voted ».

El Evento de entrada « Reset » activa un reinicio de la Variable de salida « State » seguido de un Evento de salida en « Ready ». Los diagramas de secuencia de servicio que describen este comportamiento de una manera más formal se muestran en la « Imagen Y ». Los diagramas de secuencia de servicio describen cuatro escenarios concretos. Sin embargo, estos no cubren todos los escenarios posibles. Para capturarlos, también se requeriría una especificación de todas las diferentes combinaciones de entrada posibles y flujos de sucesos posteriores. Esto daría como resultado un buen número de secuencias de servicio. Sin embargo, el subconjunto presentado aquí es suficiente para describir el comportamiento de un usuario potencial.

Tras un Evento de entrada « Vote » en el estado de inicio « Ready », se ingresa el estado « Vote » y se ejecuta el algoritmo « VoteAlg ». Este algoritmo realiza la votación y después de su finalización activa el evento de salida « Voted » para informar a los bloques de funciones siguientes sobre el resultado de la votación. Si el resultado del voto fue positivo, se activará el estado VotedPos; de lo contrario, volvemos al estado « Ready ». Cuando el estado de « VotedPos » está activo, un evento de entrada de « Reset » dará lugar a una transición al estado « Reset », invocando la « ResetAlg », desencadenando el evento de salida « Ready » y finalmente volviendo al estado « Ready » esperando la siguiente solicitud de voto.

Con todo lo descrito, los esquemas del Bloque de Función y el « ECC » de este programa serían los que se muestran en la « Imagen 139 ».

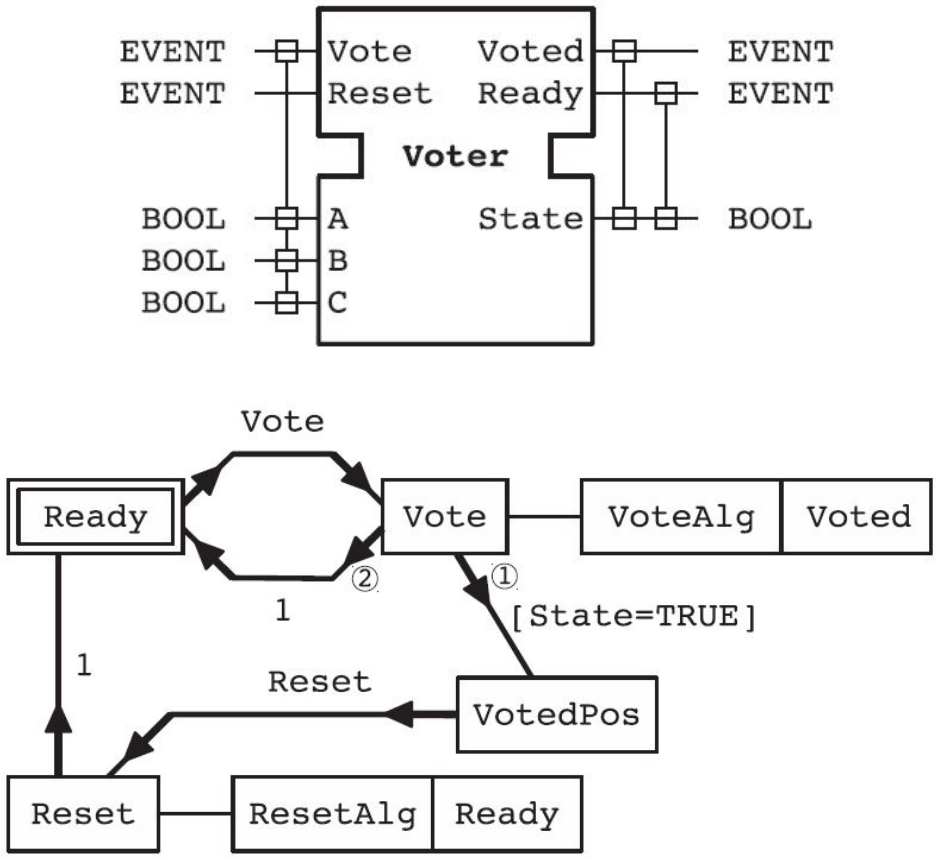

Imagen 139. Ejemplo 2, esquema del Bloque de función y el « ECC ».

La sintaxis textual para describir el Bloque de Función del « Voter », que se muestra en la « Imagen 139 », en un « Matlab Function » es la siguiente:

```
%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE
function [e_Voted, e_Ready, v_State]  = fcn (e_Vote, e_Reset, v_A, v_B, v_C)

%2-DECLARACION DE VARIABLES INTERNAS

%3-DECLARACION DEL ESTADO INICIAL
persistent Voted Ready State Vote Reset A B C E CE prev_State;

if isempty(E)
   E =[1 0 0 0]; %Estado de las etapas
   %Variables internas

   %Variables de entrada
   A = logical(0);
   B = logical(0);
   C = logical(0);
   %Variables de salida
   State = logical(0);
   %Eventos de entrada
   Vote = 0;
   Reset = 0;
   %Eventos de salida
   Voted = 0;
   Ready = 0;
end

%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)
if (e_Vote)
   Vote = 1;
   A = v_A;
   B = v_B;
   C = v_C;
end

if (e_Reset)
   Reset = 1;
end

%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)
prev_Voted=0;
prev_Ready=0;
CE=[0 0 0 0]; %Copia de etapas
while ~isequal(CE,E)
   CE=E;

   if CE(1,1) && Vote
      E(1,1)=0;
      E(1,2)=1;
   end

   if CE(1,2) && State
      E(1,2)=0;
      E(1,3)=1;
   end

   if CE(1,2) && ~E(1,3) %ya que tiene menor prioridad que la anterior
      E(1,2)=0;
      E(1,1)=1;
   end
```

```
    if CE(1,3) && Reset
        E(1,3)=0;
        E(1,4)=1;
    end

    if CE(1,4)
        E(1,4)=0;
        E(1,1)=1;
    end

    %6-EJECUCION DE ALGORITMOS
    Alg_VoteAlg = E(1,2) && ~CE(1,2);
    Alg_ResetAlg = E(1,4) && ~CE(1,4);

    %7-DECLARACION DE ALGORITMOS
    if (Alg_VoteAlg) %Algoritmo VoteAlg
        State = (A&&B) || (A&&C) || (B&&C);
    end

    if (Alg_ResetAlg) %Algoritmo ResetAlg
        State = 0;
    end

    %8-EJECUCION DE EVENTOS DE SALIDA
    Voted = E(1,2);
    Ready = E(1,4);

    %9-EJECUCION DE VARIABLES DE SALIDA (WITH)
    if (Voted)
        v_State = State;
    else
        v_State = prev_State;
    end
    prev_State = v_State;

    if (Ready)
        v_State = State;
    else
        v_State = prev_State;
    end
    prev_State = v_State;

    %10-FINAL
    Vote = 0;
    Reset = 0;

    if Voted && ~prev_Voted
        prev_Voted = 1;
    end
    if Ready && ~prev_Ready
        prev_Ready = 1;
    end
end

e_Voted = prev_Voted;
e_Ready = prev_Ready;
v_State = prev_State;
```

Dando lugar al Bloque « Matlab Function » que aparece en la « Imagen 140 ».

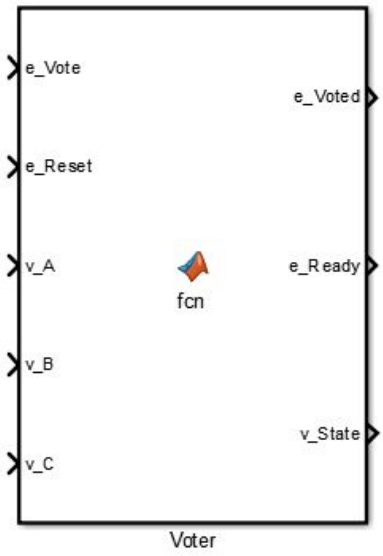

Imagen 140. Ejemplo 2, Bloque de función en Simulink.

Una vez traducido el Bloque de Función, se obtiene que la sintaxis textual que describe la estructura del « Voter » en « 4DIAC » es la siguiente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">
<FBType Comment="Voter" Name="Voter.txt">
 <Identification Standard="61499-2"/>
 <VersionInfo Author="FB_SLK24DIAC" Date="30-10-2017" Organization="UJI" Version="0.0"/>
 <InterfaceList>
  <EventInputs>
   <Event Name="Vote" Type="Event">
    <With Var="A"/>
    <With Var="B"/>
    <With Var="C"/>
   </Event>
   <Event Name="Reset" Type="Event">
   </Event>
  </EventInputs>
  <EventOutputs>
   <Event Name="Voted" Type="Event">
    <With Var="State"/>
   </Event>
   <Event Name="Ready" Type="Event">
    <With Var="State"/>
   </Event>
  </EventOutputs>
  <InputVars>
   <VarDeclaration Name="A" Type="BOOL"/>
   <VarDeclaration Name="B" Type="BOOL"/>
   <VarDeclaration Name="C" Type="BOOL"/>
  </InputVars>
  <OutputVars>
   <VarDeclaration Name="State" Type="BOOL"/>
  </OutputVars>
 </InterfaceList>
 <BasicFB>
  <InternalVars>
  </InternalVars>
  <ECC>
   <ECState Name="START" x="1000.0" y="400"/>
```

```
    <ECState Name="E12" x="1000.0" y="500">
      <ECAction Algorithm="VoteAlg" Output="Voted"/>
    </ECState>
    <ECState Name="E13" x="1000.0" y="600"/>
    <ECState Name="E14" x="1000.0" y="700">
      <ECAction Algorithm="ResetAlg" Output="Ready"/>
    </ECState>
    <ECTransition Condition="START AND Vote" Destination="E12" Source="START" x="1200.0" y="850"/>
    <ECTransition Condition="E12 AND State" Destination="E13" Source="E12" x="1200.0" y="850"/>
    <ECTransition Condition="E12" Destination="START" Source="E12" x="1200.0" y="850"/>
    <ECTransition Condition="E13 AND Reset" Destination="E14" Source="E13" x="1200.0" y="850"/>
    <ECTransition Condition="1" Destination="START" Source="E14" x="1200.0" y="850"/>
  </ECC>
  <Algorithm Name="VoteAlg">
    <ST Text="State:=(A AND B) OR (A AND C) OR (B AND C);"/>
  </Algorithm>
  <Algorithm Name="ResetAlg">
    <ST Text="State:=0;"/>
  </Algorithm>
 </BasicFB>
</FBType>
```

Una vez se obtiene dicho código, se comprueba que el programa « 4DIAC » entiende el código sin ningún problema para verificar que el funcionamiento de la traducción ha sido el correcto, obteniendo las « Imágenes 137 y 138 ».

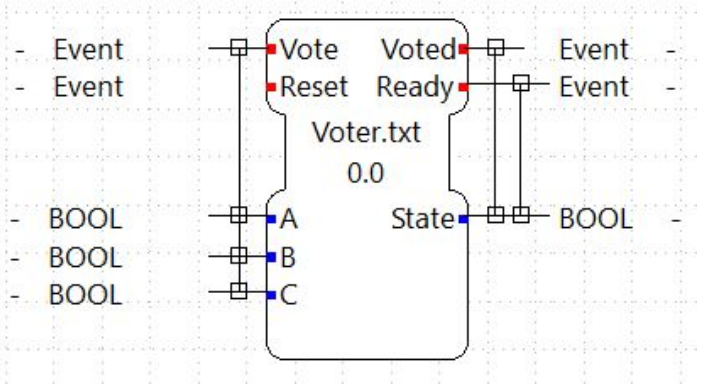

Imagen 141. Ejemplo 2, Bloque de función en « 4DIAC ».

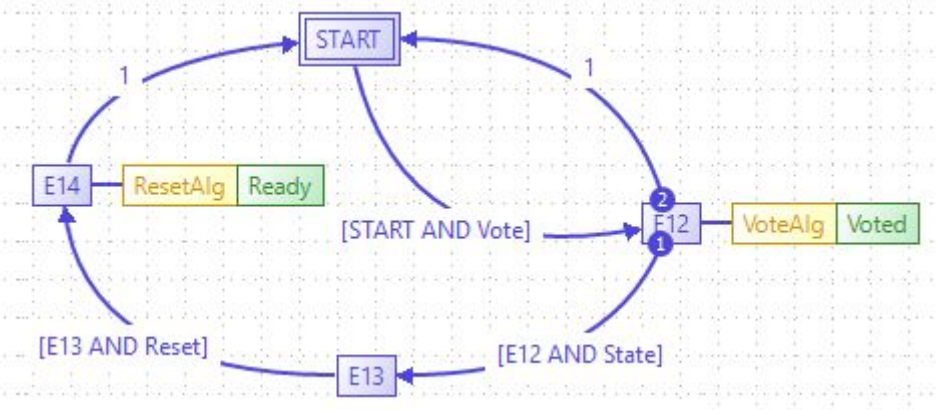

Imagen 142. Ejemplo 2, diagrama de control de ejecución en « 4DIAC ».

3.5.3. Ejemplo 3: Generador de Rampa

En éste ejemplo se considera un Bloque de Función donde la funcionalidad se define principalmente en los algoritmos. Se considera el comportamiento requerido para una rampa muy simple. Éste Bloque de Función, extrae una rampa en la Variable de salida « Out » a partir de los datos de las Variables de entrada « X0 », « X1 » y « Duration ». La Variable de entrada « Cycle » define el tiempo transcurrido entre las actualizaciones de las variables de salida de la rampa. El bloque de función también verifica si la salida excede la Variable de entrada « PV », en cuyo caso la Variable de salida « Hold » se establece en « TRUE ». Se da por supuesto que el Bloque de Función se llama repetidamente y a una velocidad de actualización marcada por « Cycle », por ejemplo, puede configurarse para ejecutarse cada 200 ms.

Cuando llega un Evento de entrada « INIT », se almacenan los valores de entrada que caracterizan el comportamiento de la rampa, es decir, « X0 », « X1 », « Cycle » y « Duration ». El Estado INIT provocando la inicialización del algoritmo « InitAlg ». Lo cual restablece la variable de temporizador interno « T ». El Evento de salida « INITO » se activa cuando finaliza el algoritmo de inicialización « InitAlg ».

De forma similar, cuando llega un Evento de entrada « REQ », se produce una transición al Estado « Ramp » que provoca la ejecución del algoritmo « RampAlg ». Esto calcula el nuevo valor de « Out » basado en los valores de « X0 », « X1 », « Cycle » y « Duration » y el tiempo en el bloque « T ». El algoritmo también verifica si la salida excede el valor de la Variable de entrada « PV », en cuyo caso el « Hold » de salida se establece en « TRUE ».

La definición gráfica de este Bloque de Función viene declarado por una interfaz externa estática, las Variables internas, el « ECC » y los algoritmos tal y como se muestra en la « Imagen 143 ».

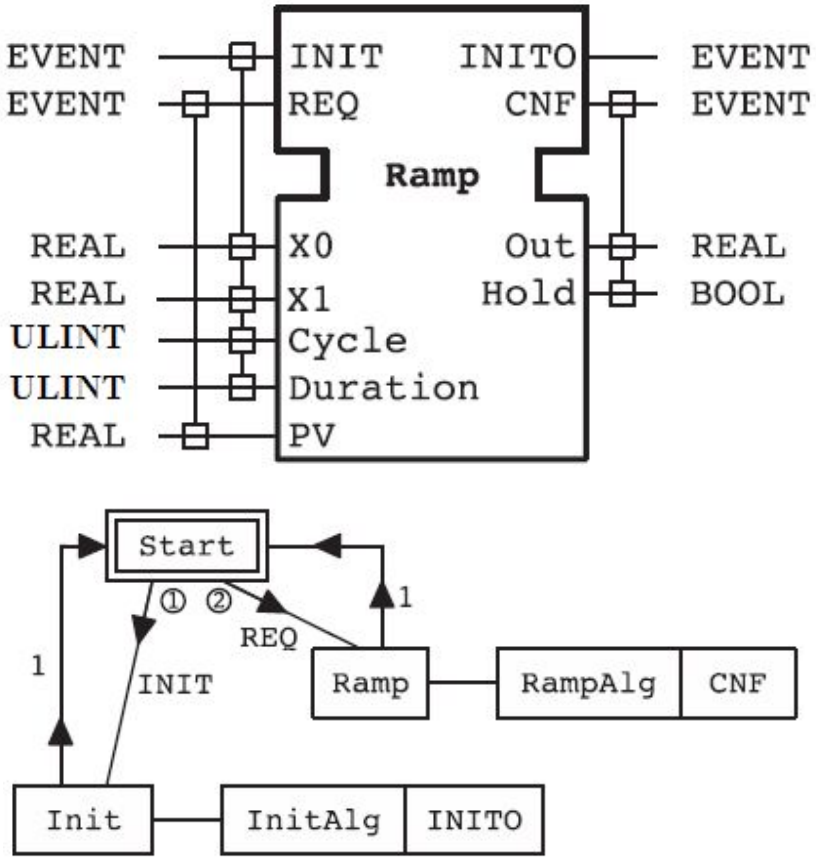

Imagen 143. Ejemplo 3, esquema del Bloque de función y el « ECC ».

La sintaxis textual para describir el Bloque de Función del « Ramp », que se muestra en la « Imagen 143 », en un « Matlab Function » es la siguiente:

```
%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE
function [e_INITO, e_CNF, v_Out, v_Hold]  = fcn (e_INIT, e_REQ, v_X0, v_X1, v_Cycle, v_Duration, v_PV)

%2-DECLARACION DE VARIABLES INTERNAS
persistent T;

%3-DECLARACION DEL ESTADO INICIAL
persistent INITO CNF Out Hold INIT REQ X0 X1 Cycle Duration PV E CE prev_Out prev_Hold;

if isempty(E)
   E =[1 0 0]; %Estado de las etapas
   %Variables internas
   T = uint64(0);
   %Variables de entrada
   X0 = 0.0;
   X1 = 0.0;
   Cycle = uint64(0);
   Duration = uint64(0);
   PV = 0.0;
   %Variables de salida
   Out = 0.0;
   Hold = 0.0;
   %Eventos de entrada
   INIT = 0;
   REQ = 0;
   %Eventos de salida
   INITO = 0;
   CNF = 0;
end

%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)
if (e_INIT)
   INIT = 1;
   X0 = v_X0;
   X1 = v_X1;
   Cycle = v_Cycle;
   Duration = v_Duration;
end

if (e_REQ)
   REQ = 1;
   PV = v_PV;
end

%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)
prev_INITO=0;
prev_CNF=0;
CE=[0 0 0]; %Copia de etapas
while ~isequal(CE,E)
   CE=E;

   if CE(1) && INIT
      E(1)=0;
      E(2)=1;
   end

   if CE(1) && REQ && ~E(2) %ya que tiene menor prioridad que el anterior
      E(1)=0;
      E(3)=1;
   end
```

```
   if CE(2)
      E(2)=0;
      E(1)=1;
   end

   if CE(3)
      E(3)=0;
      E(1)=1;
   end

   %6-EJECUCION DE ALGORITMOS
   Alg_RampAlg = E(3) && ~CE(3);
   Alg_InitAlg = E(2) && ~CE(2);

   %7-DECLARACION DE ALGORITMOS
   if (Alg_RampAlg) %Algoritmo RampAlg
      if T<Duration
         Out = X0 + (X1-X0)*T/Duration;
         T = T+Cycle;
         Hold = PV>Out;
      end
   end

   if (Alg_InitAlg) %Algoritmo InitAlg
      T = 0;
   end

   %8-EJECUCION DE EVENTOS DE SALIDA
   INITO = E(2);
   CNF = E(3);

   %9-EJECUCION DE VARIABLES DE SALIDA (WITH)
   if (CNF)
      v_Out = Out;
      v_Hold = Hold;
   else
      v_Out = prev_Out;
      v_Hold = prev_Hold;
   end
   prev_Out = v_Out;
   prev_Hold = v_Hold;

   %10-FINAL
   INIT = 0;
   REQ = 0;

   if INITO && ~prev_INITO
      prev_INITO = 1;
   end

   if CNF && ~prev_CNF
      prev_CNF = 1;
   end
end

e_INITO = prev_INITO;
e_CNF = prev_CNF;
v_Out = prev_Out;
v_Hold = prev_Hold;
```

Dando lugar al Bloque « Matlab Function » que aparece en la « Imagen 144 ».

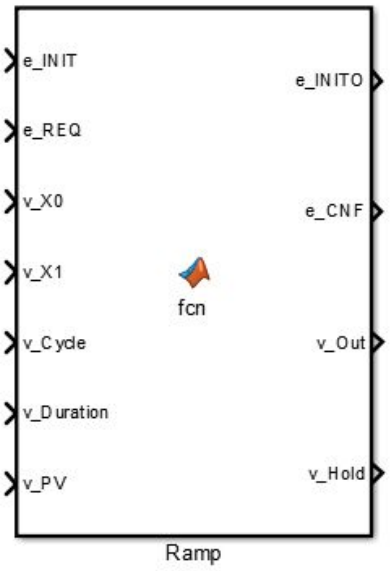

Imagen 144. Ejemplo 3, Bloque de función en Simulink.

Una vez traducido el Bloque de Función, se obtiene que la sintaxis textual que describe la estructura del « Ramp » en « 4DIAC » es la siguiente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">
<FBType Comment="Ramp" Name="Ramp.txt">
 <Identification Standard="61499-2"/>
 <VersionInfo Author="FB_SLK24DIAC" Date="5-11-2017" Organization="UJI" Version="0.0"/>
 <InterfaceList>
  <EventInputs>
   <Event Name="INIT" Type="Event">
    <With Var="X0"/>
    <With Var="X1"/>
    <With Var="Cycle"/>
    <With Var="Duration"/>
   </Event>
   <Event Name="REQ" Type="Event">
    <With Var="PV"/>
   </Event>
  </EventInputs>
  <EventOutputs>
   <Event Name="CNF" Type="Event">
    <With Var="Out"/>
    <With Var="Hold"/>
   </Event>
   <Event Name="INITO" Type="Event"/>
  </EventOutputs>
  <InputVars>
   <VarDeclaration Name="X0" Type="REAL"/>
   <VarDeclaration Name="X1" Type="REAL"/>
   <VarDeclaration Name="Cycle" Type="LINT"/>
   <VarDeclaration Name="Duration" Type="LINT"/>
   <VarDeclaration Name="PV" Type="REAL"/>
  </InputVars>
  <OutputVars>
   <VarDeclaration Name="Out" Type="REAL"/>
   <VarDeclaration Name="Hold" Type="REAL"/>
  </OutputVars>
 </InterfaceList>
 <BasicFB>
```

```
  <InternalVars>
   <VarDeclaration InitialValue="0" Name="T" Type="LINT"/>
  </InternalVars>
  <ECC>
   <ECState Name="START" x="1000.0" y="400"/>
   <ECState Name="E2" x="1000.0" y="500">
    <ECAction Algorithm="InitAlg" Output="INITO"/>
   </ECState>
   <ECState Name="E3" x="1000.0" y="600">
    <ECAction Algorithm="RampAlg" Output="CNF"/>
   </ECState>
   <ECTransition Condition="START AND INIT" Destination="E2" Source="START" x="1200.0" y="750"/>
   <ECTransition Condition="START AND REQ" Destination="E3" Source="START" x="1200.0" y="750"/>
   <ECTransition Condition="1" Destination="START" Source="E2" x="1200.0" y="750"/>
   <ECTransition Condition="1" Destination="START" Source="E3" x="1200.0" y="750"/>
  </ECC>
  <Algorithm Name="RampAlg">
   <ST Text="IF T&lt;Duration
THEN&#13;&#10;&#9;Out:=X0+(X1-X0)*ULINT_TO_REAL(T)/ULINT_TO_REAL(Duration);&#13;&#10;&#9;T:=T+
Cycle;&#13;&#10;&#9;Hold:=PV&gt;Out;&#13;&#10;END_IF;"/>
  </Algorithm>
  <Algorithm Name="InitAlg">
   <ST Text="T:=0;"/>
  </Algorithm>
 </BasicFB>
</FBType>
```

Una vez se obtiene dicho código, se comprueba que el programa « 4DIAC » entiende el código sin ningún problema para verificar que el funcionamiento de la traducción ha sido el correcto, obteniendo las « Imágenes 145 y 146 ».

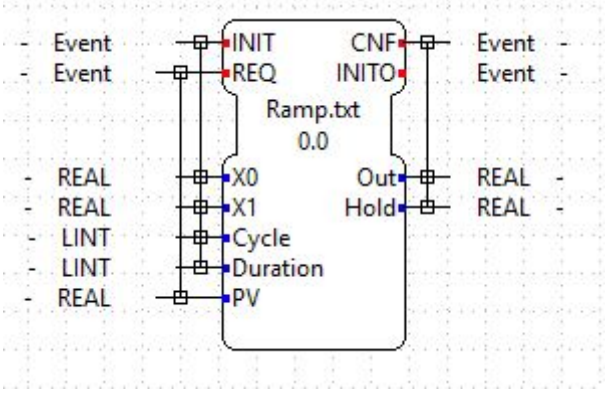

Imagen 145. Ejemplo 3, Bloque de función en « 4DIAC ».

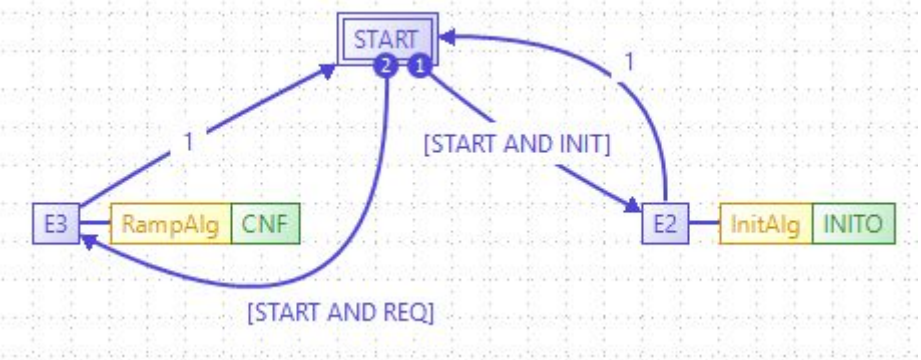

Imagen 146. Ejemplo 3, diagrama de control de ejecución en « 4DIAC ».

3.5.4. Ejemplo 4: Facturación automática

En éste ejemplo se considera un Bloque de Función cuya funcionalidad es realizar la facturación de una maleta en un aeropuerto. Donde el usuario introduce su maleta al principio de una cinta transportadora, introduce su número de identificación « ID » y pulsa un botón que activa el Evento de entrada « Facturar », haciendo así que el « ECC » avance al Estado « Comprobacion » y que se activen tanto el algoritmo « LengthAlg » como el Evento de salida « Comprobar ». En éste punto, el « ECC » espera a que le llegue la señal « BD » que hará que el « ECC » evolucione o bien al Estado « Error », si el « ID » obtenido coincide con la Variable interna « ID_ant » o supera el « Peso_Max », o bien al Estado « Listo », si el « Peso » no supera el « Peso_Max ».

Cuando el « ECC » evoluciona al Estado « Comprobacion », se activa el algoritmo « LengthAlg », el cual calcula la longitud de la cadena « ID » y guarda su valor en la Variable de salida « Longitud ». A su vez, se activa el Evento de salida « Comprobar » para verificar que aparece en la base de datos del vuelo dicha « ID ». Una vez comprobada la información en la base de datos, se genera un Evento de entrada « Listo » con el mismo valor de la Variable de entrada « ID » si es correcto y « 0 » en caso contrario.

De forma similar, al llegar dicho Evento de entrada « BD », el « ECC » puede evolucionar o bien al Estado « Error », donde se activa el Evento de salida «Senyal », o bien al Estado « Listo », donde se activa el Evento de salida « Cinta » que mueve la maleta hacia el compartimento.

La definición gráfica de este Bloque de Función viene declarado por una interfaz externa estática, las Variables internas, el « ECC » y los algoritmos tal y como se muestra en la « Imagen 147 ».

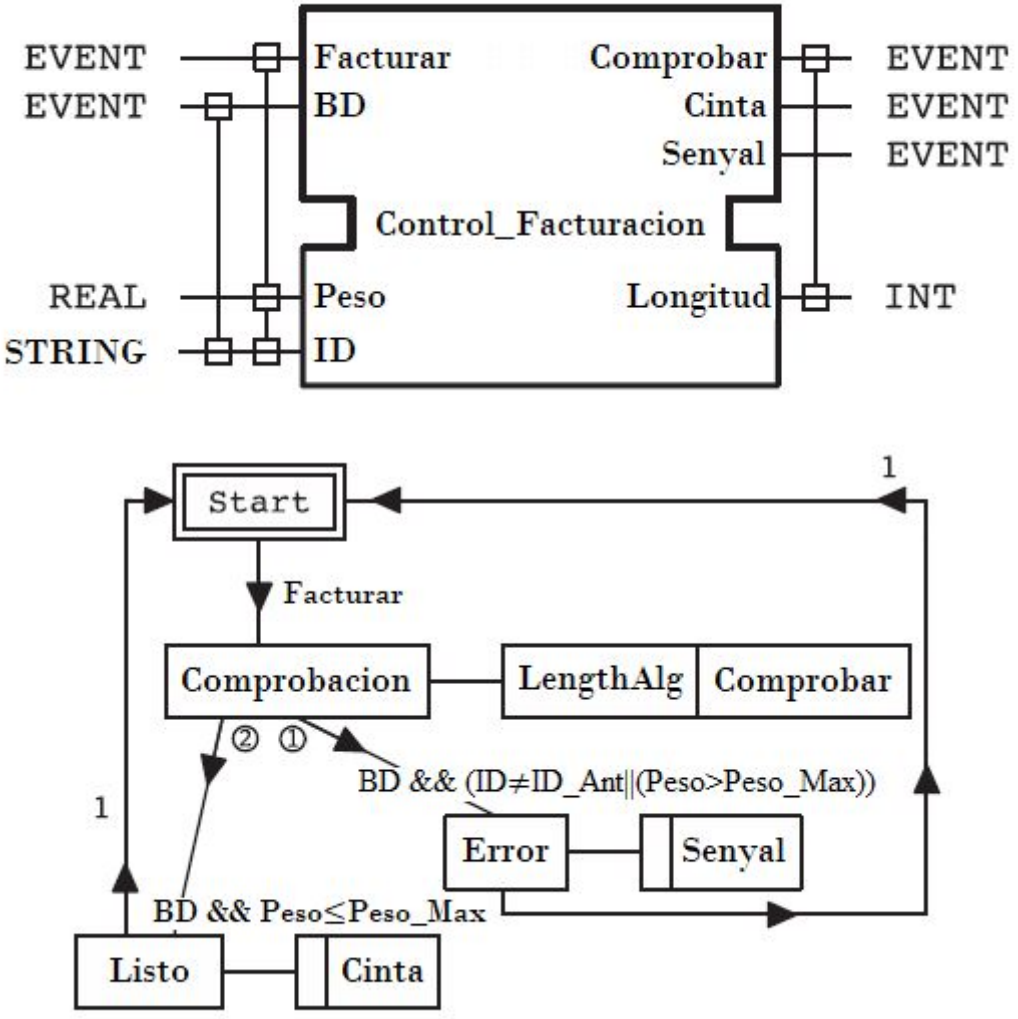

Imagen 147. Ejemplo 4, esquema del Bloque de función y el « ECC ».

La sintaxis textual para describir el Bloque de Función del « Control_Facturacion », que se muestra en la « Imagen 147 », en un « Matlab Function » es la siguiente:

```
%1-DECLARACION DE EVENTOS Y VARIABLES DEL BLOQUE
function [e_Comprobar, e_Cinta, e_Senyal, v_Longitud]  = fcn (e_Facturar, e_BD, v_Peso, v_ID)

%2-DECLARACION DE VARIABLES INTERNAS
persistent Peso_Max ID_Ant;

%3-DECLARACION DEL ESTADO INICIAL
persistent Comprobar Cinta Senyal Longitud Facturar BD Peso ID E CE prev_Longitud;

if isempty(E)
   E =[1 0 0 0]; %Estado de las etapas
   %Variables internas
   Peso_Max = 50.0;
   ID_Ant =  '_';
   %Variables de entrada
   Peso = 0.0;
   ID = '_';
   %Variables de salida
   Longitud = 0;
   %Eventos de entrada
   Comprobar = 0;
   Cinta = 0;
   Senyal = 0;
   %Eventos de salida
   Facturar = 0;
   BD = 0;
end

%4-ACTUALIZACION DE LAS VARIABLES DE ENTRADA (WITH)
if (e_Facturar)
   Facturar = 1;
   ID = v_ID;
end

if (e_BD)
   BD = 1;
   Peso = v_Peso;
   ID = v_ID;
end

%5-DIAGRAMA DE CONTROL DE EJECUCION (ECC)
prev_Comprobar=0;
prev_Cinta=0;
prev_Senyal=0;
CE=[0 0 0 0]; %Copia de etapas
while ~isequal(CE,E)
   CE=E;

   if CE(1) && Facturar
      E(1)=0;
      E(2)=1;
   end

   if CE(2) && BD && (Peso<=Peso_Max)
      E(2)=0;
      E(3)=1;
   end

   if CE(2)&& BD && ((Peso>Peso_Max) || (ID~=ID_Ant)) && ~E(3)
      E(2)=0;
```

```
    E(4)=1;
  end

  if CE(3)
    E(3)=0;
    E(1)=1;
  end

  if CE(4)
    E(4)=0;
    E(1)=1;
  end

  %6-EJECUCION DE ALGORITMOS
  Alg_LengthAlg = E(2) && ~CE(2);

  %7-DECLARACION DE ALGORITMOS
  if (Alg_LengthAlg) %Algoritmo LengthAlg
    Longitud = length(ID);
    ID_Ant = ID;
  end

  %8-EJECUCION DE EVENTOS DE SALIDA
  Comprobar = E(2);
  Cinta = E(4);
  Senyal = E(3);

  %9-EJECUCION DE VARIABLES DE SALIDA (WITH)
  if (Comprobar)
    v_Longitud = Longitud;
  else
    v_Longitud = prev_Longitud;
  end
  prev_Longitud = v_Longitud;

  %10-FINAL
  Facturar = 0;
  BD = 0;

  if Facturar && ~prev_Facturar
    prev_Facturar = 1;
  end

  if BD && ~prev_BD
    prev_BD = 1;
  end
end

e_Comprobar = prev_Comprobar;
e_Cinta = prev_Cinta;
e_Senyal = prev_Senyal;
v_Longitud = prev_Longitud;
```

Dando lugar al Bloque « Matlab Function » que aparece en la « Imagen 148 ».

Imagen 148. Ejemplo 4, Bloque de función en Simulink.

Una vez traducido el Bloque de Función, se obtiene que la sintaxis textual que describe la estructura del « Control_Facturacion » en « 4DIAC » es la siguiente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">
<FBType Comment="Control_Facturacion" Name="Control_Facturacion.txt">
 <Identification Standard="61499-2"/>
 <VersionInfo Author="FB_SLK24DIAC" Date="5-11-2017" Organization="UJI" Version="0.0"/>
 <InterfaceList>
  <EventInputs>
   <Event Name="Facturar" Type="Event">
    <With Var="ID"/>
   </Event>
   <Event Name="BD" Type="Event">
    <With Var="Peso"/>
    <With Var="ID"/>
   </Event>
  </EventInputs>
  <EventOutputs>
   <Event Name="Comprobar" Type="Event">
    <With Var="Longitud"/>
   </Event>
   <Event Name="Cinta" Type="Event"/>
   <Event Name="Senyal" Type="Event"/>
  </EventOutputs>
  <InputVars>
   <VarDeclaration Name="Peso" Type="REAL"/>
   <VarDeclaration Name="ID" Type="STRING"/>
  </InputVars>
  <OutputVars>
   <VarDeclaration Name="Longitud" Type="INT"/>
  </OutputVars>
 </InterfaceList>
 <BasicFB>
  <InternalVars>
   <VarDeclaration InitialValue="50.0" Name="Peso_Max" Type="REAL"/>
   <VarDeclaration InitialValue="" Name="ID_Ant" Type="STRING"/>
  </InternalVars>
```

```
  <ECC>
    <ECState Name="START" x="1000.0" y="400"/>
    <ECState Name="E2" x="1000.0" y="500">
      <ECAction Algorithm="LengthAlg" Output="Comprobar"/>
    </ECState>
    <ECState Name="E3" x="1000.0" y="600">
      <ECAction Output="Senyal"/>
    </ECState>
    <ECState Name="E4" x="1000.0" y="700">
      <ECAction Output="Cinta"/>
    </ECState>
    <ECTransition Condition="START AND Facturar" Destination="E2" Source="START" x="1200.0" y="850"/>
    <ECTransition Condition="E2 AND BD AND (Peso&gt;Peso_Max)" Destination="E3" Source="E2" x="1200.0"
y="850"/>
    <ECTransition Condition="E2 AND BD AND ((Peso&gt;Peso_Max) OR (ID<>ID_Ant))" Destination="E4"
Source="E2" x="1200.0" y="850"/>
    <ECTransition Condition="1" Destination="START" Source="E3" x="1200.0" y="850"/>
    <ECTransition Condition="1" Destination="START" Source="E4" x="1200.0" y="850"/>
  </ECC>
  <Algorithm Name="LengthAlg">
    <ST Text="Longitud:= LEN(ID);"/>
  </Algorithm>
 </BasicFB>
</FBType>
```

Por último, se comprueba que el programa « 4DIAC » entiende el código sin ningún problema para verificar que el funcionamiento de la traducción ha sido el correcto, obteniendo las « Imágenes 149 y 150 ».

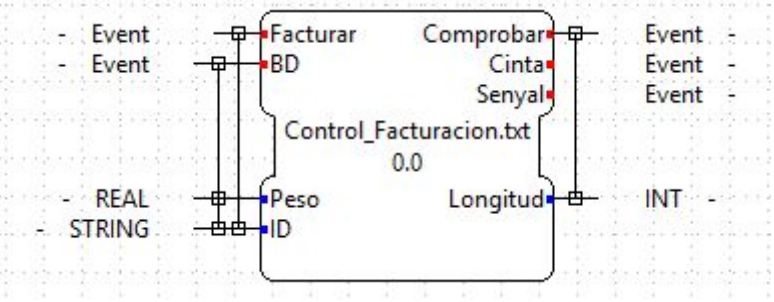

Imagen 149. Ejemplo 4, Bloque de función en « 4DIAC ».

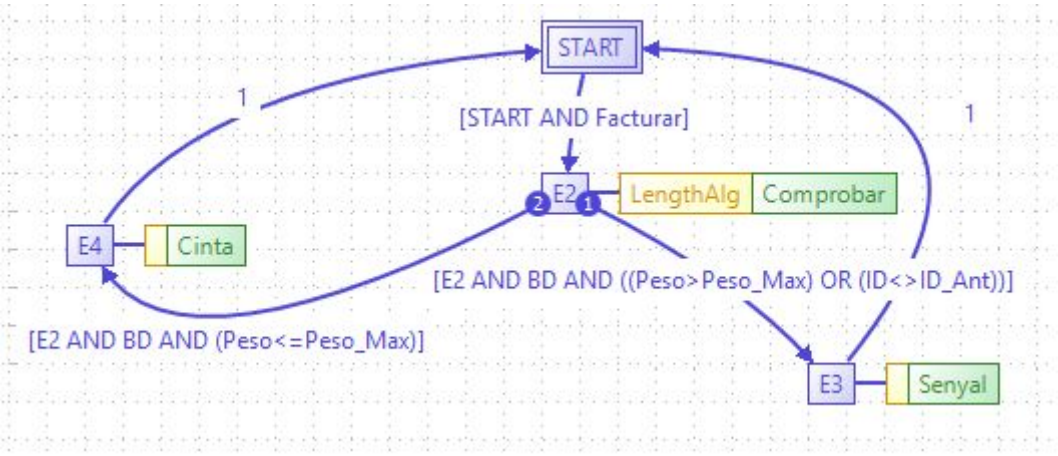

Imagen 150. Ejemplo 4, diagrama de control de ejecución en « 4DIAC ».

4. PRESUPUESTO

En este punto se detalla el presupuesto correspondiente al desarrollo del proyecto que se ha descrito, el cual se encuentra dividido en cuatro partes:

1. Gastos por mano de obra: Horas trabajadas por el ingeniero.
2. Gastos en Hardware: Equipo necesario.
3. Gastos en Software: Gastos por licencias de los programas que son necesarios.
4. Gastos en otros servicios: Gastos de Electricidad y acceso a internet.

Presupuesto			
Cantidad	Componente	Precio	
		Coste unitario	Coste total
160	Horas que dedica el ingeniero	15,20€	2.432,00€
Gastos en mano de obra			2.432,00€
1	Ordenador	700€	700€
Gastos en Hardware			700€
1	Licencia de Matlab	2.000€	2.000€
1	Licencia de Simulink	3.000€	3.000€
Gastos en Software			5.000€
35,28	Consumo Eléctrico	0,145€	5,12€
1	Acceso a internet	41,30€	41,30€
Gastos en otros servicios			46,42€

Tabla 5. Presupuesto, desglose por tipos de gasto.

Presupuesto	
Subtotal	8.178,42€
Imprevistos (15%)	1.226,76€
Ganancia (25%)	2.044,60€
TOTAL	11.449,78€

Tabla 6. Presupuesto, cálculo del total.

Los gastos asociados a Hardware y Software se imputan como costes de amortización a un periodo de 4 años, dado que el tiempo de desarrollo de una Aplicación (Conjunto de Bloques de Función), Se ha realizado el cálculo de los costes del presupuesto a dicho periodo (1 mes).

A todo ésto, cabe destacar que al invertir en la IEC-61499, se facilita el trabajo al ingeniero, además los bloques de función son reutilizables gracias al control distribuido, lo cual hace que el presupuesto para cualquier proyecto sea menor. Ya que el ingeniero necesitará menos horas para el mismo trabajo. Además, Gracias a que se pueda simular mediante simulink, se evitan posibles fallos, reduciendo así aún más el tiempo necesario para la realización del programa.

5. CONCLUSIONES

Para la realización de éste proyecto, ha sido necesario conocer el estándar IEC-61499, el cual define un entorno para el desarrollo de un control distribuido mediante bloques de función. Para el cual, se ha conseguido la integración de un método de simulación del mismo mediante el programa Matlab y su herramienta Simulink. Así mismo, se ha descrito un riguroso método de transformación entre las plataformas Simulink y « 4DIAC ». Mostrándose una serie de reglas para dicha transformación. De la misma forma, se ha realizado una transformación mutua entre ambas plataformas con el objetivo de facilitar la simulación de sistemas distribuidos. Así mismo, éste proyecto verifica la fiabilidad tanto de éstos Bloques de función como de las herramientas desarrolladas, tal y como se comprueba en los ejemplos del apartado « 3.3. Ejemplos prácticos ».

El método descrito facilita el diseño de sistemas distribuidos, ya que introduce la posibilidad de simular y analizar el comportamiento de un Bloque de función. Así pues, el hecho de tener una traducción automatizada reduce el tiempo y el esfuerzo dedicado a la hora de desarrollar los Bloques de función.

Por todo lo descrito, este proyecto se considera un avance en el despliegue de éste emergente estándar. Ya que, tal y como se muestra en el « 4. Presupuesto », es viable invertir en el control distribuido mediante el estándar IEC-61499 por su capacidad de reutilizar bloques de función, por la facilidad a la hora de realizar el programa y por la fiabilidad poder simular los bloques de función y comprobar su correcto funcionamiento. Incentivando así su uso por parte de las industrias y pudiendo de ésta manera explotar al máximo el potencial que ofrece éste estándar.

6. BIBLIOGRAFÍA

- **Alois Zoitl y Robert Lewis (2015) :** « Modelling control systems using IEC61499 ».

- **Chia-han (John) Yang y Valeriy Vyatkin (2009) :** « Automated Model Transformation between MATLAB Simulink/Stateflow and IEC 61499 Function Blocks ». Department of Electrical and Computer Engineering, University of Auckland, New Zealand.

- **Chia-han (John) Yang y Valeriy Vyatkin (2012) :** « Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems ».Department of Electrical and Computer Engineering, University of Auckland, New Zealand.

- **Esteban Querol, Julio Ariel Romero, Antonio M. Estruch y Fernando Romero (2012) :** « Norma IEC-61499 para el control distribuido. Aplicación al CNC ». Dep. Enginyeria de Sistemes Industrials i Disseny. Universitat Jaume I.