PróximaGSM v1.0

Gestión de alarmas técnicas

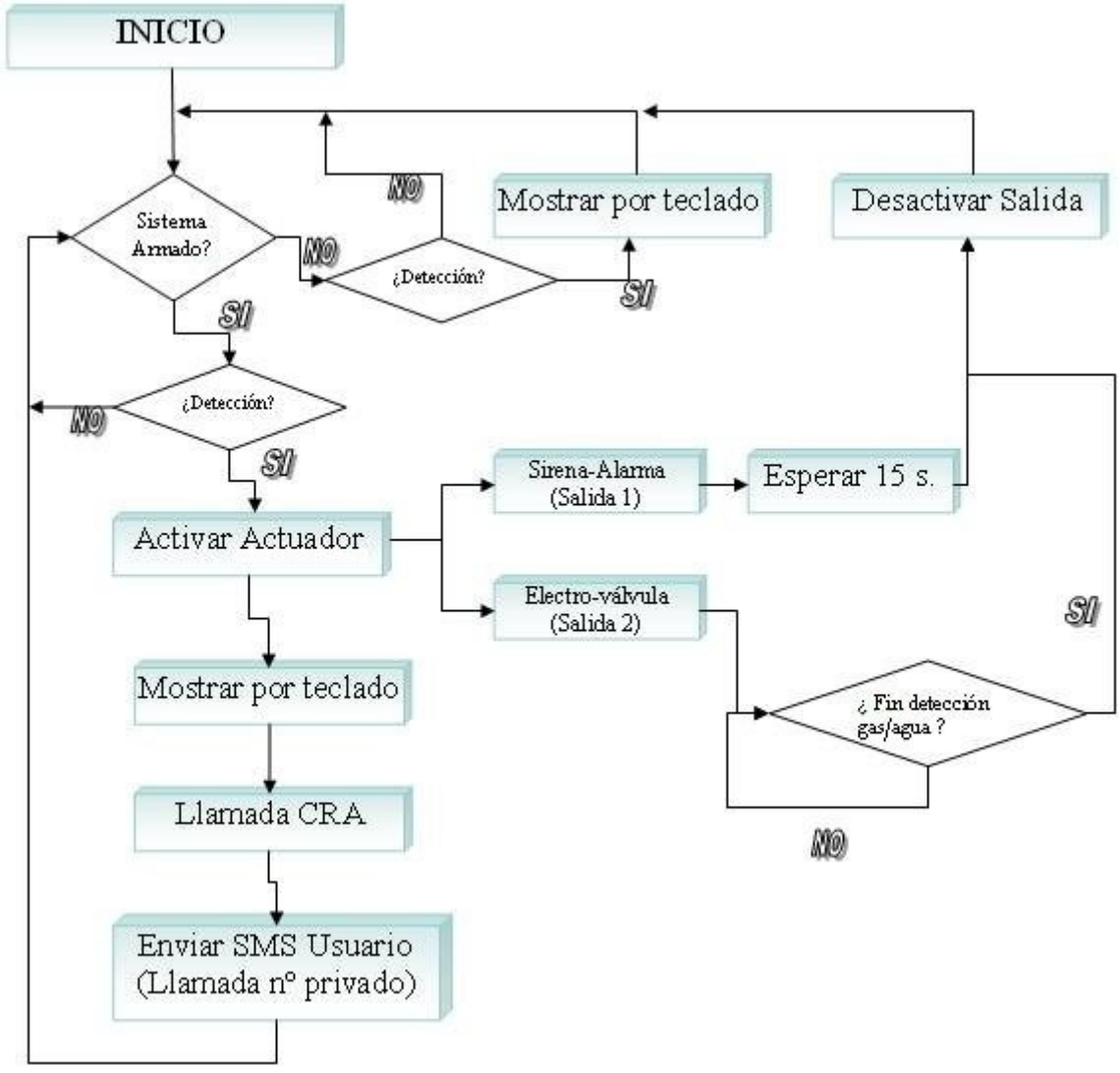
Autor: César Cuadros Cortina

ÍNDICE

1.- Introducción

1.1.- Idea básica

El proyecto PróximaGSM fusiona la capacidad de controlador de la Remota con la gestión de seguridad basada en alarmas técnicas. A través del software se ha desarrollado un módulo que hace de central gestora de alarmas, la cual es capaz de recibir las señales que provienen de los dispositivos receptores, tales como sensores de movimiento, detectores de humo, inundaciones, etc... y actuar en consecuencia, haciendo sonar una sirena, abriendo y cerrando válvulas según convenga, realizando llamadas a centrales gestoras de alarmas, a usuarios y enviando mensajes SMS.



1.2.- Características generales

El protocolo implementado trabaja mediante paquetes de información que se transmiten sobre un BUS permitiendo la comunicación de diferentes módulos, dando lugar a una funcionalidad expansible. La versión de PróximaGSM 1.0 incluye un Teclado TCL con pantalla LCD, un módulo Receptor Vía Radio (VR), un módulo GSMpro v5.12 y un controlador LPC-USB.
A través del teclado se visualiza el estado del sistema y se pueden configurar detalles de personalización. El receptor VR permite el uso de dispositivos inalámbricos que pueden trabajar en un amplio radio de acción, el módulo GSM es el encargado de realizar llamadas telefónicas a una Central Receptora de Alarmas (CRA), a usuarios particulares y enviar mensajes SMS de información de estado del sistema. El controlador USB es completamente opcional, se utiliza para configurar el sistema a través de un programa visual (JR-Eligth) conectado a un PC mediante un cable USB, todas las configuraciones necesarias se pueden realizar desde el teclado TCL.
Estos módulos están conectados a la Próxima a través de un BUS RS-485 half duplex, y los dispositivos de detección se conectan a las entradas del canal B de la Próxima.
Los disposivos utilizados en esta versión son 2 detectores volumétricos (JR-FERA y JR FERA-DOMO) cableados, y uno inalámbrico (JR-FERA VR); un detector de inundaciones inalámbrico (JR TR-CN), y un cierre magnético.
Una sirena conectada a la salida 1 de la Próxima.
Los dispositivos inalámbricos van alimentados con una batería de litio de 3V.
Tanto los módulos como los dispositivos de detección cableados funcionan con una alimentación de 13'8 V DC a 1 Amp máximo, facilitados a través de un transformador (JR-FASW-1A) conectado a la red eléctrica que acepta una entrada de 85 a 220 V AC 50/60Hz , e intensidad máxima de 300mA.
La Próxima funciona con una alimentación de 18 a 72 V DC, con un consumo máximo entre 2 y 3 Watios.

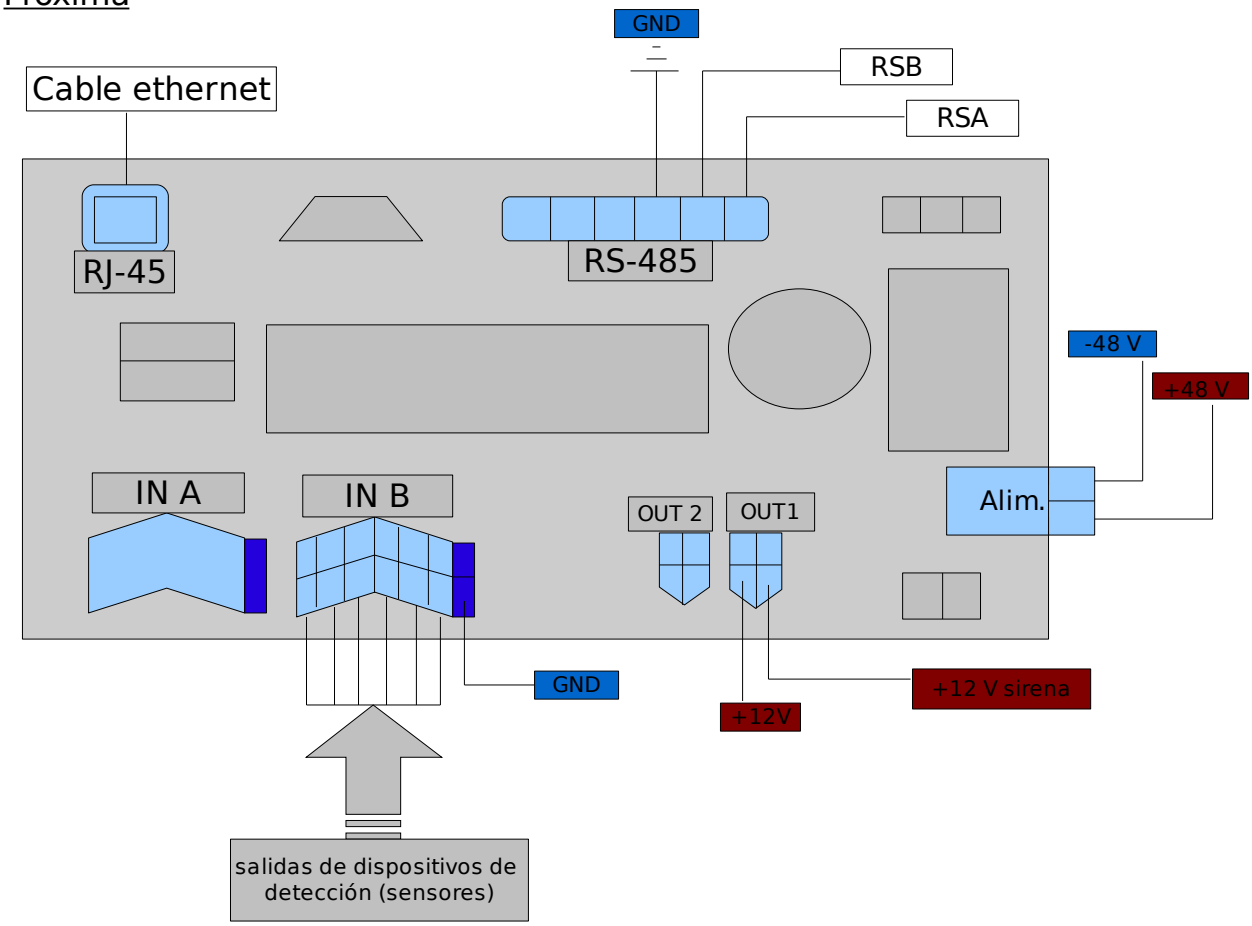
1.3.- Documentación necesaria para el desarrollo del software

Se ha desarrollado una aplicación sobre un sistema operativo Linux Kernel 2.6 Embedded, para ello se ha utilizado como entorno de enlace con el kernel el software LxNETES UNC90, es necesaria la Guía de Usuario (LxNETES TM User's Guide UNC90, Making DEVICE NETWORKING easy, de Digi International Inc. 2005) para realizar la instalación de LxNETES.
El entrono de desarrollo de software utilizado ha sido ***Code:Blocks*** *The open source, cross-platform IDE; svn 5273.*
Los protocolos de comunicaciones han sido facilitados por JR Sistemas de Seguridad, se trata del protocolo *GATEWAY EIB PARA CENTRALES CENTRUM* y *Protocolo 485 CSMA para Bus Centrum*, donde se recogen las especificaciones de los paquetes de datos, así como su formato para los diferentes tipos de tramas, sus funciones, el control de acceso al medio y el resto de definiciones necesarias para entender el protocolo.

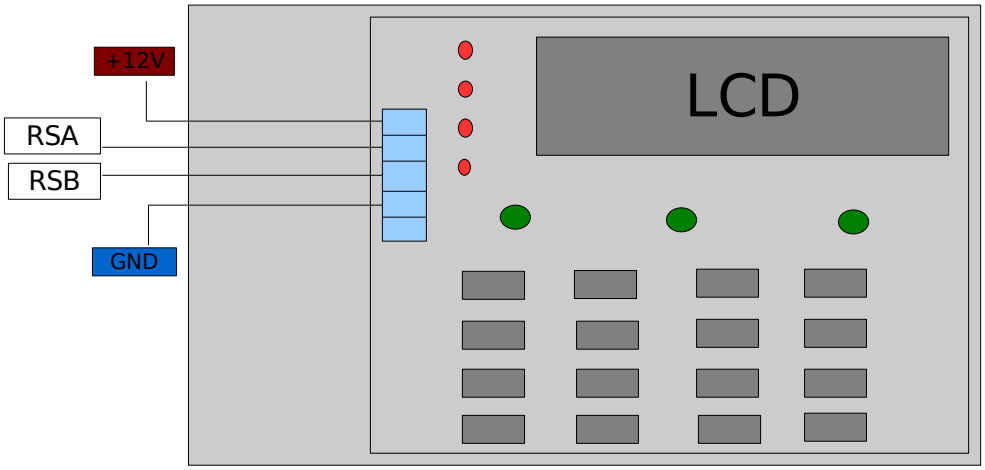
2.- Desarrollo de la aplicación

2.1.- Nivel Físico: Conexionado y cableado.

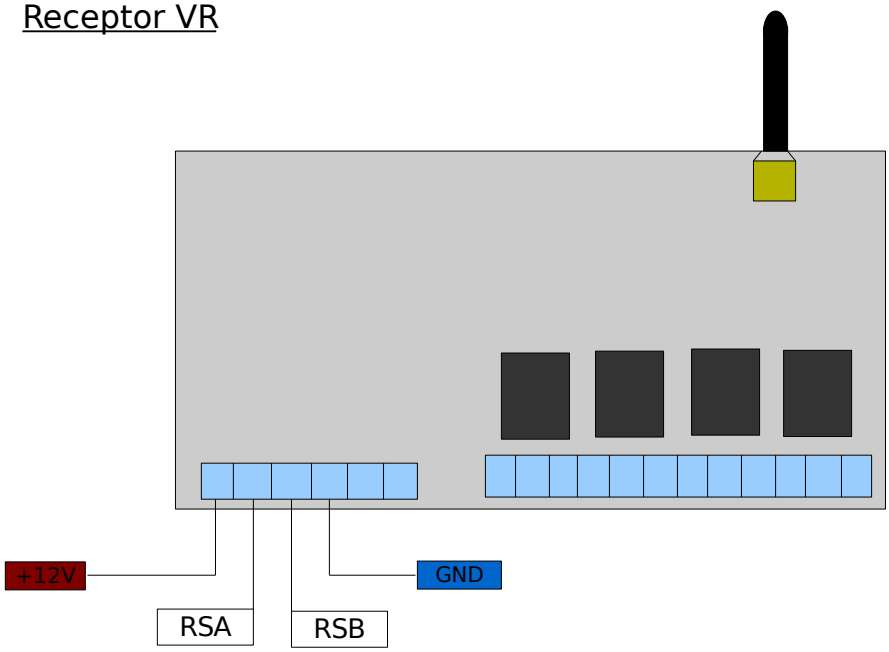
Próxima



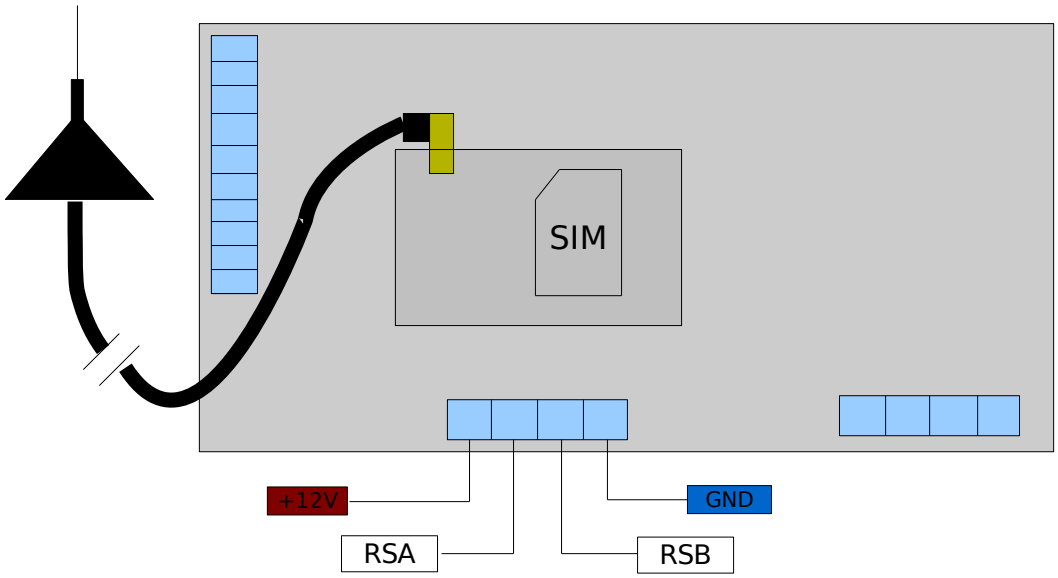
Teclado TCL



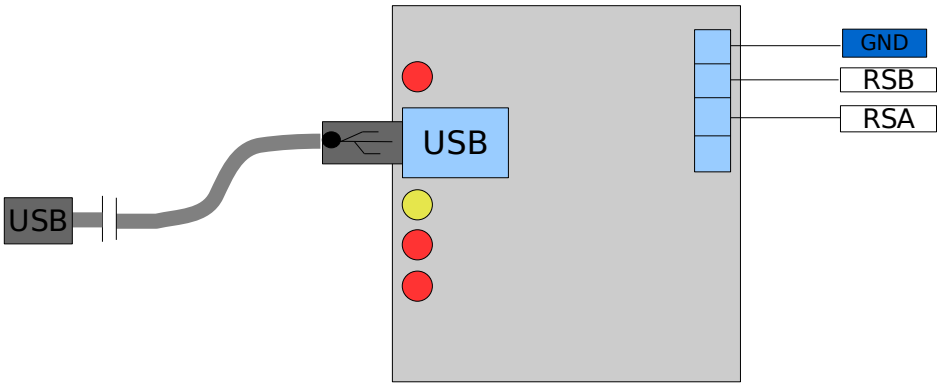
Receptor VR



Módulo GSM



Módulo LPC USB

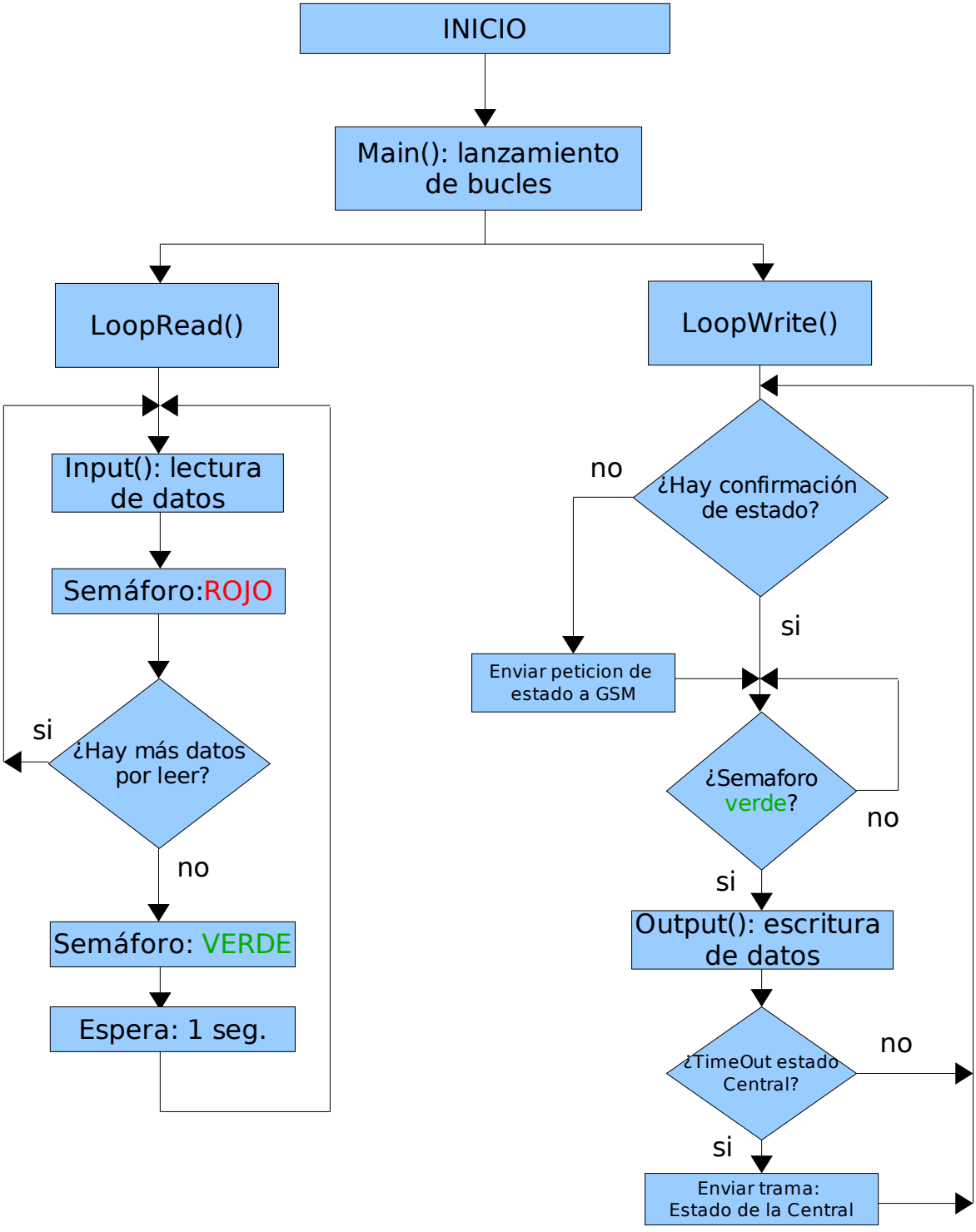


2.2.- Nivel de Enlace de datos: Control de flujo, acœso al medio y errores de tramas.

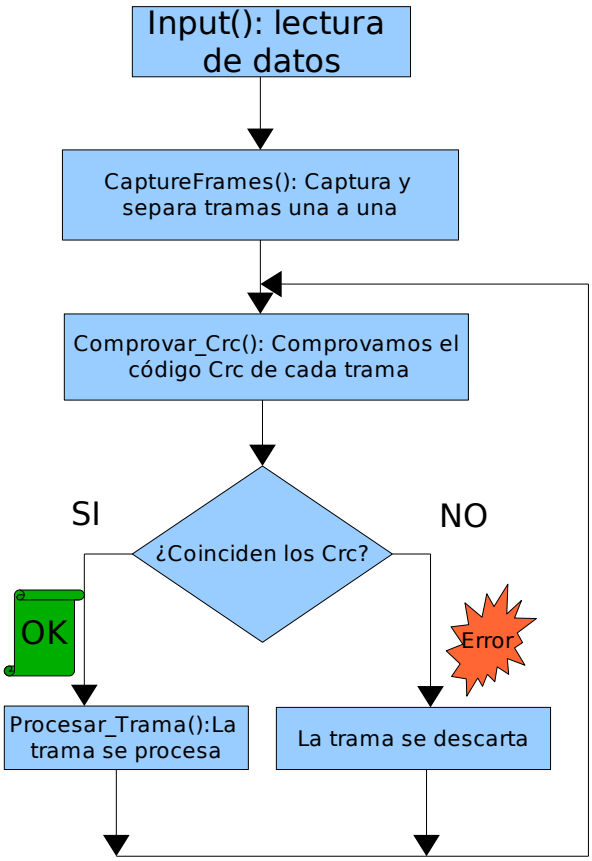
La comunicación entre los diferentes módulos se realiza a través del BUS RS-485, sin embargo, al tratarse de un bus compartido por varios módulos, sólo puede escribir uno al mismo tiempo, es por ello que se debe tratar el acceso al medio con tal de minimizar las colisiones entre paquetes. Los módulos complementarios (Receptor VR, GSM, etc..) tienen implementado su control de acceso al medio, por lo tanto pasamos a definir el control de acceso que realiza la Próxima, el cual es bastante sencillo puesto que la velocidad del procesador es diferente que la de los otros módulos, hecho que ayuda a evitar colisiones.
La gestión entre lectura y escritura viene definida en el fichero proximaGSM.c, mediante un mecanismo de exclusión mútua:
- Se ejecuta el bucle *LoopRead()* constantemente, encargado de detectar pulsos eléctricos que entran porel BUS a través de la función Input() y ReadComPort() y llamadas a sistema (en fichero proximaGSM.c). Realiza una función de escucha o lectura de datos, activando un semáforo (variable TimerOK), el cual bloquea al proceso que escribe siempre que hay datos entrantes. Las tramas que se reciben son capturadas y separadas una a una mediante la función CaptureFrames(), se comprueban y en caso de ser correctas pasan a procesarse una a una(Procesar_Trama()). En cuanto el bus queda libre, este bucle de lectura pone el semáforo en verde durante un segundo aproximandamente, permitiendo que el proceso de escritura realice la transmisión de datos que tenga pendiente.
- De forma concurrente se ejecuta el bucle *LoopWrite()*, este es el encargado de transmitir datos a traves del BUS RS-485. Esta función comprueba 3 puntos principales: en cada iteración verifica si se ha recibido confirmación de estado por parte del módulo GSM, si no es así se envia una petición para que este módulo conteste. Seguidamente comprueba el estado del semáforo hasta que se pone verde, en tal caso cede el control a la función de escritura de tramas generales en el bus *Do_frames(),* implementada en el fichero serial.c, esta función llama al resto de funciones encargadas de montar los diferentes paquetes de datos. . Finalmente, realiza un control temporal para enviar una trama del estado del sistema de alarmas cada 8 minutos aproximadamente (por convenio, para evitar un bloqueo de estado del sistema).

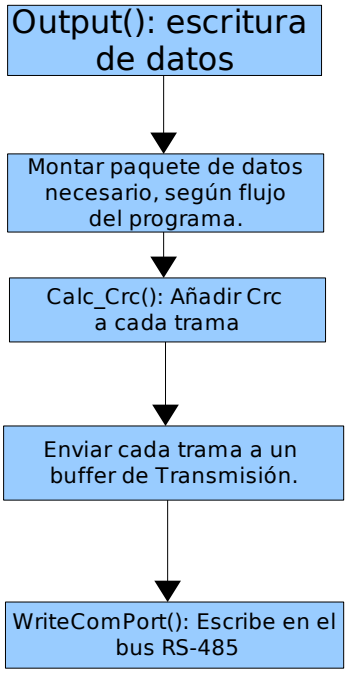
Cada paquetede datos que se recibe pasa por un control de errores para verificar que no haya sufrido daños (funciones CaptureFrames() y Comprovar_Crc() en serial.c).
Las tramas que se envían son construidas junto unos valores calculados mediante una función de código de redundancia cíclica (CRC).
Este código se vuelve a calcular cuando se recibe la trama y se compara con el código que venía en el paquete de datos, si son diferentes significa que la trama ha sido corrompida debido a alguna colisión y se descarta. La función encargada de realizar los cálculos de CRC es Calc_Crc() y se encuentra en el fichero Crc.c .

DIAGRAMA DEL CONTROL DE FLUJO, ACCESO AL MEDIO Y ERRORES



NOTA: LoopRead() se ejecuta primero, y LoopWrite() inmediatamente después, de manera que al principio este último siempre encuentra el semáforo en rojo hasta que se realiza la primera comprobación de datos por leer. LoopWrite() se queda en un bucle anidado hasta que el semáforo le cede el paso.





NOTA: Las tramas descartadas (generalmente corruptas a causa de alguna colisión) no reciben confirmación y por tanto serán reenviadas.

2.3.- Nivel de Red: Procesado y construcción de paquetes de datos.

2.3.1.- Procesado de Tramas

Todas las tramas que transmiten los módulos (Teclado, GSM, ReceptorVR) entran a la próxima a través del puerto de comunicaciones correspondiente al canal de recepción del BUS RS-485. Como ya se ha comentado, la Próxima está constantemente en escucha, y en cuanto hay datos entrantes se guardan en un búfer de recepción para ser tratados paquete a paquete.

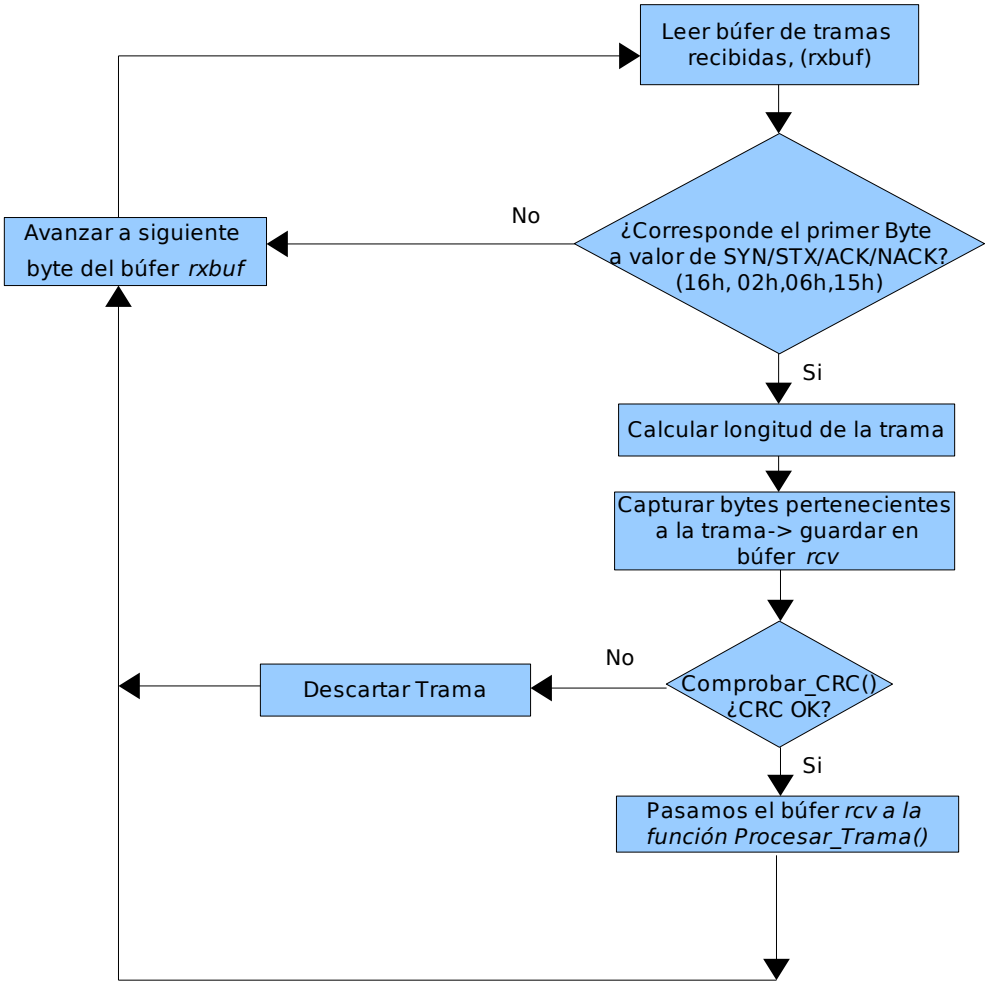
Input() es la función que hace la llamada a la responsable de la lectura de datos entrantes: ReadComPort(), y ésta realiza una llamada a sistema (“read”) para leer los datos que se reciben, los cuales son guardados en un búfer de recepción. Este búfer se retorna a la función Input(), y ésta le pasa el búfer en cuestión a otra función llamada CaptureFrames() para fragmentar dicho búfer y separar todas las tramas. De este modo Input() puede volver a realizar las siguientes lecturas, reescribir el búfer con los nuevos datos y volver a repetir el proceso de captura de tramas sin perder datos.

- CaptureFrames(): Recibe una estructura que corresponde al búfer con los datos que se han leído por el puerto de entrada 485. Este búfer contiene todas las tramas leídas hasta el momento; el resultado es la fragmentación del búfer para guardar cada trama por separado en un vector y así poder diferenciar los paquetes uno a uno. Una vez hemos separado las tramas se pasará a comprobar si están corruptas o han llegado bien. Encaso de que la trama sea correcta se le pasa a la función Procesar_Trama(). Para entender bien el proceso de Captura de tramas necesitamos conocer el formato de las tramas, y la función de sus campo. (Consultar - Anexo 1)

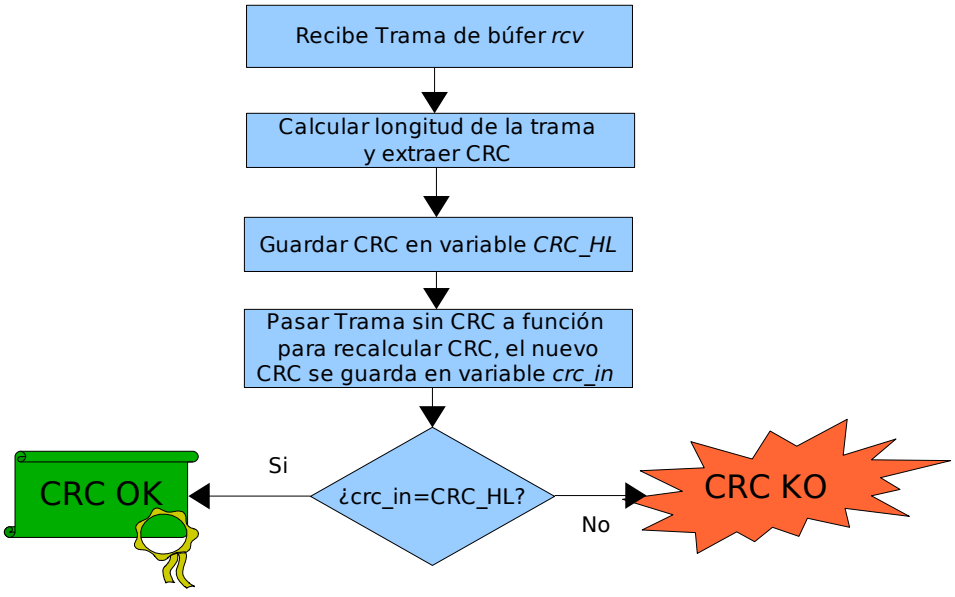
- Procesar_Trama(): Esta función analiza los bytes que pertenecen a la cabecera, origen, destino e indetificador de cada trama que le entra. Escribe mensajes por pantalla según el tipo de trama. Para calcular el Origen, Destino e Identificador de Función llama a la función Extrae_Dirección(), la cual verifica las direcciones y funciones, mostrando mensajes con el origen, el destino y el tipo de función. Una vez verificado el identificador de trama, actúa en consecuencia según el tipo de trama recibida:
  - Si es On_User: envía paquete para armar o desarmar sistema.
  - Si es AREA_ARM: envía paquete para forzar el armado del sistema.
  - Si es AREA_DISARM: envía paquete para forzar el desarmado del sistema.
  - Si es VOID: envía paquete con el estado de la central
  - Si es paquete de órdenes: responde con paquete test_answer o con memoria de alarmas, según tipo de orden.
  - Si es confirmación positiva de trama (ACK), cambia variables para dejar de enviar tramas al destino correspondiente.
  - Si se trata de un paquete de origen VR, llama a la función encargada de procesar los paquetes Vía Radio.

- Paquetes_VR(): Esta función es llamada cuando se detecta que el origen del paquete es el Receptor VR, dicha función comprueba que se trate de una detección de algún sensor VR o de su restauración (fin de detección), y seguidamente se modifica una variable para gestionar las alarmas VR según proceda (activar/restaurar alarma). Cuando se procesa el paquete VR se debe enviar confirmación al módulo Receptor VR, en caso contrario dicho módulo puede comenzar a inundar el sistema con tramas de detección VR.
En esta versión se trabaja con un sensor de detección de movimiento y un sensor de inundaciones, ambos inalámbricos. Este módulo es perfectamente ampliable con más dispositivos de detección (detector de humo, rotura de cristales, sensor de presión…)

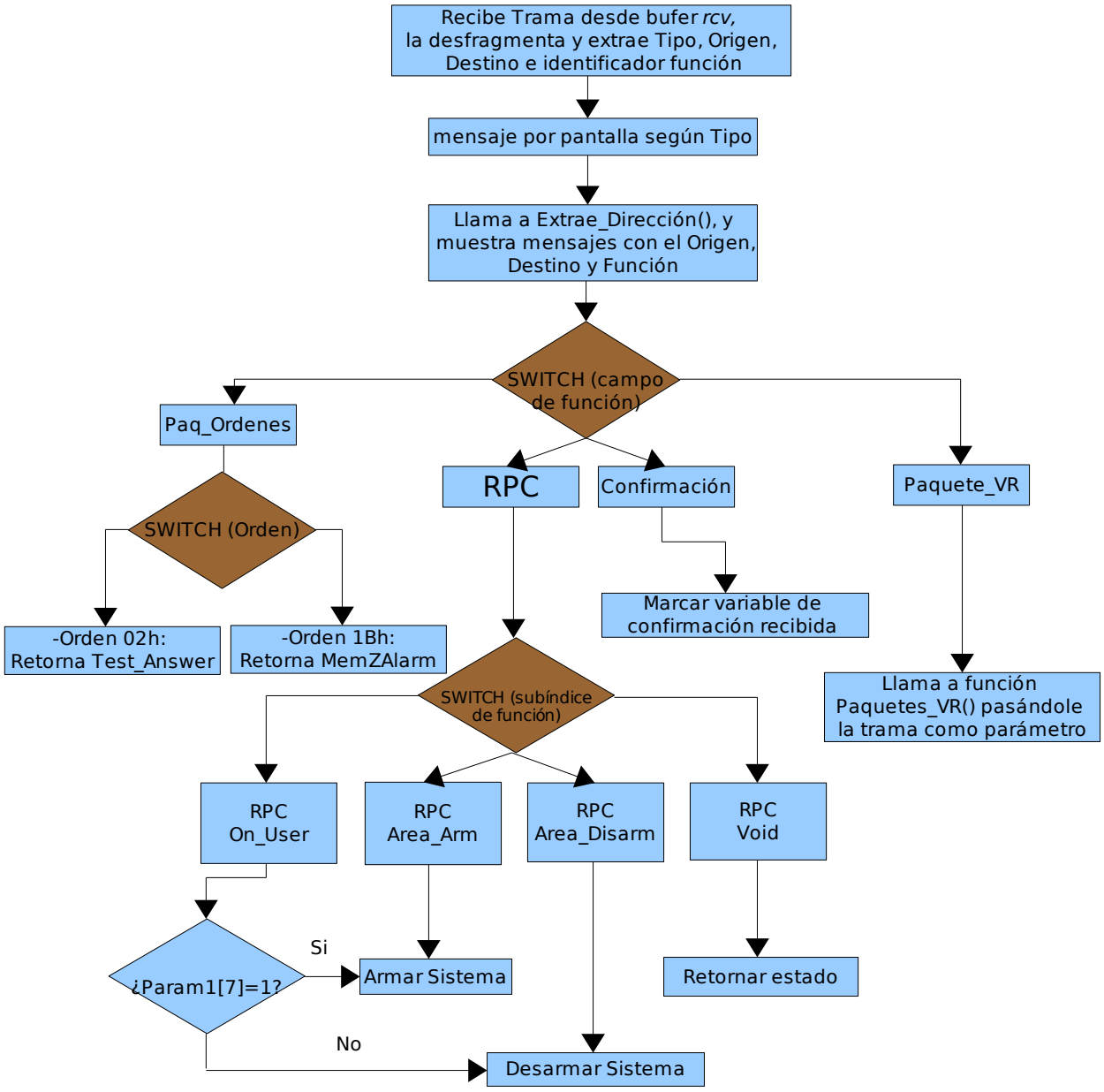
CAPTURA DE TRAMAS
[CaptureFrames()]



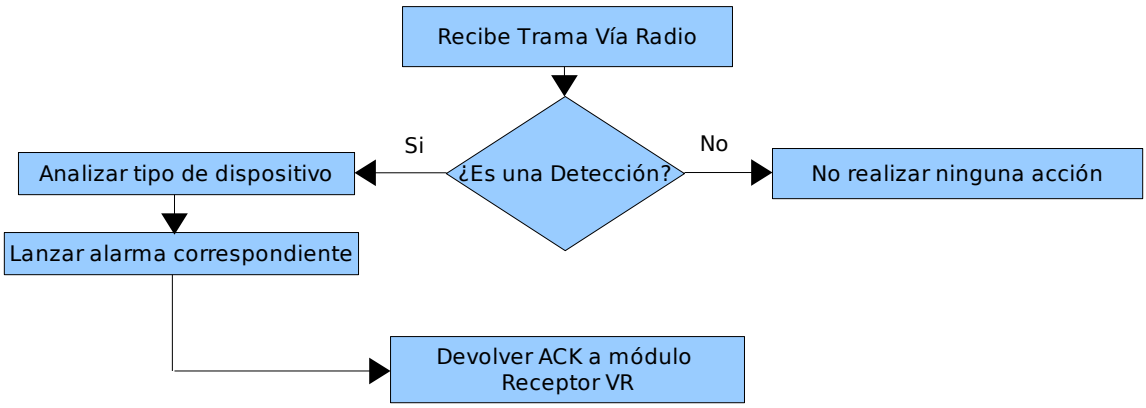
[Comprovar_Crc()]



PROCESADO DE TRAMAS
[Procesar_Trama()]



[Paquetes_VR()]



2.3.2.- Construcción de Tramas

Los paquetes de datos que salen de la Próxima hacia los módulos deben ser construidos en base a criterios de reacción. Es decir, no se genera tráfico porque sí, sino que los paquetes que se envían son respuestas a peticiones, confirmaciones o consecuencias de entradas espontáneas por activación de algún dispositivo de seguridad.
Nos encontramos con diferentes tipos de tramas necesarias para la comunicación con los módulos, cada una de las diferentes tramas se construyen en funciones específicas que comentaremos a continuación, siguiendo todas ellas un criterio común de construcción:

- cuando se llama a una función para construir una trama, se le pasa un búfer () al que se le irá introduciendo datos en forma de bytes, empezando por los campos que ya conocemos: Tipo de trama, Destino, Origen; y continuando con el resto: identificador de función (si procede), datos (si procede) y CRC, respetando siempre este orden de campos, ya que forma parte del protocolo de comunicaciones.
  Una vez la trama está construida, se envía a través de la función de Output() y se confirma la transmisión con éxito mediante un mensaje por pantalla.

Las razones para la construcción de tramas pueden ser varias: si se detectan entradas de detecciones nuevas (cambios de estado), si se debe ejecutar alguna activación de salida (actuadores), para notificar eventos (conexión o desconexión de un usuario, activación o restauración de alarmas, etc...), estas causas vienen definidas en el nivel de aplicación.

Los diferentes paquetes de datos que se construyen en esta versión y su definición (llamada, causa y consecuencias) son los siguientes:

- Paq_RPC(): Esta función monta diferentes tipos de tramas que realizarán llamadas de procedimiento remoto, se construyen desde la Próxima, en esta versión son:

  - función SET_ALARM (índice 0Ah): Esta función activa la alarma de la zona que se desee. Se llama desde la función Do_Frames(), la cual es lanzada desde el thread de escritura LoopWrite(). Para construir este paquete es necesario que se produzca una o más detecciones, las cuales se recogen por las entradas del sistema y quedan registradas en una variable global llamada *vector_Detecciones,* esta variable contiene todas las detecciones de los dispositivos del sistema.

  - función SEND_SMS (índice 0Fh): se ejecuta la orden de enviar un mensaje SMS a través del módulo GSM. Los caracteres que se envían deben haber sido transferidos

previamente a la memora RAM del GSM, a partir de la posición 62. Acepta un máximo de 24 caracteres.

- función VOID (ídince FFh): provoca el retorno del estado del módulo que la recibe. Este módulo debe responder obligatoriamente con una trama de estado.

- Estado_Central(): Esta función construye la trama que contiene el estado actual de la central. Se llama desde 4 puntos diferentes del programa:

  1) desde la función Procesar_Trama(), cuando se recibe un paquete RPC con índice FFh (Void, retorno de estado);

  2) desde la función Detección_entradas(), la cual es llamada a su vez en cada iteración del thread LoopEntradas(), sólo si se ha producido alguna detección, en este caso puede parecer redundante pero no lo es, ya que este paso se realiza también en Do_Frames(), lo que se consigue es un mejor sincronismo con la realidad, ya que Do_Frames() debe esperar a que se le ceda el turno para que se ejecute (Semáforo), y podrían haber retrasos no deaseados entre que se produce una detección y se informa a los teclados;

  3) desde Do_Frames(), en caso que se produzca una detección nueva, se envía un total de 5 tramas de estado, para asegurar su recepción, se controlan las repeticiones para evitar exceso de tráfico;

  4) desde el thread LoopWrite(), donde hay un control de tiempo que lanza este paquete cada 8 minutos aproximadamente, con la idea de informar del estado actual y así evitar bloqueos permanentes de estado.

- Paq_TestAns(): Monta la trama que representa la respuesta a una orden de test, llamada desde Procesar_Trama(), al detectar que un paquete entrante era paquete de órdenes con petición de Test. Significa que existe (a modo ping). Se rellena con la identidad del dispositivo que la envía.

- Paq_MEMZALARM(): Crea el paquete de datos que informa de la memoria de Alarmas pendientes desde la Próxima. Se envía paquete tipo 0x20 con las máscaras de las alarmas de zona 1..8 y 9..16. Es llamada desde Procesar_Trama(), al detectar que un paquetes entrante era paquete de órdenes con petición de Memoria de alarmas.

- Paq_NotifEvent(): Monta la trama encargada de notificar al interfaz EIB de cualquier evento (activación/desactivación) de alarmas técnicas. Incluye estructura con tipos de evento, tipo de alarma, y detalles de usuario, área, hora y fecha del evento. Se llama desde 3 puntos del programa:

  1) desde la función On_UserResult, cuando se produce una conexión y desconexión del sistema;

  2 y 3) también es llamada desde los threads LoopSalidas() y LoopAlarmas(), notificando de que se ha activado o restaurado alguna alarma o electroválvula. Estos paquetes hacen de disparador para llamadas a particulares y a la Central Receptora de Alarmas si esta opción ha sido activada en la configuración del GSM (a través de JR eLight o por teclado TCL).

- Paq_TrasnferenciaInfo(): La función construye la trama que transfiere información hacia una memoria (RAM o EEPROM), en esta versión, se utilizan estos paquetes para transferir a partir de la posición 62 de la memoria RAM del módulo GSM los caracteres apropiados para cada tipo de mensaje SMS que se desea enviar. Se llama desde la función MensajeSMS(), la cual a su vez es llamada desde el thread LoopSalidas(), cuando se produce una salida o restauración de la misma.

- Paq_On_UserResult(): Monta la trama que se obtiene después de haber recibido una llamada a RPC de tipo On_User. Dependiendo de los datos del Param1, esta trama es la directamente responsable de armar y desarmar el sistema, modificando una variable global que es accesible desde cualquier punto del programa y que controla la primera capa de funcionamiento del sistema (si el sistema está desarmado, no se producen alarmas, llamadas ni mensajes SMS).

- Do_AcK(): la fución recibe como parámetros los campos de origen y destino para montar trama de reconocimiento positivo. Se llama a esta función desde 2 puntos:

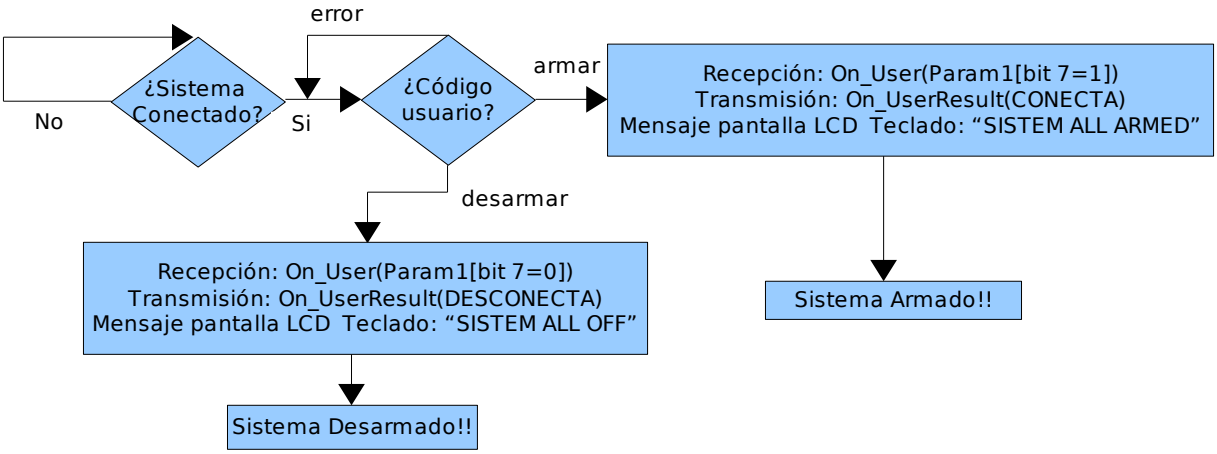
  1) desde Procesar_Trama(), al recibir un paquete RPC o paquete de órdenes que exigen confirmación;

  2) y desde Paquetes_VR(), que es cuando se recibe un paquete vía radio y debe ser confirmado para que el Receptor VR no continue enviando los mismo paquetes de información.

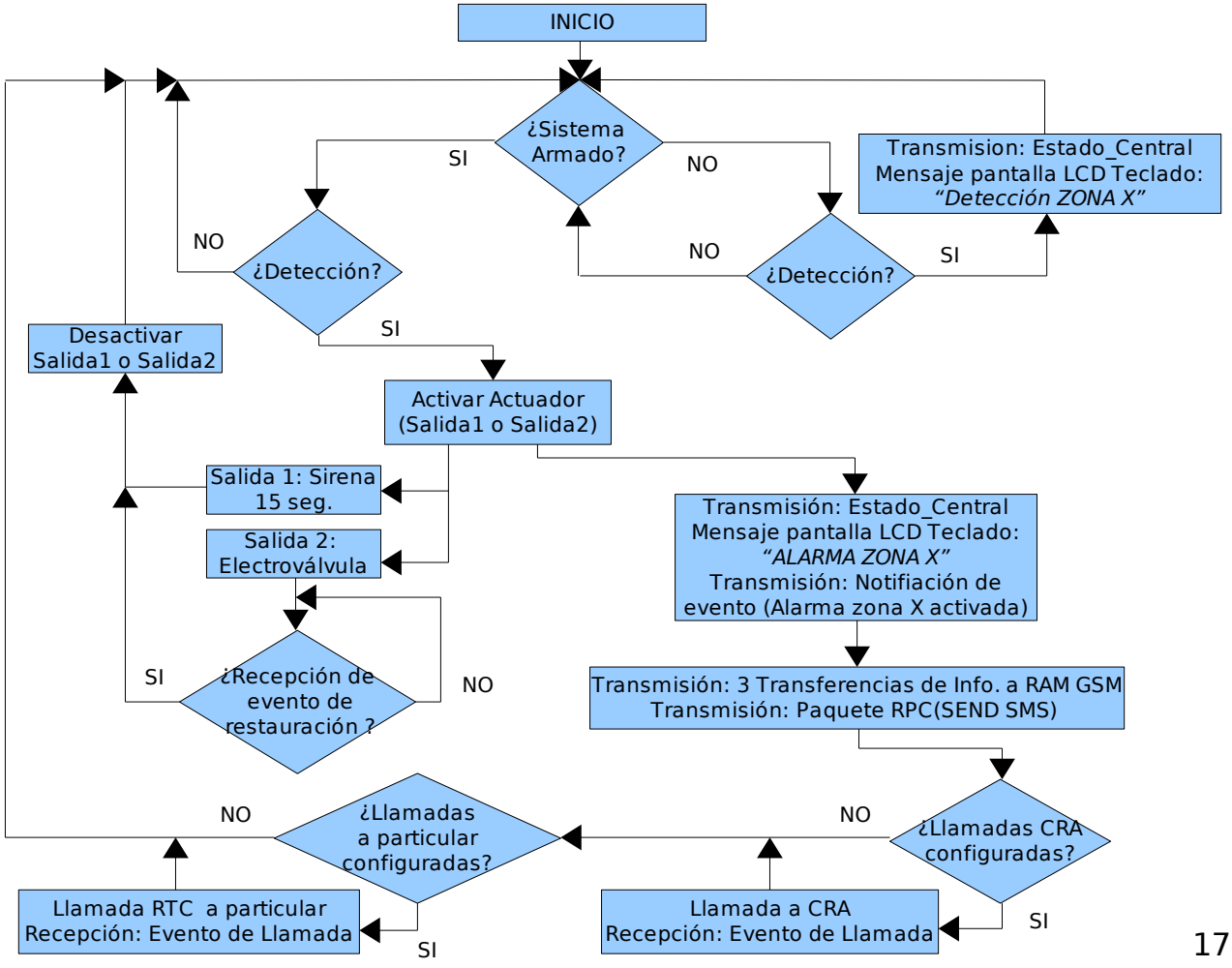
2.4.- Nivel de aplicación: Sistema Armado/Desarmado. Gestión de E/S. Alarmas, llamadas y SMSs.

El siguiente diagrama define cómo son tratados los paquetes de información que llegan de la capa de red y tienen efecto en la capa de aplicación, o, del mismo modo, los efectos que derivan de la aplicación cuando interviene un usuario, generando los paquetes de información que pasarían a la capa de red anteriormente comentada.

CAPA DE APLICACIÓN
[Armado/Desarmado del sistema. INICIALIZACIÓN]



[Gestion de E/S. Alarmas, llamadas y SMS]



2.5.- Compilación y ejecución del programa.

La implementación del programa *proximaGSM* se ha realizado mediante la interfaz de entomo de desarrollo Code::Blocks, sin embargo la compilación se realizaba desde consola, para ello se necesitó modificar el archivo Makefile.in añadiendo los ficheros objeto del proyecto(fichero.o), ya que la aplicación de inicio generaba un Makefile por defecto (mirar la guía de LxNETES), seguidamente se sube a un directorio de acceso externo mediante TFTP, y se ejecuta desde la remota, a la cual accedemos mediante telnet, las intrucciones necesarias para todo ello son las siguientes:

MODIFICACIÓN DE *Makefile.in*:

##modificamos las líneas correspondientes a los CFLAGS, OBJS y proximaGSM, quedando del siguiente modo:

```
# Makefile for proximaGSM
# @SET_MAKE@

subdir = apps/proximaGSM
top_builddir = @top_builddir@
srcdir = @srcdir@
top_srcdir = @top_srcdir@

CROSS_COMPILE ?=

CC = $(CROSS_COMPILE)gcc
# CFLAGS += -lpthread $(if $(filter apps $(subdir) 1, $(DEBUG)),-g)
CFLAGS := -lpthread -o1 -Wall
CXXFLAGS := $(CFLAGS)
TARGETS=proximaGSM
OBJS=$(TARGETS:%=%.o)serial.o proximaGSM.o Crc.o

all:$(TARGETS)

proximaGSM: serial.o proximaGSM.o Crc.o

install: all
	$(call cmd,install_program,proximaGSM,/usr/bin/proximaGSM)
	include $(top_builddir)/Makefile.lib
```

COMPILACIÓN:

 # desde consola es necesario trabajar como super usuario:
user:~$ sudo su

 # acceder al directorio build de LxNETES:
root# cd LxNETES-3.2/build/

#ejecutar make:
root@/LxNETES-3.2/build# make

#es necesario linkar los ficheros objeto y las opciones de threads a mano, introduciendo la ruta de los ficheros y la opción de threads:
root@/LxNETES-3.2/build# ../bin/arm-linux-gcc ./apps/proximaGSM/proximaGSM.o ./apps/proximaGSM/serial.o ./apps/proximaGSM/Crc.o -o proximaGSM -lpthread

#finalmente, se copia el fichero compilado al directorio /tftpboot:
root@/LxNETES-3.2/build# cp ./proximaGSM /tftpboot/

EJECUCIÓN:

#desde consola accedemos a la remota mediante telnet:
user:~$ sudo telnet 172.25.97.119

#cambiamos al directorio donde queremos que resida el programa:
/ # cd usr/bin/

#importamos el ejecutable que dejamos en /tftpboot:
/usr/bin # tftp -r proximaGSM 172.25.97.197 -g

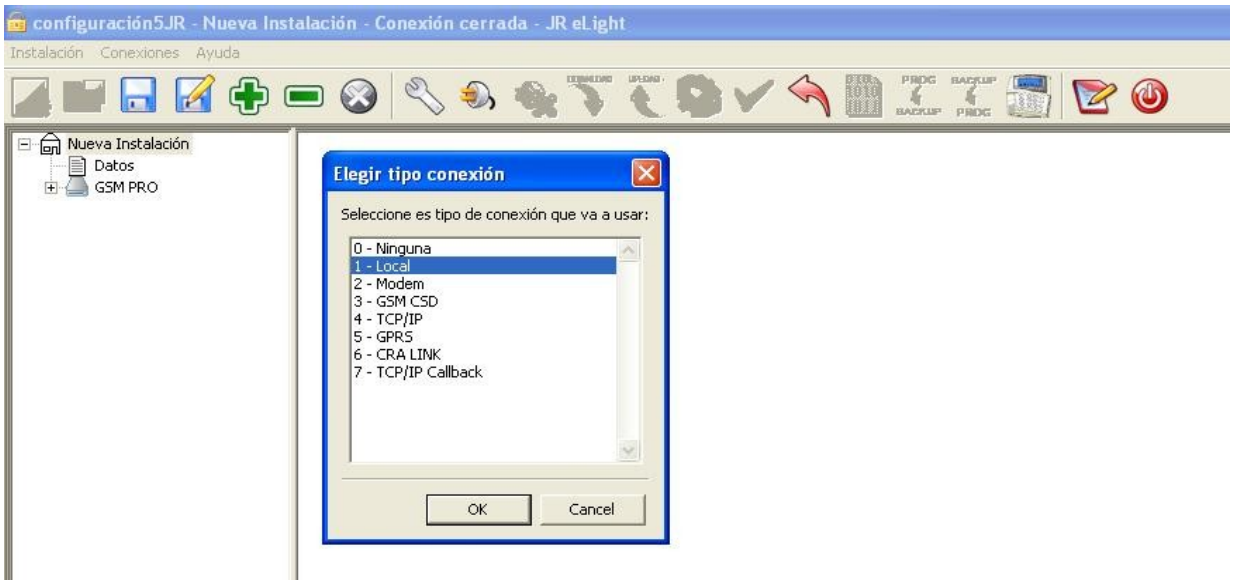
#ejecutamos el programa desde este punto:
/usr/bin # ./proximaGSM

NOTA: Se debe tener en cuenta que las direcciones ip pueden cambiar dependiendo de la configuración de la red.

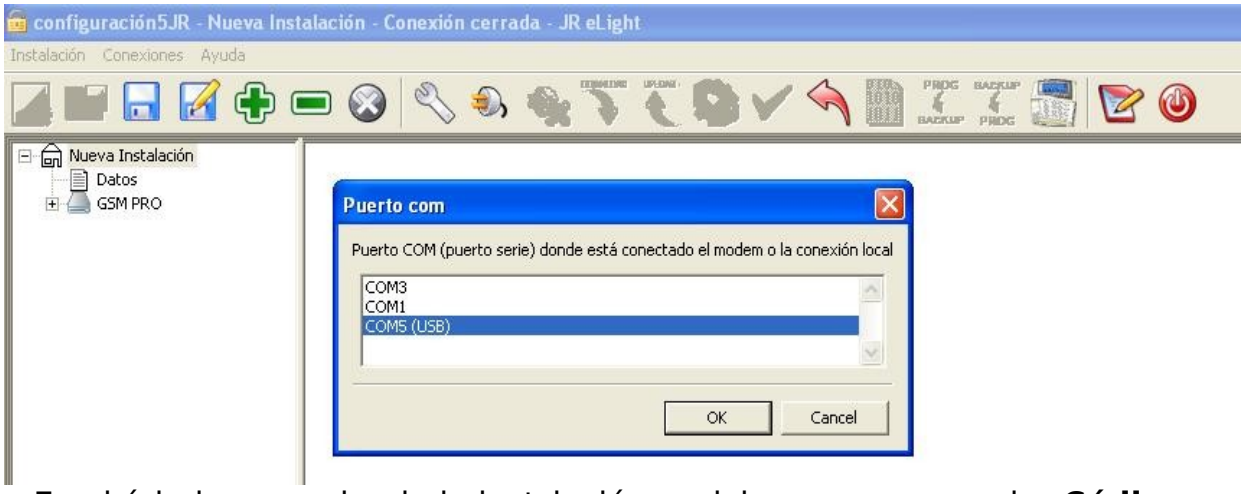
2.6.- Configuración del módulo GSM a través de JR-Elight.

Se adjunta documentación sobre el programa de configuración de módulos GSM: " *JR eLight JR Express Light v1.0 Manual de Usuario*". De todos modos se pasa a comentar las opciones de configuración más relevantes y necesarias para conseguir realizar llamadas y envíos de mensajes SMS.

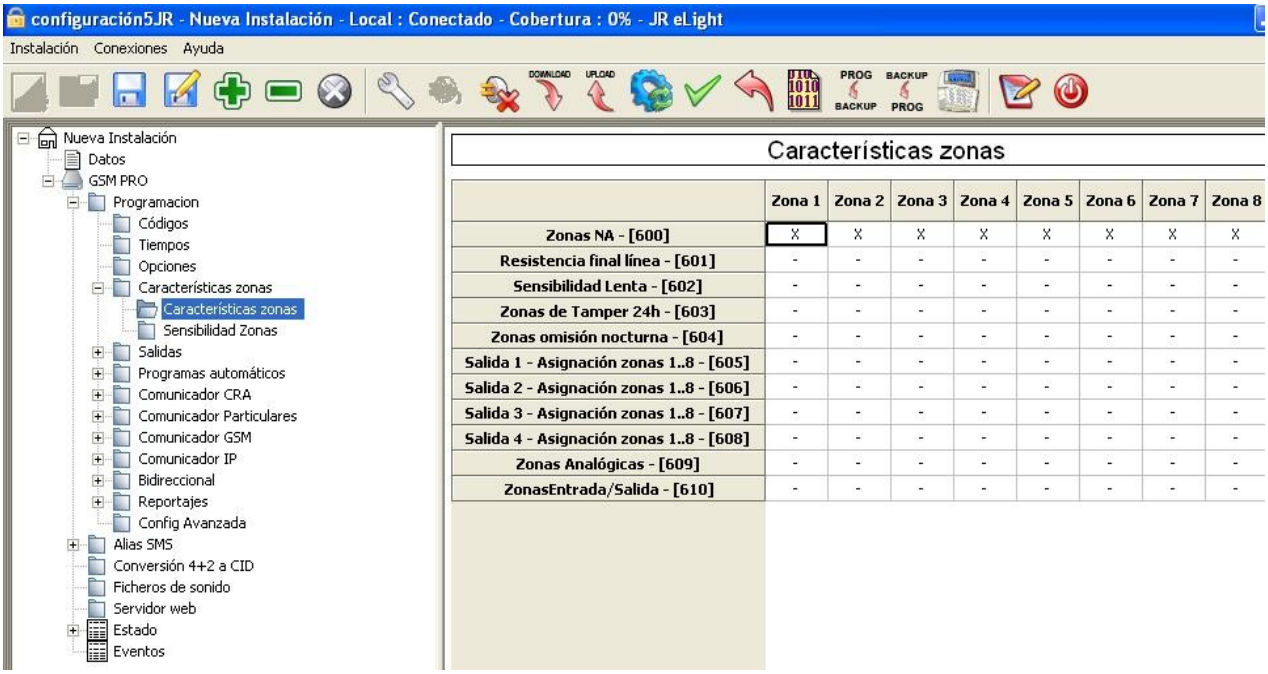
Para crear una primera conexión es necesario configurar, haciendo clic en el icono de la llave inglesa, eligiendo tipo de conexión Local en nuestro caso y el puerto com USB. Una vez configurada la conexión podemos aceptar y hacer clic en el icono del enchufe para conectar.



Se puede observar en las capturas de pantalla.



En el árbol navegador de la instalación se debe asegurar que los **Códigos** sean los siguientes:
000 - Código instalador: 0011
001 - Código usuario 1: 1111
Las **Características zonas** deben estar todas las **Zonas NA** (Zonas normalmente abiertas) seleccionadas, ya que el módulo GSM no gestionará ninguna zona, lo hará la próxima:



El **Comunicador CRA**, para realizar llamadas a la central receptora de alarmas, necesita definir varias opciones de configuración:

- La subcarpeta **Teléfonos:** se debe añadir el nº de teléfono de la CRA.
- **Intentos:** como mínimo, cada teléfono asociado debe realizar 1 intento de llamada.
- **Formato/Protocolo:** Los teléfono activados necesitan un 07 en este campo.
- Las **Opciones:** Las que vienen por defecto.
- Las **Opciones de envío:** Para cada teléfono se activan los eventos que provocaran llamada a CRA.
- **Áreas/Zonas que envían:** Para cada teléfono, se activan las zonas que deseamos que produzcan llamadas a CRA.

El **Comunicador a Particulares**, es el apratado donde se configuran las llamadas de teléfono a particulares, estas llamadas pueden incorporar un mensajes predefinido o se puede configurar dicho mensaje mediante buzones de voz. Las opciones a configurar son las siguientes:

- **Teléfonos:** Se añaden los teléfonos, ya sean móviles o fijos, a los que se realizarán las llamadas.
- **Opciones:** Como mínimo un intento teléfonico de llamada, el resto de parámetros no se deben modificar a no ser que se sepa perfectamente lo que se hace.
- Las **Opciones de envío:** Para cada teléfono se activan los eventos que provocaran las llamadas a Particular.
- **Áreas que envían particulares:** Para cada teléfono, se activan las zonas que deseamos que produzcan llamadas a particulares.

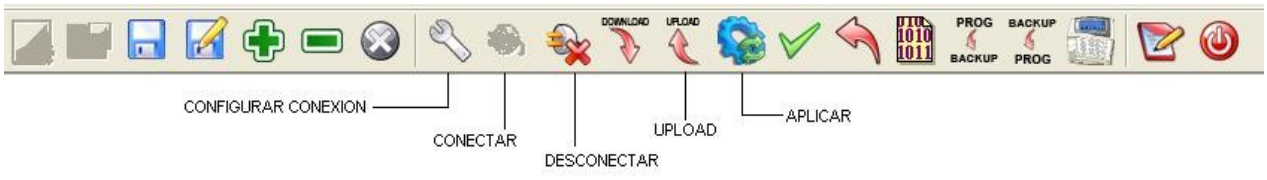
El **Comunicador GSM** se encarga de realizar los envíos de mensajes SMS, las opciones de configuración son las que se comentan a continuación:

- **PIN:** Se debe introducir el PIN de la targeta SIM que se use en el módulo GSM. Si la opción PIN de la targeta está desabilitada, en este campo no se introduce nada.
- **Dígitos:** Estos valores vienen por defecto, no se deben modificar a no ser que se sepa lo que se hace.
- **Opciones de comunicador GSM:** No es necesario activar ninguna de estas opciones.
- **Teléfonos SMS:** Se añaden los teléfonos destino a los que queremos que lleguen los mensajes SMS. En este caso los teléfonos deben ir precedidos del prefijo 34, quedando: 346XXXXXXX. Se pueden añadir hasta 5 teléfonos. También se debe incluir el centro servidor de mensajes correspondiente a la compañía de la targeta SIM.
- **Opciones de envío SMS:** Se pueden añadir opciones de eventos para enviar SMS, en nuestro caso no se activa ninguna opción ya que el envío de SMS está gestionado mediante software desde la Próxima.
- **Áreas que envían SMS:** Están todas las áreas activadas por defecto, aunque en nuestro caso es redundante por la misma razón que el punto anterior.
- **Control de salidas:** En esta opción no se activa ninguna casilla.

En el apartado **Reportajes**, se deben activar con el valor 07 todos aquellos campos que generen algún evento de llamada o envío de SMS:

- Las **Zonas:** Todas a 07
- **Eventos Zonas:** Las que se deseen activar, añadir 07. en nuestro caso activamos tan sólo Rest/Omisión de zona.
- **Eventos Averías:** En nuestro caso no activamos ninguna opción (todo FF).
- **Eventos on/off:** En nuestro caso no activamos ninguna opción (todo FF).
- **Miscelánea:** En nuestro caso no activamos ninguna opción (todo FF).
- **Códigos de abonado:** Es necesario introducir un código de abonado de área 1 por defecto, en nuestro caso añadimos 1234.
- **Test Telefónico:** Estos parámetros no se deben tocar a no ser que se sepa lo que se hace.

NOTA: Los Apartados y subapartados de los que no se comenta nada no se deben tocar. Los parámetros que vienen por defecto son suficiente para un correcto funcionamiento.
Cada vez que se modifica un campo se debe hacer un **upload** y después **Aplicar**, si no se realizan estos 2 pasos no se configurará correctamente. El **upload**, y **Aplicar** afecta a una carpeta y a todas sus subcarpetas, recursivamente.(podemos observarlos botones de la barra de herramientas en la siguiente figura)



- Anexo 1

FORMATO DE TRAMAS EN PRÓXIMAGSM v1.0 PARA BUS RS-485

Tramas SYN/STX(longitud variable):

Tramas ACK/NACK (no tienen campo Identif. ni DATOS, longitud fija de 5 bytes):

Cabecera: La cabecera es un Byte que indica si la trama corresponde a un primer envío de información (para lo cual se necesitará sincronización) o se trata de un reenvío. También puende indicar si el paquete de datos es una confirmación positiva o negativa. Los diferentes tipos de cabecera y sus valores en hexadecimal correspondientes son los siguientes:

- **SYN**: sincronización, primer paquete -> 16h.
- **STX**: retransmisión -> 02h.
- **ACK**: confirmación positiva -> 06h.
- **NACK**: confirmación negativa -> 15h.
- **ACKW**: confirmación positiva -> 07h. (No se usa en esta versión)
- **NACKW**: confirmación negativa -> 13h. (No se usa en esta versión)

Destino y Origen: Ambos son bytes que indican la dirección correspondiente al módulo al cual va dirigida la trama (Destino) y el módulo que la envió (Origen). Todas las direcciones vienen definidas en el documento *Protocolo 485 csma para BUS Centrum*, facilitado por JR Sistemas de Seguridad, las direcciones más relevantes en el proyecyo PróximaGSM v1.0 son las siguientes:

- **Próxima**: 00h.
- **GSM**: 12h.
- **ReceptorVR**: 01h.
- **Teclado TCL**: 38h.
- **Controlador LPC-USB (JR-eLight en modo local)**: 31h.
- **A TODOS los módulos (Broadcast):** 40h.

Identif.: Se trata de un Byte que realiza doble función, identificador de tipo de trama y longitud del campo de Datos (que puede ser variable). La longitud en cantidad de bytes, que viene implícita en el identificador, se obtiene desplazando 4 bits a la derecha, es decir, viene definida por los 4 bits más significativos de este campo.

Ejemplo: Identif. : B1 . Desplazamos 4 bits B1>>4= B (en decimal 11)
- Obtenemos una longitud del campo de datos igual a 11 bytes.

Todos los identificadores vienen definidos en cada tipo de trama en los documentos *Protocolo 485 csma para BUS Centrum y Gateway EIB para Centrales Centrum.* Las tramas más relevantes junto a la definición de su función del identificador son las siguientes:

i - Estado_Central: A1h. Contiene una estructura de tamaño fijo de 10 bytes que definen el estado de la central, mostrando las zonas en detección, en alarma, en omisión, zonas en avería, estado de las diferentes áreas, etc... (*Protocolo 485 csma para BUS Centrum*, pg. 4). Se envía esta trama cada vez que se detecta un cambio de estado de la central y cada 8 minutos para evitar bloqueos permanentes de estado.

ii - Paq_*RPC (Llamada a procedimiento Remoto)*: 5Bh y 7Bh. Esta trama concentra diferentes funciones de llamadas a procedimientos. El formato de esta trama y las funciones RPC más importantes son las siguientes:

FORMATO DE TRAMA 5Bh (RPC)

Cabecera	Destino	Origen	0x5B	Indice función	PARAM 1	PARAM 2	PARAM 3	PARAM 4	CRC_H	CRC_L

(es necesario consultar *Protocolo 485 csma para BUS Centrum, pg 5 a 8*)

Índices de funciones:

- *Omisión de Zona*: 00h. Realiza omisión/restauración de zonas. Usa 2 parámetros: - Param1 (modo nocturno, bloques de zonas,usuario)
  - Param2 (máscara de zonas a omitir)
- On_User: 01h. Conecta/Desconecta a los usuarios del sistema. Usa 1 parámetro: - Param1 (fuerza armado/desarmado, índice de usuario)
- *Armado de Área*: 02h. Permite armado de áreas. Un parámetro:
  - Param1 (Máscara de áreas a conectar)
- *Desarmado de Área*: 03h. Permite desarmado de áreas. Un parámetro:
  - Param1 (Máscara de áreas a desconectar)
- *Set_Alarm*: 0Ah. Activa la alarma en la zona que se desee. Un parámetro: - Param1(índice de la zona, de 0 a N-1)
- *Enviar SMS*: 0Fh. Ejecuta el envío de un SMS. Usa 2 parámetros:
  - Param1 (longitud de caracteres a enviar, se deben haber memorizado en RAM previamente)
  - Param2 (índice del número de teléfono destino, el teléfono en si se configura a través del JR-eLight o el Teclado)
- *Void*: FFh. Provoca el retorno del estado.

iii – Paquete de órdenes: Dicho paquete gestiona el control de estado/presencia de elementos en el sistema. Al enviar este paquete a un módulo, éste debe responder, actualizarse, pedir datos de memoria, etc... dependiendo del tipo de orden. El formato de esta trama es el siguiente:

FORMATO DE PAQUETE DE ÓRDENES

Cabecera	Dest.	Origen	0x10	Orden	CRC_H	CRC_L

Los tipos de órdenes de importancia son:

- ORDER_TEST: 02h. Obliga al receptor a responder con un paquete TEST_ANSWER.
- ORDER_AL: 1Bh. Realiza una petición de memoria de alarmas.

iv – Paquete TEST_ANSWER: 61h. Se trata de una respuesta a una orden de test. El paquete tiene una estructura similar al de órdenes, con identificador 0x61, y el campo de Orden se sustituye por el de Identidad, el cual consta de 6 Bytes, los cuales definen los datos del dispositivo: tipo, versión, día, mes y año de compilación, y tensión de alimentación.(Consultar Gateway EIB para Centrales Centrum)

DATOS: Este campo contiene una cantidad variable de bytes que corresponden a los datos que transporta la trama. La longitud de este campo se define en el campo Identif.

CRC: Es el campo que contiene el código de redundancia cíclica. Se compone de 2 bytes: CRC_H, el cual contiene la parte alta del código y CRC_L, que contiene la parte baja.
