Índice de contenidos

Índice de Figuras

PARTE I
(MEMORIA)

Capitulo 1. Introducción

En la investigación marina experimental, cada vez más, van aumentando las necesidades de obtener mayor resolución y volumen de información en los sistemas de observación oceanográfica. Tradicionalmente se han ido obteniendo estos datos mediante buques oceanográficos y/o boyas, sistemas de observación y adquisición de datos que presentan serios inconvenientes respecto a los costes, volumen de datos almacenados o autonomía de las baterías. En los observatorios submarinos cableados como el proyecto OBSEA (Observatorio Submarino ExpAndible) no es necesario que los científicos estén en los barcos sujetos a las malas condiciones meteorológicas.

Los observatorios submarinos son modulares, extensibles y se adaptan a los diferentes usos y requisitos para satisfacer las necesidades de la comunidad científica.

El proyecto OBSEA (Observatorio Submarino ExpAndible) es una iniciativa conjunta CSIC_UPC para diseñar y desarrollar un observatorio submarino delante de la costa de Vilanova i la Geltrú. La plataforma submarina se localiza a 4 km de la costa y a 20 metros de profundidad.
El proyecto OBSEA está formado por dos estaciones, una estación terrestre (ET) y una estación submarina (ES). La ET es la que se encarga de proporcionar energía a los instrumentos submarinos y de recibir información tanto de los sensores como del estado de los dispositivos conectados a la ES. En cambio, la ES está formada por los propios sensores y por un sistema de control que proporciona tanto el estado de éstos como el estado de las fuentes de alimentación para una futura monitorización en la ET. Además, en el sistema de control de la ES está implementado un protocolo encargado de enviar alarmas en cuanto ocurre algún evento que anteriormente se ha programado. Este protocolo se llama SNMP. La actividad desarrollada en este proyecto final de carrera se centra en el diseño y programación de la unidad de control de la estación submarina. Así como en la colaboración de las tareas derivadas de la puesta en marcha y construcción del observatorio.

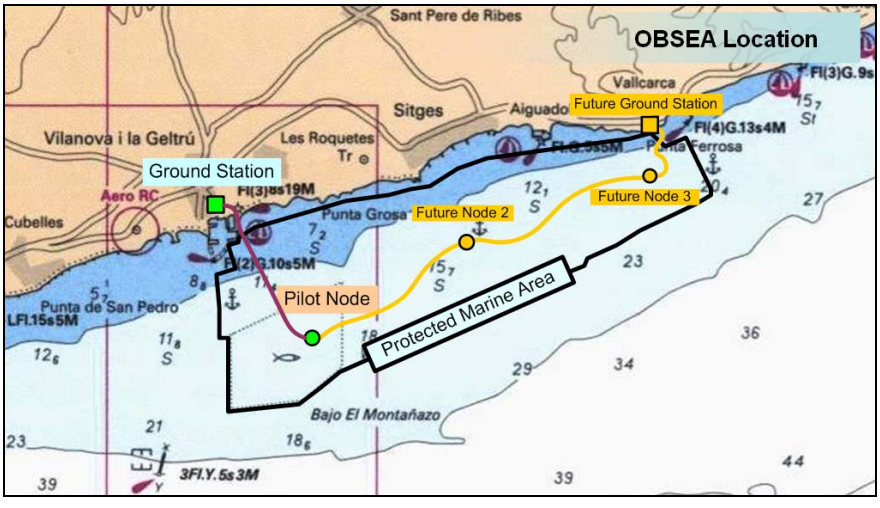

Figura 1.1: Localización OBSEA

Cabe decir que actualmente se ha desplegado el primer prototipo OBSEA y que en un futuro está previsto ampliar el número de estaciones. En la figura 1.1 se observa el nodo piloto y los futuros nodos.

1.1. Objetivos

El objetivo del proyecto OBSEA es la creación de una red de nodos submarinos en los que poder conectar diferentes sensores de acuerdo con las necesidades de los científicos. En cada nodo se pueden conectar hasta 8 instrumentos como por ejemplo, CTD (Conductivity, Temperature Depth), cámara submarina, hidrófono... La principal ventaja que aportan los observatorios es la de poder obtener los datos de los instrumentos en tiempo real, las 24 horas y los 365 días de año. De ésta forma, se pueden guardar los datos recibidos y analizar las singularidades de los eventos.

1.2. Especificaciones

Cómo se ha dicho, OBSEA está compuesto por dos estaciones. A continuación se va a realizar una breve descripción de los componentes de estas y seguidamente se analizará más detalladamente.

La Estación Terrestre está compuesta por:

- Sistema de alimentación: formado por una fuente de alimentación (FA) que puede ser de hasta 1500Vdc que alimenta la ES. Actualmente se generan hasta 320Vdc 11A.
- Sistema de comunicaciones: para poder acceder a la ES. Se dispone de un switch ethernet encargado de recibir toda la información de la ES y a la vez de hacer de interfaz óptica hacia el cable submarino. La comunicación entre las estaciones se realiza mediante un cable híbrido de fibra óptica y cobre.
- Sistema de control: se dispone de un servidor con un software de gestión. Se supervisan las variables de los sensores de la ES.

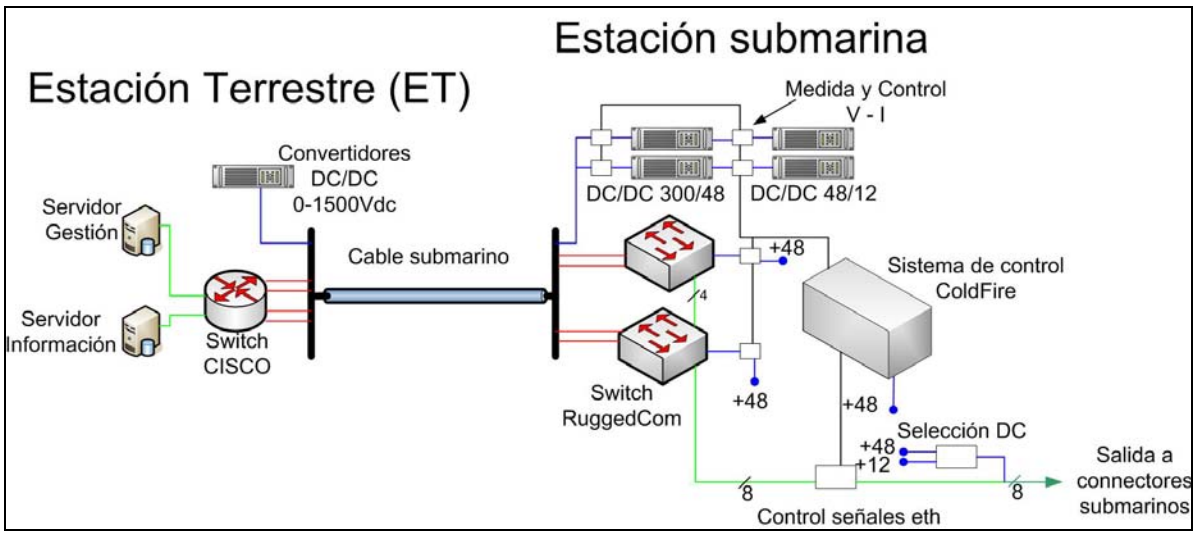

Figura 1.2: Esquema General del Observatorio

La Estación Submarina está compuesta por:

- Sistema de alimentación: formado por 2 Fuentes de Alimentación (FA) de 48V, 3 FA de 12V y baterías.

- El sistema de control submarino está compuesto por una placa que integra un microcontrolador de 32 bits basado en la tecnología Coldfire MCF5282. Éste dispone de las funciones necesarias para implementar el protocolo SNMP (Simple Network Management Protocol).
- Sistema de comunicaciones: Formado por 2 switch ethernet industrial encargados de realizar la interfaz óptica hacia el cable submarino y a la vez de conectar todas las señales del sistema de control y sensores.

En la figura 1.2 podemos ver un esquema de la configuración general de las 2 estaciones.

1.2.1. Estación terrestre (ET)

La estación terrestre dispone de los elementos necesarios para la operación remota de la estación submarina. Estos elementos son:

- Sistema de alimentación: formado por un conjunto de 12 fuentes de 27Vdc controladas por un PLC, formando un sistema de 320Vdc y 11A (Figura 1.3).

Figura 1.3: Fuentes de Alimentación de la Estación Terrestre

- Sistema de comunicaciones: formado por 3 switches ethernet CISCO con supervisión remota y capacidad para 4 puertos ópticos de 1Gb/s y 8 eléctricos 10/100/1000BaseT (Figura 1.4).

Figura 1.4: Switch ethernet CISCO

- Sistema de control: formado por un conjunto de servidores dedicados a la gestión y supervisión de los equipos de la estación submarina y administración de los datos de los sensores.

1.2.2. Estación submarina (ES)

La estación submarina está compuesta por sensores y por un sistema que controla el correcto funcionamiento de los dispositivos conectados (Figura 1.6). Estos son los elementos de los que está compuesta:

- Sistema de alimentación: Formado por 3 fuentes de alimentación de 12Vdc y 2 fuentes de 48Vdc.
- Sistema de comunicaciones: formado por 2 switches ethernet con supervisión remota y capacidad para 2 puertos ópticos de 1Gb/s y 8 eléctricos 10/100BaseT. Se alimentan a 48Vdc (Figura 1.5).

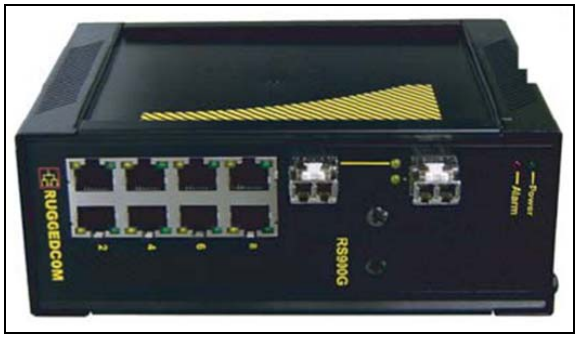

Figura 1.5: Switch Ethernet Ruggedcom

- Sistema de control: formado por un modulo ColdFire encargado de supervisar y controlar las alimentaciones internas, externas, parámetros ambientales y también de realizar el control de conexión y desconexión de los instrumentos (Figura 1.7). Además, informa a la ET del estado de los dispositivos conectados a la ES.

Figura 1.6: Chasis de la electrónica incorporada en OBSEA

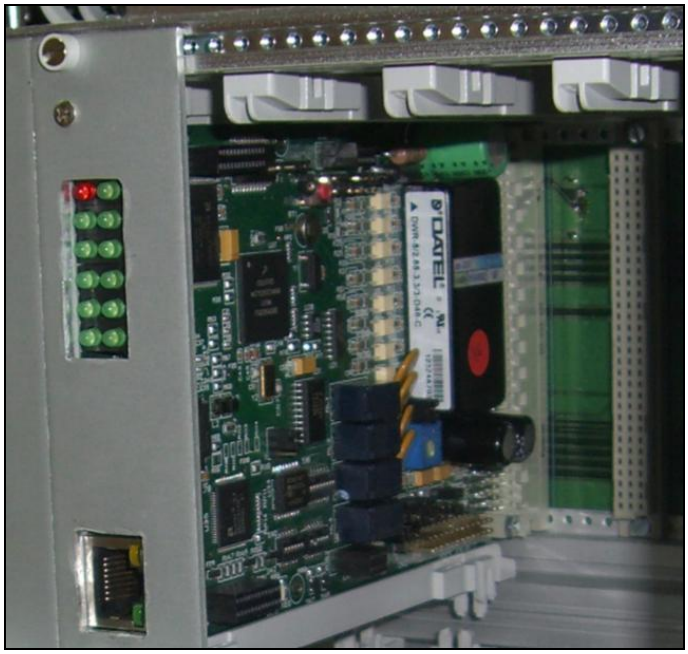

Figura 1.7: Placa ColdFire

1.2.3. Otros elementos necesarios

Además de lo comentado anteriormente son necesarios los siguientes elementos:

- Cable hibrido de fibra óptica / cobre de 5 km (Figura1.8): cable de conexión entre la ET y la ES. Éste nos lo ha proporcionado Telefónica y sus características son las siguientes:
  - 31.8mm

- 6 fibras ópticas
- 1 tuvo conductor de cobre
- 2 capas de tracción de hilos de acero
- Aislante eléctrico de polietileno
- Tuvo de aluminio
- Capa exterior de polietileno

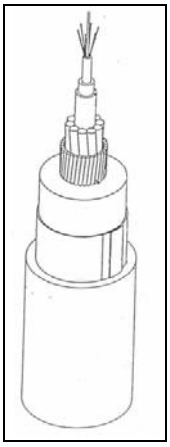

Figura 1.8: Cable Fibra Óptica Submarina

- Cable terrestre: Cable de 1.5 km que cubre el tramo entre la estación terrestre y el inicio del cable submarino.
- Conectores submarinos:

  - Conectores submarinos híbridos fibra / eléctrico. Para la interconexión del cable submarino y OBSEA (Figura 1.9).

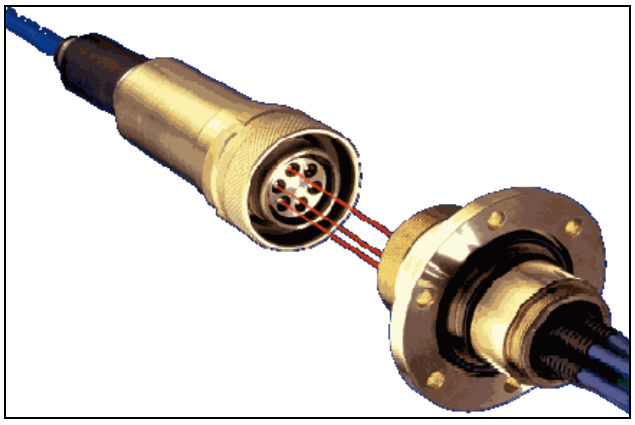

Figura 1.9: Conectores híbridos fibra / eléctricos

  - Conectores eléctricos WET-MATEABLE para los sensores. Permiten la conexión y desconexión en mojado de instrumentos oceanográficos sin necesidad de sacar la estación de dentro del agua (Figura 1.10).

Figura 1.10: Conectores eléctricos para agua

- Estructura de anclaje: proporciona el soporte físico del nodo submarino. Se ha construido con acero inoxidable 316L. Protege los elementos e impide el acceso no autorizado (Figura 1.11).

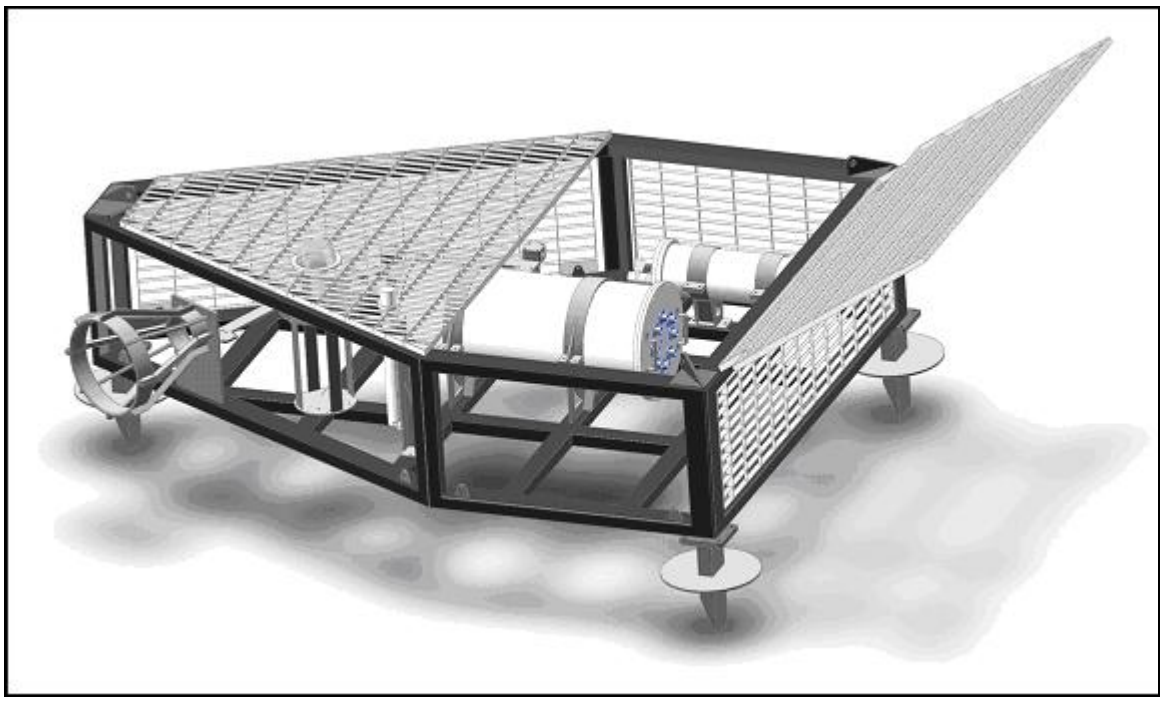

Figura 1.11: Estructura de anclaje del Observatorio

- Estructuras de soporte de los sistemas submarinos: cilindros presurizados que contienen la electrónica de la estación (Figura 1.12).

Figura 1.12: Cilindro que contiene la electrónica

- Sensores oceanográficos:

  Actualmente hay 3 tipos de sensores:

  - CTD: Sensor de temperatura y humedad. Modelo “SBE 37-SMP” de MicroCAT. Envía datos vía serie y el cable de conexión al OBSEA los convierte a ethernet (Figura 1.13).

Figura 1.13: CTD

  - Cámara submarina: Modelo “OPT-06. Se reciben los datos mediante ethernet (Figura 1.14).

Figura 1.14: Cámara Submarina

- Hidrófono: Transductor de sonido a electricidad. Modelo “Hydrophone 02345” de Bjorge. Se reciben los datos mediante ethernet (Figura 1.15).

Figura 1.15: Hidrófono

1.3. Sistema de control de la estación submarina

El sistema de control de la ES se encarga de supervisar y controlar una serie de parámetros ambientales y eléctricos que aseguran el correcto funcionamiento del observatorio submarino.
Este sistema, está compuesto por un Microcontrolador MCF5282 de la familia ColdFire de Motorota, el cual tiene instalado el sistema operativo uClinux que soporta el protocolo SNMP (Simple Network Management Protocol). Con la utilización de este protocolo se pueden enviar alarmas mediante UDP para que la ET sepa si se ha producido algún error y para poder conectar o desconectar diferentes sensores. Como alternativa al protocolo SNMP, el sistema de control de OBSEA basado en el Coldfire, acepta la comunicación RS232; conexión con menos funcionalidades pero muy extendida entre los fabricantes de instrumentación.

Para poder implementar todo esto es necesario obtener los datos del estado de sensores. Por eso, se ha incorporado (Figura 1.16):

- Conversores Analogico – Digital (National Control Devices AD1232PROXR)
- Banco de Reles (National Control Devices XR16xDPDT).
- Sensor Temperatura / Humedad situado dentro del cilindro donde está la electrónica de control de la ES.

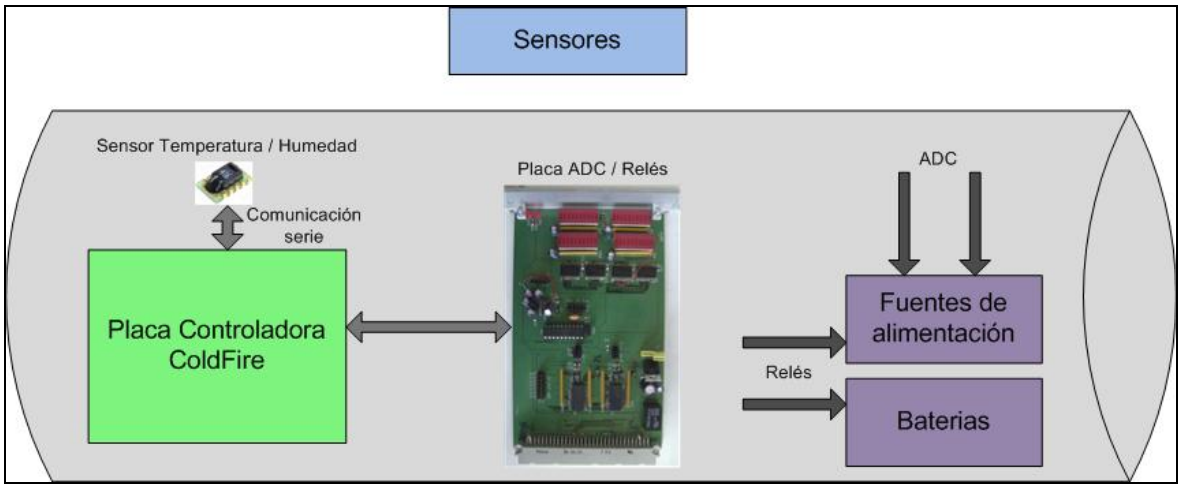

Figura 1.16: Esquema básico de la electrónica incorporada en OBSEA

Las placas ADC y relés se comunican con la controladora mediante el puerto RS232.

En el módulo conversor AD1232PROXR (8 a 10 bits resolución) se disponen 16 conversores Analógico a Digital para muestrear y cuantificar la señal analógica procedente de los sensores. La placa de relés XR16xDPDT, dispone de dos bancos de ocho relés cada uno.

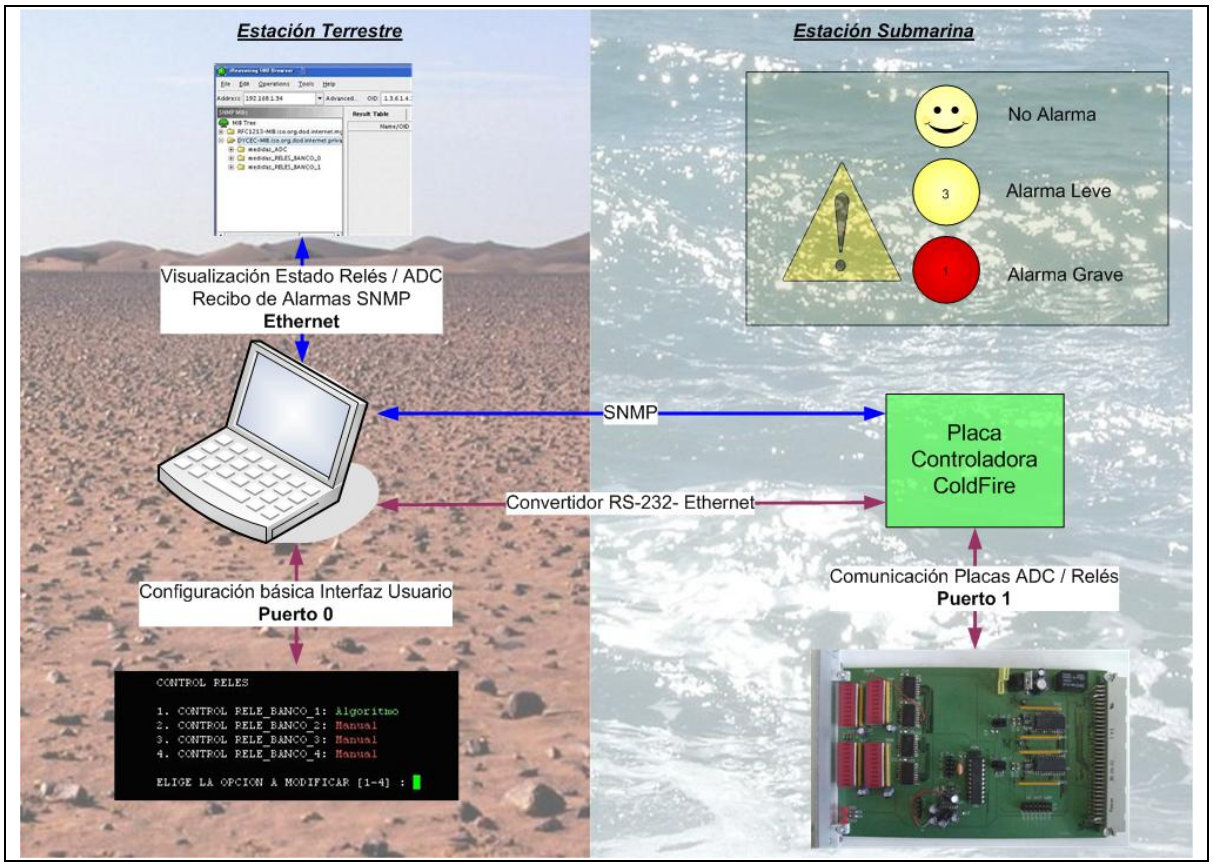

Figura 1.17: Esquema del sistema de control de la Estación Submarina

1.4. Resumen de las especificaciones del algoritmo de control de la estación submarina

El algoritmo de control de la ES está dividido en tres grandes bloques. El primero controla la alimentación, el segundo controla los elementos internos asociados a la electrónica y el tercero realiza el control de los instrumentos a conectar.

1- Control de Alimentación: El objetivo es detectar el mal funcionamiento de la fuente de alimentación o batería, detectar si la tensión de entrada está fuera de los márgenes preestablecidos, proteger todos los elementos si se produce una fallada en un dispositivo y evitar que un cortocircuito pueda desactivar todo el sistema. Para realizar dicho algoritmo se han definido una serie de condiciones que si se sobrepasan ciertos límites se envía una alarma leve o grave hacia la ET.

2- Control de los elementos internos de la ES: Los elementos a controlar son los conversores Analógico- Digital y los relés. Los objetivos son los mismos que en el algoritmo de alimentación.

3- Control de los instrumentos conectados.

   i. Alimentar y controlar el funcionamiento de los instrumentos oceanográficos conectados a la ES.
   ii. Mantener los conectores no utilizados libres de tensión.
   iii. Realizar la conexión / desconexión de instrumentos en el agua.
   iv. Evitar que un instrumento externo pueda afectar al funcionamiento interno de la estación.

Capitulo 2. Componentes del sistema

2.1. Microcontrolador ColdFire MCF5282

2.1.1. Introducción

A finales de los años 70 fue cuando aparecieron los microcontroladores de 16 bits cómo Intel 8086 o Motorota 68000. Se produjeron evoluciones, primero pasando a micros de 32 bits y luego a microprocesadores y microcontroladores que ya incluían diversos periféricos integrados.
Dentro de esta clase, en 1994 Motorota introduce la familia ColdFire de Freescale. ColdFire dispone de varias versiones de núcleo.
En los núcleos v1. los periféricos y las herramientas de desarrollo son comunes a la arquitectura de 8 bits. Además proporcionan compatibilidad de conectores para una mejor migración a los 32 bits. La v1 es una versión simplificada de v2. El microcontrolador utilizado para el proyecto pertenece a la versión 2 y la frecuencia de reloj puede llegar hasta los 66MHz.

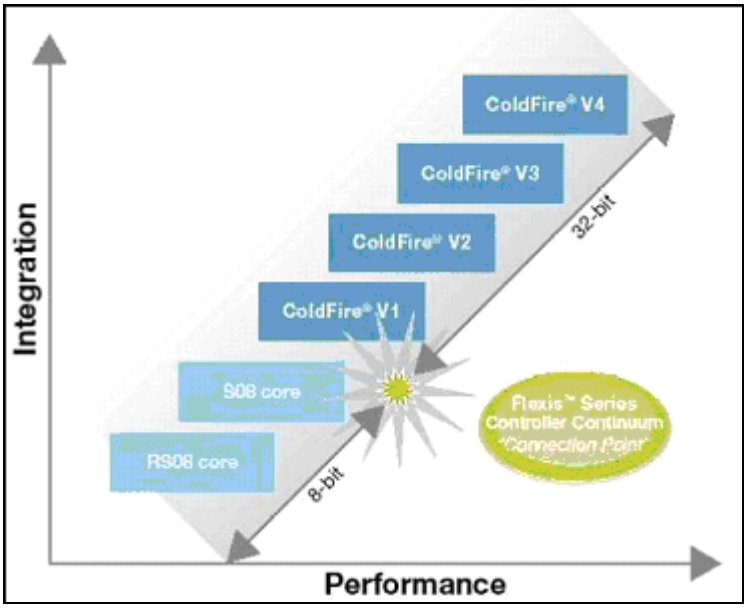

Figura 2.1: Evolución de los microcontroladores Motorola

2.1.2. Características placa Dycec

Las características de la placa controladora Dycec son las siguientes:

- Microcontrolador MCF5282 de la familia ColdFire de Motorola (Figura 2.2).
- Interfaces
    - UART con interfaces RS-232

- 2 UART´s con interfaces RS-232 o RS-485
- Bus I2C
- Bus CAN
- Bus QSPI
- 1 Ethernet. 10/100

- 16 MB de memoria SDRAM
- 8 MB de memoria FLASH
- RTC (Reloj de Tiempo Real)
- 8 entradas digitales
- Capacidad para dos módulos de expansión
- Alimentaciones de entrada de 12V ó 48V
- Opción de alimentación a través del interfaz de Ethernet (PowerEthernet).

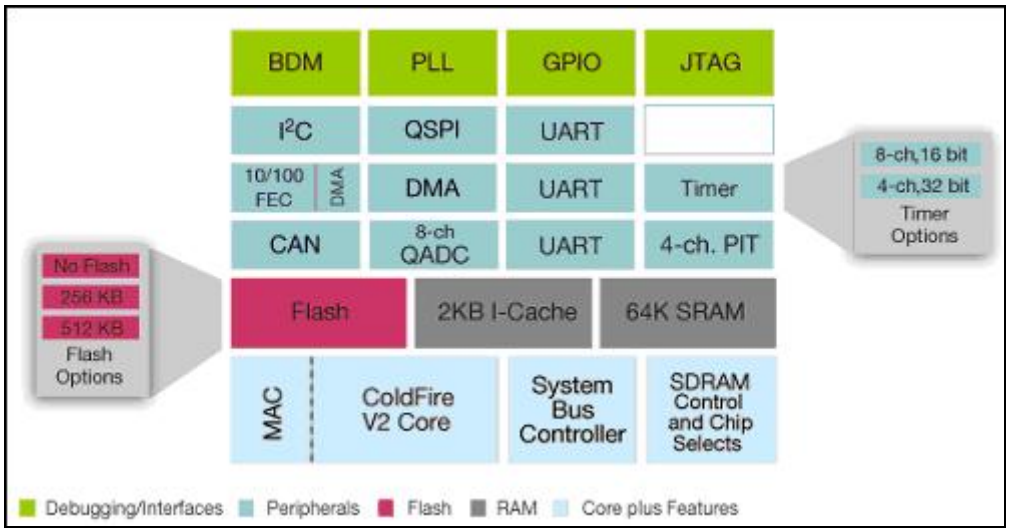

Figura 2.2: Características del microcontrolador MCF5282

2.2. Herramientas de programación

2.2.1. Entorno de desarrollo Kdevelop

El código del proyecto se ha desarrollado bajo el sistema operativo Suse y se ha empleado el entorno de desarrollo Kdevelop.
Kdevelop se emplea principalmente para sistemas GNU/Linux bajo entorno gráfico KDE; como su propio nombre indica, KDE Development Environment (Entorno de Desarrollo para KDE).
Soporta 15 lenguajes de programación y tiene varios editores internos. Cabe decir que es un entorno totalmente de libre distribución. Además, éste puede configurarse también para conectarnos a un servidor CVS donde guardar las fuentes.

A parte de desarrollar el proyecto en un PC con Suse instalado, como se ha dicho anteriormente, el propio microcontrolador posee otro sistema operativo llamado uClinux. La diferencia contra un sistema linux convencional es que uClinux no tiene MMU, o sea unidad de gestión de memoria.

La MMU es un dispositivo hardware formado por un grupo de circuitos integrados responsables del manejo de los accesos a memoria por parte de la CPU. Las funciones del dispositivo son las siguientes:

- Traducción de direcciones lógicas a direcciones físicas.
- Protección de la memoria.

- Control de la memoria caché.

Cuando la CPU intenta acceder a una dirección de memoria lógica, la MMU realiza una búsqueda en la memoria caché. En esa memoria se mantienen las entradas de la tabla de páginas donde se rescatan las direcciones físicas correspondientes a las direcciones lógicas. Cuando la dirección requerida por la CPU se encuentra en esa tabla, la traducción es casi inmediata, aunque en el caso de que no esté disponible la dirección, el procesador busca en la tabla de páginas del proceso tal y como se ve en la figura 2.3.

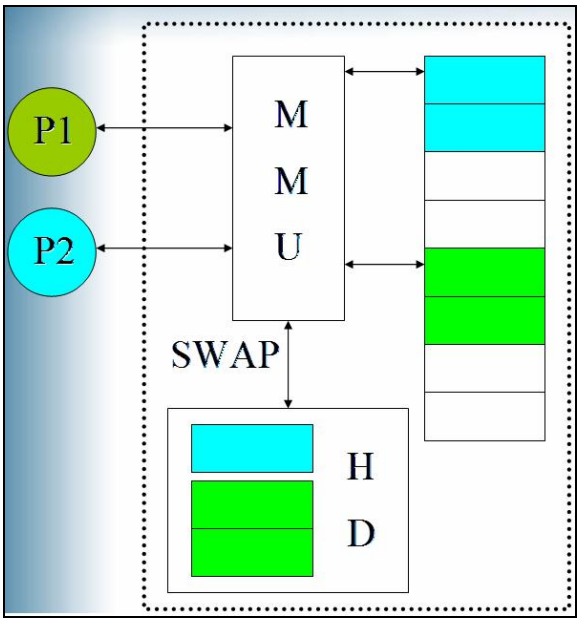

Figura 2.3: Microcontrolador con MMU

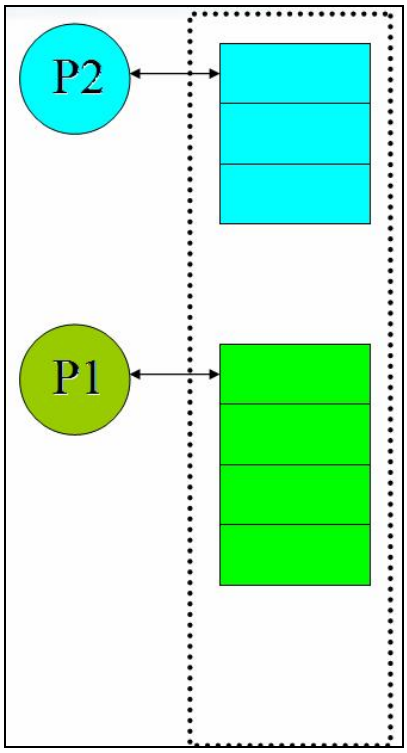

Figura 2.4: Microcontrolador sin MMU

Como se puede intuir, un beneficio fundamental de la MMU es la posibilidad de implementar protección de memoria, evitando que los programas accedan a porciones de memoria prohibidos. Por ejemplo se puede evitar que un programa acceda o modifique sectores de memoria de otros programas. Es por esto, que al programar bajo un sistema uClinux se debe tener mucho cuidado con la asignación de memoria para cada programa. En la figura 2.3, la MMU gestiona la memoria para cada programa P1 y P2, por el contrario, en la figura 2.4, no hay MMU y la memoria asignada a cada programa puede ser problemática al no permitir la ampliación de ésta para cada proceso.

2.3. Familia ProXR

2.3.1. Introducción

La familia ProXR ha sido la elegida en este proyecto, pero antes de elegir la placa de adquisición de datos AD1232PROXR se hizo un estudio de las posibilidades del mercado.

Se miraron las siguientes opciones:

- NI USB-6229 (DAQ multifunción de la Serie M de 16 bits):
    - 32 entradas analógicas (16 bits, 250 kS/s)
    - 4 salidas analógicas (16 bits a 833 kS/s), 48 E/S digitales (32 a hasta 1 MHz) y contadores de 32 bits
    - NI Signal Streaming para transferencia de datos sostenida a alta velocidad en USB; la versión OEM está disponible
    - Compatible con LabVIEW, LabWindows/CVI y Measurement Studio para Visual Studio .NET
    - El software de NI-DAQmx y software interactivo NI LabVIEW SignalExpress LE para registro de datos

Figura 2.5: NI USB-6229

- PIC24HJ:
    - 32 entradas analógicas
    - Resolución 10/12 bits

Figura 2.6: PIC24HJ

- AD1232PROXR:
    - Interfaz RS-232 a una velocidad como máximo de 115.2KBaudios
    - 32 canales ADC de 8 o 12 bits
    - Capacidad de expansión de hasta 48 canales más.
    - Posibilidad de añadir hasta 256 relés utilizando el puerto de expansión XR.

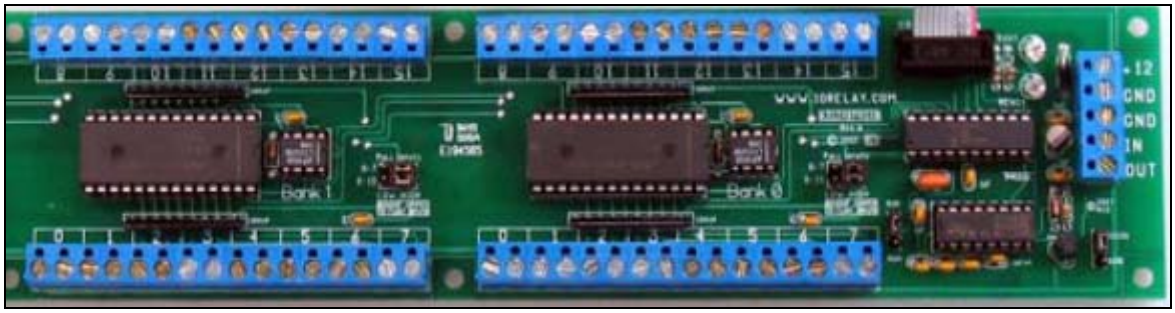

Figura 2.7: AD1232PROXR

- XR16xDPDT:
    - Interfaz RS-232 a una velocidad como máximo de 115.2KBaudios

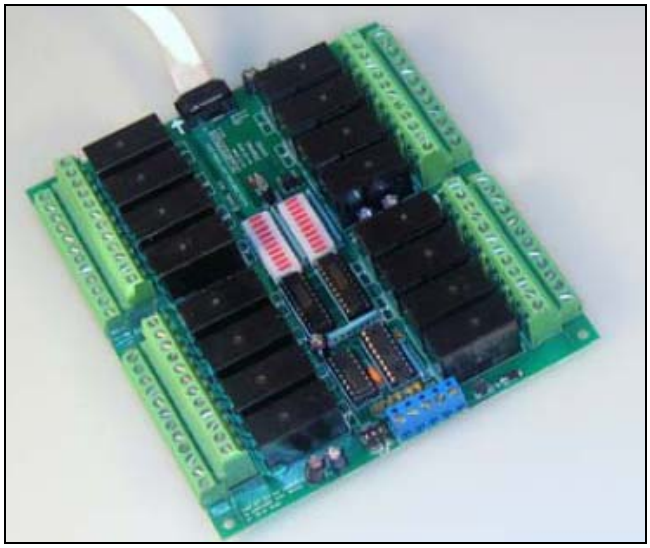

Figura 2.8: XR16xDPDT

Finalmente se optó por escoger las dos últimas placas por los siguientes motivos:

- En un primer momento se estudió la posibilidad de añadir NI USB-6229 que cumple todas las necesidades en cuanto al número de canales ADC disponibles. Por otro lado, es un dispositivo que se debe conectar vía USB para el intercambio de datos, aunque la mayor desventaja que presenta en nuestro caso, es que el software que soporta sólo está disponible para Windows, y no para trabajar con la plataforma uClinux.
- Se pensó en el PIC24HJ aunque finalmente se optó por la serie ProXR (AD1232PROXR y XR16xDPDT). Se vio que era la mejor opción por su facilidad de implementación y porque además permite conectar mas módulos ADC y relés de una forma muy fácil.

2.3.2. Principio de funcionamiento

Los reguladores de la serie ProXR están equipados con un puerto de expansión XR. Este puerto se utiliza para encadenar placas de reguladores de relés. De esta forma es fácil agregar más bancos para un posible incremento en las necesidades. Se pueden agregar distintos tipos de bancos de expansión en cualquier combinación. La figura 2.9 muestra una posible combinación entre módulos.

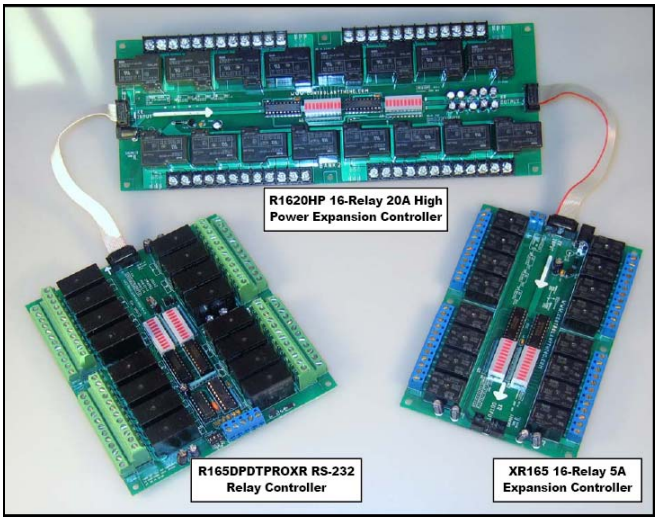

Figura 2.9: Combinación entre módulos ProXR

Comunicación a 2 hilos: Los reguladores se deben conectar según la figura 2.10 si se está utilizando el dispositivo por primera vez o si se pretenden conectar varios reguladores con un solo puerto serie. En tal caso, se debe programar un número para cada dispositivo.

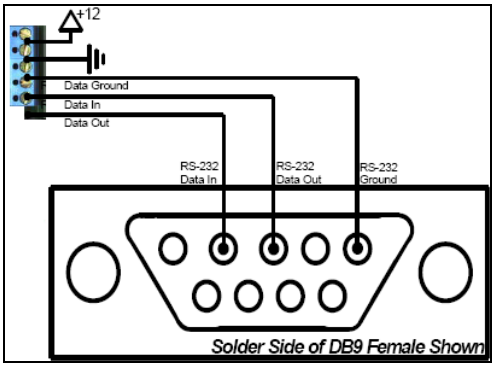

Figura 2.10: Comunicación a 2 hilos

Comunicación con 1 hilo: Los reguladores se pueden conectar a un ordenador o a un microcontrolador utilizando tan sólo 1 hilo. Cuando se utiliza este método se pueden enviar los comandos mucho más rápidamente pero no se le puede pedir información del estado de los relés. Por tanto, sólo hace una de las dos funciones. En la figura 2.11 se ve el conexionado.

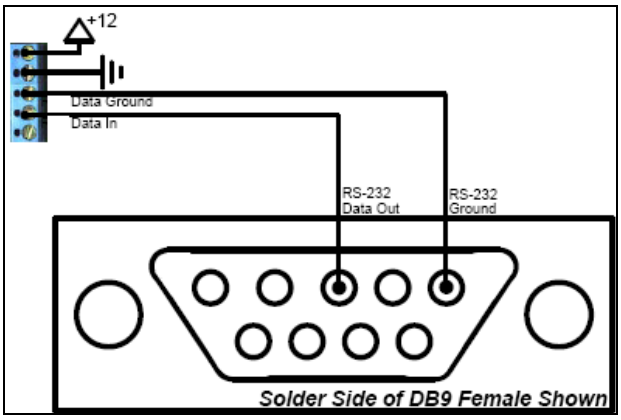

Figura 2.11: Comunicación con 1 hilo

Comunicación hibrida a 1 y 2 hilos: Los distintos dispositivos se pueden conectar con un solo puerto serie y controlarlos individualmente. Antes de utilizar la configuración del cableado que se muestra en la figura 2.12 es necesario programar un único número de dispositivo para cada placa. Una vez guardado el número de dispositivo se pueden utilizar hasta 256 placas distintas.
En la figura 2.12 vemos cómo el dispositivo 1 sólo puede enviar comandos pero no recibir, en cambio, el dispositivo 2 puede enviar y a la vez recibir.

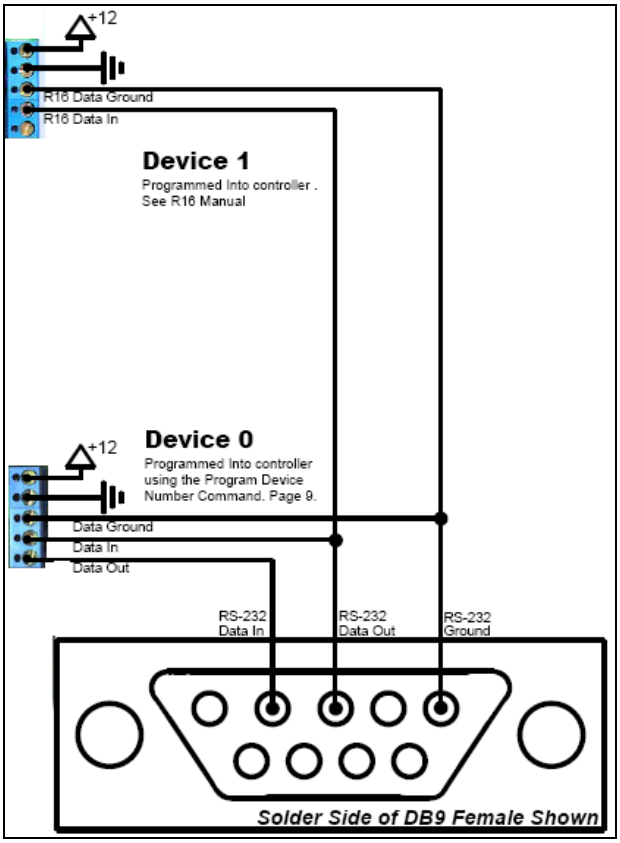

Figura 2.12: Comunicación hibrida

2.3.3. Sistema de comandos E3C

El sistema de comandos E3C permite controlar hasta un total de 256 dispositivos con un solo puerto serie. Se pueden mezclar diferentes dispositivos siempre y cuando soporten E3C.
Los reguladores soportan el sistema completo de comandos E3C más un sistema de comandos extendido para almacenar y recordar el número de dispositivo.
Para la correcta utilización primero se le debe asignar un número entre el 0 y 255 a cada dispositivo.

Comandos E3C:

254, 248 Enable All Devices:
Tells all devices to respond to your commands.

254, 249 Disable All Devices:
Tells all devices to ignore your commands.

254, 250 Enable a Selected Device:
Tells a specific device to listen to your commands.

254, 251 Disable Selected Device:
Tells a specific device to ignore your commands.

254, 252 Enable Selected Device Only:
Tells a specific device to listen to your commands, all other devices will ignore your commands.

254, 253 Disable a Selected Device Only:
Tells a specific device to ignore your commands, all others will listen.

Figura 2.12: Comandos E3C básicos

Comandos E3C extendidos:

La serie de controladores ProXR soporta tres comandos adicionales. Éstos se deben utilizar sólo cuando hay un dispositivo conectado al puerto serie.

254, 255 Store Device Number:
Stores the device number into the controller. The device number takes effect immediately. The enabled/disabled status of the device is unchanged.

254, 246 Recall Device Identification:
This command reports back 4 bytes of data:
ProXR Device ID Part 1: 1
ProXR Device ID Part 2: 0
ProXR Firmware Version: 17 (or newer)
ProXR Year of Firmware Production: 205 (or newer)
ProXR E3C Device Number

254, 247 Recall Device Number:
Reads the stored device number from the controller.

Figura 2.13: Comandos E3C extendidos

2.3.4. Características

Un banco de relés es un grupo de 8 relés. La serie ProXR permite controlar hasta un máximo de 256 relés (que corresponde a 32 bancos).
Hay dos maneras distintas de especificar de qué banco se está hablando.
Cuando debamos especificar el banco en el que queremos leer o escribir deberemos poner un número entre 0 y 32. Un valor de 1 hablará del banco 1, pudiendo seleccionar hasta 8 relés. Lo mismo para los 31 bancos restantes.
Hay que decir que si seleccionamos el banco 0, se estará leyendo o escribiendo en todos los bancos disponibles.

2.3.5. Comandos E3C utilizados

A continuación se va a ver cuál es el proceso tanto de inicialización cómo de escritura / lectura de las placas ADC / Relés. Las tramas que se muestran a continuación se han sacado gracias al analizador de protocolos para el puerto serie.

Inicialización de las placas:

Los pasos necesarios para una buena inicialización de los dispositivos es la siguiente:

Test de comunicación a 2 hilos

Puerto abierto mediante el proceso "ProXR COM1.exe" (PID: 3704)

Pedido: 30/07/2008 16:56:06.515625064

254 33

Respuesta: 30/07/2008 16:56:06.531250064 (+0.0156250000 seconds)

85

A continuación se da la descripción de los dispositivos conectados.

Pedido: 30/07/2008 16:56:06.531250064 (+0.0000000000 seconds)

254 246

Respuesta: 30/07/2008 16:56:06.531250064 (+0.0000000000 seconds)

4 0 207 23 1

Antes de leer o escribir a un dispositivo es necesario saber a qué banco nos dirigimos. En este caso, al poner un “0” nos estamos refiriendo a todos los bancos.

Pedido: 30/07/2008 16:56:12.312500064 (+1.4843750000 seconds)

254 49 0

Respuesta: 30/07/2008 16:56:12.328125064 (+0.0156250000 seconds)

85

El siguiente comando nos da el estado de todos los bancos de relés

Pedido: 30/07/2008 16:56:12.328125064 (+0.0000000000 seconds)

254 24

Respuesta: 30/07/2008 16:56:12.328125064 (+0.0000000000 seconds)

252 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Antes de leer o escribir en un dispositivo se debe decir a qué banco nos dirigimos. Al poner un "1" nos estamos refiriendo al banco 1 de relés.

Pedido: 30/07/2008 16:56:13.906250064 (+1.4218750000 seconds)

254 49 1

Respuesta: 30/07/2008 16:56:13.921875064 (+0.0156250000 seconds)

85

El siguiente comando nos da el estado de los relés del banco 1. Un 252 en decimal equivale a 11111100. Eso quiere decir que solamente hay 2 relés en OFF y el resto en ON

Pedido: 30/07/2008 16:56:13.953125064 (+0.0312500000 seconds)

254 24

Respuesta: 30/07/2008 16:56:13.968750064 (+0.0156250000 seconds)

252

A continuación preguntamos en qué banco de relés estamos. Al retornarnos un "1" sabemos que estamos en el banco 1.

Pedido: 30/07/2008 16:56:13.968750064 (+0.0000000000 seconds)

254 34

Respuesta: 30/07/2008 16:56:13.968750064 (+0.0000000000 seconds)

1

Recuperamos el número del dispositivo en el que estamos.

Pedido: 30/07/2008 16:56:13.968750064 (+0.0000000000 seconds)

254 247

Respuesta: 30/07/2008 16:56:13.968750064 (+0.0000000000 seconds)

1

Los relés del banco 1 están en OFF: se ponen a ON todos:

Ponemos en ON todos los relés del banco 1.

Pedido: 30/07/2008 16:59:06.640625064 (+25.7968750000 seconds)

254 30

Respuesta: 30/07/2008 16:59:06.656250064 (+0.0156250000 seconds)

85

Leemos el estado de los relés y comprobamos que están todos en ON

Pedido: 30/07/2008 16:59:06.703125064 (+0.0468750000 seconds)

254 24

Respuesta: 30/07/2008 16:59:06.703125064 (+0.0000000000 seconds)

255

Los relés del banco 2 están en OFF: se ponen a ON todos:

Nos posicionamos en el banco 2 de relés.

Pedido: 30/07/2008 16:59:36.468750064 (+29.7656250000 seconds)

254 49 2

Respuesta: 30/07/2008 16:59:36.468750064 (+0.0000000000 seconds)

85

Leemos su estado y vemos que están todos a OFF.

Pedido: 30/07/2008 16:59:36.515625064 (+0.0468750000 seconds)

254 24

Respuesta: 30/07/2008 16:59:36.515625064 (+0.0000000000 seconds)

0

Activamos todos los relés del banco 2.

Pedido: 30/07/2008 16:59:40.250000064 (+3.7343750000 seconds)

254 30

Respuesta: 30/07/2008 16:59:40.265625064 (+0.0156250000 seconds)

85

Leemos el estado y comprobamos que se ha escrito correctamente.

Pedido: 30/07/2008 16:59:40.312500064 (+0.0468750000 seconds)

254 24

Respuesta: 30/07/2008 16:59:40.312500064 (+0.0000000000 seconds)

255

Lectura del estado de los ADC:

Leemos el estado de los 16 canales de ADC.

Pedido: 30/07/2008 17:01:22.859375064 (+61.1562500000 seconds)

254 192

Respuesta: 30/07/2008 17:01:22.859375064 (+0.0000000000 seconds)

0 0 0 0 0 0 0 0 244 236 255 217 249 255 224 240

Capitulo 3. Adaptación SNMP para ColdFire

3.1. Introducción

SNMP es un protocolo que facilita el intercambio de información entre dispositivos de una misma red. Es parte de la familia de protocolos TCP/IP y permite a los administradores del sistema supervisar tanto el estado de la red como buscar y resolver posibles problemas.

SNMP es un protocolo que funciona a nivel de aplicación, generalmente sobre UDP. Utiliza los puertos 161 y 162. El puerto 161 se utiliza para las transmisiones normales de SNMP y el 162 para el envío de traps o alarmas.

El protocolo SNMP está compuesto por dos elementos básicos, un agente y un gestor.

- El agente es un programa que ha de ejecutarse en cada nodo de la red que se desea gestionar o monitorizar. En nuestro caso se ejecuta en la placa controladora (ES).

- El gestor es el software que se ejecuta en la estación encargada de monitorizar la red. Su tarea consiste en consultar los diferentes agentes que se encuentran en los nodos. En nuestro caso se ubica en la ET.

Se pueden diferenciar dos formas de funcionamiento del protocolo SNMP:

- Comandos de Lectura /Escritura (GET / SET): es necesario que el elemento gestor tenga una base de datos llamada MIB (Management Information Base) que contiene la información de todos los dispositivos gestionados. Se pueden hacer peticiones de lectura del valor de las variables que se desean obtener o hacer peticiones de escritura a las variables sobre las que se desea actuar. En el caso de OBSEA el elemento gestor (ET) puede activar o desactivar los sensores o las fuentes de alimentación que se ubican en la ES.

- Envió de Alarmas (Traps): El elemento gestor es el receptor de dichas alarmas (llamadas traps) enviadas por la ES. Para que esto sea posible es necesario la creación del código encargado de actuar según las necesidades especificadas.

Tal y como se ha comentado anteriormente, en la figura 3.1 vemos la utilización tanto de los puertos utilizados para el envío de traps como para la recepción de las variables a petición de la entidad gestora.
Además, en la figura 3.2 hay otro esquema de comunicación entre ET y ES del OBSEA. Mediante el cable de fibra óptica unimos las dos estaciones. El nodo agente está en la estación submarina y se encarga de enviar las alarmas o traps a la ET según lo programado. Por otro lado, el nodo gestor ubicado en la estación terrestre es el encargado de recibir dichas alarmas y de pedirle al agente las variables que crea oportunas para su monitorización.

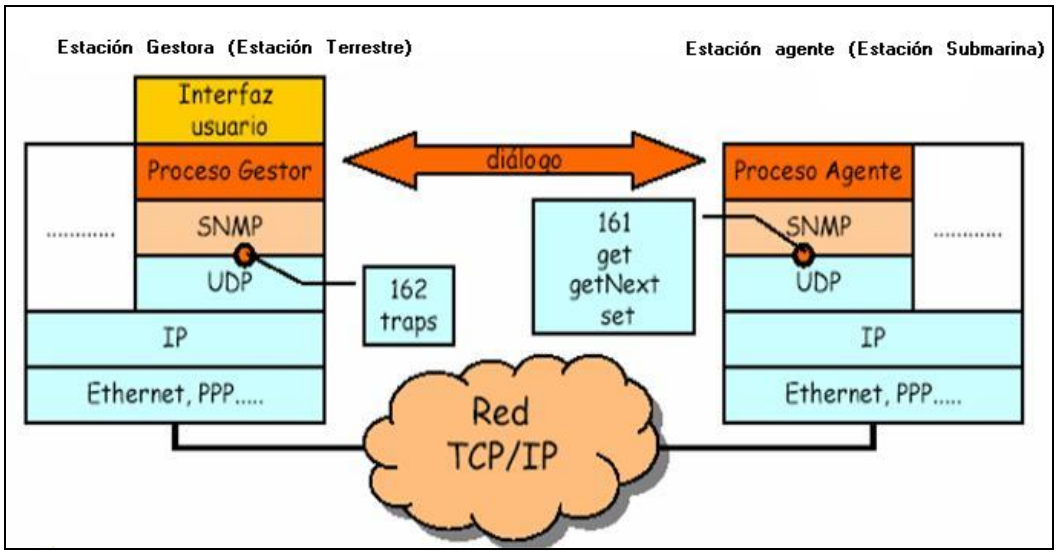

Figura 3.1: Esquema de utilización de puertos para el protocolo SNMP

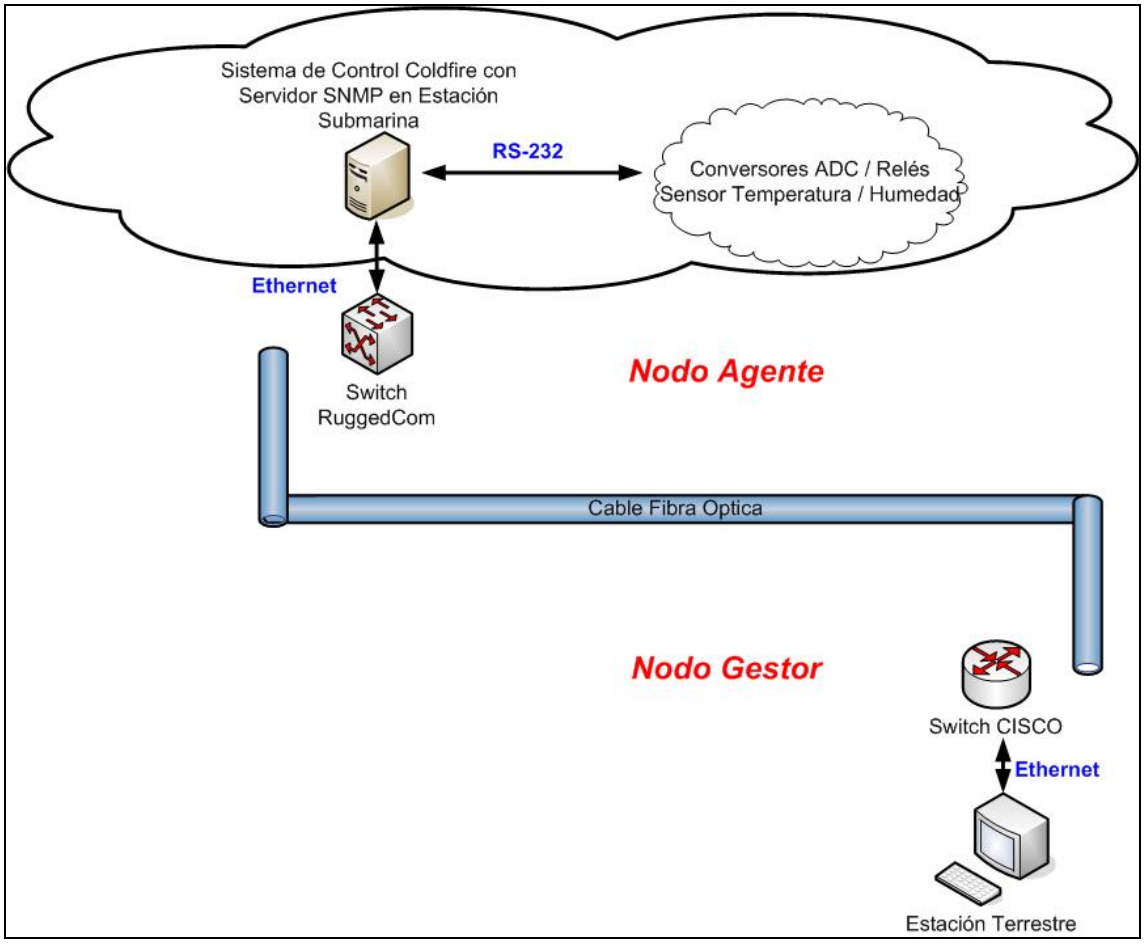

Figura 3.2: Esquema básico de la comunicación entre ET y ES

3.2. Estructura de la información

Como se ha dicho, para poder funcionar mediante comandos de Lectura / Escritura es necesario que el elemento gestor tenga una base de datos (MIB) para poder obtener la información de los dispositivos gestionados. Para ello, es necesario guardar un único identificador para cada objeto gestionado. Éste identificador se conoce con el nombre OID (Identificador de Objeto) y está formado por una secuencia de números enteros positivos separados por puntos que construyen un árbol tal y como se muestra en la figura 3.4. En la figura 3.3 podemos ver la declaración de éstos.

```
static oid dycec_oid[] = { 1,3,6,1,4,4,12592 };
static oid adc00_oid[] = { 1,3,6,1,4,1,12592,1,1, 0 };
static oid adc01_oid[] = { 1,3,6,1,4,1,12592,1,2, 0 };
static oid adc02_oid[] = { 1,3,6,1,4,1,12592,1,3, 0 };
.....
static oid adc13_oid[] = { 1,3,6,1,4,1,12592,1,14, 0 };
static oid adc14_oid[] = { 1,3,6,1,4,1,12592,1,15, 0 };
static oid adc15_oid[] = { 1,3,6,1,4,1,12592,1,16, 0 };
static oid alarma00_oid[] = { 1,3,6,1,4,1,12592,2,1, 0 };
static oid alarma01_oid[] = { 1,3,6,1,4,1,12592,2,2, 0 };
.....
static oid alarma16_oid[] = { 1,3,6,1,4,1,12592,2,17, 0 };
static oid alarma17_oid[] = { 1,3,6,1,4,1,12592,2,18, 0 };
static oid adc2_00_oid[] = { 1,3,6,1,4,1,12592,3,1, 0 };
static oid adc2_01_oid[] = { 1,3,6,1,4,1,12592,3,2, 0 };
.....
static oid adc2_14_oid[] = { 1,3,6,1,4,1,12592,3,15, 0 };
static oid adc2_15_oid[] = { 1,3,6,1,4,1,12592,3,16, 0 };
static oid R_BANCO_0_ALL_oid[] = { 1,3,6,1,4,1,12592,4,1, 0 };
static oid R_BANCO_1_ALL_oid[] = { 1,3,6,1,4,1,12592,4,2, 0 };
static oid R_BANCO_2_ALL_oid[] = { 1,3,6,1,4,1,12592,4,3, 0 };
static oid R_BANCO_3_ALL_oid[] = { 1,3,6,1,4,1,12592,4,4, 0 };
```

Figura 3.3: Declaración del OID para cada objeto

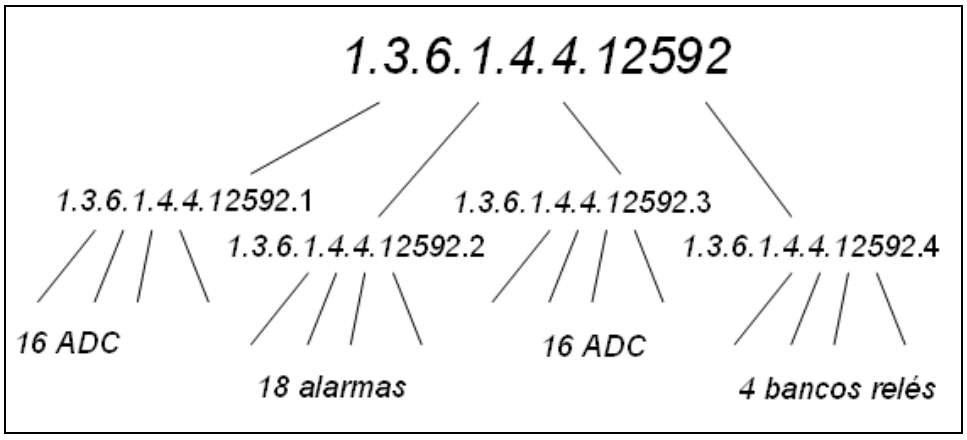

Figura 3.4: Árbol de OIDs

La MIB está escrita utilizando la sintaxis ASN.1. Ésta es una norma para representar datos independientemente de la maquina que se esté utilizando. Los tipos de datos se clasifican de la siguiente manera:

- Tipos primitivos: almacenan un único valor como por ejemplo un entero o una cadena de caracteres. Los más importantes son los siguientes:
    - *INTEGER*: para representar números enteros
    - *OCTET STRING*: para almacenar una secuencia de bytes. Se dividen en:
        - DsiplayString: para cadenas ASCII
        - OctetBitString: para cadenas de bits mayores de 32
        - PhysAddress: para representar direcciones
    - *OBJECT IDENTIFIER*: para representar los identificadores de los objetos o OID
    - *BOOLEAN*: representan valores que pueden ser verdadero o falso
    - *NULL*: representa la ausencia de valor
- Tipos construidos: almacenan tipos compuestos tales como tablas o arrays. Los más importantes son los siguientes:
    - *SEQUENCE*: representa una estructura de datos como por ejemplo una fila de una tabla
    - *SEQUENCE OF*: es similar al SEQUENCE excepto que todos los tipos tiene que ser iguales
    - *SET*: es similar al SEQUENCE pero la lista no tiene porque estar ordenada
    - *SET OF*: es similar al SEQUENCE OF pero la lista no tiene porque estar ordenada
    - *CHOICE*: representa un tipo de datos a elegir entre los disponibles en una lista
- Tipos definidos: almacenan tipos derivados de los anteriores pero con un nombre mas descriptivo. Los más importantes son:
    - *IpAddress*: representa una dirección IP. Son 4 bytes y se definen como “OCTET STRNG (SISE (4))”
    - *Counter*: representa un contador que únicamente puede incrementar su valor hasta llegar a su valor máximo, que vuelve a cero.
    - *Gauge*: representa un indicador de nivel. Es un valor que puede incrementarse o decrementarse.
    - *TimeTicks*: es un tipo de datos que se utiliza para medir tiempos. Es un entero de 32 bits

3.3. Operaciones del protocolo

En el protocolo SNMP existen los siguientes comandos básicos:

- *GET* (obtener información): el nodo gestor obtiene datos del dispositivo agente.
- *PUT* (actualizar información): el nodo gestor establece los valores de variables del dispositivo agente.
- *TRA*P (interrupción): el dispositivo agente notifica al gestor acerca de ciertos sucesos de importancia que se definen previamente.

3.4. Primitivas

SNMP define 8 tipos de primitivas:

- GET REQUEST: Solicita un atributo de un objeto. La transmite el nodo administrador y contesta el nodo agente.
- GET NEXT REQUEST: Solicita el siguiente atributo de un objeto. La transmite el nodo administrador y contesta el agente.
- GET BULK REQUEST: Solicita un conjunto amplio de atributos en lugar de irlo solicitando de uno en uno.
- SET REQUEST: Actualiza un atributo de un objeto.
- SET NEXT REQUEST: Actualiza el siguiente atributo del objeto.
- GET RESPONSE: Transmitida por el agente. Devuelve los atributos solicitados. La recibe el nodo administrador.

- TRAP: Informa de fallos. La transmite el agente.

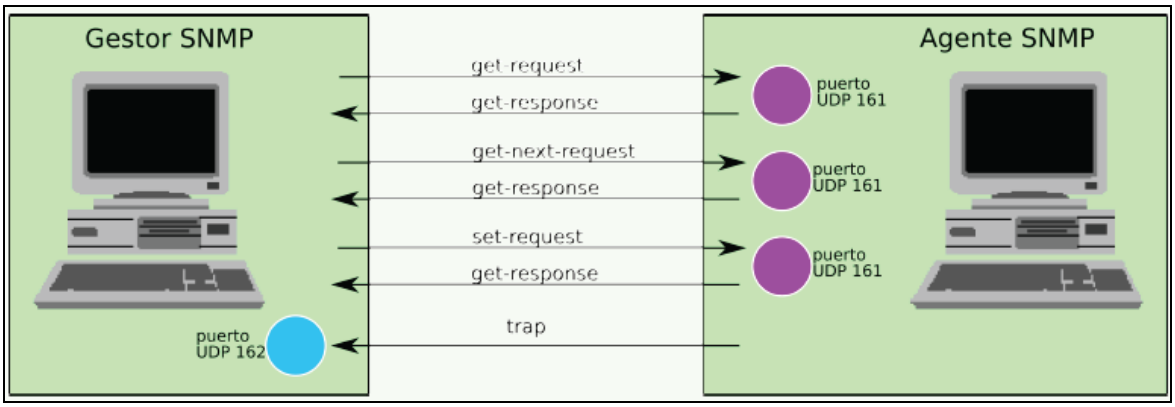

Figura 3.5: Visualización de primitivas SNMP

3.5. Formato de mensajes SNMP

El formato de los mensajes SNMP se muestran en la figura 3.6 y 3.7. El primero de ellos corresponde a mensajes SNMP y el segundo a alarmas o Traps.

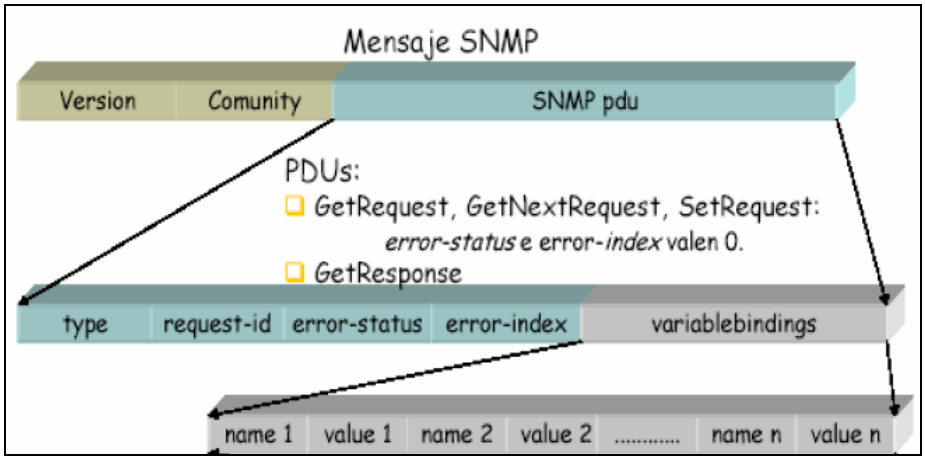

Figura 3.6: Formato mensaje SNMP

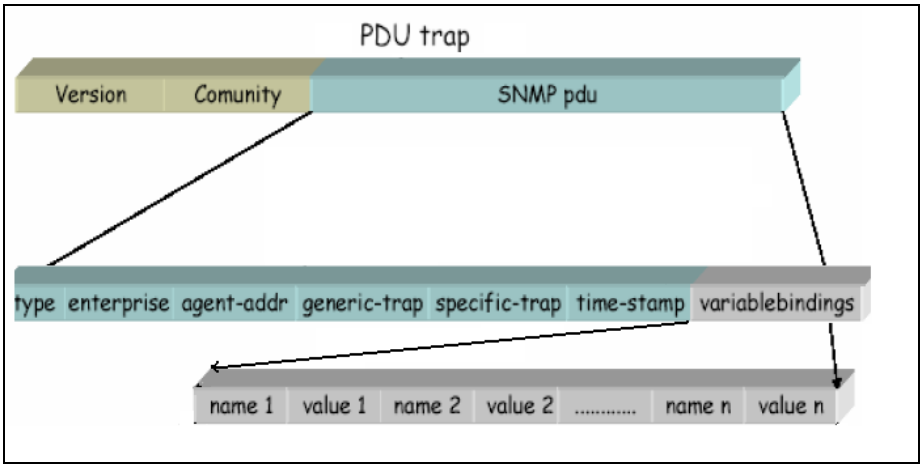

Figura 3.7: Formato mensaje trap

El significado de cada campo es el siguiente:

- **Versión:** Versión del protocolo SNMP.
- **Community:** Comunidad que permite la autenticación del gestor ante el agente y el control de acceso.
- **Request-id:** Identificador único para cada petición, es el mismo en la respuesta correspondiente.
- **Error-status:** Indica si se ha producido un error cuando se procesa una petición. Los código son: noError(0), tooBig(1), noSuchName(2), badValue(3), readOnly(4), genErr(5). Vale 0 en caso de getRequest, getNextRequest y setRequest.
- **Error-index:** Cuando se ha producido un error, puede aportar información, indicando en qué instancia de la lista de variablebindings se ha producido el error. Vale 0 en caso de getRequest, getNextRequest y setRequest.
- **Variablebindings:** Lista de nombres de variables (OIDs) y sus correspondientes valores. Los valores son nulos (NULL) en casos como GetRequest, GetNextRequest o GetResponse con error. Permite múltiples operaciones en el mismo mensaje.
- **Enterprise:** Tipo de sistema que genera el trap.
- **Agent-addr:** Dirección IP del agente que genera el trap.
- **Generic-trap:** Código de trap genérico.
- **Specific-trap:** Código de trap específico.
- **Time-stamp:** Tiempo desde la última inicialización del agente y la generación del trap (valor de sysUpTime).

En las figuras 3.8 y 3.9 se puede ver el formato de un trap GET visualizado con el analizador de redes ethereal. En la figura 3.8 se ha realizado un GET sobre un objeto y en la figura 3.9 se muestra lo que el controlador ColdFire ha retornado.

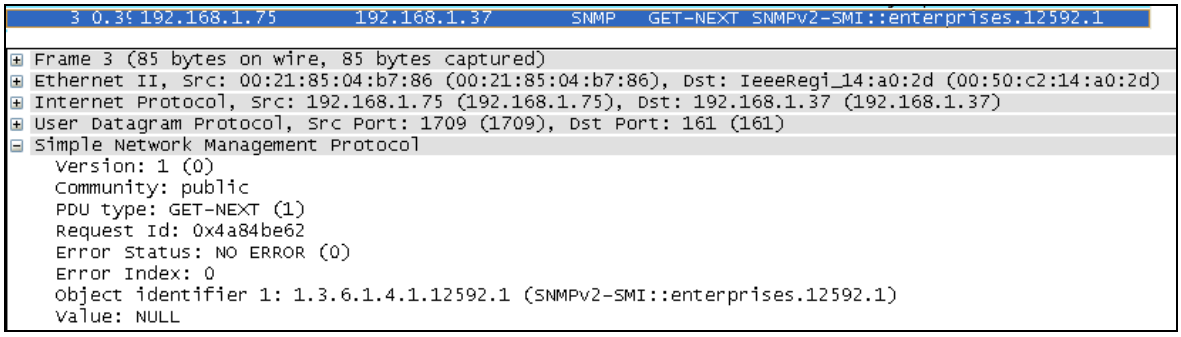

Figura 3.8: Formato mensaje GET con ethereal

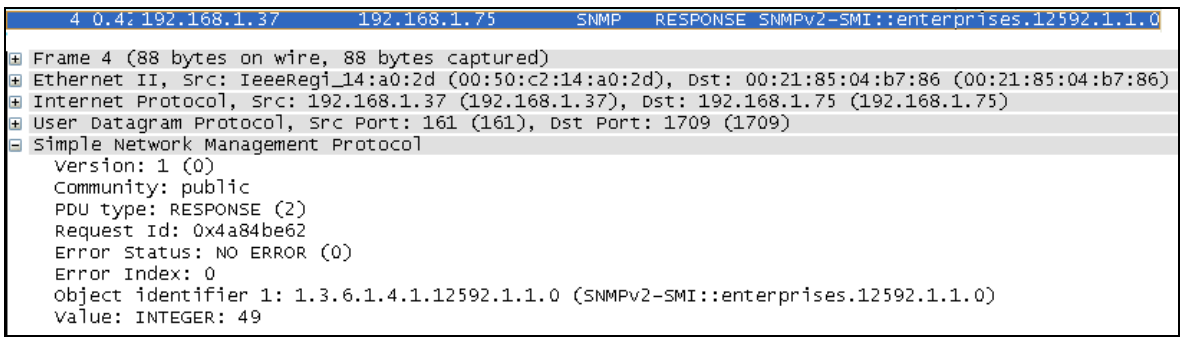

Figura 3.9: Formato mensaje GET- RESPONSE con ethereal

En cambio, en la figura 3.10 podemos ver el formato de un trap.

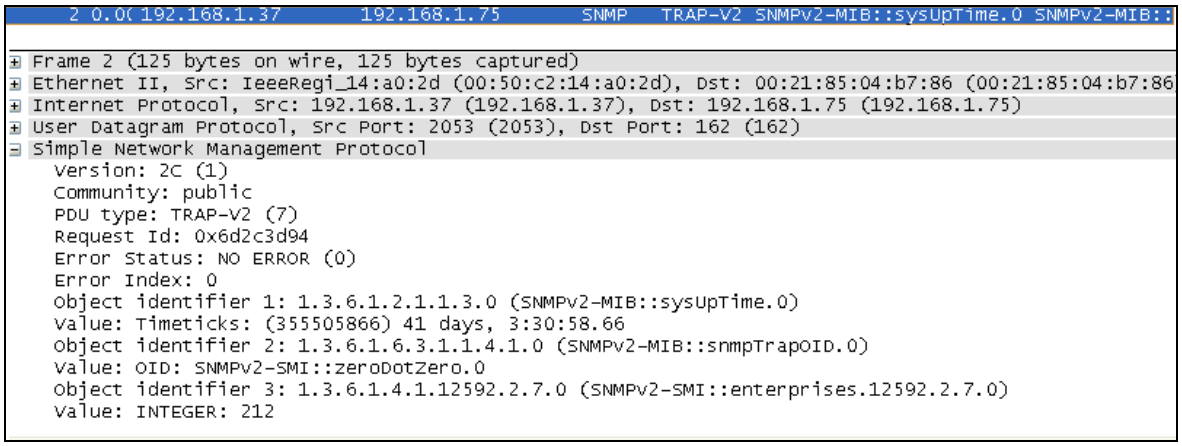

Figura 3.10: Formato mensaje trap SNMP con ethereal

3.6. MIB del proyecto

Una vez se ha creado la MIB (Anexo B) y compilado el proyecto sin errores podemos ver los resultados obtenidos de dos formas:

- Podemos visualizar la jerarquía de las variables SNMP en formato árbol utilizando el software adecuado. En nuestro caso se ha utilizado “MIB Browser”.
- Podemos visualizar las alarmas que se reciben en el caso de que ocurra algún evento. En nuestro caso se han utilizado 2 analizadores de redes distintos, “wireshark” y “ethereal”.

3.7. Visualización de variables SNMP en formato árbol

A continuación se hace una breve descripción del software “MIB-Browser” en el que se visualizan las variables SNMP en formato árbol. El software permite explorar la estructura de una MIB de forma amigable. También ofrece la posibilidad de enviar peticiones de lectura o escritura a un agente SNMP.

En la figura 3.11 se muestra el aspecto del programa. Primero de todo debemos introducir la dirección IP del agente del que vamos a extraer la información. En nuestro caso el agente tiene la dirección 192.168.1.34.
Debajo de la IP se ve la base de datos MIB cargada. Ésta se llama “DYCEC-MIB” y es la base de datos que se ha modificado para la realización del proyecto. La raíz de la librería DYCEC-MIB tiene el siguiente OID .1.3.6.1.4.1.12592. Cómo se ha dicho, éste es un número único que identifica la base de datos por la que vamos a navegar.

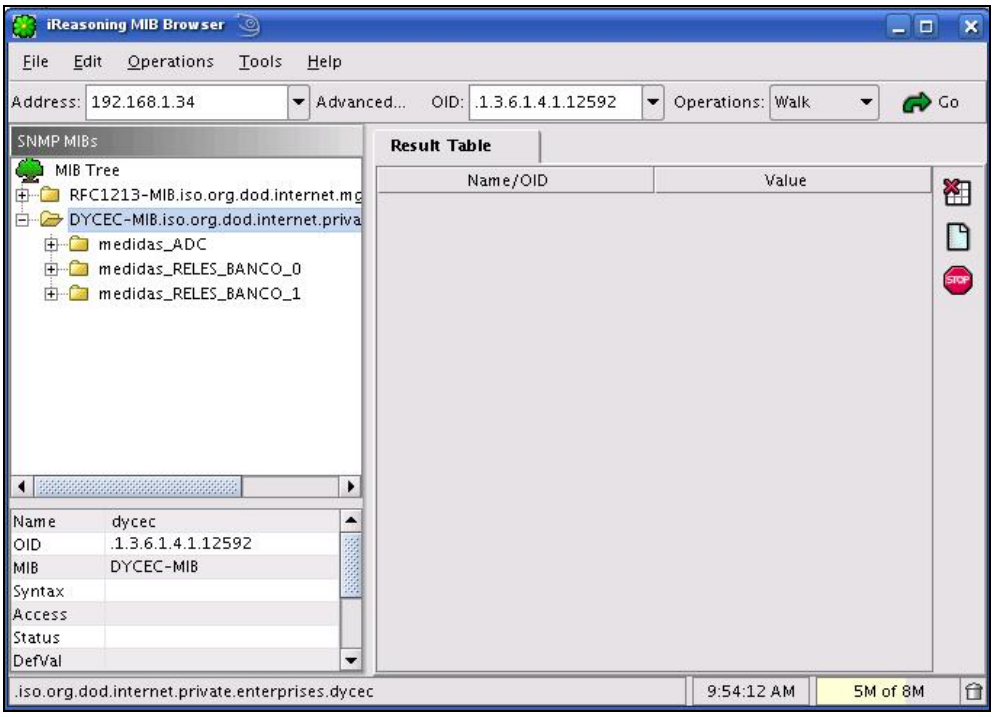

Figura 3.11: Visualización de la MIB en iReasoning

Para poder extraer la información que se desea, se deben configurar los parámetros relativos al protocolo de gestión. Estos parámetros son las comunidades de lectura y escritura que nos permiten leer o fijar parámetros mediante *gets* o *sets*. Una comunidad es un nombre o palabra clave que se utiliza como “contraseña” para tener acceso a la visualización o actuación sobre los objetos. Generalmente existe una comunidad de lectura llamada "public" y una comunidad de escritura llamada "private" aunque por motivos de seguridad es aconsejable cambiarlas.

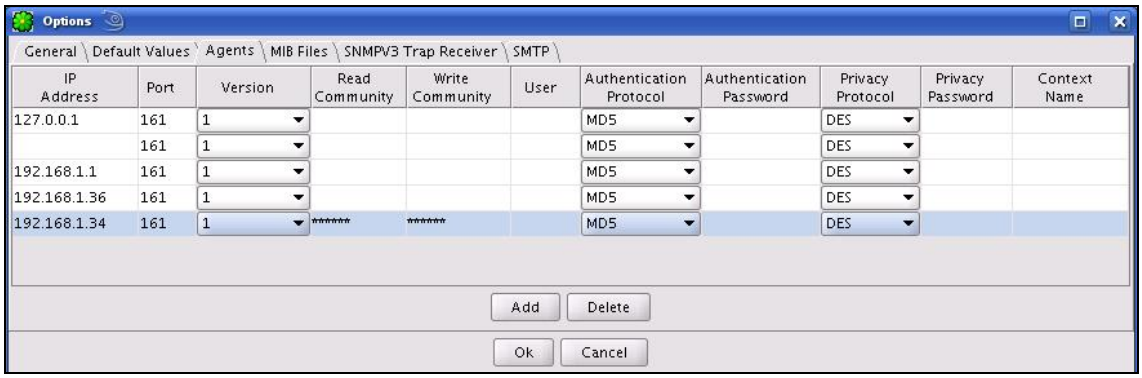

Figura 3.12: Configuración de parámetros en iReasoning

En la MIB “DYCEC-MIB” podemos acceder a los valores de los conversores ADC y al valor del estado de los Reles (del Banco 0 y 1).
Si clicamos con el botón derecho en “medidas_ADC” y hacemos un “Get Subtree” accederemos a los 16 valores ADC a la vez tal y como se muestra en la figura 3.13.

Además también podemos acceder a los valores de los relés. La diferencia que tienen éstos sobre los conversores ADC es que hay la posibilidad de actuar remotamente sobre ellos. En el caso de los relés podemos leer y/o cambiar su estado.

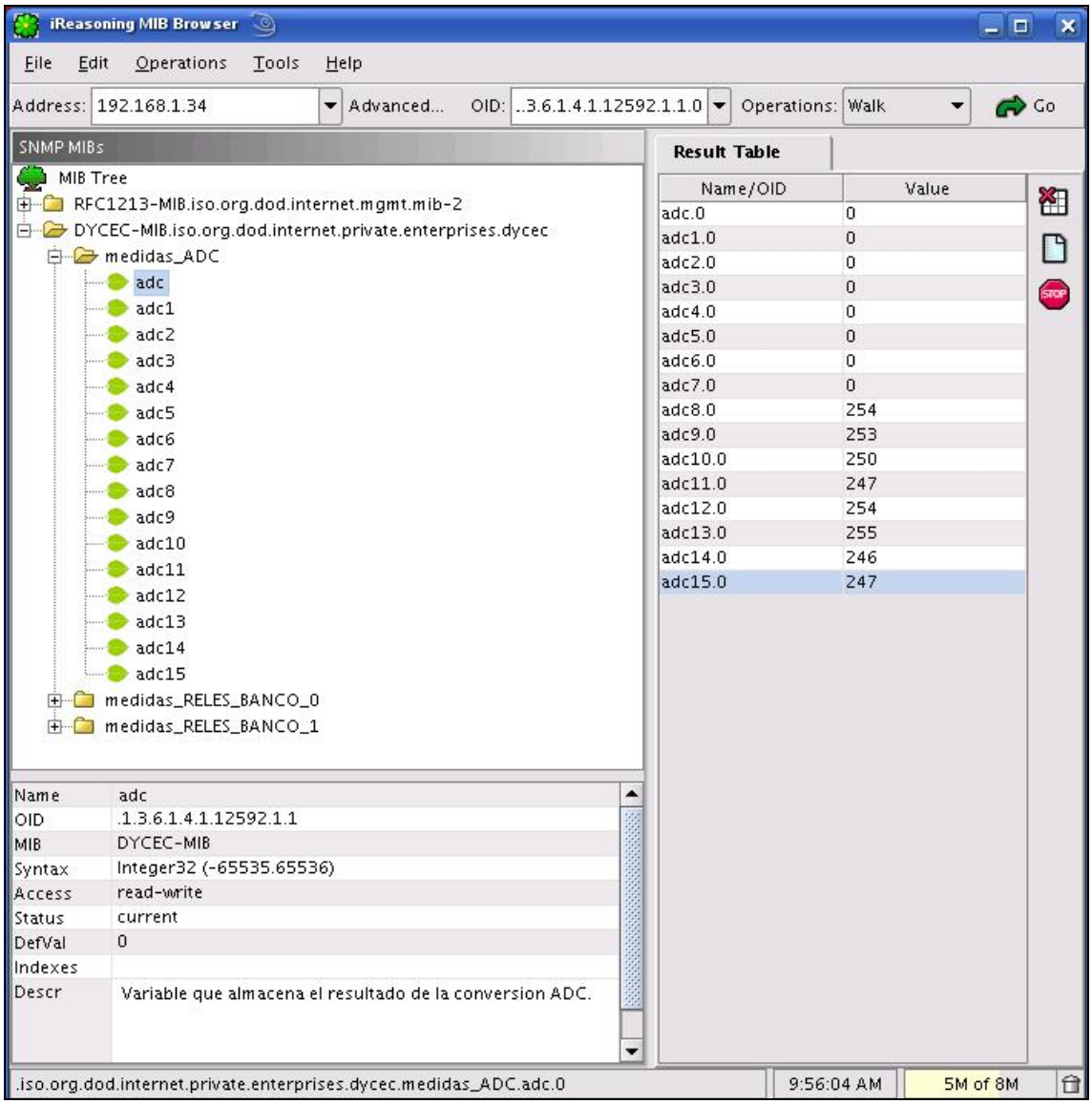

Figura 3.13: Visualización valores ADC en iReasoning

Para poder fijar el valor de un relé es necesario saber el tipo de dato que se debe introducir. En nuestro caso, cuando se ha creado la base de datos MIB y el código necesario para que la aplicación funcione se han definido los relés como enteros, por lo tanto, para cambiar su estado, en "Data Type" debemos seleccionar que sea del tipo "Integer". Los pasos necesarios para cambiar el estado de un relé son los siguientes, además también se muestra en la figura 3.14.

- Clicamos con el botón derecho sobre el relé al que se quiere actuar, por ejemplo en "reles00" y seleccionamos SET.
- Seleccionamos el tipo de dato ("Integer").
- Fijamos el valor deseado. En nuestro caso si introducimos un "0" significa que queremos apagar el relé y si introducimos un "1" significa que queremos activar el relé.
- Por último, para comprobar si su estado ha cambiado, clicamos con el botón derecho sobre el relé que hemos apagado o encendido y realizamos un "GET.

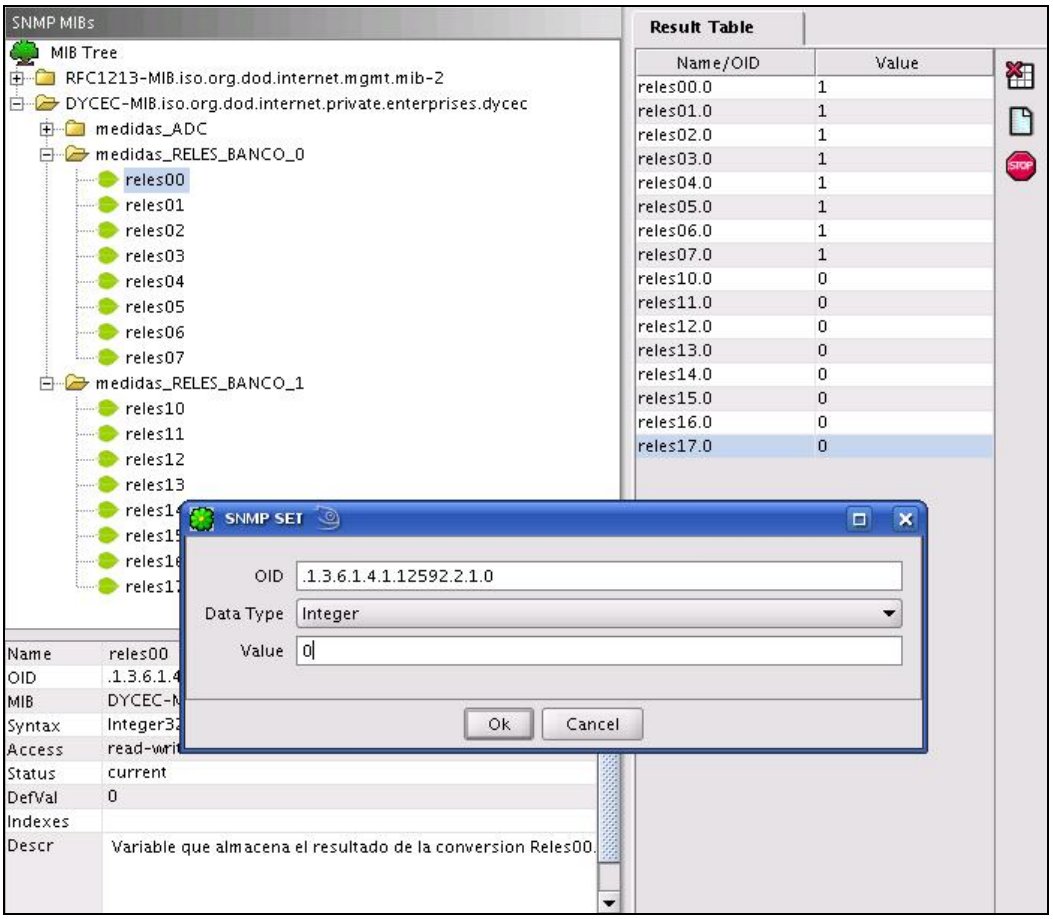

Figura 3.14: Modificación del valor de los relés en iReasoning

3.8. Visualización de alarmas o traps

Como ya sabemos, el protocolo SNMP puede funcionar de dos formas distintas. Anteriormente hemos comentado cómo leer o fijar valores de objetos. Ahora pasamos a explicar cómo visualizar alarmas o "traps" que nos puede enviar el agente en el caso de que algún parámetro no funcione correctamente.

Para visualizar dichas alarmas es necesario obtener un programa que nos permita ver el tráfico que hay en la red. En nuestro caso se ha utilizado un programa llamado "wireshark". Éste es un analizador de protocolos utilizado para hacer análisis y solucionar problemas en redes. De esta forma podemos saber en todo momento si se están enviando correctamente las alarmas.
Para utilizar el software debemos configurarlo de manera que sólo recibamos paquetes UDP. Para ello debemos ir a "Capture"→ "Interfaces" y aparecerá una ventana como la de figura 3.15. Luego debemos entrar en las opciones de "eth1" que aparecerá una ventana parecida a la de la figura 3.16.

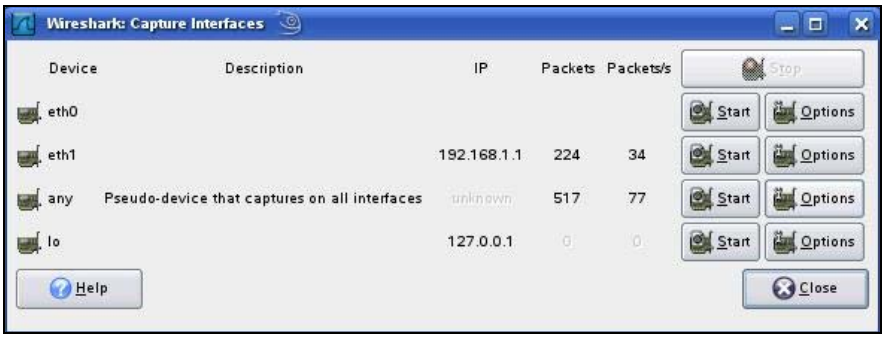

Figura 3.15: Configuración de paquetes en wireshark

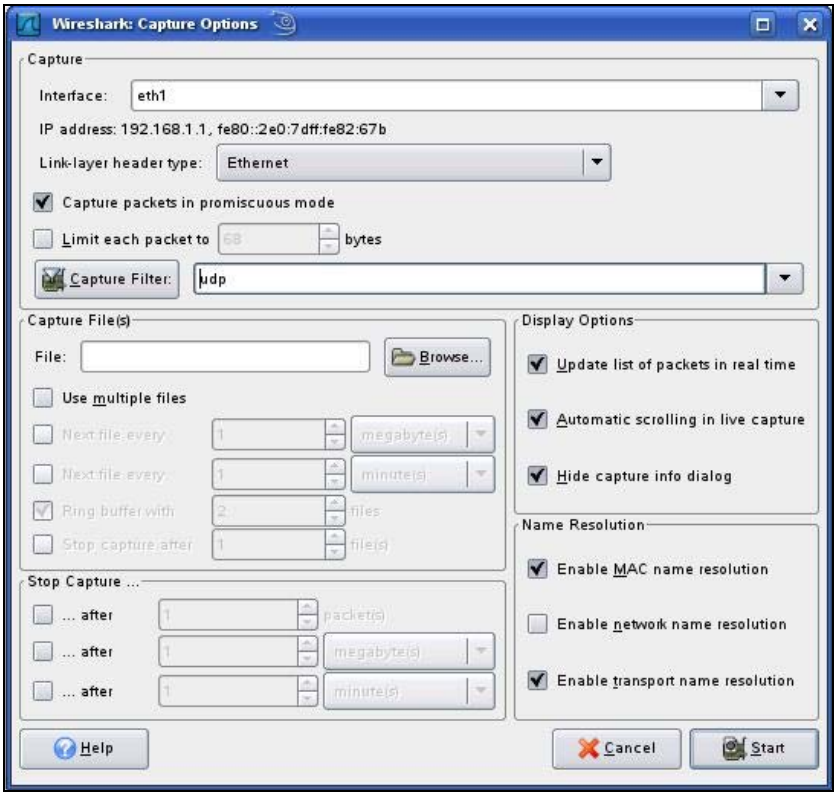

Figura 3.16: Configuración wireshark para filtrar paquetes UDP

Una vez se ha configurado correctamente el software y empezamos a recibir traps nos aparecerá una ventana como la de la figura 3.17 que está dividida en tres zonas. En la zona superior vemos los envíos de los paquetes capturados. En la zona del centro tenemos toda la información desglosada de dichos paquetes. Y en la zona inferior tenemos el contenido del paquete en formato hexadecimal.

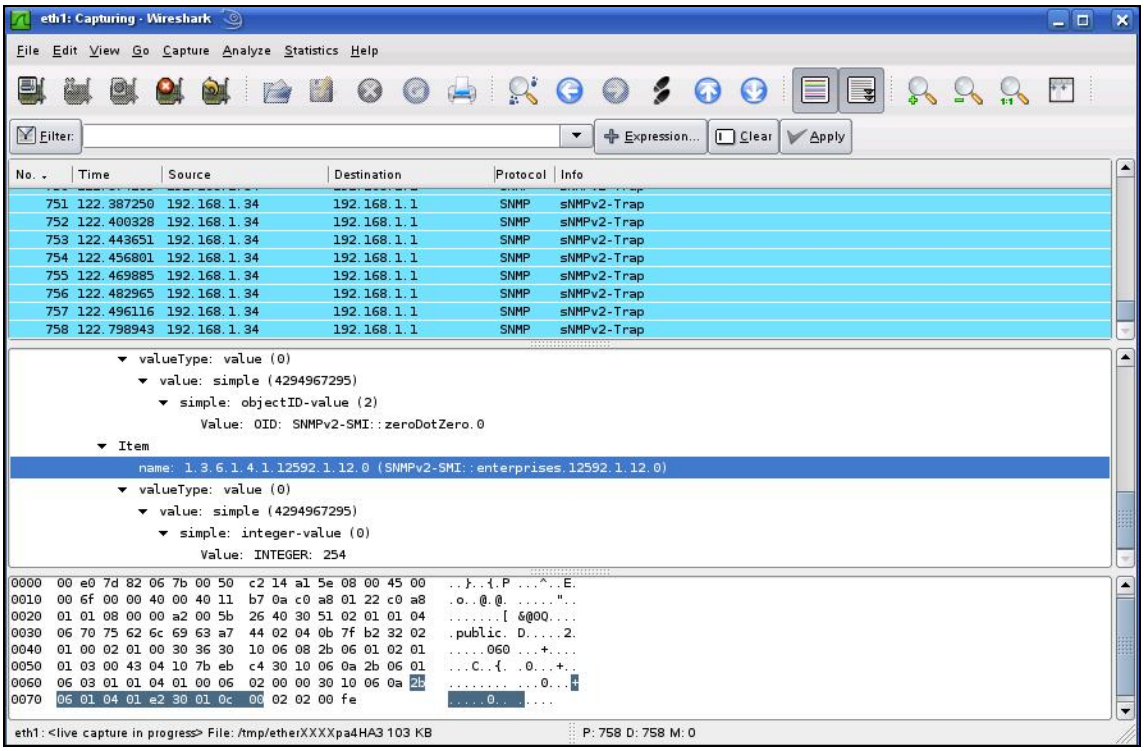

Figura 3.17: Ventana de recibo de paquetes en wireshark

Capitulo 4. Características del software

La aplicación que corre sobre la placa consiste en un programa que monitoriza la temperatura, el estado de los relés y los datos de los conversores ADC. Además, envía traps SNMP si los valores superan unos rangos predeterminados. Está compuesta también de un menú de configuración para los parámetros más comunes de la placa.

4.1. Estructura de los directorios del Proyecto

El entorno de desarrollo utilizado ha sido kdevelop. A continuación se describe la estructura de los directorios de la aplicación.

1. aplicacion: Directorio del programa principal de la aplicación
2. exec: Aquí están los ejecutables tras la compilación del proyecto. Son los que se vuelcan a la placa.
3. include: Archivos cabecera del proyecto.
4. menu: Directorio para la programación de interfaz de usuario.
5. modulos: Módulos software que cubren diferentes aspectos de la aplicación (memoria compartida, RS-232, etc.).
6. snmp: Directorio del demonio SNMPD.

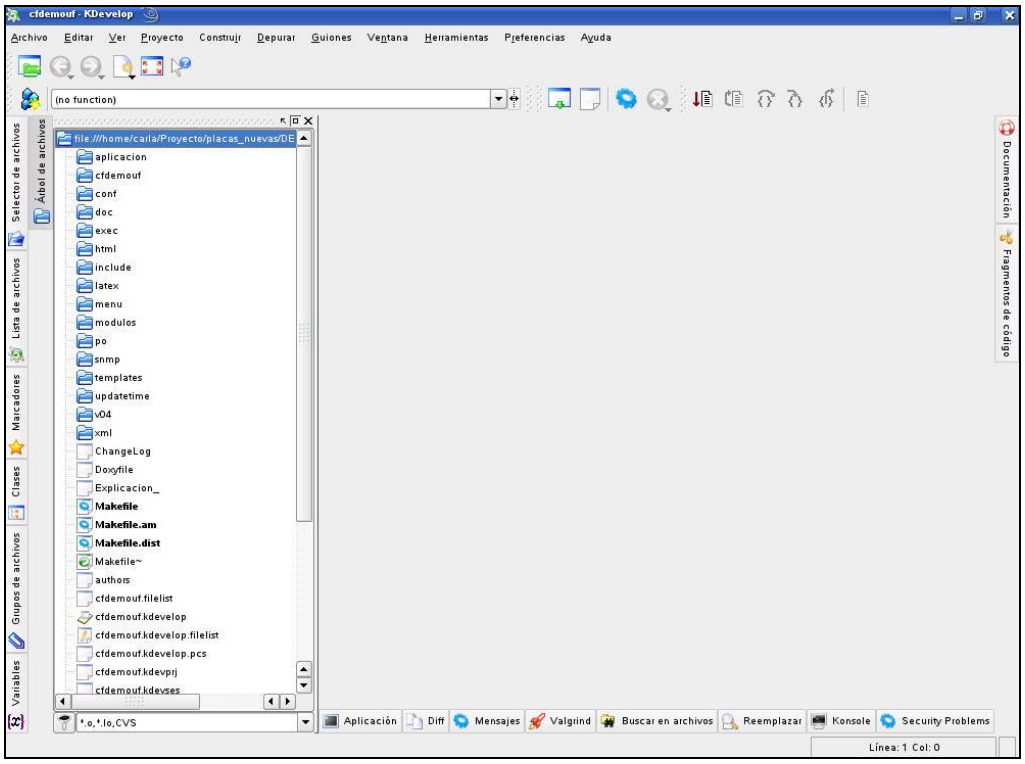

Figura 4.1: Entorno de desarrollo kdevelop

4.2. Makefile

4.2.1. Introducción

Uno de los aspectos más importantes del proyecto y con lo que se debe tener mucho cuidado son los Makefile.
Los Makefiles son ficheros de texto que utilizan el comando "make" para llevar la gestión de la compilación de programas. Es una herramienta que nos ayuda cuando estamos ante grandes proyectos.

Antes de entrar en detalle, decir que GCC es un compilador rápido, flexible, y riguroso con el estándar de C ANSI. GCC se encarga de realizar el pre procesado del código, la compilación, y el enlazado. Nosotros le proporcionamos a GCC el código fuente en C, y él nos devuelve un archivo binario compilado para nuestra maquina. Dicho esto, a continuación se van a ver los aspectos más importantes a la hora de realizar una compilación.

4.2.1.1. Clases de ficheros para el compilador

En el transcurso de la ejecución de GCC aparecen varias clases de ficheros, que tal y como podemos ver a continuación pueden ser fuentes, objetos, bibliotecas o ejecutables.

.c	fichero fuente en C
.h	fichero cabecera fuente en C (sólo útil para los **`#include`**)
.s	fichero fuente en ensamblador (también reconocido)
.i	fichero fuente tras ser pre procesado (raramente empleado por el usuario)
.o	fichero objeto
.a	fichero de biblioteca

4.2.1.2. Opciones para el compilador

Algunas de las opciones básicas para el compilador son las siguientes:

`-Aa`	compila en C ANSI
`-Ac`	compila en C de K&R
`-c`	sólo compila, no hace el montaje
`-I` ***directorio***	define directorios para buscar los **`#includes`**
`-l` ***lib***	monta la biblioteca ***lib***
`-L` ***directorio***	define directorios para buscar las bibliotecas
`-o` ***fichero***	establece el nombre del ejecutable

En el siguiente ejemplo se muestra un ejemplo muy básico de compilación:

cc -o nombre main.c main2.c nombre.o

El ejecutable de la compilación de “main” y “main2” se llamará “nombre”, igual que el objeto.
En las figuras 4.2 y 4.3 se pueden ver cómo sería una compilación simple y una en la que intervienen varios archivos.

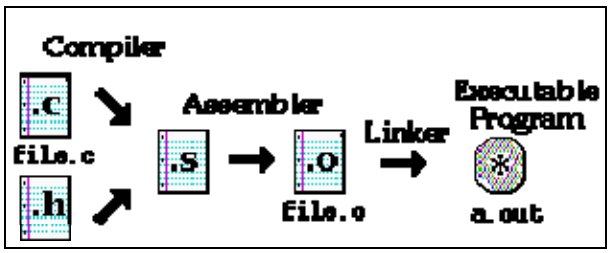

Figura 4.2: Compilación Simple

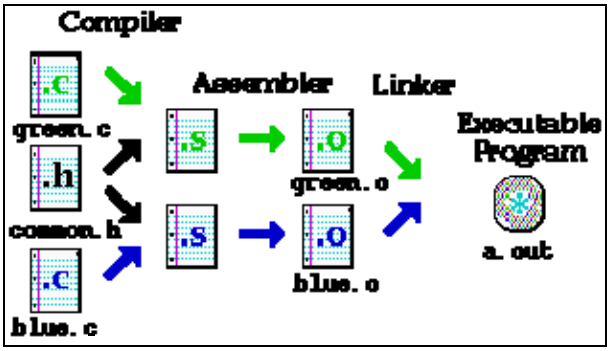

Figura 4.3: Compilación de varios archivos

En nuestro caso, el fichero Makefile principal es el que se encuentra en el directorio base del proyecto y detalla los distintos directorios que contiene software a compilar. Además, en cada uno de los directorios también se debe incluir un Makefile secundario con las reglas de compilación de ese directorio.

4.2.2. Reglas Makefile

Todos los Makefiles están ordenados en forma de reglas. El formato de cada una de esas reglas es el siguiente:

objetivo: dependencias
comandos

En “objetivo” definimos el módulo o programa que queremos crear, después de los dos puntos y en la misma línea podemos definir qué otros módulos o programas son necesarios para conseguir el “objetivo”. Por último, en las líneas siguientes y sucesivas indicamos los comandos necesarios para llevar esto a cabo. Es muy importante que los comandos estén separados por un tabulador de comienzo de línea. Un ejemplo podría ser el siguiente:

coche : ventana.o motor.o centralita.o
gcc –O2 –c coche.c –o coche.o
gcc –O2 coche.o ventana.o motor.o centralita.o –o coche

Para crear “coche” es necesario que se hayan creado “ventana.o”, “motor.o” y “centralita.o” (típicamente habrá una regla para cada uno de esos ficheros objeto en ese mismo Makefile).
Es muy habitual que existan variables en los ficheros Makefile, para facilitar su portabilidad a diferentes plataformas y entornos. La forma de definir una variable es muy sencilla, basta con indicar el nombre de la variable y su valor, de la siguiente forma:

```
CC = gcc –O2
```

Cuando queramos acceder al contenido de esa variable, lo haremos así:

```
$(CC) coche.c –o coche
```

Es relativamente habitual que además de las reglas normales, los ficheros Makefile puedan contener reglas virtuales, que no generan un fichero en concreto, sino que sirven para realizar una determinada acción dentro de nuestro proyecto software. Normalmente estas reglas suelen tener un objetivo, pero ninguna dependencia. El ejemplo más típico de este tipo de reglas es la regla “clean” que incluyen casi la totalidad de Makefiles, utilizada para “limpiar” los ficheros ejecutables y ficheros objeto de los directorios que haga falta, con el propósito de rehacerlo todo la próxima vez que se llame a “make”:

```
clean:
        rm –f coche *.o
```

Esto provocaría que cuando alguien ejecutase “make clean”, el comando asociado se ejecutase y borrase el fichero “coche” y todos los ficheros objeto.

A continuación se explica de forma genérica en qué consisten el fichero Makefile principal de las figura 4.4. Éste se encuentra en el directorio principal del proyecto.

Makefile

```
1  PROYECTODIRS = modulos  menu aplicacion snmp
2
3  -include config_m68k
4
5  PWD = $(shell pwd)
6  CFLAGS +=  -I $(PWD)/include
7
8
9  all romfs:
10         for i in $(PROYECTODIRS) ; do make -C $$i $@ || exit $?; done
11
12 clean:
13         for i in $(PROYECTODIRS) ; do make -C $$i clean ; done
14
15 main:
16         make -C principal  all;
17
18 snmpd:
19         make -C snmp all;
20
21 modulos:
22         make -C modulos all;
23
24
25
26
```

Figura 4.4: Makefile Principal

Primero de todo, comentar la parte más sencilla del fichero. Al hacer “make clean” se borraran todos los ficheros compilados y sus respectivos objetos.
Al realizar “make main” se compilará lo que hay en el directorio de proyectos “principal”. Lo mismo ocurre cuando se hace “make snmpd” o “make modulos”.
En la primera línea tenemos la variable PROYECTORDIRS que indica los directorios (con sus respectivos ficheros) que forman parte del proyecto.
La variable CFLAGS le indica al compilador los directorios donde debe buscar los **`#includes.`**
La línea “all romfs” está diciendo que se compilen todos los directorios con los archivos del proyecto.

Seguidamente se explica en qué consiste el Makefile secundario de la Figura 4.6, que se encuentra dentro del directorio “modulos”. Antes que nada lo mas importe es saber cómo están estructurados los ficheros en ese directorio. Como se ve en la Figura 4.5, el directorio “modulos” está compuesto por un Makefile y una serie de directorios que contienen los códigos de diferentes módulos. Y dentro de cada uno de esos módulos también están sus respectivos Makefile.

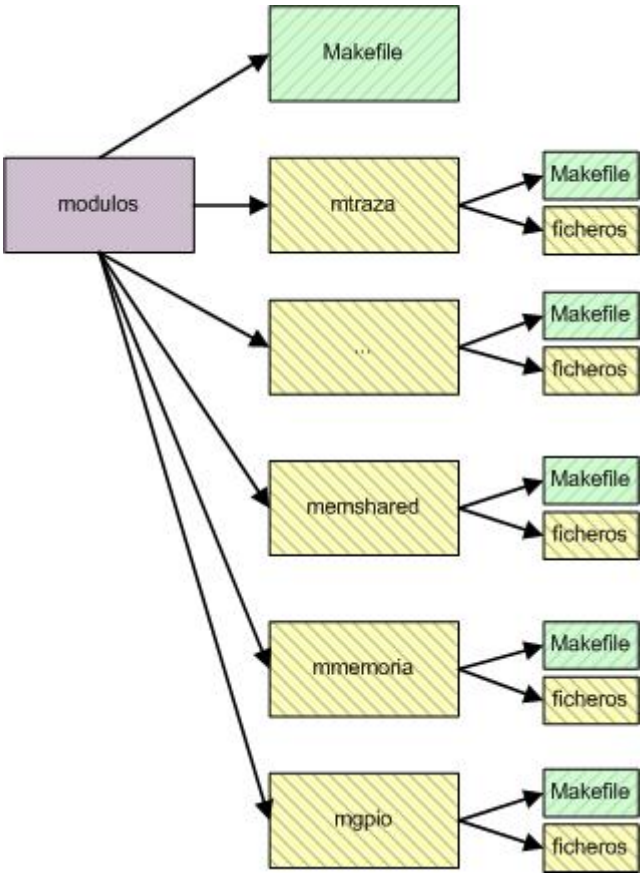

Figura 4.5: Directorio “modulos”

```
EXEC = main
MODULOS_DIR = ../modulos
EXEC_DIR = ../exec

OBJS = main/main.o \
       $(MODULOS_DIR)/mtraza/mtraza.o \
       $(MODULOS_DIR)/memshared/memshared.o \
       $(MODULOS_DIR)/uwatchdog/uwatchdog.o \
       $(MODULOS_DIR)/servtrazas/servtrazas.o \
       $(MODULOS_DIR)/mconfiguracion/mconfiguracion.o \
       $(MODULOS_DIR)/mmedida/mmedida.o \
       $(MODULOS_DIR)/mb485/enlace.o \
       $(MODULOS_DIR)/mb485/i_es.o \
       $(MODULOS_DIR)/mb485/m_es.o \
       $(MODULOS_DIR)/mb485/mensajes.o \
       $(MODULOS_DIR)/mmemoria/mmemoria.o \
       $(MODULOS_DIR)/mrs232/mserie.o \
       $(MODULOS_DIR)/mtime/time.o \
       $(MODULOS_DIR)/mrtc/mrtc.o \
       $(MODULOS_DIR)/mi2c/mi2c.o \
       $(MODULOS_DIR)/mgpio/mgpio.o \
       $(MODULOS_DIR)/mseguridad/mseguridad.o \


all : $(EXEC)

$(EXEC) : $(OBJS)
    $(CC) $(LDFLAGS) -o $(EXEC_DIR)/$@ $(OBJS) $(LDLIBS)
    cp -rf $(EXEC_DIR)/$@ /export

clean:
    rm -rf $(OBJS);
```

Figura 4.6: Makefile del directorio “modulos”

La línea “all” nos está diciendo que se compilan todos los ficheros dentro del directorio “modulos”. Del mismo modo que antes, al realizar “make clean” se borraran los ficheros compilados y sus respectivos objetos.
Las 3 primeras líneas son muy importantes, ya que sin ellas las compilaciones no llegarían a estar nunca ordenadas. Cómo se ha dicho antes, hay un directorio en el proyecto llamado “exec” que es donde se encuentran los ejecutables después de la compilación. En la primera línea, la variable EXEC contiene el nombre main, que será el nombre de uno de los ejecutables principales.
Con la variable “MODULOS_DIR” se le dice al compilador dónde se encuentran los ficheros de cada modulo.

4.3. Diseño de la aplicación

En este capítulo se realiza una descripción del programa que contiene el microcontrolador y que hace que se realicen todas las operaciones, tanto de captura de datos, visualización de estos y de su envío mediante ethernet. A lo largo del capítulo se verá con detalle el funcionamiento de los protocolos utilizados (RS-232 i SNMP), así como la metodología para realizar la lectura de los datos de las placas ADC y relés.
Una de las cosas más importantes es que la aplicación corre bajo un sistema operativo llamado uClinux. Es por eso por lo que “no” estamos delante de una aplicación común en la que tenemos un código que se ejecuta secuencialmente. En nuestro caso, la aplicación consta de 3 partes diferenciadas, y cada una de estas tiene sus propios ejecutables. El método de comunicación entre procesos se realiza mediante memoria compartida.

En la Figura 4.7 se muestran cada uno de los programas de nuestra aplicación, que como se ha dicho anteriormente se comunican entre sí mediante memoria compartida.

- Programa principal (main)
- Interfaz de usuario (menu)
- SNMP (snmpd)

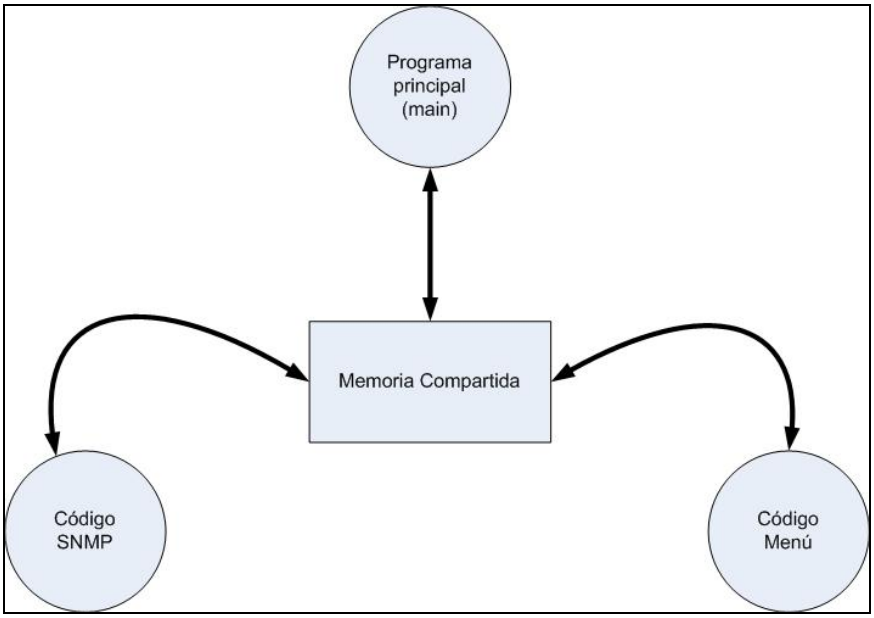

Figura 4.7: Procesos comunicados mediante memoria compartida

4.3.1. Diagrama de flujo del Programa principal (main)

El programa principal del microcontrolador está formado por dos partes diferenciadas: la inicialización y el bucle principal del programa. La inicialización es la rutina que se ejecuta una sola vez al iniciar la operación.
Esta rutina especial sirve para inicializar tanto el microcontrolador como los diferentes dispositivos que se conectan a él y necesitan de una serie de operaciones antes de su funcionamiento normal. La otra parte es el bucle principal del programa, dónde está la parte donde las instrucciones (dentro del ejecutable “main”) se ejecutan por orden.
En la figura 4.8 se puede observar el diagrama de flujo que representa el programa principal del microcontrolador.

Al inicio del programa, se ejecutan las rutinas de inicialización. Cabe decir que todas estas rutinas ya estaban programadas. A continuación se hará una pequeña descripción de la función que realizan.

- **Abrir Cola de Mensajes**: Abre una cola de mensajes donde irán destinadas las trazas.
  Esta es una función proporcionada por el módulo de Trazas. Este modulo se encarga de centralizar las trazas de los diferentes procesos y/o ejecutables. Se procesan las trazas enviadas por el módulo de traza a una cola de mensajes.

- **Activar Trazas**: Activa el nivel de traza especificado accediendo a la una estructura interna de estado.

- **Establecer Memoria compartida**: El proceso para utilizar memoria compartida es:

• Abrir semáforo: Inicializa el semáforo que va a controlar el acceso a memoria compartida.
• Abrir Memoria. Reserva el espacio en memoria.
• Asociar memoria. “Asocia” el espacio reservado a una estructura.

A la hora de usar memoria compartida el proceso es cerrar el semáforo, acceder a memoria y liberar el semáforo.
Las funciones implementadas son:

• int P(int semaforo)
Función para bloquear el semáforo.
• int V(int semaforo)
Función para desbloquear el semáforo
• int AbrirSemaforo(int key)
Crea un identificador de semáforo para el valor de key dado.

- **Activar Watchdog**: El microcontrolador Coldfire5282 posee dos watchdogs configurables.
  Aprovechando esta característica se ha utilizado un watchdog para el sistema operativo y otro para controlar fallos de aplicación. En el caso de que la aplicación deje de funcionar la placa se reseteará.

- **Leer Configuración**: El módulo de configuración se utiliza para leer y escribir la configuración de la aplicación. Se leen los parámetros de los ficheros de configuración y los pasa a memoria compartida. Los ficheros de configuración son los siguientes:
  - /user/etc/pdcf.conf: Fichero de configuración de la aplicación principal.
  - /user/etc/snmpd.conf: Fichero de configuración del proceso snmpd.
  - /user/etc/start: fichero de configuración de ethernet.

- **Iniciar I2C**: El módulo RTC da soporte al reloj en tiempo real. El acceso a este hardware se realiza a través del bus I2C.

- **UpdateTime**: Módulo que obtiene o modifica la hora del reloj de tiempo real.

- **Configurar Puerto Serie**: Configuración del puerto Serie para la comunicación con las placas ADC y reles.
  ConfigurarPuertoSerie(2, 115200, PARIDAD_NULO, 0, 8, FLUJO_NULO);

- **Iniciar sensor Temperatura / Humedad**: El sensor está conectado a la placa como si fuese un dispositivo hardware más. Por esta razón se ha creado un driver para el sensor de Temperatura que se debe cargar en el kernel del sistema operativo. El driver está asociado al dispositivo /dev/sht11.

Seguidamente se entra en el bucle principal de la aplicación que se explicará en el siguiente apartado.

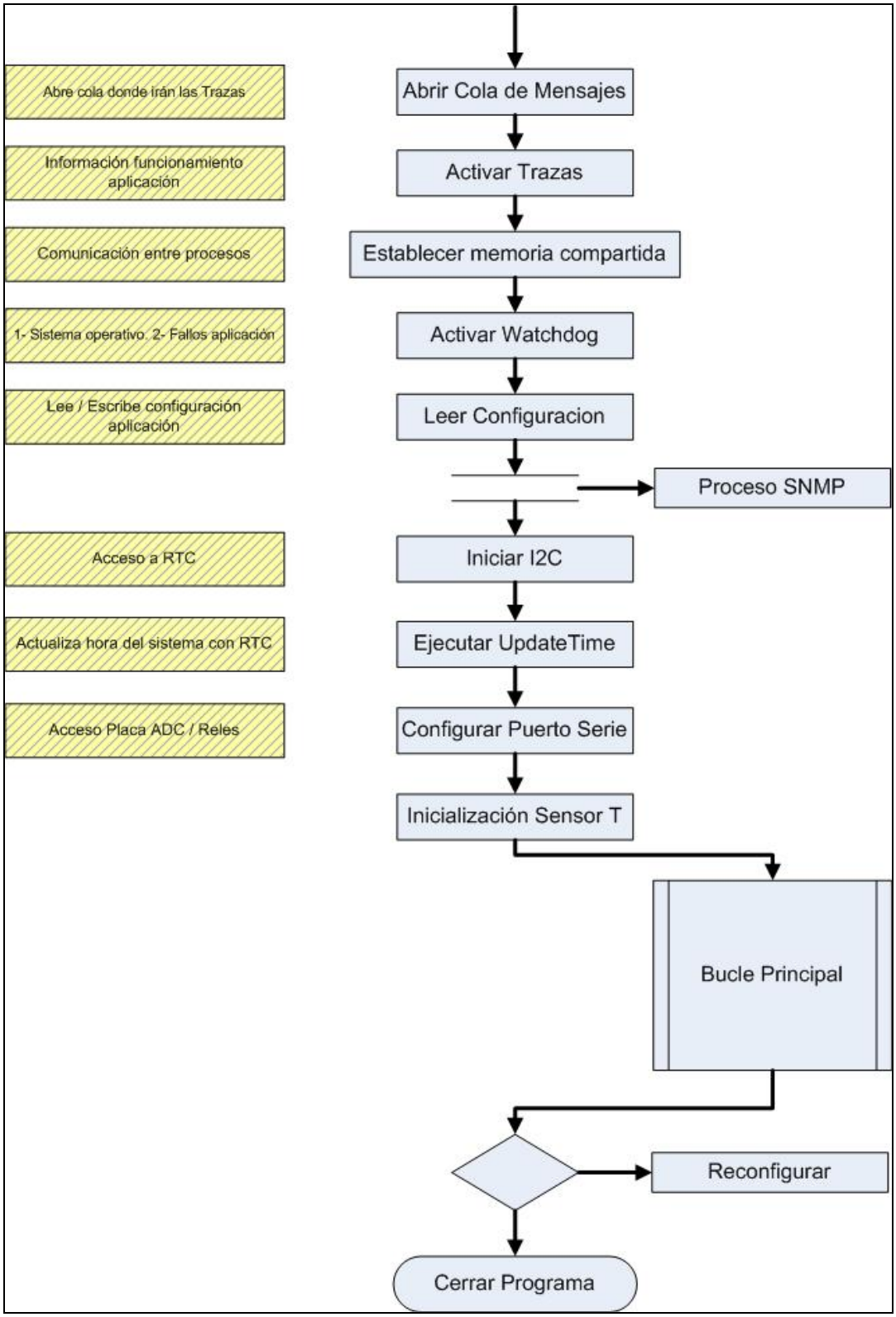

Figura 4.8: Diagrama de flujo del programa principal

4.3.2. Diagrama de flujo del Bucle Infinito del programa principal (main)

Una vez inicializados tanto el microcontrolador como los diferentes dispositivos y funciones, se llega al bucle principal de la aplicación. A continuación se explica el diagrama de flujo de la figura 4.9.

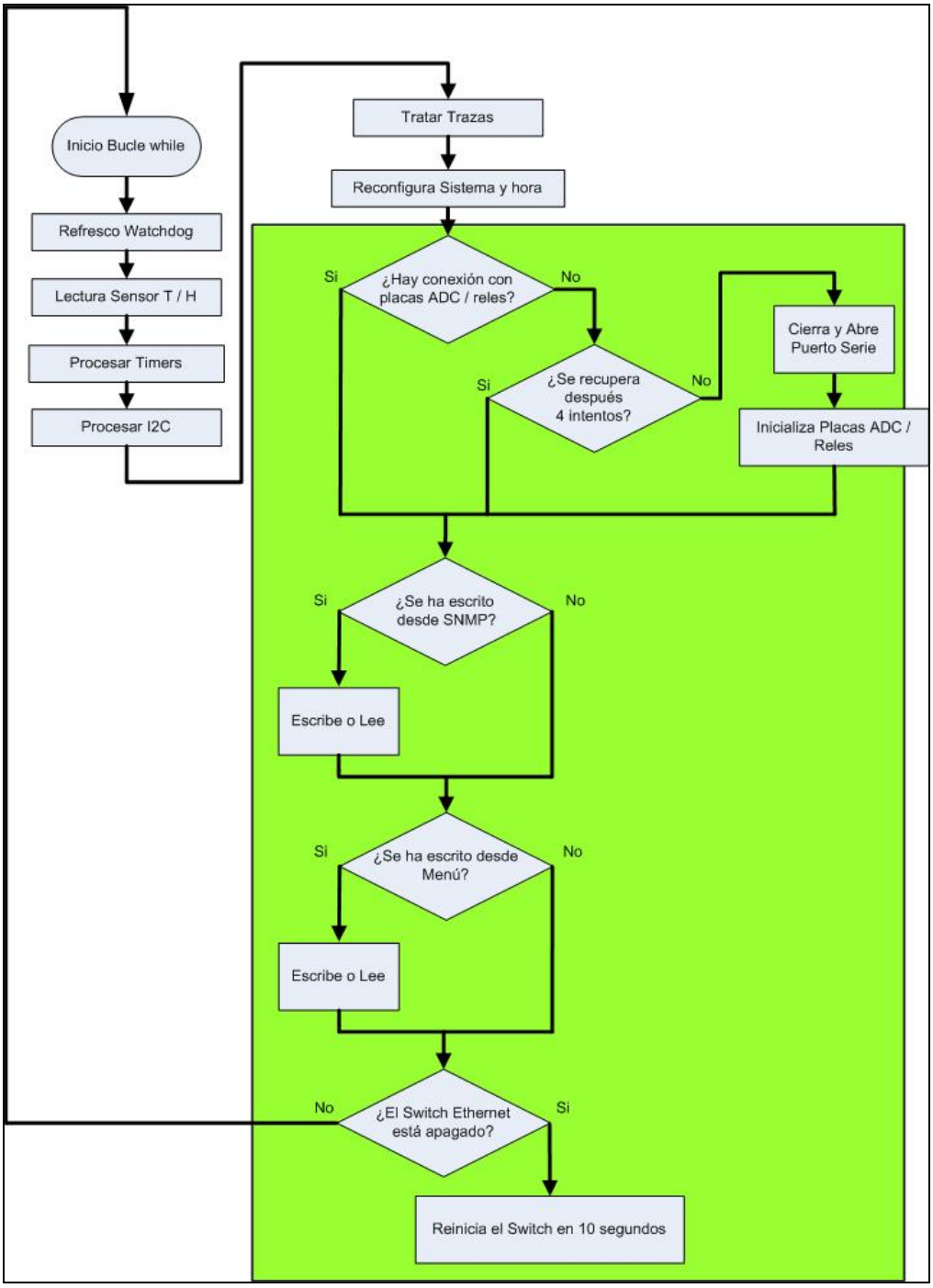

Figura 4.9: Diagrama de flujo del bucle infinito del programa principal

Antes que nada se van a describir los puntos más importantes anteriores al cuadro verde, que cómo se ha dicho en el punto anterior, su código ya estaba programado.
Al inicio del bucle principal, se ejecutan las siguientes rutinas:

- **Refresco Watchdog**: La utilización de un Watchdog es para el sistema operativo y el otro para controlar fallos de aplicación. En este caso se está refrescando el segundo, y refresca el Watchdog de usuario.
- **Lectura Sensor T / H:** Se ha introducido el código para obtener los valores de temperatura y humedad de los sensores detectados en el proceso de inicialización. Para evitar sobrecargar en la aplicación se ha establecido un proceso de lectura periódica de los sensores. En este caso, se realiza una lectura cada 25 pasos por el bucle infinito.
- **Procesar Timers:** Función que actualiza los temporizadores. Hay que llamarla sucesivamente en tiempo de ejecución. Devuelve el número de temporizadores lanzados.
- **Tratar Trazas:** Se tratan las trazas recibidas y se envían a la cola.
- **Comprobamos si reconfiguramos:** Comprobamos si debemos reconfigurar el sistema o actualizar la hora.

Una vez comentados los anteriores puntos pasamos a explicar el recuadro verde de la figura 4.9.

- **Comprobamos si hay conexión ADC / relés:**

La variable “respuesta_ADC” nos retorna si hay o no conexión con las placas. Se pueden dar hasta 3 situaciones:

1- Si el valor de “respuesta_ADC” es igual al valor de la variable de reconfiguración “ADC_RECONFIG”, significa que es necesario reconfigurar el sistema.
2- Si la variable “respuesta_ADC” es igual al valor de una variable de no respuesta llamada “ADC_NO_ANS”, significa que no hay conexión con las placas.
3- En el caso de que “respuesta_ADC” sea igual a “ADC_RECONFIG” o a “ADC_OK” podremos decir que se ha recuperado la conexión con las placas.

```
        respuesta_ADC = ActualizarLecturaADC(m,semid);
        if(respuesta_ADC == ADC_RECONFIG)
        {
                Reconfigurar();
        }
        if(respuesta_ADC == ADC_NO_ANS)
        {
                if(fallos_Preles >= NUM_FALLOS_PRELES)
                {
                        P(semid);
                         if ((*m).com_ADC == 1)
                                Traza(t_INFORMATION, "Perdida de conexion ADC");
                         (*m).com_ADC = 0;
                        V(semid);
                         CerrarPuerto(2);
                         ConfigurarPuertoSerie(2, 115200, PARIDAD_NULO, 0, 8,
FLUJO_NULO);
                        Configuracion_placa_reles();
                        ActualizarSalidasMem_1(m,semid);
                        ActualizarSalidasMem_2(m,semid);
                        ActualizarSalidasMem_3(m,semid);
                        ActualizarSalidasMem_4(m,semid);
                }
                else
                        fallos_Preles++;
                }
        else
```

```
if ((respuesta_ADC == ADC_OK) || (respuesta_ADC == ADC_RECONFIG))
{
        fallos_Preles = 0;
        P(semid);
        if((*m).com_ADC == 0)
                Traza(t_INFORMATION, "Recuperacion de conexion ADC");
        (*m).com_ADC = 1;
        V(semid);
        if (lecturas_adc >= 5)
        {
                if (bloqueo_ini_algoritmos == 1)
                {
                        Traza(t_INFORMATION, "Desbloqueo algoritmos");
                        bloqueo_ini_algoritmos = 0;
                }
        }
        else
        {
                lecturas_adc++;
                Traza(t_INFORMATION,"Incrementamos lecturas_adc");
        }
  }
```

- **Comprobamos si se ha escrito desde SNMP:**

En el siguiente código se actúa sobre los relés en el caso de que se haya realizado algún cambio desde SNMP. Cada vez que se cambia algún objeto desde SNMP, la variable “flag_snmp” se pone a “1”. Cuando esto ocurre, entramos en el correspondiente bucle poniendo “flag_snmp” a “0” y pasamos a realizar la escritura de relés (ActualizarSalidasMem_x).
Estas funciones lo que hacen es mirar si ha habido cambios en el estado de los relés, y solamente en ese caso, se escribe en ellos.
La variable “com2_inuse_LEEestado” nos dice si se está intentando escribir en algún banco por 2 lados de código distinto. Es decir, actúa como un semáforo.

```
        P(semid);
        flag_snmp = m->flag_snmp;
        V(semid);
        if (flag_snmp)
        {
                if ((com2_inuse_LEEestado == 0))
                {
                        P(semid);
                        m->flag_snmp = 0;
                        V(semid);
                        ActualizarSalidasMem_1(m,semid);
                        ActualizarSalidasMem_2(m,semid);
                        ActualizarSalidasMem_3(m,semid);
                        ActualizarSalidasMem_4(m,semid);
                        Reconfigurar();
                  }
        }
```

- **Comprobamos si se ha escrito desde menú:**

El siguiente código es casi parecido al anterior, la única diferencia es que en vez de ejecutarse cuando hay cambios desde SNMP, se ejecuta cuando hay cambios en el menú.

```
if (flag)
{
        if ((com2_inuse_LEEestado == 0))
        {
                P(semid);
                m->flag = 0;
                V(semid);
                ActualizarSalidasMem_1(m,semid);
                ActualizarSalidasMem_2(m,semid);
                ActualizarSalidasMem_3(m,semid);
                ActualizarSalidasMem_4(m,semid);
                auxiliar = Reconfigurar();
        }
}
```

- **Miramos si los 2 switches ethernet están activos:**

Lo que se pretender con el siguiente código es mirar el estado de los switches de comunicación con la Estación Terrestre. Si por algún motivo no se tiene conexión con los switches, se deben reiniciar. Uno de los switch está conectado al rele25 (banco 2, relé 5) y el otro al rele26 (banco 2, relé 6). Cuando estadoReles25 y SW_1 = 0, significa que no hay comunicación con el switch. En este caso se activa un flag que cuenta hasta 10 segundos para comprobar que eso sea realmente es cierto. Una vez pasados los 10 segundos se vuelve a activar el switch.

```
if (((*m).estadoReles25 == 1) && (SW_1 == 0))
{
        SW_1 = 1;
        ActivarFlag(diez_seg,FLAG_RELES_SW1);
 }
if ((ComprobarFlag(FLAG_RELES_SW1) == 1) ||
            (ComprobarFlag(FLAG_RELES_SW1) == -1) && (SW_1 == 1))
{
        DesactivarFlag(FLAG_RELES_SW1);
        SW_1 = 0;
        P(semid);
        salidas_aux = ((*m).R_BANCO_2) & (0x000000df);
        V(semid);
           ActivarSalidas_3(salidas_aux, m, semid);
 }

if (((*m).estadoReles26 == 1) && (SW_2 == 0))
{
        SW_2 = 1;
        ActivarFlag(diez_seg,FLAG_RELES_SW2);
 }
if ((ComprobarFlag(FLAG_RELES_SW2) == 1) ||
            (ComprobarFlag(FLAG_RELES_SW2) == -1) && (SW_2 == 1))
{
```

```
                DesactivarFlag(FLAG_RELES_SW2);
                SW_2 = 0;
                P(semid);
                salidas_aux = ((*m).R_BANCO_2) & (0x000000bf);
                V(semid);
                ActivarSalidas_3(salidas_aux, m, semid);
        }
```

4.4. Escritura en los relés

A continuación se describen los pasos necesarios para la activación de los relés. La función "EscribirReles2" se utiliza para minimizar trozos de código. Cuando debemos realizar una escritura será necesario introducir el siguiente comando: 254, valor del relé. De esta forma solo será necesario introducir EscribirReles2 (num_e).

```
void EscribirReles2(char num_e)
{
   char cadena_auxi[4];
   cadena_auxi[0] = 254;
   cadena_auxi[1] = num_e;
   cadena_auxi[3] = '\0';
   EscribirCaracteres(2, cadena_auxi, 2);
}
```

La función ActivarSalidas_1 se utiliza para escribir en los relés del banco 1. Existen 3 funciones más para el resto de los bancos.
Al principio, con la variable "com2_inuse" se comprueba que no se esté escribiendo en el mismo banco desde otro trozo de código.
Con las variables salidas_x se pretende comprobar cuales han sido los relés que han cambiado de estado. De esta forma solamente se escribe en el relé que sea necesario y no en todos. La variable "estadoRelesxx" contiene el estado real de los relés. Por tanto, solamente en el caso de que las 2 variables no coincidan se realizará la escritura.

```
void ActivarSalidas_1(unsigned char salidas_1, medidas *m, int semid)
{
   if ((com2_inuse_LEEestado == 0))
   {
        if (((salidas_1 & 0x01) != (*m).estadoReles00) ||
        (((salidas_1 & 0x02)>>1) != (*m).estadoReles01) ||
        (((salidas_1 & 0x04)>>2) != (*m).estadoReles02) ||
         (((salidas_1 & 0x08)>>3) != (*m).estadoReles03)||
        (((salidas_1 & 0x10)>>4) != (*m).estadoReles04) ||
        (((salidas_1 & 0x20)>>5) != (*m).estadoReles05) ||
        (((salidas_1 & 0x40)>>6) != (*m).estadoReles06) ||
        (((salidas_1 & 0x80)>>7) != (*m).estadoReles07))
        {
                if ((salidas_1 & 0x01) != (*m).estadoReles00)
                {
                        if ((salidas_1 & 0x01) != 0)
                        {
                                EscribirReles2(8);
                        }else{
```

```
                        EscribirReles2(0);
                }
                        usleep(20000);
        }
        if (((salidas_1 & 0x02)>>1) != (*m).estadoReles01)
        {
                if ((salidas_1 & 0x02) != 0)
                {
                        EscribirReles2(9);
                }else{
                        EscribirReles2(1);
                }
                usleep(20000);
        }
        if (((salidas_1 & 0x04)>>2) != (*m).estadoReles02)
        {
                if ((salidas_1 & 0x04) != 0)
                {
                        EscribirReles2(10);
                }else{
                        EscribirReles2(2);
                }
                usleep(20000);
        }
        if (((salidas_1 & 0x08)>>3) != (*m).estadoReles03)
        {
                if ((salidas_1 & 0x08) != 0)
                {
                        EscribirReles2(11);
                }else{
                        EscribirReles2(3);
                }
                usleep(20000);
        }
        if (((salidas_1 & 0x10)>>4) != (*m).estadoReles04)
        {
                if ((salidas_1 & 0x10) != 0)
                {
                        EscribirReles2(12);
                }else{
                        EscribirReles2(4);
                }
                usleep(20000);
        }
        if (((salidas_1 & 0x20)>>5) != (*m).estadoReles05)
        {
                if ((salidas_1 & 0x20) != 0)
                {
                        EscribirReles2(13);
                }else{
                        EscribirReles2(5);
                }
                usleep(20000);
        }
        if (((salidas_1 & 0x40)>>6) != (*m).estadoReles06)
```

```
                {
                        if ((salidas_1 & 0x40) != 0)
                        {
                                EscribirReles2(14);
                        }else{
                                EscribirReles2(6);
                        }
                        usleep(20000);
                }
                 if (((salidas_1 & 0x80)>>7) != (*m).estadoReles07)
                {
                        if ((salidas_1 & 0x80) != 0)
                        {
                                EscribirReles2(15);
                        }else{
                                EscribirReles2(7);
                        }
                        usleep(20000);
                }
                EscribirReles2(24);
        }
    }
}
```

4.5. Lectura del estado de los ADC y relés

A continuación se describe cómo se realiza la lectura de los conversores ADC y los estados de los relés.

- Case 0:
    a) Nos mantenemos a la espera de que no queden caracteres en el buffer.
    b) Cuando esto ocurre lanzamos una orden de lectura para el banco 0 de ADC.
    c) Activamos el temporizador de seguridad.
    d) Cambiamos de estado.
- Case 1:
    a) Esperamos a haber recibido los 16 caracteres de respuesta del ADC del banco 0 o bien a que el temporizador de seguridad salte.
    b) Si el temporizador salta, significa que ha habido algún problema con la lectura del ADC y procedemos a enviar el comando de lectura del estado de los relés. Seguidamente saltamos al estado 2.
    c) En caso de que se reciba la respuesta, se lee, se almacena en memoria compartida y se pasa al estado 2.
- Case 2:
    a) Esperamos a haber recibido el estado de los relés o bien a que el temporizador de seguridad salte.
    b) Si el temporizador salta, significa que ha habido algún problema con la lectura de los relés y procedemos a realizar la lectura del banco 1 de los ADC. Seguidamente saltamos al estado 3.
    c) En caso de que se reciba la respuesta, se lee el estado del banco 1 de ADC, se almacena en memoria compartida y se pasa al estado 3.
- Case 3:
    a) Esperamos a haber recibido los 16 caracteres de respuesta del ADC del banco 1 o bien a que el temporizador de seguridad salte.

b) Si el temporizador salta, significa que ha habido algún problema con la lectura de los ADC del banco 1 y procedemos a realizar la lectura de los banco 2 de los ADC. Seguidamente saltamos al estado 0 para empezar de nuevo.
c) En caso de que se reciba respuesta, se lee el estado, se almacena en memoria compartida y se pasa al estado 0 para empezar de nuevo.

```
case 0:
        Nos aseguramos de que no se está leyendo ni escribiendo en el puerto
        Esperamos hasta que no haya ningun caracter en el buffer
        {
                Comando Lectura ADC banco 0
                Activacion temporizador
                estado = 1;
        }
case 1:
        Comprobamos temporizador
        {
                Comando Lectura estado relés
                Activacion temporizador
                estado = 2;
        }else
        {
                Esperamos hasta que no haya ningun caracter en el buffer
                {
                        Almacenamiento en memoria compartida ADC banco 0
                        Activacion temporizador
                        Comando Lectura estado relés
                        estado = 2;
                }
        }
break;
case 2:
        Comprobamos temporizador
        {
                Comando Lectura ADC banco 1
                Activacion temporizador
                estado =3;
        }else
        {
                Esperamos hasta que no haya ningun caracter en el buffer
                {
                        Almacenamiento en memoria compartida estado relés
                        Activacion temporizador
                        Comando Lectura ADC banco 1
                        estado = 3;
                }
        }
break;
case 3:
        Comprobamos temporizador
        {
                Si no hay respuesta de las placas ADC y relés
                estado = 0;
        }else
        {
```

```
            Esperamos hasta que no haya ningun caracter en el buffer
            {
                    Almacenamiento en memoria compartida ADC banco 2
               estado = 0;
            }
}
```

4.6. Diagrama de flujo Proceso SNMP

En la figura 4.10 se muestra el diagrama de flujo del proceso SNMP. Primero hay una inicialización de los objetos, a continuación está el código al que se entra cuando hay una petición de lectura o escritura.
Después está la inicialización del propio modulo SNMP, el cual contiene el código para el envío de los valores de las variables y también para el envío de traps. En el caso del envío de traps, es necesario configurar los tiempos entre envíos de alarmas.

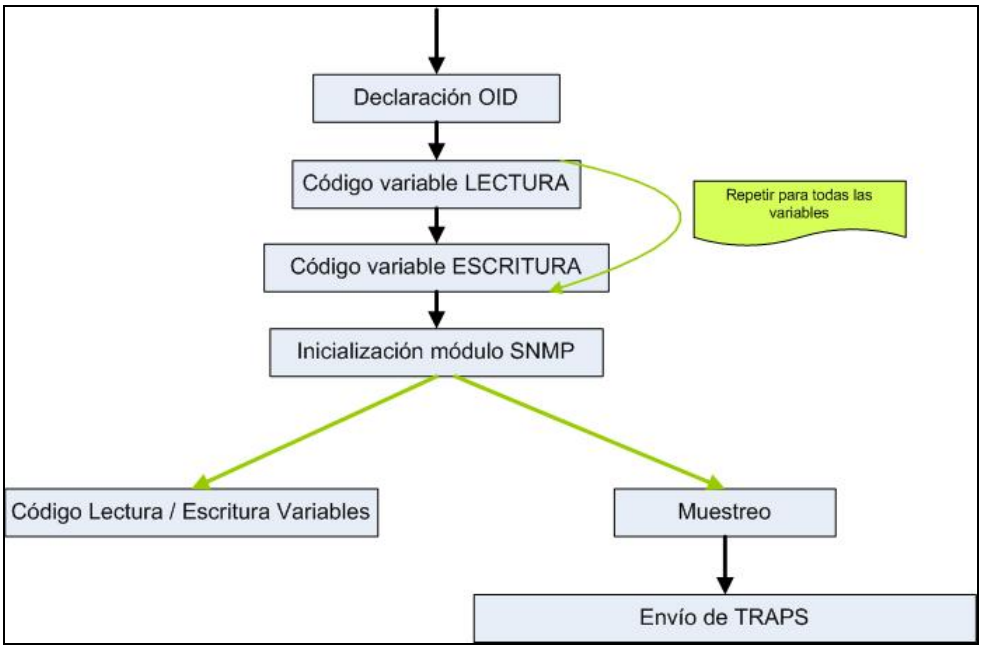

Figura 4.10: Diagrama de flujo del proceso SNMP

A continuación se muestra la inicialización de los objetos:

```
char buftraza[256];
unsigned int com2_inuse_LEEestado;

static oid dycec_oid[] = { 1,3,6,1,4,4,12592 };
static oid adc00_oid[] = { 1,3,6,1,4,1,12592,1,1, 0 };
.....
static oid adc15_oid[] = { 1,3,6,1,4,1,12592,1,16, 0 };
static oid alarma00_oid[] = { 1,3,6,1,4,1,12592,2,1, 0 };
.....
static oid alarma17_oid[] = { 1,3,6,1,4,1,12592,2,18, 0 };
static oid adc2_00_oid[] = { 1,3,6,1,4,1,12592,3,1, 0 };
```

```
.....
static oid adc2_15_oid[] = { 1,3,6,1,4,1,12592,3,16, 0 };
static oid R_BANCO_0_ALL_oid[] = { 1,3,6,1,4,1,12592,4,1, 0 };
static oid R_BANCO_1_ALL_oid[] = { 1,3,6,1,4,1,12592,4,2, 0 };
static oid R_BANCO_2_ALL_oid[] = { 1,3,6,1,4,1,12592,4,3, 0 };
static oid R_BANCO_3_ALL_oid[] = { 1,3,6,1,4,1,12592,4,4, 0 };
```

Después de la inicialización debe estar el código al que se entra cuando se hace una petición de lectura o escritura desde SNMP.

```
int read_adc()
{
        int buff_sin0=0;
        P(semid);
        buff_sin0=(*m).adcc0;
        V(semid);
        return buff_sin0;
}
int write_adc(int adc00)
{
        return -1;
}
int read_adc1()
{
        int buff_sin1=0;
        P(semid);
        buff_sin1=(*m).adcc1;
        V(semid);
         return buff_sin1;
}
                    ……..
                    ……..
int write_R_BANCO_0_ALL(unsigned int R_BANCO_0_ALL)
        {
                 if (m->reles_manual_banco_0 == 0)
                {
                        return(-1);
                }
                 else
                {
                        P(semid);
                        m->R_BANCO_0_aux = R_BANCO_0_ALL;
                        m->flag_snmp = 1;
                        V(semid);
                        return SNMP_ERR_NOERROR;
                }
        }
```

A continuación se presenta la inicialización del módulo SNMP.
La función más interesante es “netsnmp_register_instance”, que es creada para registrar los objetos. Ésta recibe como parámetros el nombre del handler (“adcxx”), el handler utilizado para manejar las solicitudes (do_adcxx), el objeto (adcxx_oid), su longitud y el nivel de acceso que se tiene, de escritura y lectura (HANDLER_CAN_RWRITE).

Decir, que el handler es el que se encarga de administrar como se van a manejar los datos de las variables.

```
void
init_dycec(void)
{
        ……..……..……..……..……..……..
        ……..……..……..……..……..……..
        adc00 = read_adc();
        adc15 = read_adc15();
        ……..……..……..……..……..……..
        ……..……..……..……..……..……..
        alarma00 = read_adc();
        alarma16 = (*m).alarma16;
        P(semid);
        alarma17 = (*m).com_ADC;
        V(semid);
        adc2_00 = read_adc2_();
        ……..……..……..……..……..……..
        ……..……..……..……..……..……..
        adc2_15 = read_adc2_15();
        R_BANCO_0_ALL = read_R_BANCO_0_ALL();
        R_BANCO_1_ALL = read_R_BANCO_1_ALL();
        R_BANCO_2_ALL = read_R_BANCO_2_ALL();
        R_BANCO_3_ALL = read_R_BANCO_3_ALL();

        reg_res = netsnmp_register_instance(netsnmp_create_handler_registration
                    ("adc00",
                 do_adc,
                 adc00_oid,
                 OID_LENGTH(adc00_oid),
                HANDLER_CAN_RWRITE));
                 if (reg_res != 0) {
                        Traza(t_INFORMATION, "Fallo al registrar adc00");
                }
                usleep(150000);
        reg_res = netsnmp_register_instance(netsnmp_create_handler_registration
                    ("adc01",
                  do_adc1,
                 adc01_oid,
                 OID_LENGTH(adc01_oid),
                 HANDLER_CAN_RWRITE));
                 if (reg_res != 0) {
                        Traza(t_INFORMATION, "Fallo al registrar adc01");
                 }
                usleep(150000);
        ……..……..……..……..……..……..
        ……..……..……..……..……..……..
}
```

En la función “muestreo” nos encontramos con el código necesario para el envío de traps SNMP. La función “snmp_varlist_add_variable” recibe como parámetros la lista de variables, el OID del objeto, su longitud, el tipo de dato del objeto, el valor del objeto y su longitud. Seguidamente se envía el trap (send_v2trap).

```
void
muestreo(unsigned int clientreg, void *clientarg)
{
   size_t        oid_adc_len = OID_LENGTH(adc00_oid);
……
   size_t        oid_adc15_len = OID_LENGTH(adc15_oid);
   size_t        oid_alarma00_len = OID_LENGTH(alarma00_oid);
……
   size_t        oid_alarma17_len = OID_LENGTH(alarma17_oid);
   size_t        oid_adc2__len = OID_LENGTH(adc2_00_oid);
……
   size_t        oid_adc2_15_len = OID_LENGTH(adc2_15_oid);
   size_t        oid_R_BANCO_0_ALL_len = OID_LENGTH(R_BANCO_0_ALL_oid);
   size_t        oid_R_BANCO_1_ALL_len = OID_LENGTH(R_BANCO_1_ALL_oid);
   size_t        oid_R_BANCO_2_ALL_len = OID_LENGTH(R_BANCO_2_ALL_oid);
   size_t        oid_R_BANCO_3_ALL_len = OID_LENGTH(R_BANCO_3_ALL_oid);

   adc00=read_adc();
   adc15=read_adc15();
   alarma00 = read_adc();
   alarma16 = (*m).alarma16;
   P(semid);
   alarma17 = (*m).com_ADC;
   V(semid);
   adc2_00 = read_adc2_();
   adc2_15 = read_adc2_15();
   R_BANCO_0_ALL = read_R_BANCO_0_ALL();
   R_BANCO_1_ALL = read_R_BANCO_1_ALL();
   R_BANCO_2_ALL = read_R_BANCO_2_ALL();
   R_BANCO_3_ALL = read_R_BANCO_3_ALL();

//ALARMA GRABE--> No se detectan las placas de relés y ADC
         if (alarma17 == 0)
         {
                  snmp_varlist_add_variable(&alarma17_vars,
                  alarma17_oid, oid_alarma17_len,
                  ASN_INTEGER,
                  (u_char *) &alarma17,
                  sizeof(alarma17));
                  send_v2trap(alarma17_vars);
                   snmp_free_varbind(alarma17_vars);
          }

//ALARMALEVE--> Fuente entrada a punto de salir del margen (136-334)
          if ((adc00 <= ((*m).umbralADC[0] / (*m).factt0)) ||
               (adc00 >= ((*m).umbralADC[1] / (*m).factt0)))
         {
                  snmp_varlist_add_variable(&alarma06_vars,
                   alarma06_oid, oid_alarma06_len,
                   ASN_INTEGER,
                  (u_char *) &alarma06,
                  sizeof(alarma06));
                  send_v2trap(alarma06_vars);
                   snmp_free_varbind(alarma06_vars);
         }
```

```
//Tensió d'entrada a punt de sortir de marge (124 – 370 Vdc), ALARMA LEVE
        if (((*m).activar_trap_in_0 == 1))
        {
                snmp_varlist_add_variable(&adc_vars,
                adc00_oid, oid_adc_len,
                ASN_INTEGER,
                (u_char *) &adc00,
                sizeof(adc00));
                send_v2trap(adc_vars);
                snmp_free_varbind(adc_vars);
        }

//Tensió d'entrada fora de marge (124 – 370 Vdc), ALARMA GRABE
        if (((*m).activar_trap_in_1 == 1))
        {
                snmp_varlist_add_variable(&alarma00_vars,
                alarma00_oid, oid_alarma00_len,
                ASN_INTEGER,
                (u_char *) &alarma00,
                sizeof(alarma00));
                send_v2trap(alarma00_vars);
                snmp_free_varbind(alarma00_vars);
        }
            ………………………………………..
            ……………………………………….
```

También aparece el código que identifica el tipo de solicitud SNMP.
En el caso de que el tipo de petición sea MODE_GET, se llama a la función "snmp_set_var_typed_value", que se encarga de obtener el valor del objeto solicitado. Recibe como parámetros la lista de variables solicitadas (requests->requestvb), el tipo de dato del objeto (ASN_INTEGER), un apuntador al dato del objeto (&adc00) y la longitud de los datos en bytes sizeof(adc00)).
En el caso de que la solicitud sea del tipo MODE_SET lo que se hace es coger el valor introducido y llamar a la función *"write_adcxx"* que con el valor de la variable, que realizará la escritura.

```
  int do_adc(netsnmp_mib_handler *handler,
             netsnmp_handler_registration *reginfo,
             netsnmp_agent_request_info *reqinfo,
             netsnmp_request_info *requests)
   {
     switch(reqinfo->mode) {

     case MODE_GET:
             adc00=read_adc();
             snmp_set_var_typed_value(requests->requestvb,
                                 ASN_INTEGER, (u_char *) &adc00, sizeof(adc00) );
     break;
     case MODE_SET_RESERVE1:
             if (requests->requestvb->type != ASN_INTEGER)
             netsnmp_set_request_error(reqinfo, requests,SNMP_ERR_WRONGTYPE);
     break;
```

```
        case MODE_SET_RESERVE2:
                if (write_adc(*(requests->requestvb->val.integer)) == -1)
                {
                        netsnmp_set_request_error(reqinfo,
requests,SNMP_ERR_RESOURCEUNAVAILABLE);
                return SNMP_ERR_NOERROR;
                 }
                 break;
                case MODE_SET_FREE:
                break;
                case MODE_SET_ACTION:
                        adc00=*(requests->requestvb->val.integer);
                break;
                case MODE_SET_COMMIT:
                break;
                case MODE_SET_UNDO:
                break;
                default:
                        if (handler->next && handler->next->access_method)
                        return netsnmp_call_next_handler(handler, reginfo, reqinfo,requests);
                return
               SNMP_ERR_GENERR;
                  }
        return SNMP_ERR_NOERROR;
        }
```

4.8. Estructura del Menú

El menú es un programa que se ejecuta cuando realizamos un telnet sobre la placa.
El programa presenta un menú principal con más o menos opciones en función del usuario que acceda. Existen dos tipos de usuarios, normales y administradores. Los usuarios normales solo tendrán acceso a los cuatro primeros submenús:

0.- Mostrar Medidas y Estado.
1.- Configuración de Medidas y Puerto.
2.- Configuración de Alarmas y Telemandos.
3.- Conexión a Puerto.

Los usuarios administradores tendrán acceso a todos los submenús y podrán así configurar totalmente la placa. El código está estructurado de forma que cada submenú está contenido en un fichero, esta estructura se muestra en la Figura 4.11.

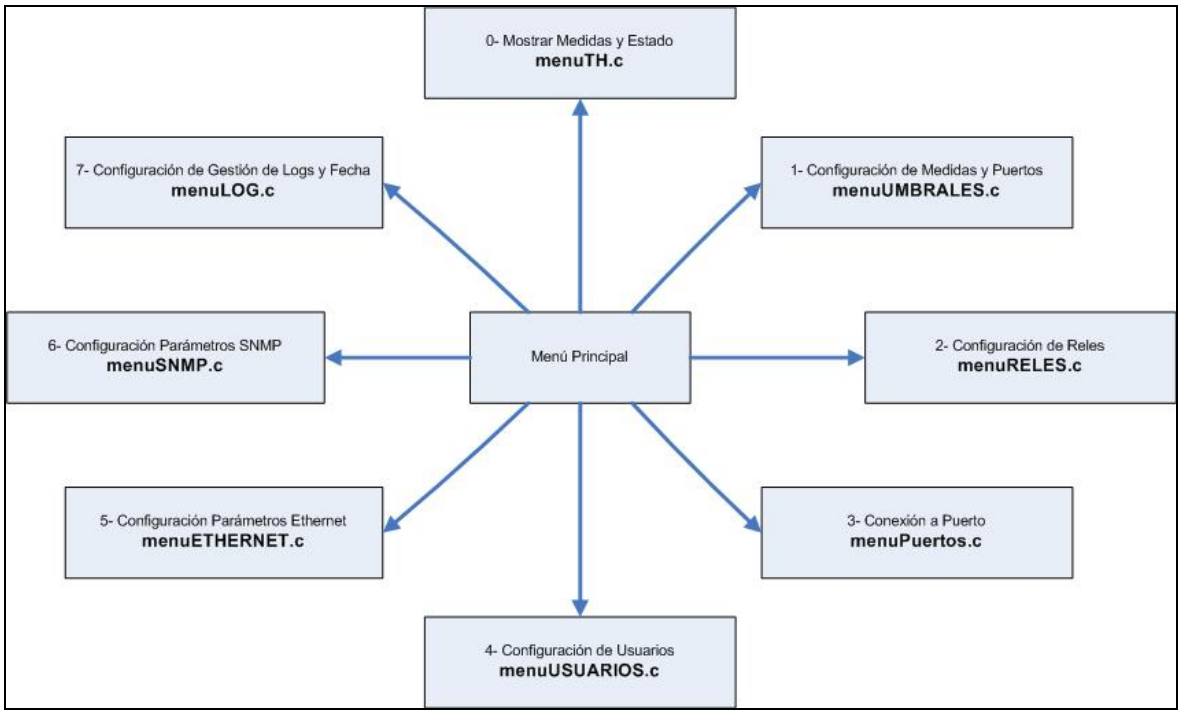

Figura 4.11: Estructura del menú

El menú captura las señales de finalización de programa y cuelgue de conexión para finalizar de forma controlada.
Todo el menú está estructurado de forma secuencial con esperas. Tiene implementado un mecanismo paralelo de temporización controlado por señales y temporizadores del sistema para que el programa finalice de forma automática a los 15 minutos de la última selección.

Para acceder al interfaz de usuario necesitamos realizar un telnet a la placa. Después de autentificarnos obtenemos un menú en pantalla sobre el que podemos navegar. El menú distingue entre administradores y usuarios.

Existe otro método de acceso al menú en conexión local a través del puerto 2. Conectamos un cable serie y configuramos un “Hyperterminal” o una aplicación similar en modo 19200 8N1 sin control de flujo.

Capitulo 5. Experimentos realizados

En las instalaciones del grupo de investigación SARTI-UPC, se dispone de una cámara hiperbárica Iberco (20 bar) en la que se simula el comportamiento de los elementos que forman la Estación Submarina. Gracias a ésta se han realizado las primeras pruebas para comprobar la resistencia de los materiales utilizados, estanqueidad de los equipos e infraestructura de la estación submarina.

En la figura 4 podemos ver una de las pruebas realizadas en la cámara hiperbárica:

Figura 5.1: Cámara hiperbática

Se han realizado las siguientes pruebas en los módulos que componen la Estación Submarina:

- Verificación del correcto funcionamiento de los módulos de conversores Analógico-Digital (ADC) y relés.
- Primeras pruebas de comunicación entre la placa controladora y los módulos ADC y relés.
- Implementación del envío de alarmas mediante SNMP sobre los datos obtenidos por los módulos ADC y relés.
- Implementación de peticiones de lectura y escritura mediante el protocolo SNMP sobre los mismos módulos (figura 5).

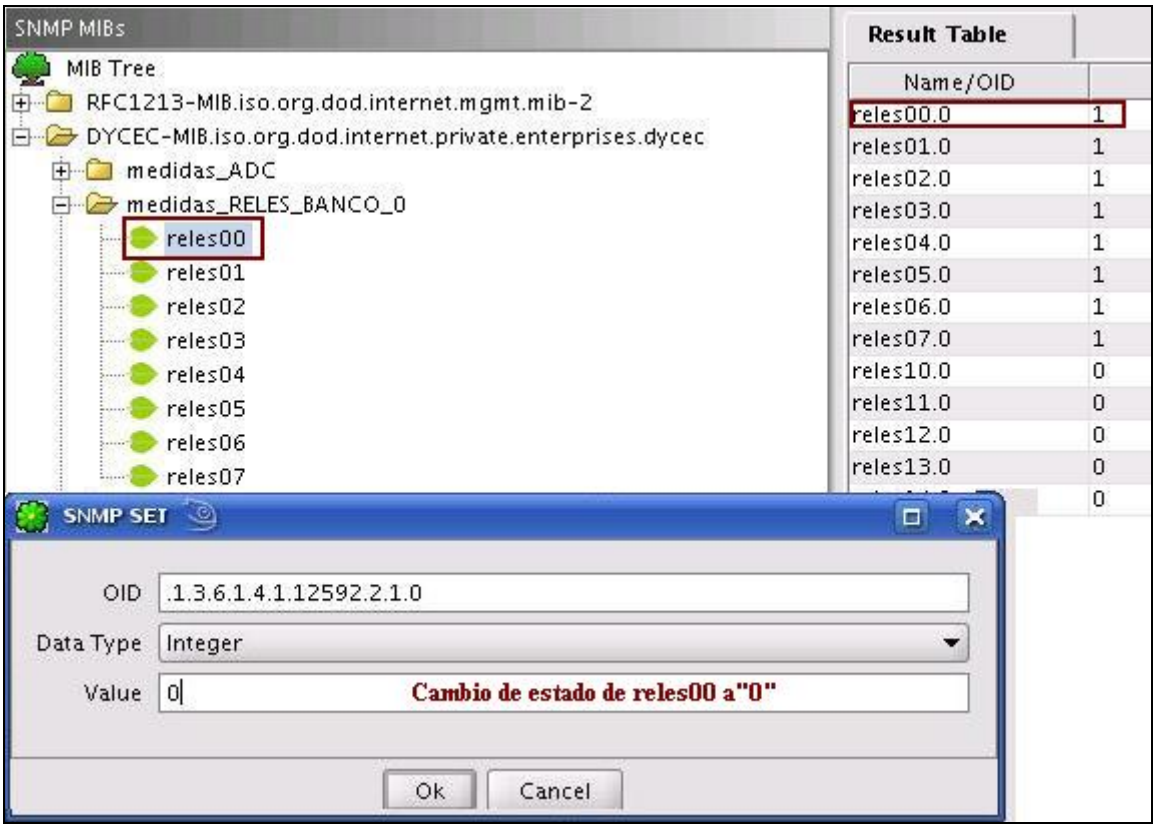

Figura 5.2: Cambio de variables mediante SNMP

- Realización de un menú al que se puede acceder mediante RS232 o vía Telnet; y en el que se pueden visualizar los datos obtenidos por los conversores Analógico-Digital y leer o actuar sobre los estados de los relés (figura 6).

Figura 5.3: Menú visualización

- Realización de un menú para la configuración de factores de escala, que luego se utilizan para la calibración de las fuentes de alimentación.
- Implementación del algoritmo de control de las fuentes de alimentación para la Estación Submarina.

Referente a la estructura mecánica de la ES se han realizado con éxito las siguientes pruebas:

- Verificación de la compatibilidad estructural: las diferentes piezas de la estructura encajan correctamente.
- Estanquidad del cilindro principal de la ES: Comprobación de que el cilindro impide la entrada de agua en su interior. Se ha comprobado hasta 15 BAR de presión.
- Sistema de comunicaciones: Compatibilidad entre los distintos dispositivos de comunicaciones, pruebas de transferencia de datos entre switches de la ET y ES.

Capitulo 6. Conclusiones

En el presente proyecto se ha realizado una aplicación informática para el control y gestión de los equipos de medida instalados en un observatorio submarino. El observatorio submarino OBSEA, construido por el grupo SARTI de la UPC, se desplegó el 19 de mayo de 2009 con 3 instrumentos submarinos de medida (CTD, hidrófono, cámara submarina). Desde ese preciso instante está completamente operativo y funcionando correctamente.

Para conseguir el objetivo propuesto de desarrollar una aplicación informática para la gestión y control del observatorio submarino OBSEA, ha sido preciso realizar una serie de actuaciones:

- Familiarización con un microcontrolador de 32 bits trabajando bajo sistema operativo. Era la primera vez que se trabajaba de este modo y aparecieron algunas complicaciones.
- Además, la placa controladora utilizada para el desarrollo del OBSEA nos llegó al laboratorio con el sistema operativo funcionando y con un código fuente. Por esto, fue necesario entender ese código escrito por otro ingeniero. Una tarea bastante complicada.
- También, se realizó un estudio del protocolo SNMP para el envío de las posibles alarmas desde la estación submarina hasta la estación terrestre del OBSEA.
- Se hizo un estudio previo de las placas ADC y relés. En un principio se utilizó el software proporcionado por el fabricante y se visualizó con el analizador de protocolos las tramas enviadas y recibidas a través de las placas. Cabe decir, que el analizador de protocolos fue de gran ayuda.
- Tras hacer los anteriores pasos, se empezó a incluir el código para controlar las placas ADC / relés en el propio microcontrolador ColdFire MCF5282. Al tener un sistema operativo corriendo sobre la placa se vio que no era tan inmediato incluir el código ya que siempre se debe tener en cuenta que un sistema operativo debe realizar otras tarea para que su funcionamiento sea el correcto.

Una vez solucionados los anteriores puntos el resultado conseguido es un sistema robusto con las siguientes prestaciones:

- Control de los sensores conectados en el observatorio.
- Monitorización de los parámetros de control mediante SNMP.
- Envío de alarmas desde la Estación Submarina en el caso de que haya un mal funcionamiento.
- Visualización de la temperatura y humedad dentro del cilindro que contiene la electrónica de la Estación Submarina.

Los resultados del trabajo realizado han sido presentados a tres congresos internacionales:

- *Marc Nogueras; Carola Artero; Joaquín del Rio; Antoni Mànuel; David Sarrià.* Control and acquisition system design for an Expandable Seafloor Observatory (OBSEA). *IEEE Conference & Exhibition OCEANS 2009. May 11-14, 2009 Bremen, Germany. ISBN: 978-1-4244-2523-5*

- Diseño del sistema de control y adquisición de datos del Observatorio Submarino ExpAndible (OBSEA). *C. Artero; M. Nogueras; S. Shariat-Panahi; A. Mànuel; P. Santamaria; J. Cadena; O. Gualdo. Congreso: SAAEI08 ISBN 13:978-8496997-04-2. Lugar de celebración: Cartagena (España). Fecha: septiembre 9-11 de 2008. SAAEI.*
- *Oceans, poster VISO:* OBSEA: A Submarine Laboratory at the Spanish Mediterranean Coast *(http://www.oceanobservatory.com/news/viso_posters)*

De cara a futuras nuevas versiones, una de las posibles mejoras en la programación sería ampliar las funcionalidades de algunos de los comandos SNMP, como enviar el valor de los conversores ADC con una resolución de 12 bits o implementar más comandos para modificar otras variables del sistema.
Además, en un futuro se podría programar el microcontrolador que tiene las placas ADC / relés, exclusivamente para cubrir nuestras necesidades.

Capitulo 7. Bibliografía y Congresos

7.1. Bibliografía

NOGUERAS M, «*Especificación sistema OBSEA v0*»

STALLINGS W, «*SNMP, SNMPv2, SNMP v3 and RMON 1 and 2*»

«*MCF5282 ColdFire Microcontroller User's Manual*»

7.2. Bibliografía electrónica

68K/ColdFire, www.freescale.com/coldfire

Catalyst express 500G, http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps6545/product_data_sheet0900aecd80322aeb.html

DYCEC, http://www.dycec.com/

Iberco- Cámaras Hiperbáricas e Ingenieria, http://www.iberco.es/

uClinux- http://www.uclinux.org/

Kdevelop , http://www.kdevelop.org/

KDE, http://www.kde.org/

LE-3200-E, http://www.lineeye.co.jp/english/html/p_3200.html

iReasoning, http://www.ireasoning.com/

Ethereal, http://www.ethereal.com/

Wireshark, http://www.wireshark.org/

Nacional Control Devices "*ProXR Series*", http://controlanything.com/

Monterey Bay Aquarium Research Institute (MBARI), http://www.mbari.org/

NEMO- The New Millenium Observatory, http://www.pmel.noaa.gov/vents/nemo/index.html

7.3. Congresos

MARC NOGUERAS, CAROLA ARTERO, JOAQUÍN DEL RIO, ANTONI MÀNUEL, DAVID SARRIÀ. «Control and acquisition system design for an Expandable Seafloor Observatory (OBSEA) ». IEEE Conference & Exhibition OCEANS 2009. May 11-14, 2009. Bremen, Germany. ISBN: 978-1-4244-2523-5

C. ARTERO, M. NOGUERAS, S. SHARIAT-PANAHI, A. MÀNUEL, P. SANTAMARIA. J. CADENA. O. GUALDO. «Diseño del sistema de control y adquisición de datos del Observatorio Submarino ExpAndible (OBSEA) ». Congreso: SAAEI08 ISBN 13:978-8496997-04-2. Lugar de celebración: Cartagena (España). Fecha: septiembre 9-11 de 2008.

SAAEI, Oceans, poster VISO, (http://www.oceanobservatory.com/news/viso_posters)

PARTE II
(ANEXOS)

Anexo A. Acceso a la placa de control

A.1. Introducción

En este apartado se describen los diferentes accesos a la placa controladora. Podemos acceder de las siguientes formas:

- Acceso a uClinux: descargar de los ejecutables de la aplicación, consultar parámetros de configuración y/o procesos…
- Acceso a nuestra aplicación.

Antes de empezar con la explicación se va a hacer una breve descripción de las particiones montadas en el sistema:

- */etc/config*: Disco sobre memoria RAM. Al ser RAM, su contenido se pierde al rebotar la placa.
- */user*: Disco sobre memoria FLASH. En este disco están los ficheros de configuración.
- */user1*: Disco sobre memoria FLASH. En este disco se encuentran los logs del sistema.
- */user2/aplicación*: Se encuentran los ejecutables de nuestra aplicación.

A.2. Acceso a uClinux para descargar los ejecutables de la aplicación

Una vez compilado el proyecto es necesario descargar el ejecutable a la placa controladora mediante algún protocolo de transferencia de archivos. En este caso se utiliza Telnet para acceder al sistema operativo que tiene la placa controladora y luego se utiliza el protocolo FTP para descargar nuestra aplicación.
Para acceder a uClinux debemos hacer un telnet a la dirección IP de la controladora (en nuestro caso 192.168.1.36) y acceder como “root” tal y como se muestra en la A.1.

Figura A.1: Acceso a uClinux mediante telnet

Una vez hemos entrado en el sistema operativo de la controladora debemos irnos al directorio */user2/aplicación.* En este directorio lo más seguro es que ya exista algún ejecutable de una versión anterior, por tanto, si lo que deseamos es cargar otra versión primero tenemos que borrar la existente. Simplemente hay que escribir en la consola *rm* * para borrar todo el contenido del directorio.

Figura A.2: Borrado de ejecutables

Una vez tenemos el directorio listo lo que debemos hacer es acceder a la máquina de la que vamos a coger los ejecutables compilados. Por tanto debemos hacer un FTP desde la placa controladora a la dirección IP de nuestro PC (192.168.1.1).

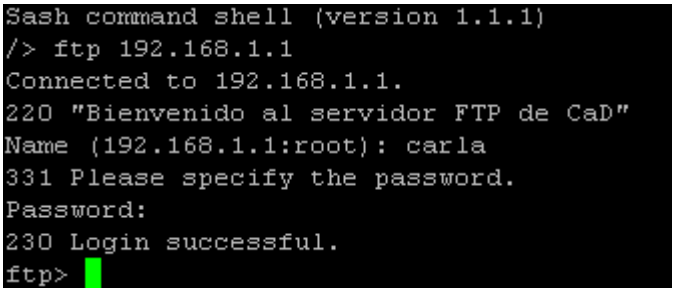

Figura A.3: Acceso por FTP desde la placa controladora hasta el PC

Ahora pasamos a realizar la descarga de los ejecutables. Es muy importante tener en cuenta cómo se realiza la transferencia de archivos. Para una correcta recepción de los ejecutables debemos realizar la transferencia en modo binario (bin).
Luego cogeremos los ejecutables de nuestra maquina haciendo un *"get"*. Escribimos get y seguidamente el nombre del archivo que vamos a descargar. Por ejemplo, *"get main"*, *"get menú"*, *"get snmpd"*.

Figura A.4: Transferencia de archivos en binario

Una vez hemos descargado los ejecutables salimos del FTP escribiendo *"bye"*. Como se ve en la siguiente figura en uClinux están los 3 ejecutables.

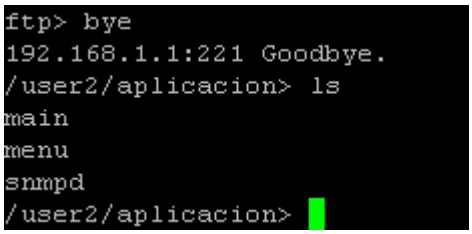

Figura A.5: Ejecutables copiados a la placa controladora

A.3. Acceso a uClinux para la consulta de parámetros de configuración y/o procesos

Como se puede ver en la figura A.6, el directorio de archivos de uClinux es muy similar al de un sistema linux de un PC.

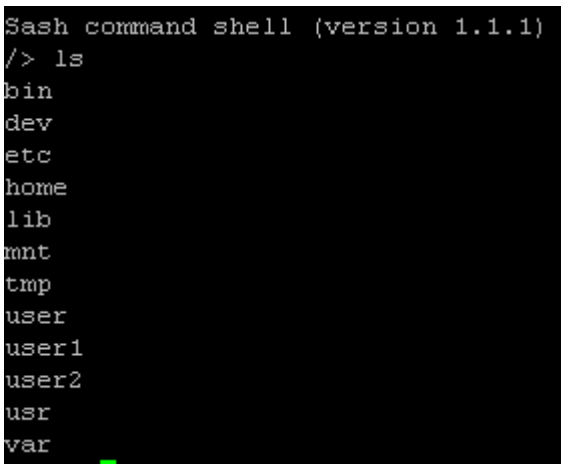

Figura A.6: Directorio principal de uClinux

bin	Binarios de comandos esenciales
dev	Archivos de dispositivos
etc	Configuración del sistema local-máquina
home	Directorios /home de los usuarios
lib	Librerías compartidas
mnt	Punto de montaje de particiones temporales
tmp	Archivos temporales
user	Ficheros de configuración de procesos
user1	Ficheros de logs del sistema
user2	Se encuentran los ejecutables de la aplicación
usr	Segunda jerarquía mayor
var	Información variable

A.4. Acceso a las opciones de la placa controladora mediante menú Telnet

A continuación se van a explicar las opciones de las que dispone la placa controladora y cómo accedemos a ellas mediante telnet.
Para acceder a la placa hay que seguir los siguientes pasos:

- Debemos conectarnos mediante telnet (telnet 192.168.1.34)
- Para tener acceso a la configuración de los parámetros, es necesario logearse con el usuario “admin”.

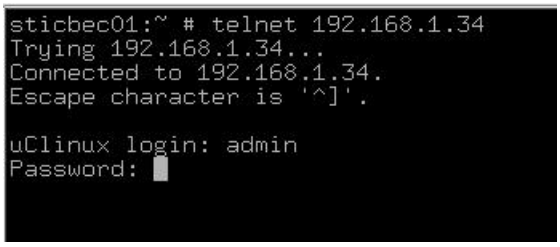

Figura A.7: Acceso mediante telnet al menú de usuario

Una vez se ha introducido la contraseña aparecerá un menú como el de la Figura A.8.

Figura A.8: Visualización de parámetros

0. MOSTRAR MEDIDAS Y ESTADO

En este apartado se muestra:

- Lectura de los 2 bancos de conversores ADC. El resultado varía entre 0 y 255
- Lectura del estado de 4 bancos de relés. El resultado aparece como “A” en el caso de que el relé esté encendido y “R” en el caso de que el relé esté en reposo.

Figura A.9: Visualización desde menú del estado relés y ADC

1. CONFIGURACION DE MEDIDAS Y PUERTO

En este apartado se muestra:

- Umbral de alarma para un sensor de temperatura. Actualmente el sensor no existe y en el proyecto OBSEA no se utiliza.
- Umbral de alarma para un sensor de humedad. Actualmente el sensor no existe y en el proyecto OBSEA no se utiliza
- Periodo de muestreo para el envío de traps de alarma.

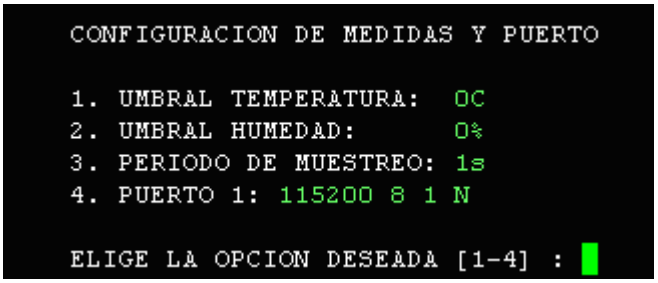

Figura A.10: Configuración parámetros

2. CONFIGURACION FACTORES DE ESCALA Y CONTROL RELES

En este apartado se muestra:

- Configuración Factores de Escala: Estos factores son necesarios para implementar el algoritmo de control. Se accede a un submenú en el que se pueden modificar hasta 8 factores de escala (figuras A.11 y A.12).

- Control Relés: Apartado necesario para la configuración del algoritmo de control. Se accede a un submenú en el que se puede optar por poner en funcionamiento el algoritmo de control, o dejar que la actuación de los dispositivos sea manual (figura A.13).

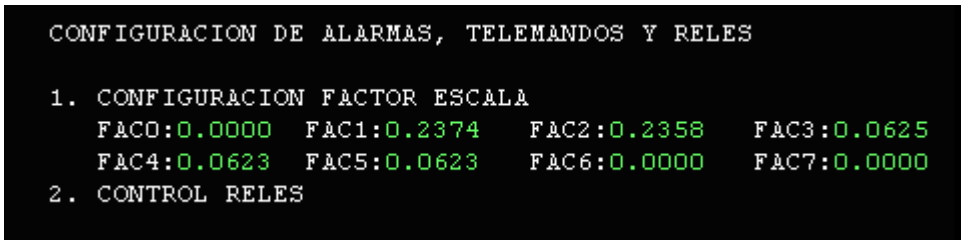

Figura A.11: Configuración factores de escala

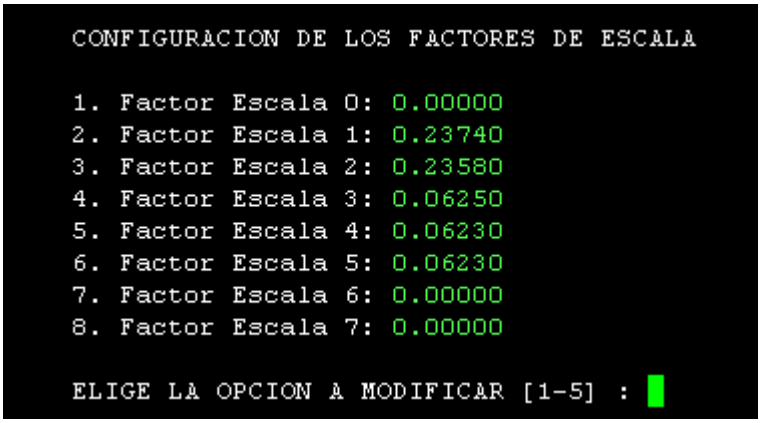

Figura A.12: Menú factores de escala

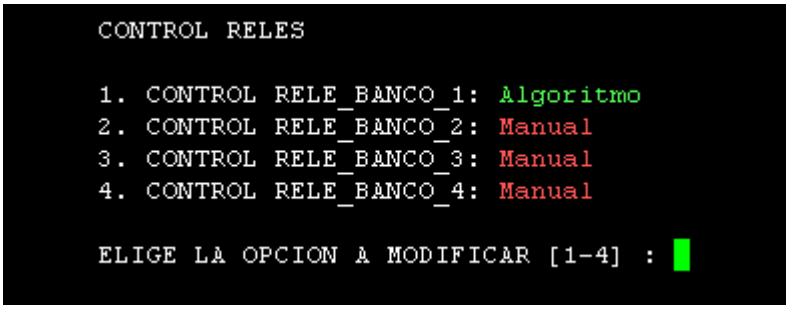

Figura A.13: Menú configuración de relés

3. ___CONFIGURACION ESTADO RELES

Apartado actualmente deshabilitado

4. CONFIGURACION DE USUARIOS

En este apartado, un usuario administrador puede gestionar la creación, modificación y borrado del resto de usuarios.

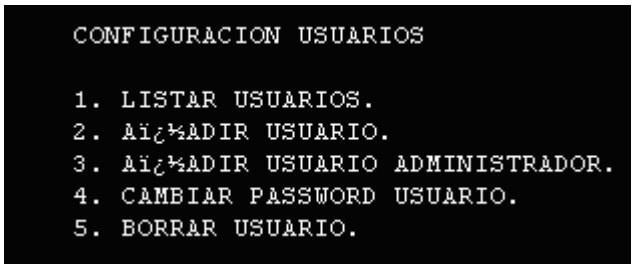

Figura A.14: Configuración de usuarios

5. CONFIGURACION DE PARAMETROS ETHERNET

En este apartado se realiza la configuración de la dirección IP del equipo. El administrador puede fijar una dirección IP o puede activar el DHCP de forma que sea el servidor en la red el que le proporcione una dirección IP disponible.

Figura A.15: Configuración parámetros ethernet

6. CONFIGURACION DE PARAMETROS SNMP

En este apartado se realiza la configuración de los parámetros SNMP.

- Configuración del destino de los traps o alarmas (figura A.17).
- Configuración de la comunidad de lectura para poder acceder a las variables (figura A.18).
- Configuración de la comunidad de escritura para poder modificar el estado de las variables (figura A.19).

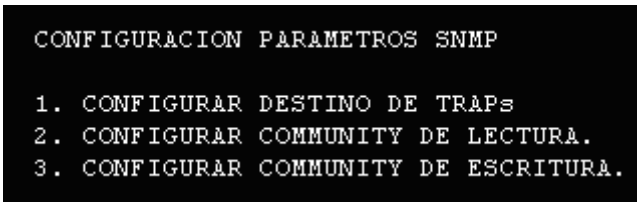

Figura A.16: Configuración parámetros SNMP

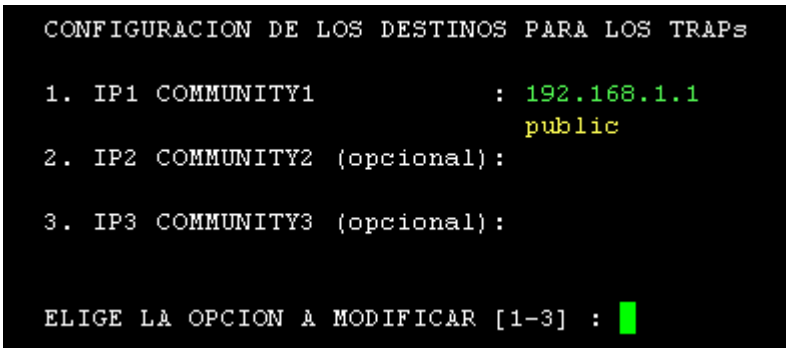

Figura A.17: Configuración destino traps

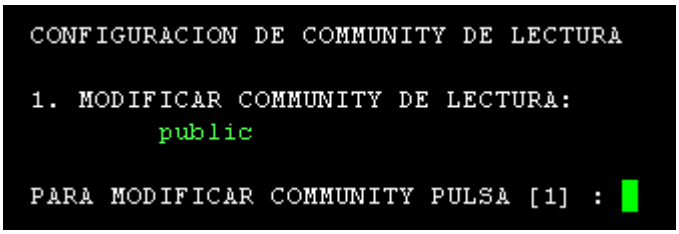

Figura A.18: Configuración comunidad de lectura

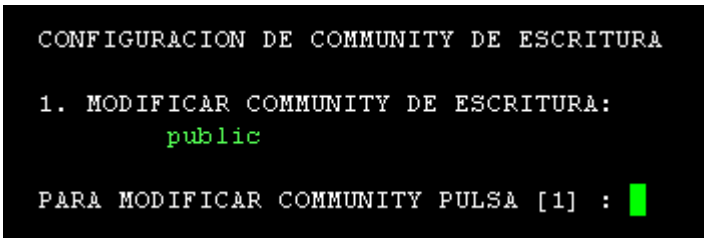

Figura A.19: Configuración comunidad de escritura

7. CONFIGURACION DE GESTION DE LOGS Y FECHA

En este apartado se realiza:

- Configuración del destino de los logs del sistema en los que se muestra información del funcionamiento de la aplicación
- Los niveles de traza permitidos son los siguientes:
    - FATAL: Avería o mal funcionamiento por lo que la aplicación no pueda continuar.
    - ERROR: Mal funcionamiento de un módulo software o dispositivo
    - WARNING: Avisos sobre sucesos anómalos y ocasionales.
    - INFORMATION: Estado de la aplicación (máquinas de estado, etc.)
    - DEBUG: Información de depuración (se muestran valores de variables, etc.)
- Modificación de fecha y hora del sistema.

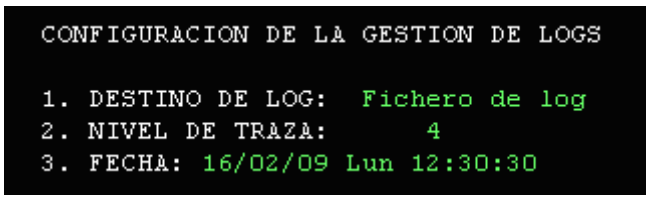

Figura A.20: Configuración destino de trazas

8. SALIR GUARDANDO LOS CAMBIOS / ACTUALIZAR

En este apartado se pueden guardar los cambios que se han realizado en la configuración de la aplicación. Una vez seleccionada esta opción es posible abandonar o volver a la aplicación.

9. SALIR SIN GUARDAR LOS CAMBIOS

Permite abandonar la aplicación sin guardar ningún cambio.

Anexo B. Implementación de una MIB

```
DYCEC-MIB DEFINITIONS ::= BEGIN

IMPORTS
        enterprises                     FROM SNMPv2-SMI,
        MODULE-IDENTITY                 FROM SNMPv2-SMI
        MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF;

dycec MODULE-IDENTITY
   ORGANIZATION "SARTI"
   CONTACT-INFO "
            Author: Carla Artero Delgado


            email:     carola.artero@upc.edu
            phone:
           "
   DESCRIPTION  "MIB Control Dispositivos
           "
   ::= { enterprises 12592 }

medidas_ADC                     OBJECT IDENTIFIER ::= { dycec 1 }
ALARMAS                         OBJECT IDENTIFIER ::= { dycec 2 }
medidas_ADC_2                   OBJECT IDENTIFIER ::= { dycec 3 }
medidas_R_BANCO_ALL             OBJECT IDENTIFIER ::= { dycec 4 }

adc00          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
        "Variable que almacena el resultado de la conversion ADC."
   DEFVAL { 0 }
   ::= { medidas_ADC 1 }

adc01          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
        "Variable que almacena el resultado de la conversion ADC1."
   DEFVAL { 0 }
   ::= { medidas_ADC 2 }
```

…… *(codigo parecido para el resto)*

```
adc14          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Variable que almacena el resultado de la conversion ADC14."
   DEFVAL { 0 }
   ::= { medidas_ADC 15 }

adc15          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Variable que almacena el resultado de la conversion ADC15."
   DEFVAL { 0 }
   ::= { medidas_ADC 16 }

alarma00          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Alarma00."
   DEFVAL { 0 }
   ::= { ALARMAS 1 }

alarma01          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Alarma01."
   DEFVAL { 0 }
   ::= { ALARMAS 2 }
```

…… *(codigo parecido para el resto)*

```
alarma16          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Alarma08."
   DEFVAL { 0 }
   ::= { ALARMAS 17 }

alarma17          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
```

```
   DESCRIPTION
         "Alarma08."
   DEFVAL { 0 }
   ::= { ALARMAS 18 }

adc2_00          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Variable que almacena el resultado de la conversion ADC_2."
   DEFVAL { 0 }
   ::= { medidas_ADC_2 1 }

adc2_01          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Variable que almacena el resultado de la conversion ADC1_2."
   DEFVAL { 0 }
   ::= { medidas_ADC_2 2 }

……. (codigo parecido para el resto)

adc2_14          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Variable que almacena el resultado de la conversion ADC14_2."
   DEFVAL { 0 }
   ::= { medidas_ADC_2 15 }

adc2_15          OBJECT-TYPE
   SYNTAX      Integer32 (-65535..65536)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Variable que almacena el resultado de la conversion ADC15_2."
   DEFVAL { 0 }
   ::= { medidas_ADC_2 16 }

R_BANCO_0_ALL          OBJECT-TYPE
   SYNTAX      PositiveInteger
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Todo Banco 0"
   DEFVAL { 0 }
   ::= {medidas_R_BANCO_ALL 1 }

R_BANCO_1_ALL          OBJECT-TYPE
   SYNTAX      PositiveInteger
   MAX-ACCESS  read-write
```

```
   STATUS      current
   DESCRIPTION
         "Todo Banco 1"
   DEFVAL { 0 }
   ::= {medidas_R_BANCO_ALL 2 }

R_BANCO_2_ALL          OBJECT-TYPE
   SYNTAX      PositiveInteger
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Todo Banco 2"
   DEFVAL { 0 }
   ::= {medidas_R_BANCO_ALL 3}

R_BANCO_3_ALL          OBJECT-TYPE
   SYNTAX      PositiveInteger
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
         "Todo Banco 3"
   DEFVAL { 0 }
   ::= {medidas_R_BANCO_ALL 4}
```

Anexo C. Calibración

A continuación se presentan algunas de las tablas de calibración de las placas ADC / relés.
En la tabla C.1 tenemos la calibración de la fuente de 350V. Con una tensión conocida (Vin) se han sacado los valores medios leídos en los ADC y se ha hecho un promediado.
Los valores de la columna "Vo placa control fuentes" se sacan de la división de la columna de promediado entre 819. Éste valor se saca así:

- Los valores medidos están con una precisión de 12 bits. $2^{12} = 4096$
- 4096 / 5V = 819

Lo mismo se ha realizado para el cálculo de las dos tablas restantes, C.2 y C.3.

Vin	Vo placa control fuentes	Promediado	Valores medidos						
48,12	0,488	400	400	404	399	401	397	400	401
76,3	0,753	617	617	619	615	616	620	616	614
105,99	1,039	851	851	848	853	850	851	849	853
133,82	1,292	1058	1058	1059	1055	1057	1056	1063	1060
161,54	1,578	1292	1292	1294	1290	1293	1295	1287	1290
189,27	1,850	1515	1515	1511	1518	1514	1517	1510	1517
216,8	2,121	1737	1737	1740	1737	1735	1736	1738	1737
244,4	2,392	1959	1959	1959	1962	1964	1961	1951	1958
272,1	2,667	2184	2184	2185	2181	2186	2187	2183	2181
296,6	2,939	2407	2407	2409	2405	2406	2407	2405	2410
327,1	3,214	2632	2489	2490	2489	2487	2490	2489	2490
Vin =	101,162	1,611							

Tabla C.1: Calibración de la fuente de entrada de 350 Voltios

Bateria	Vo	Valor A/D 1	Valores medidos						
0	0	0							
25,008	1,83028083	1499	1499	1500	1499	1499	1500	1497	1501
50,66	3,716727717	3044	3044	3045	3043	3042	3047	3042	3045
Vbat =	13,637	x Vo							

Tabla C.2: Calibración de la batería

PW2	Vo	Valor A/D 2	Valores medidos						
39,97	2,931623932	2401	2401	2401	2404	2401	2401	2399	2401
42,08	3,085470085	2527	2527	2526	2527	2526	2528	2529	2524
44,03	3,228327228	2644	2644	2643	2642	2645	2644	2643	2647
45,99	3,373626374	2763	2763	2762	2763	2764	2761	2763	2763
48,04	3,523809524	2886	2886	2887	2885	2884	2887	2888	2887
50,02	3,67032967	3006	3006	3005	3013	3007	3002	3008	3003
52,16	3,825396825	3133	3133	3131	3132	3133	3133	3134	3133
54,14	3,970695971	3252	3252	3252	3250	3251	3251	3253	3252
Vps2 =	13,6340884	x Vo							

Tabla C.1: Calibración de la fuente de 48 Voltios

Anexo D. Código de lectura del estado de los ADC y relés

```
unsigned char ActualizarLecturaADC(medidas *m, int semid)
{
    // Inicialización de  las variables
    switch (estado)
   {
   case 0:
         com2_inuse_LEEestado = 1;
         leidos = LeerCaracteres(2, buf_adc, 16);
         if (leidos <= 0)
         {
                  cadena_au[0] = 254;
                  cadena_au[1] = 196;
                  cadena_au[2] = '\0';
                  EscribirCaracteres(2, cadena_au, 2);
                  DesactivarFlag(FLAG_ADC);
                  ActivarFlag(TIEMPO_ESPERA_ADC,FLAG_ADC);
                  reconfig = 0;
                  estado = 1;
            }
   break;
   case 1:
         if ((ComprobarFlag(FLAG_ADC) == 1) || (ComprobarFlag(FLAG_ADC) == -1))
         {
                  DesactivarFlag(FLAG_ADC);
                  no_respuesta_ADC = 1;
                  leidos = LeerCaracteres(2, buf_reles, 1);
                  if (leidos <= 0)
                  {
                           cadena_aux[0] = 254;
                           cadena_aux[1] = 124;
                           cadena_aux[2] = 0;
                           cadena_aux[3] = '\0';
                           EscribirCaracteres(2, cadena_aux, 3);
                           DesactivarFlag(FLAG_RELES);
                           ActivarFlag(TIEMPO_ESPERA_RELES,FLAG_RELES);
                           estado = 2;
                  }
            }else
            {
                  leidos = CaracteresDisponibles(2);
                  if (leidos >= 32)
                  {
```

```
                    no_respuesta_ADC = 0;
                    LeerCaracteres(2, buf_adc, 32);
                    P(semid);
                    m->adcc15 = (((buf_adc[31] & 0x0f) << 4) |
                                        ((buf_adc[30] & 0xf0) >> 4));
                        …….……..……..……..…
                        …….……..……..……..…
                    m->adcc0 = (((buf_adc[1] & 0x0f) << 4) | ((buf_adc[0] & 0xf0) >> 4));
                    V(semid);
                    DesactivarFlag(FLAG_ADC);
                    leidos = LeerCaracteres(2, buf_reles, 1);
                     if (leidos <= 0)
                    {
                            cadena_aux2[0] = 254;
                            cadena_aux2[1] = 124;
                            cadena_aux2[2] = 0;
                            cadena_aux2[3] = '\0';
                            EscribirCaracteres(2, cadena_aux2, 3);
                            DesactivarFlag(FLAG_RELES);
                            ActivarFlag(TIEMPO_ESPERA_RELES,FLAG_RELES);
                            estado = 2;
                     }
            }
    }
    break;
    case 2:
            if ((ComprobarFlag(FLAG_RELES) == 1) ||
                     (ComprobarFlag(FLAG_RELES) == -1))
            {
                    DesactivarFlag(FLAG_RELES);
                    no_respuesta_RELES = 1;
                    leidos = LeerCaracteres (2, buf_adc_2,1);
                    if (leidos <= 0)
                    {
                             cadena_au_2[0] = 254;
                            cadena_au_2[1] = 197;
                             cadena_au_2[2] = '\0';
                            EscribirCaracteres(2, cadena_au_2, 2);
                            DesactivarFlag(FLAG_ADC);
                            ActivarFlag(TIEMPO_ESPERA_ADC,FLAG_ADC);
                            estado =3;
                    }
            }else
             {
                    leidos = CaracteresDisponibles(2);
                    if (leidos >= 32)
                    {
                            LeerCaracteres(2, buf_reles, 32);
                            P(semid);
                             m->R_BANCO_0 = buf_reles[0];
                             m->R_BANCO_1 = buf_reles[1];
                            m->R_BANCO_2 = buf_reles[2];
                            m->R_BANCO_3 = buf_reles[3];
                            V(semid);
```

```
                    if ((buf_reles[0] & 0x01) != 0)
                    {
                            intermedio1[0] = 0x01;
                     }
                    else
                     {
                            intermedio1[0] = 0x00;
                    }
                     if ((buf_reles[0] & 0x02) != 0)
                     {
                            intermedio1[1] = 0x01;
                     }
                     else
                    {
                            intermedio1[1] = 0x00;
                     }
                    …….……..……..……..……..…
                    …….……..……..……..……..…
                     if ((buf_reles[3] & 0x80) != 0)
                    {
                            intermedio4[7] = 0x01;
                    }
                    else
                    {
                            intermedio4[7] = 0x00;
                    }
                    if ((m->estadoReles00 != intermedio1[0]) ||
                    (m->estadoReles01 != intermedio1[1]) ||
                     …….……..……..……..……..…
                     …….……..……..……..……..…
                    (m->estadoReles06 != intermedio1[6]) ||
                    (m->estadoReles37 != intermedio4[7]))
                    {
                            reconfig = 1;
                            P(semid);
                            m->estadoReles00 = intermedio1[0];
                            m->estadoReles01 = intermedio1[1];
                            m->estadoReles36 = intermedio4[6];
                            m->estadoReles37 = intermedio4[7];
                            V(semid);
                    }
                     no_respuesta_RELES = 0;
                    DesactivarFlag(FLAG_RELES);
                    leidos = LeerCaracteres (2, buf_adc_2,1);
                    if (leidos <= 0)
                     {
                            cadena_au_22[0] = 254;
                            cadena_au_22[1] = 197;
                            cadena_au_22[2] = '\0';
                            EscribirCaracteres(2, cadena_au_22, 2);
                            DesactivarFlag(FLAG_ADC);
                            ActivarFlag(TIEMPO_ESPERA_ADC,FLAG_ADC);
                            estado = 3;
                     }
            }
```

```
                }
                 break;
                case 3:
                if ((ComprobarFlag(FLAG_ADC) == 1) ||
                                (ComprobarFlag(FLAG_ADC) == -1))
                {
                        DesactivarFlag(FLAG_ADC);
                        if ((no_respuesta_ADC == 1) && (no_respuesta_RELES == 1))
                                retorno = ADC_NO_ANS;
                        else
                        {
                                if(reconfig == 0)
                                        retorno = ADC_OK;
                                else
                                        retorno = ADC_RECONFIG;
                        }
                        estado = 0;
                 }else
                 {
                        leidos = CaracteresDisponibles(2);
                        if (leidos >= 32)
                        {
                                 if(reconfig == 0)
                                        retorno = ADC_OK;
                                 else
                                        retorno = ADC_RECONFIG;
                                LeerCaracteres(2, buf_adc_2, 32);
                                P(semid);
                                 m->adcc2_15 = (((buf_adc_2[31] & 0x0f) << 4) |
                                                ((buf_adc_2[30] & 0xf0) >> 4));
                                 …….……..……..……..……
                                ……..……..……..……..……
                                m->adcc2_0 = (((buf_adc_2[1] & 0x0f) << 4) |
                                                ((buf_adc_2[0] & 0xf0) >> 4));
                                V(semid);
                                DesactivarFlag(FLAG_ADC);
                                 com2_inuse_LEEestado = 0;
                                 estado = 0;
                        }
                 }
                break;
                default:
                        DesactivarFlag(FLAG_ADC);
                        DesactivarFlag(FLAG_RELES);
                        com2_inuse_LEEestado = 0;
                        estado = 0;
                break;
        }
        return(retorno);
}
```

Anexo E. Analizador de Protocolos LE 3200E

Figura E.1: Analizador de Protocolos LE 3200E

LE-3200 es un analizador de protocolos que permite monitorizar un puerto RS-232. Soporta una velocidad máxima de 1.544Mbps.

E.1. Características de LE-3200

- Mediadas con velocidad máxima de 1.5Mbps.
- Equipado con RS-232C y RS-422/485.
- Memoria de 3.6MB
- Software que permite la comunicación entre el analizador y el PC.
- Duración de la batería aproximadamente de 8 horas.
- Peso: 950g
- LE-3200 viene con el interfaz para los estándares RS-232C (V.24) y RS-422/485 (RS-530). X.21, RS-449 y V.35 requieren solamente un cable dedicado (Figura C.2).

Figura E.2: Conexiones Analizador de Protocolos

- Alguno de los parámetros a configurar tales como la velocidad de conexión, la paridad… se pueden ver en la figura E.3.

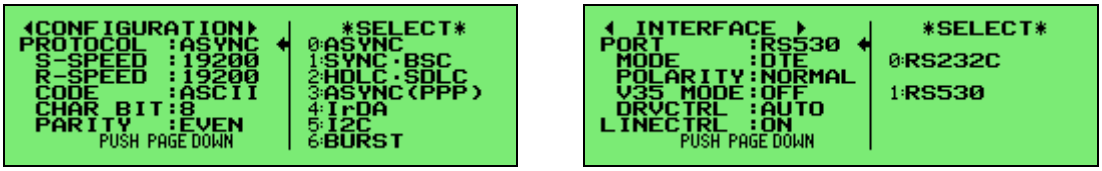

Figura E.3: Visualización parámetros Analizador Protocolos

- Como se puede ver soporta varios estándares e interfaces con solo añadir el modulo correspondiente (Figura E.4 y E.5).

Figura E.4: Módulos de expansión para Analizador de Protocolos

Figura E.5: Protocolos expansión para Analizador de Protocolos

Anexo F. Servidores utilizados

F.1. Introducción

A continuación se hace una breve descripción de los protocolos utilizados en el presente proyecto:

Telnet-> Protocolo de red para acceder a otra maquina
FTP-> Protocolo de transferencia de archivos
SSH-> Protocolo de red para acceder a otra máquina pero con más ventajas que Telnet

F.2. Protocolo Telnet

Telnet (TELecommunication NETwork) es un protocolo que sirve para acceder a una máquina a través de una red. Este acceso se realiza en modo terminal (sin gráficos).
En general, telnet se utiliza para abrir una sesión en una máquina linux sin estar físicamente en ella. De este modo múltiples usuarios con cuenta de usuario se conectan y abren una o varias sesiones.
El protocolo telnet está dentro del nivel de aplicación de la Pila OSI. Además, en la capa de aplicación también aparecen otros de los protocolos utilizados (SSH o FTP).

Figura F.1: Pila OSI

Para acceder a una maquina mediante telnet solamente hay que escribir telnet seguido de la dirección IP del dispositivo. Por ejemplo, “telnet 192.168.1.36”.
Como se verá a continuación, otro de los protocolos utilizados, SSH, tiene muchas más ventajas respecto al protocolo telnet. Entre otras, telnet carece de seguridad y SSH no. La placa controladora utilizada para OBSEA no tiene el protocolo SSH activado, por esto, cuando queramos acceder a la controladora lo haremos mediante telnet y cuando debamos descargar archivos lo haremos por FTP.

F.3. Protocolo FTP

F.3.1. Introducción

Como se ha comentado anteriormente, debemos conectarnos a la placa vía telnet. Una vez conectados, vamos a realizar la transferencia de archivos mediante FTP.
A continuación, se realiza una breve explicación del protocolo FTP y de los archivos de configuración necesarios para el correcto funcionamiento del servidor.

FTP (File Transfer Protocol) es un protocolo de transferencia de archivos entre sistemas conectados a una red TCP basada en la arquitectura cliente-servidor.
Desde un equipo cliente nos conectamos a un equipo servidor para descargar o subir archivos.

Uno de los problemas que presenta este servicio FTP es la falta de seguridad. Todo el intercambio de información, desde que el usuario se conecta hasta la transferencia de información se realiza en texto plano, es decir, sin ningún tipo de cifrado.
Para poder solucionar el problema de la seguridad se utiliza una variante del protocolo, llamado VSFTPD. Éste, permite gestionar el ancho de banda entregado para evitar posibles saturaciones.

F.3.2. Configuración del servicio FTP

Es importante saber que la siguiente configuración solamente es aplicable a maquinas linux.

A continuación se muestran los principales ficheros de configuración:

/etc/vsftpd/vsftpd.conf	Fichero de configuración principal del demonio FTP.
/etc/vsftpd.ftpusers	Usuarios no permitidos para hacer FTP.
/etc/vsftpd.user_list	Usuarios permitidos o no para hacer FTP. Depende de la directiva *userList_deny* si está a NO o YES respectivamente.
/etc/vsftpd.chroot_list	Lista de usuarios que al conectar estarán excluidos de tomar cómo raíz el directorio *home*.
/var/ftp	Directorio para el usuario anónimo.
/etc/pam.d/vsftpd	Fichero PAM que indica dónde se encuentran los usuarios del servidor FTP. Por defecto los usuarios son los usuarios del sistema linux. Permite configurarse para tener usuarios FTP diferentes a los usuarios del sistema.
/etc/vsftpd_login.db	Fichero que contiene una Base de Datos de usuarios distintos del sistema.

Para poder arrancar o parar el servicio debemos abrir una consola de comandos linux y escribir lo siguiente:

```
/sbin/service vsftpd {start | stop | restart | status}
```

Es decir, si vamos a arrancar el servicio debemos escribir:

/sbin/service vsftpd start

en cambio, si lo que deseamos es restaurar el servicio debemos escribir:

/sbin/service vsftpd restart

Para la activación del servicio es necesario editar un archivo con la configuración. Se edita el archivo:

/etc/vsftpd/vsftpd.conf

L*os* parámetros a modificar son los siguientes:

- Parámetro *anonymous_enable*

  Se utiliza para definir si se permitirán los accesos anónimos al servidor. Se debe establecer el valor a YES o NO de acuerdo a lo que se quiera.

  anonymous_enable=YES

  El directorio para usuarios anónimos es */var/ftp*

- Parámetro *local_enable*

  Establece si se van a permitir los accesos autentificados de los usuarios locales del sistema. Se establece el valor YES o NO de acuerdo a lo que se requiera.

  local_enable=YES

  Si la directiva *chroot_local_user* no está puesta a YES, hace que el usuario pueda navegar por el árbol completo de directorios del sistema, esto lógicamente es muy peligroso, por lo tanto, es conveniente colocar la siguiente directiva para enjaular al usuario autenticado en su directorio */HOME*.

  chroot_local_user=YES

- Cambio del puerto del servidor:

  La directiva *listen_port* permite cambiar el puerto por el que escucha el servidor. Por defecto es el puerto 21. La directiva *listen=YES* permite configurar el servidor en modo standalone (solitario).

  connect_from_port_20=YES
  listen_port=21
  listen=YES

- Parámetro *write_enable*

  Este parámetro establece si se permite la escritura en el servidor. Se establece el valor YES o NO de acuerdo a lo que se requiere.

- Parámetro *ftpd_banner*

  Establece el banderín de bienvenida que será mostrado cada vez que un usuario acceda al servidor. Se puede establecer cualquier frase breve que se considere conveniente.

  ftpd_banner=Bienvenido al servidor FTP de Carla.

- Parámetro *anon_max_rate*. (Control del ancho de banda.)

  Utilizado para limitar la tasa de transferencia en bytes por segundo para los usuarios anónimos, algo sumamente útil en servidores FTP de acceso público. En el siguiente ejemplo, se limita la tasa de transferencia a 5 Kb por segundo para los usuarios anónimos:

  anon_max_rate=5120

- Parámetro *local_max_rate*

Este parámetro hace lo mismo que *anon_max_rate*, pero lo aplica a usuarios locales. En el siguiente ejemplo se muestra la limitación de la tasa de transferencia a 5 Kb por segundo para los usuarios locales:

local_max_rate=5120

- Parámetro *max_clients*

  Establece el número máximo de clientes que podrán acceder simultáneamente al servidor FTP. En el siguiente ejemplo se limita el acceso a 5 clientes simultáneos.

  max_clients=5

- Parámetro *max_per_ip*

  Establece el número máximo de conexiones que se pueden realizar desde una misma dirección IP. En el siguiente ejemplo se limita el número de conexiones por IP simultáneas a 5.

  max_per_ip=5

F.3.3. Ejemplo de configuración de ficheros:

A continuación se muestra un ejemplo de una posible configuración de un servidor FTP.

Fichero "vsftpd.conf"

```
ftpd_banner="Bienvenido al servidor FTP de Carla"
local_enable=YES
chroot_list_enable=YES
chroot_local_user=YES
chroot_list_file=/etc/vsftpd.chroot_list
anonymous_enable=NO
connect_from_port_20=YES
pam_service_name=vsftpd
listen=YES
ssl_enable=NO
user_config_dir=/etc/vsftpd/users
userlist_enable=YES
```

Fichero "vsftpd.user_list"

```
dirlist_enable=YES
download_enable=YES
local_root=/home/carla
write_enable=NO
anon_world_readable_only=NO
```

F.4. Protocolo SSH

F.4.1. Introducción

SSH (Secure SHell o intérprete de comandos seguro) es el nombre del protocolo utilizado para acceder a maquinas remotas a través de una red.
SSH trabaja de forma similar a como lo hace telnet. La principal diferencia es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio lo haga de manera no legible. De esta forma, nadie puede descubrir el usuario y contraseña de la conexión ni lo que se escribe durante la sesión.

F.4.2. Ficheros de configuración

A continuación se muestran los principales ficheros de configuración.

/etc/ssh/sshd_config	Fichero de configuración principal del servidor SSH
/etc/ssh/ssh_config	Fichero de configuración principal del cliente SSH

Para poder arrancar o parar el servicio debemos abrir una consola de comandos Linux y escribir lo siguiente:

```
/sbin/service sshd {start | stop | restart | status}
```

El anterior comando se utiliza de la siguiente forma:

```
/sbin/service sshd start
```

en cambio, si lo que deseamos es restaurar el servicio debemos escribir:

```
/sbin/service sshd restart
```

F.4.4. Procedimientos

Antes que nada, es necesario editar el siguiente archivo de configuración:

/etc/ssh/sshd_config

A continuación analizaremos los parámetros a modificar o añadir según sea necesario.

- Parámetro *Port*

  Se puede cambiar el puerto por el cual escucha SSH. Por defecto es el puerto 22.

  Port 22

- Parámetro *ListenAddress*

  Se especifica bajo qué interfaces responderá a las peticiones.

  ListenAddress 192.168.1.20

- Parámetro *PermitRootLogin*

  Especificamos si el súper usuario root podrá conectarse mediante SSH. Por razones de seguridad siempre es mejor que no se pueda realizar la conexión con root.

  PermitRootLogin NO

- Parámetro *X11Forwarding*

  Especificamos si es necesario que se ejecuten aplicaciones graficas.

  X11Forwarding NO

- Parámetro *AllowUsers*

  Especificamos qué usuarios se conectarán mediante SSH.

  AllowUsers carla, juan

Anexo G. Presupuesto

A continuación se detallan los costes del nodo sumergido del proyecto OBSEA.
En él están incluidos desde la construcción de la estructura que lleva la electrónica dentro, hasta el material para realizar las pruebas de test.
Todo lo explicado en la memoria de este proyecto entra dentro del punto 1.2 (Electrónica nodo).

Además, cabe decir que el presupuesto está hecho sobre el nodo prototipo OBSEA. En el caso de que se amplíen los nodos, los costes invertidos tales como equipos de prueba o desarrollo software ya no se contarían. Sólo debería añadirse los costes extras de desarrollo software, mantenimiento de material de pruebas y nuevo material para el siguiente nodo.

Presupuesto Proyecto OBSEA

Estimación total coste material OBSEA - €

1	Nodo Sumergido	48.514,00 €
1.1	Estructura	
1.1.1	Gabia base estructura	8.514,00 €
1.1.2	Cilindro	5.544,80 €
1.1.3	Modificaciones estructura	4.822,00 €
1.1.4	Cilindro adicional empalme óptico	2.000,00 €
1.2	Electrónica nodo	
1.2.1	Subrack y electrónica control	3.000,00 €
1.2.2	Swiches comunicaciones	4.300,00 €
1.2.3	Fuentes alimentación y baterias	1.600,00 €
1.2.4	Cableado y conectores	500,00 €
1.2.5	Desarrollo Microcontrolador ColdFire	11.217,20 €
1.2.6	Sensores internos	300,00 €
1.2.7	Desarrollo fuente alimentación	3.000,00 €
1.2.8	Conversores serie ethernet	200,00 €
1.2.9	Entradas A/D y salidas relé	1.266,00 €
1.2.10	Fabricación placas electrónica	2.250,00 €

	2 Equipamiento para las pruebas		59.492,91 €
2.1	Cámara hiperbarica		
2.1.1	Fabricación cámara	42.230,00 €	
2.1.2	Infrastructura soporte	3.000,00 €	
2.1.3	Accesorios	1.207,87 €	
2.2	Instrumentación laboratorio		
2.2.1	Instrumentos medida óptica	3.387,20 €	
2.2.2	Intrumentos medida comunicaciones serie	3.000,00 €	
2.2.3	Fuente alimentación 1500 Vdc	5.857,00 €	
2.2.4	Cámara IP	810,84 €	
	3 Otras Gastos		5.389,62 €
3.1	Consultoria y asesoramiento externo	1.392,00 €	
3.2	Viajes y desplazamientos	997,62 €	
3.3	Varios	3.000,00 €	
	TOTAL		113.396,53 €

Anexo H: GLOSARIO

ADC: conversores analógico a digital.

ANSI: lenguaje de programación.

CSIC: Consejo Superior de Investigaciones Científicas.

CTD: conductividad, temperatura y profundidad.

ES: Estación submarina.

ET: Estación terrestre.

GCC: Colección de Compiladores GNU.

I2C: protocolo de comunicación.

MIB: Base de Información Gestionada

OBSEA: Observatorio submarino expandible.

OID: identificador de objeto.

PLC: controlador lógico programable.

QSPI: protocolo de comunicación.

RTC: reloj en tiempo real.

SARTI: sistema de adquisición remota y tratamiento de la información.

SNMP: Protocolo Simple de Administración de Red

TCP / IP: protocolo de control de transmisión / protocolo de internet.

UART: Transmisor-Receptor Asíncrono Universal

UDP: Protocolo de datagrama de usuario

UPC: Universidad Politécnica de Catalunya.