

Máster Dual en Industria 4.0

Universidad de Córdoba
Instituto de Estudios de Posgrado

Trabajo Fin de Máster

**Adaptación de un Supermercado
a la Industria 4.0**

Autor: Cristian Cuesta Sánchez

Director: José Manuel Palomares Muñoz

Córdoba, Agosto de 2020

Agradecimientos

A mi madre, por su confianza y apoyo siempre en mis estudios.

*A la Universidad de Córdoba, por la oportunidad que ha supuesto el
Máster Dual de Industria 4.0.*

*A mis compañeros Paco, Gaspar, Antonio y Abraham, por toda su ayuda
y guía durante la etapa de prácticas en la empresa Intarcon.*

Índice general

| | |
|---|------------|
| Índice general | I |
| Índice de figuras | III |
| Índice de tablas | V |
| Resumen | VII |
| Abstract | IX |
| Introducción, Objetivos, Metodología y Planificación | 1 |
| 1. Objetivos | 3 |
| 2. Metodología de trabajo | 3 |
| 3. Estructura de la memoria | 4 |
| 1. Estado del arte | 5 |
| 1.1. iPro y su pantalla | 7 |
| 1.1.1. Programación iPro | 8 |
| 1.1.2. Configuración de la pantalla | 10 |
| 1.2. Kiconex | 11 |
| 1.3. ESP32-PoE | 14 |
| 1.4. Protocolo Modbus | 15 |
| 2. Desarrollo del proyecto | 17 |
| 2.1. Programación iPro | 17 |
| 2.2. Diseño Pantalla para el control | 25 |

| | | |
|-----------------|--|------------|
| 2.3. | Elaboración de librerías en Kiconex | 28 |
| 2.4. | Programación dispositivo wireless | 32 |
| 2.4.1. | Librería para interfaz de configuración | 33 |
| 2.4.2. | Librería para maestro Modbus RTU | 37 |
| 2.4.3. | Librería para esclavo Modbus TCP | 38 |
| 2.4.4. | Código principal | 40 |
| 2.4.5. | Integración de todos los elementos del sistema | 42 |
| 3. | Pruebas y Resultados | 47 |
| 3.1. | Pruebas control UTA | 47 |
| 3.2. | Pruebas kiwi | 48 |
| 3.3. | Puesta en marcha de la instalación | 52 |
| 4. | Conclusiones y Trabajos Futuros | 55 |
| Anexo A. | Aplicación del protocolo Modbus | 59 |
| Anexo B. | Las Unidades de Tratamiento de Aire | 111 |
| Anexo C. | Programa Unidad de Tratamiento de Aire | 115 |
| Anexo D. | Registros Control Unidad de Tratamiento de Aire | 121 |
| Anexo E. | Manual funcionamiento Unidad de Tratamiento de Aire | 145 |

Índice de figuras

| | | |
|-------|---|----|
| 1.1. | Plano de equipos del supermercado | 5 |
| 1.2. | iPro Series | 7 |
| 1.3. | Aspecto tabla de variables en ISaGRAF | 9 |
| 1.4. | Ventanas pantalla iPro | 10 |
| 1.5. | Aspecto interfaz Visoprog | 11 |
| 1.6. | Red kiconex | 12 |
| 1.7. | Aspecto librería en la plataforma IoT | 13 |
| 1.8. | Olimex ESP32-PoE | 14 |
| 1.9. | Conversor UEXT-RS485 | 14 |
| 1.10. | Pines ESP32-PoE y conector UEXT | 15 |
| 2.1. | Modos de funcionamiento | 19 |
| 2.2. | Diseño de las ventanas de la pantalla | 28 |
| 2.3. | Añadir parámetro a librería kiconex - Pestaña general | 30 |
| 2.4. | Añadir parámetro a librería kiconex - Pestaña categoría | 31 |
| 2.5. | Añadir parámetro a librería kiconex - Pestaña unidades | 31 |
| 2.6. | Añadir parámetro a librería kiconex - Pestaña modbus | 31 |
| 2.7. | Añadir parámetro a librería kiconex - adición de metadato | 32 |
| 2.8. | Ventanas interfaz kiwi | 33 |
| 2.9. | Flujo de programa pasarela TCP-RTU | 40 |
| 2.10. | Flujo de programa del código principal de kiwi | 41 |
| 2.11. | Esquema conexión equipos supermercado | 42 |
| 2.12. | Creación instalación en plataforma IoT - Datos generales | 43 |
| 2.13. | Creación instalación en plataforma IoT - Hardware kibox | 43 |

| | |
|---|-----|
| 2.14. Adición de dispositivo a la instalación - Datos generales | 44 |
| 2.15. Adición de dispositivo a la instalación - Datos Modbus dispositivo | 44 |
| 2.16. Instalación supermercado creada | 45 |
| 3.1. Simulador de pruebas para iPro | 47 |
| 3.2. Desconexiones kiwi en red de Intarcon | 50 |
| 3.3. Desconexiones kiwi en red de router kiconex | 51 |
| 3.4. Instalación supermercado conectada y funcionando | 53 |
| B.1. Plano de partes de una UTA | 113 |
| C.1. Plano de partes de una UTA | 115 |
| C.2. Flujo de programa para gestión de ventiladores | 118 |
| C.3. Flujo de programa de gestión de la velocidad | 119 |
| C.4. Curvas de control automático de la velocidad en ventiladores, en función de la temperatura | 120 |
| C.5. Curva de control del humectador | 120 |

Índice de tablas

| | |
|--|-----|
| 1.1. Especificaciones de E/S para distintos modelos de iPro. | 8 |
| 2.1. Especificaciones E/S UTA. | 17 |
| 2.2. Parámetros de configuración de la UTA. | 19 |
| 3.1. Direcciones Modbus equipos supermercado. | 52 |
| C.1. Variables configuración control | 117 |

Resumen

En el presente proyecto se realiza la integración de los equipos de climatización y refrigeración de un supermercado en la plataforma IoT de kiconex.

Disponer de una red de este tipo permite la recolección de datos con los que conocer el comportamiento de los distintos equipos de la instalación frente a distintas condiciones, además de poder realizar la gestión remota de la misma. Es por ello que kiconex supone un paso más en la innovación tecnológica de cualquier empresa dedicada al sector del frío y clima, marcando una clara diferencia de la misma frente a su competencia.

En este proyecto también se lleva a cabo el desarrollo de un nuevo dispositivo de comunicación inalámbrica, que actúa como pasarela de las tramas Modbus TCP transmitidas vía WiFi, a tramas Modbus RTU vía RS485. Un dispositivo kiconex de este tipo facilita la integración de equipos, al evitar el uso de cables y disminuir la mano de obra con una instalación más limpia y rápida. Para este desarrollo se emplea hardware de la marca Olimex, basado en el chip ESP32.

Por último, se ha programado el control de la Unidad de Tratamiento de Aire del establecimiento, a través del software ISaGRAF, que emplea varios lenguajes de la norma IEC61131.

Palabras clave

kiconex, Modbus, ESP32, iPro, Dixell, Olimex

Abstract

This project involves the integration of the air conditioning and refrigeration equipment of a supermarket into the kiconex IoT platform.

Having this type of network allows the collection of data with which to know the behavior of the different equipment in the installation under different conditions, in addition to being able to perform a remote management. That is why kiconex is another step in the technological innovation of any company dedicated to the cold and climate sector, marking a clear difference from its competition.

In this project is also carried out the development of a new wireless communication device, which acts as a gateway from Modbus TCP frames transmitted via WiFi, to Modbus RTU frames via RS485. A kiconex device of this type facilitates equipment integration by avoiding the use of cables and reducing labor with a cleaner and faster installation. For this development, Olimex brand hardware is used, based on the ESP32 chip.

Finally, the control of the Air Handling Unit of the establishment has been programmed, through the ISaGRAF software, which uses several languages of the IEC61131 standard.

Keywords

kiconex, Modbus, ESP32, iPro, Dixell, Olimex

Introducción, Objetivos, Metodología y Planificación

El término Industria 4.0 implica la innovación a través de los nuevos conceptos tecnológicos como son el IoT (Internet of Things), el Big Data, Machine Learning, etc. Se trata de buscar lo que busca cualquier empresa: reducir costes e incrementar beneficios. Es por ello que cualquier empresa que no entienda este concepto, corre el riesgo de perder su cuota de mercado en un mundo tan competitivo.

La recolección de datos en la industria, y en concreto en el sector de la refrigeración, supone, en la mayoría de casos, comunicarse con PLCs que emplean el protocolo Modbus. Para ello, son cada vez más las empresas que crean sistemas gateway que traduzcan la información que viaja a través de Modbus, a un protocolo más rápido, innovador y propio de la Industria 4.0, en la mayoría de casos para intercambiar información con la nube. Es así como surgen productos y servicios completamente nuevos, como es el caso de kiconex [1].

Kiconex es un grupo de ingenieros que busca aplicar esos nuevos conceptos de la Industria 4.0 a los equipos de climatización y refrigeración, mediante un sistema de monitorización, supervisión y control. Un sistema de este tipo permite la adquisición de datos de los equipos comercializados, mediante cuyo análisis se puede estudiar el comportamiento de los mismos ante distintos ambientes y crear modelos predictivos, así como proporcionar un soporte técnico rápido y eficaz a los clientes. A esto hay que añadir la comodidad y tranquilidad del cliente, que desde la palma de su mano y desde cualquier parte del mundo, puede comprobar que sus instalaciones están funcionando correctamente, en tiempo real y sin necesidad de una supervisión presencial de las mismas.

Para todo ello, kiconex dispone de distintas opciones hardware [2] que se conectan a una instalación con un controlador previamente instalado e intercambian información con sus re-

gistros. La información recogida es enviada a un servidor en la nube y puede ser visualizada en una plataforma IoT [1]. Para realizar esta operación, el hardware dispone de puertos para la conexión de equipos de climatización y refrigeración mediante el protocolo de comunicaciones industrial Modbus [3]. Para ello, el software ha sido diseñado para poder transmitir todos los datos obtenidos por Modbus TCP o RTU a través de MQTT.

En estas situaciones se tiene muy en cuenta el concepto inglés retrofit, que consiste en la adaptación de la maquinaria ya existente a las nuevas tecnologías, sin modificar lo que ya hay o con unos mínimos cambios. Esto, en el entorno en que se mueve kiconex, supone acceder a máquinas en muchas ocasiones aisladas, a las que el tendido de cableado hace que el proceso sea difícil o impracticable. Es por ello, por lo que se necesita desarrollar un nuevo dispositivo que permita mejorar las máquinas a través de una comunicación Modbus inalámbrica.

Parte del contenido del proyecto que en este documento se desarrolla, es precisamente el diseño de un equipo wireless cuya aplicación concreta será la de dotar de conectividad inalámbrica a una serie de equipos frigoríficos de un supermercado. Las aplicaciones de un sistema de este tipo son múltiples, dada la facilidad con la que se podrían conectar equipos distribuidos, ampliando el alcance a esas zonas aisladas que se mencionan anteriormente. Un sistema inalámbrico supone una instalación rápida, limpia y económica, téngase en cuenta lo que supondría establecer comunicaciones por cable con, por ejemplo, unas islas frigoríficas situadas en la parte central de un supermercado. Significaría desmantelar dicho supermercado para la instalación de los canales y cables necesarios, implicando mayor mano de obra y posiblemente horas de cierre del establecimiento para poder realizar las labores necesarias.

El resto del proyecto desarrollado está compuesto de la parte de climatización del supermercado, del diseño de todas las comunicaciones y su integración con la plataforma IoT de kiconex. Aquí es donde entra en juego *kicontrol*, otro servicio que da kiconex para aquellos clientes que prefieren un control diseñado a medida para su instalación, con el añadido de que ese control va acompañado de un dispositivo kiconex que lo conecta con la nube. En el supermercado que se trata, las Unidades de Tratamiento de Aire (en adelante UTAs, Véase Anexo B, explicativo sobre las Unidades de Tratamiento de Aire) son nuevas pero carecen de controlador, por lo que se aprovechará el diseño de un control nuevo y a medida para facilitar su posterior implantación en la red de kiconex.

1. Objetivos

El objetivo de este proyecto es la implantación de una red IoT para la climatización y refrigeración de un supermercado, usando la plataforma kiconex.

El objetivo principal puede dividirse en los siguientes subobjetivos:

1. Realizar un análisis de todos los puntos a tener en cuenta en la implantación del sistema.
2. Programación de un controlador para una o varias UTAs.
3. Elaboración de las librerías necesarias para la integración de todos los elementos en kiconex.
4. Desarrollo de un nuevo sistema de comunicación inalámbrico vía WiFi.
5. Integración de todos los elementos del sistema.

2. Metodología de trabajo

Para alcanzar los objetivos planteados se ha seguido la siguiente metodología de trabajo:

1. Analizar cada uno de los componentes del supermercado, los controles empleados, como se comunicarán entre ellos, y qué es necesario para diseñar todo el sistema y sus componentes.
2. Estudio del funcionamiento del control programable *iPro* [4], de Dixell [5].
3. Estudiar el funcionamiento del software necesario para la programación del *iPro* y su pantalla: *ISaGRAF* y *Visoprog*.
4. Desarrollo de las librerías necesarias para integrar cada uno de los controladores programables en la red de kiconex [6].
5. Diseño completo de un dispositivo de comunicación inalámbrica basado en ESP32, con un sistema que facilite su configuración e implantación por parte de los instaladores.
6. Pruebas de funcionamiento de los nuevos desarrollos realizados, resultados y conclusiones.

7. Integración de los elementos y puesta en marcha.
8. Documentar todo el proyecto realizado.

3. Estructura de la memoria

Para la documentación del proyecto, la memoria se ha organizado en los siguiente capítulos:

- **Capítulo 1. Estado del Arte:** Se describe el estado del arte de cada parte del proyecto y los elementos necesarios para llevarla a cabo.
 - **1. Kiconex:** Se explica cómo se estructura una red con kiconex.
 - **2. iPro y Pantalla:** Se presenta el software usado para la programación del control y la pantalla de visualización, además de las librerías y códigos de los que se parte.
 - **3. ESP32-PoE:** Se presenta el hardware empleado para la programación del dispositivo kiconex wireless, así como las librerías que han ayudado a su desarrollo.
- **Capítulo 2. Desarrollo del proyecto:** En este capítulo se desarrolla el proyecto de forma estructurada:
 - **1. Programación iPro.**
 - **2. Diseño Pantalla para el control.**
 - **3. Elaboración de librerías en kiconex.**
 - **4. Programación dispositivo wireless.**
- **Capítulo 3. Pruebas y resultados:** Se describen las pruebas realizadas para verificar el correcto funcionamiento del control de la UTA y del dispositivo inalámbrico. También se comentan los resultados obtenidos.
- **Capítulo 4. Puesta en marcha:** Se describe el modo de instalación, la comunicación e integración de cada elemento y el resultado final.
- **Capítulo 5. Conclusiones y Trabajos Futuros:** Se indican las conclusiones extraídas del trabajo realizado, aportando nuevas y posibles líneas de desarrollo.

Capítulo 1

Estado del arte

El presente proyecto parte de un pedido realizado por un cliente que pretende comunicar con kiconex los distintos equipos de un supermercado: muebles frigoríficos y máquinas de climatización. La mayoría de estos equipos dispone de un controlador propio, sin embargo, en el caso de la climatización, se ha solicitado el diseño de un control a medida para el funcionamiento de una UTA. Aunque no se dispone de un plano específico del supermercado y su equipamiento, la Figura 1.1 es un buen ejemplo. La única diferencia está en la climatización.

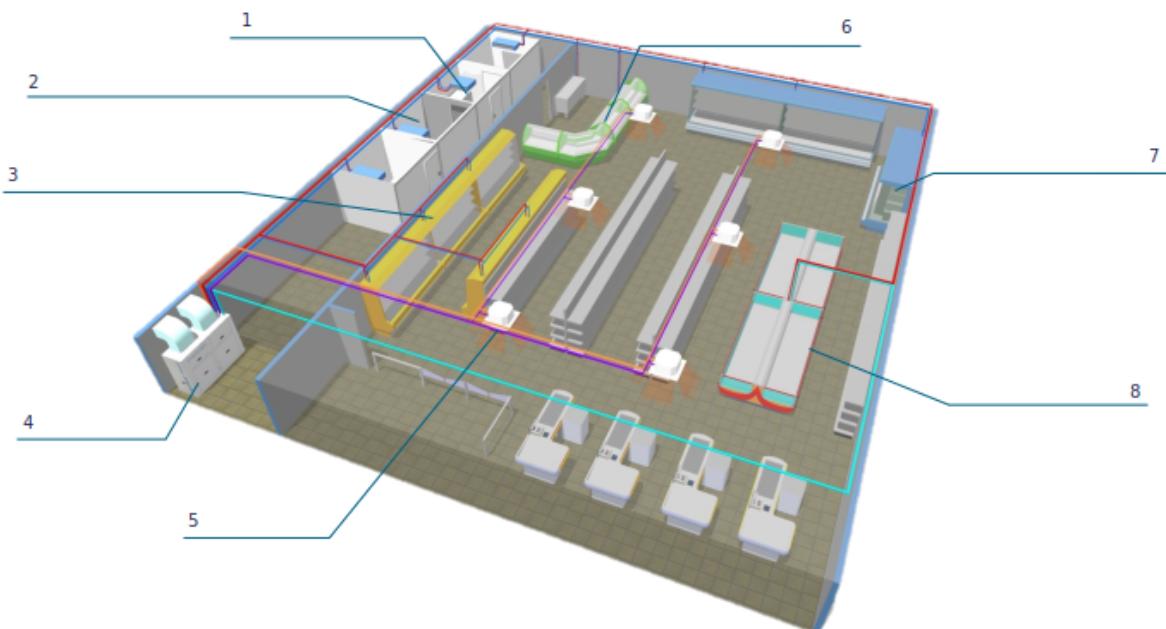


Figura 1.1: Plano de equipos del supermercado

1. Obradores.
2. Cámaras frigoríficas.
3. Murales de lácteos.
4. Bomba de calor.
5. Climatización*.
6. Vitrinas expositoras.
7. Semimurales de carne, frutas y verduras.
8. Islas de congelados.

Kiconex emplea en sus desarrollos controladores de Dixell [5], en concreto los modelos *IPG208* e *IPG215* de *iPro* [4], con los módulos de expansión *IPX206* e *IPX215* [4]. El modelo a emplear dependerá de las especificaciones de entradas y salidas de la UTA que aparece en la Figura 1.1. A continuación, en el apartado 1 de este capítulo se describe el software necesario para la programación de un control de este tipo.

El dispositivo inalámbrico a diseñar, mencionado en el apartado de introducción anterior comunicará los muebles frigoríficos: Murales de lácteos, vitrinas expositoras, semimurales e islas de congelados. El hardware empleado, descrito en el apartado 3 de este capítulo, se basa en el chip ESP32 [7]. Para entender el papel de este dispositivo en la red, es necesario conocer el funcionamiento de una instalación desde el punto de vista de kiconex, atendiendo a cómo se integra cada elemento. El apartado 2 de este capítulo se dedica por completo a describir la estructura de un sistema basado en kiconex.

*En la Figura 1.1 aparecen unidades tipo fancoil, pero lo que en realidad hay es una serie de conductos conectados a una UTA central

1.1. iPro y su pantalla



Figura 1.2: iPro Series

iPro (Figura 1.2) es la gama de controladores programables ofrecida por Dixell. La gama consta de controladores programables, ampliaciones de E/S, controladores para válvulas electrónicas e interfaces gráficas adaptadas para cubrir cualquier tipo de aplicación en el sector del aire acondicionado, el sector de la refrigeración y cualquier área relativa. Algunas de sus especificaciones son:

- Alimentación a 24Vac/dc.
- Microprocesador ARM9 de 32 bits (200MHz).
- El programa y los parámetros se almacenan en una memoria flash permanente. No se pierden datos en caso de fallo de alimentación.
- Servidor web interno.
- Hasta 80 Mb de memoria flash, dependiendo del modelo.

- Entradas y salidas completamente configurables.
- Conexiones:
 - Puerto Ethernet.
 - Puerto USB.
 - Conexión dedicada para un display LCD.
 - CANBus.
 - RS485 Master.
 - RS485 Slave.

Los modelos se diferencian en el tamaño (10 DIN o 4 DIN) y en el número de entradas y salidas (analógicas y digitales). La Tabla 1.1 recoge las especificaciones de los modelos empleados por kiconex.

| | Controlador | | Módulo de Expansión | |
|----------------------------------|-------------|--------|---------------------|--------|
| | IPG208 | IPG215 | IPX206 | IPX215 |
| Entradas analógicas | 6 | 10 | 7 | 10 |
| Salidas analógicas | 4 | 6 | 3 | 6 |
| Entradas digitales | 11 | 20 | 3 | 20 |
| Salidas digitales (Relés) | 8 | 6 | 8 | 15 |

Tabla 1.1: Especificaciones de E/S para distintos modelos de iPro.

Dixell dispone de dos modelos de displays compatibles con el *iPro*: *VGIPG* y *VTIPG* [8]. Para el diseño de dichas pantallas, se emplea el software *Visoprog* [9] de Dixell, que importa las variables creadas en la programación del controlador, para poder configurar en la pantalla la interacción con las mismas.

1.1.1. Programación iPro

Para la programación del *iPro* se emplea el software *ISaGRAF* [10], ya que ofrece un entorno de desarrollo estándar e internacional, soportando varios lenguajes de programación diferentes según las normas IEC61131. Se trata de un software usado en todo el mundo y que

también permite simular el sistema programado. Los estándares de programación soportados son:

- **(LD)** Escalera
- **(FBD)** Diagrama a Bloques
- **(SFC)** Tabla de Funciones Secuenciales
- **(ST)** Texto Estructurado
- **(IL)** Lista de Instrucciones
- **(FC)** Diagrama de flujo

Para comenzar a realizar un programa se parte de una plantilla ya preparada previamente por kiconex para configurar entradas, salidas, parámetros, alarmas, avisos, etc.. Es por ello que primero es necesario conocer las especificaciones del sistema a programar. En el código es necesario distinguir lo que es la configuración de entradas/salidas de lo que es el valor leído en las mismas.

A cada variable de parámetro, alarma, aviso, estado y entradas/salidas se le asigna una dirección de registro, como se puede ver en la Figura 1.3. Lo habitual es destinar distintos rangos de direcciones a cada tipo de variable.

| ENTRADAS_SALIDAS | | | | | | | |
|-----------------------|------|------------|-----------|---------------|-----------|------------|---|
| Nombre | Tipo | Comentario | Dirección | Valor inicial | Dimensión | No volátil | 0 |
| Pb_T_deposito_1 | DINT | | 1000 | | | No | |
| Pb_T_deposito_2 | DINT | | 1001 | | | No | |
| Pb_T_deposito_3 | DINT | | 1002 | | | No | |
| Pb_T_deposito_4 | DINT | | 1003 | | | No | |
| Pb_T_ACS_impul | DINT | | 1004 | | | No | |
| Pb_T_ACS_retor | DINT | | 1005 | | | No | |
| Pb_T_REC_impul | DINT | | 1006 | | | No | |
| Pb_T_REC_retor | DINT | | 1007 | | | No | |
| AO_TEMP_CAL | DINT | | 1080 | | | No | |
| DI_on/off | BOOL | | 1100 | | | No | |
| DI_cal | BOOL | | 1101 | | | No | |
| DI_cal_status | BOOL | | 1102 | | | No | |
| DI_contador | BOOL | | 1103 | | | No | |
| DI_B01 | BOOL | | 1104 | | | No | |
| DI_B02 | BOOL | | 1105 | | | No | |
| DI_B03 | BOOL | | 1106 | | | No | |
| DI_B04 | BOOL | | 1107 | | | No | |
| RELE_VALV_ACS | BOOL | | 1200 | | | No | |
| RELE_VALV_EQUILIBRADO | BOOL | | 1201 | | | No | |
| RELE_RESIS_DEPOS_ACS | BOOL | | 1202 | | | No | |

Figura 1.3: Aspecto tabla de variables en ISaGRAF

1.1.2. Configuración de la pantalla

Para la configuración de la pantalla, se emplea *Visoprog*, un software ofrecido por la misma marca Dixell, especialmente para su uso con los modelos *VGIPG* e *VTIPG*. En este caso también se ha partido de una plantilla previamente preparada por kiconex con las distintas pantallas que necesita un control como el *iPro*:

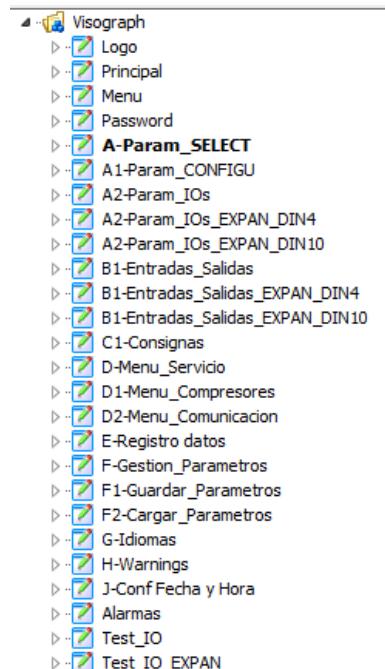


Figura 1.4: Ventanas pantalla iPro

Para usar este programa, en primer lugar se importa un fichero de registros generados por *ISaGRAF*. Así es como *Visoprog* los conoce y permite que el usuario interactúe con los mismos en el proceso de navegación por la pantalla, habiendo configurado previamente dicha interacción.

Visoprog tiene también una sección destinada a textos estándar, así se puede cambiar un texto en un solo sitio, y no en los cincuenta sitios en los que se está empleando. La Figura 1.5 representa el aspecto de la interfaz del programa.

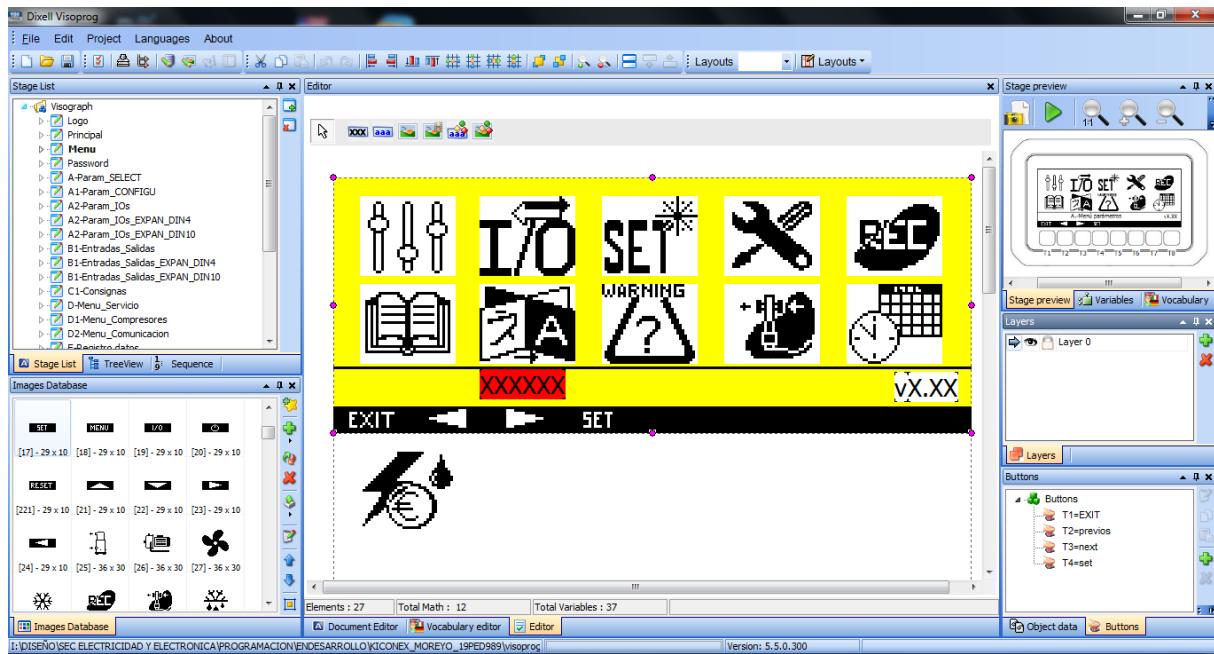


Figura 1.5: Aspecto interfaz Visoprog

1.2. Kiconex

Como ya se ha mencionado en la introducción, kiconex es una plataforma de supervisión y control para equipos de climatización y frío industrial. Entre sus funciones se encuentra:

- **Almacenamiento en la nube** de datos de temperatura, estado de funcionamiento, presiones, etc.
- **Gráficas** para visualizar la evolución de los datos en el tiempo, permitiendo compararlos.
- **Control remoto**: marcha/paro, cambio de consigna, etc.
- **Reglas** con programaciones horarias o acciones frente a condiciones.
- **Diagramas** para introducir un plano del edificio y sobre ese plano ver la temperatura de las salas y, por ejemplo, encender y apagar.
- **Alarmas** con posibilidad recibir alertas en caso de fallo del equipo.

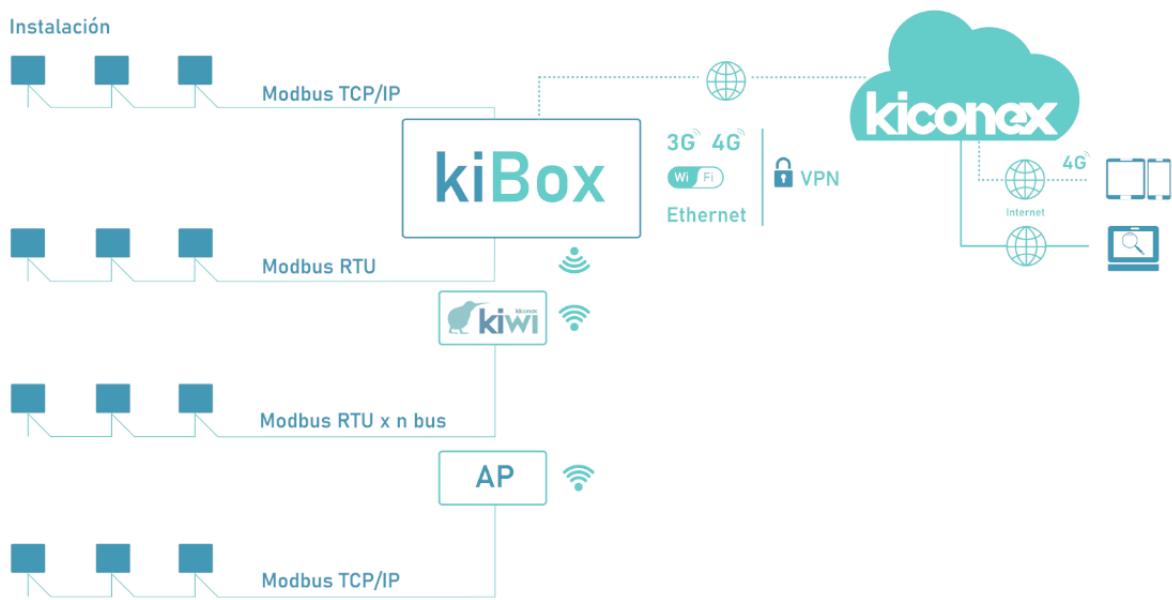


Figura 1.6: Red kiconex

Para su funcionamiento, kiconex se compone de los siguiente elementos:

1. **Servidor en la nube**, donde se almacenan los datos.
2. **Plataforma de visualización y control**.
3. **Kibox**: hardware que actúa como pasarela entre la nube y el equipo de frío o clima. Recibe mensajes de la nube a través de MQTT, los procesa y los envía en formato Modbus al equipo final. Este último, recibe el mensaje, lo procesa y envía una respuesta que sigue el camino inverso hasta que la nube recibe el dato solicitado.
4. **Equipos** de climatización o frío industrial: emplean el estándar industrial Modbus. La mayoría emplean Modbus RTU pero kiconex también funciona a través de Modbus TCP. Estos equipos reciben los mensajes del *kibox* y le envían una respuesta.
5. **Kiwi**: aún no está en venta, ya que se trata del dispositivo inalámbrico que se ha desarrollado en este proyecto, pero el objetivo es su actuación como esclavo Modbus, recibiendo mensajes del maestro *kibox* a través de TCP-IP. *Kiwi* retransmite estos mensajes actuando como maestro, a los dispositivos de frío y clima (esclavos). Es decir, *kiwi* es un maestro-esclavo: maestro de cara a los equipos y esclavo de cara al *kibox*.

La plataforma de supervisión, para poder comunicarse con los controles de los equipos de frío y clima, necesita conocer sus registros. Para ello, en la plataforma existe un apartado llamado "librerías", en el cual se pueden ver y crear los mapeados de registros de los controles que se necesiten. En la Figura 1.7 se puede ver el aspecto de una librería en la plataforma.

| Nombre | Protocolo | Grupo | Intervalo | Visibilidad |
|---|---|-------|-----------|---|
| Transductor alta presión C1 (P) Transductor de alta presión circuito 1 (Presión) | 1 (0x1) | | 0 | <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> |
| Transductor alta presión C1 (T) Transductor de alta presión circuito 1 (Temperatura) | 2 (0x2) | | 0 | <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> |
| Lectura Escritura Val. min.: 32788 Val. máx.: 32787 U. medida: °C Notas | Func. lectura: 3 Func. escritura: 0 Offset: 0.1 Máscara: 0 Valor: 0 Longitud: 16 | | | |
| Transductor alta presión C2 (P) Transductor de alta presión circuito 2 (Presión) | 3 (0x3) | | 0 | <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> |
| Transductor alta presión C2 (T) Transductor de alta presión circuito 2 (Temperatura) | 4 (0x4) | | 0 | <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> |

Figura 1.7: Aspecto librería en la plataforma IoT

Para el caso del *iPro*, en el cual el programa se diseña desde cero por kiconex, el mapeado de registros se extrae directamente del software *ISaGRAF*, y se introduce en una nueva librería en la plataforma. Cuando se trata de otros controles, es el cliente quien lo consigue a través del fabricante. kiconex, gracias a su trayectoria, ya ha recopilado una gran cantidad de librerías para multitud de controles, lo que facilita mucho el proceso.

1.3. ESP32-PoE

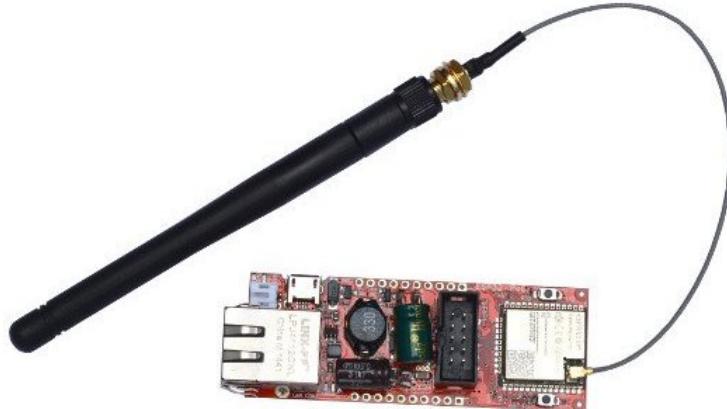


Figura 1.8: Olimex ESP32-PoE

El hardware empleado para el desarrollo del kiconex inalámbrico (*kiwi*) se basa en el chip ESP32 [7]. El modelo empleado es el Olimex ESP32-PoE (Figura 1.8). Olimex [11] es una empresa de Bulgaria líder en fabricación electrónica para el mercado integrado. Su modelo ESP32-PoE ha sido elegido por disponer de un puerto UEXT desde el que se puede obtener una interfaz RS-485 a través de un conversor [12] (Figura 1.9), para la comunicación a través de Modbus RTU. También dispone de puerto ethernet que puede usarse bien como conexión a internet vía ethernet o bien para conectar un equipo por modbus TCP. La Figura Figura 1.10 representa los pines de entrada y salida de la placa y del conector UEXT.



Figura 1.9: Conversor UEXT-RS485

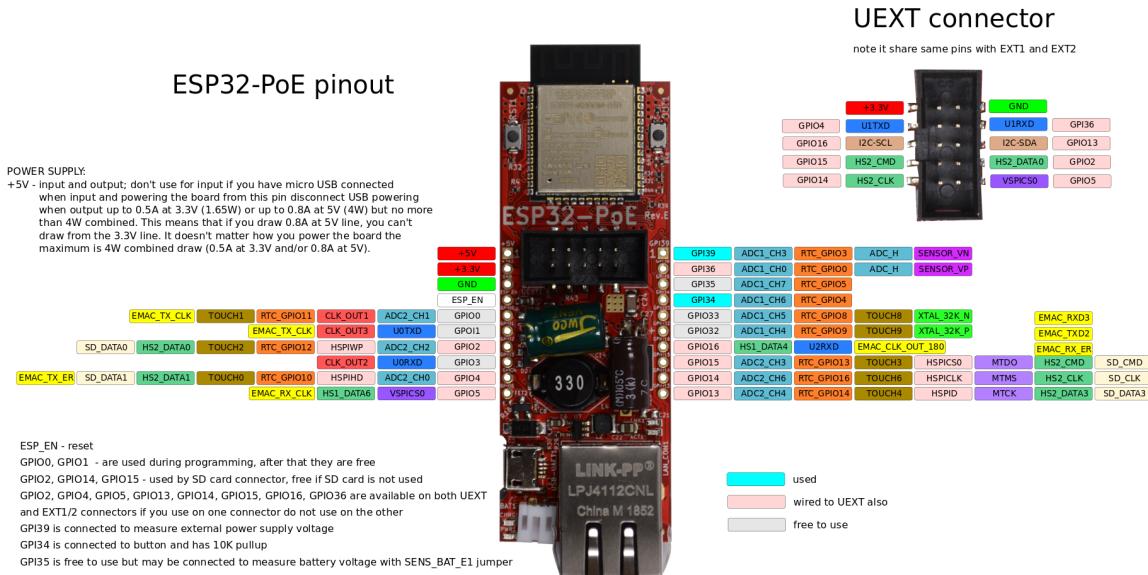


Figura 1.10: Pines ESP32-PoE y conector UEXT

Para la elaboración del código de programación del kiwi, se va a aprovechar parte del código empleado en las siguientes librerías ya existentes en github **githubweb**:

- Librería Modbus RTU, de Samuel Marco [13]: librería necesaria para enviar mensajes vía Modbus RTU.
 - Librería WiFiManager, de Khoi Hoang [14]: Servirá de punto de partida para diseñar una interfaz de configuración de *kiwi*.

1.4. Protocolo Modbus

Hasta este punto del documento se ha mencionado varias veces al protocolo Modbus y sus variantes: Modbus TCP y Modbus RTU, por lo que es en esta sección, y dada la importancia de este protocolo en el proyecto, donde se va a exponer de forma muy resumida sus características principales:

- Distingue entre el dispositivo que solicita la información (**maestro**) y los dispositivos que proporcionan dicha información (**esclavos**). Esto significa que un esclavo no puede

ofrecer información si no se le ha solicitado antes.

- Para la comunicación, cada esclavo dispone de una dirección única de **8 bits**, por lo que un maestro puede tener hasta **254** esclavos.
- Se puede distinguir entre **Modbus TCP**, para comunicaciones vía ethernet a través de la red LAN, y **Modbus RTU**, para comunicaciones serie a través de RS232/RS485.
- Estructura del mensaje: el mensaje siempre se compone de: *dirección del esclavo*, código de *función Modbus*, *dirección de registro* y *cantidad de registros* a leer. En el caso del Modbus RTU, se añaden dos bytes que contienen un *código CRC* para la detección de errores.
- Funciones: dispone de distintas funciones en función de la naturaleza de la información compartida o de los registros con los que se desea compartir información:
 - **01 (0x01) Read Coils:** lee de 1 a 2000 bits de un dispositivo remoto.
 - **02 (0x02) Read Discrete Inputs:** lee de 1 a 2000 bits de registro en un dispositivo remoto.
 - **03 (0x03) Read Holding Registers:** lee de 1 a 125 registros de 16 bits continuos en un dispositivo remoto.
 - **04 (0x04) Read Input Registers:** lee de 1 a 125 registros de 16 bits continuos en un dispositivo remoto.
 - **05 (0x05) Write Single Coil:** escribe un solo bit de registro en el dispositivo remoto.
 - **06 (0x06) Write Single Register:** escribe un solo registro de 16bits en el dispositivo remoto.
 - **15 (0x0F) Write Multiple Coils:** escribe de 1 a 2000 bits de registro consecutivos en un dispositivo remoto.
 - **16 (0x10) Write Multiple registers:** escribe de 1 a 123 registros 16 bits consecutivos en un dispositivo remoto.

En el Anexo A, extraído directamente de la web de la Modbus Organization [3] se describe de forma detallada cada una de las funciones Modbus, y que información exacta contiene cada campo (byte) del mensaje.

Capítulo 2

Desarrollo del proyecto

En las siguientes secciones se detalla de forma estructurada el desarrollo de cada una de las partes que componen este proyecto.

2.1. Programación iPro

Como ya se ha explicado en los capítulos anteriores, el destino del iPro es la programación de la UTA existente en el supermercado. En el Anexo B se describe detalladamente qué es una UTA y todos los elementos de los que se puede componer. Para el caso que nos atañe, la Tabla 2.1 recoge las especificaciones concretas del control.

| GRUPO | Entrada analógica | Salida analógica | Entrada digital | Salida digital |
|-------------------------|--|--|--|-------------------|
| Ventiladores | | Ventilador impulsión Ventilador retorno | Seguridad Ventilador impulsión Seguridad Ventilador retorno | |
| Filtros | | | Filtro entrada Filtro salida Filtro retorno | |
| Humectador | Humedad de retorno Humedad exterior | | Indicador nivel agua | ON/OFF Humectador |
| Intercambiador de calor | Temp. impulsión agua Temp. retorno agua | | | Válvula de agua |
| E/S de aire | Temp. impulsión aire Temp. retorno aire Temp. extracción aire Temp. aire exterior | | | |

Tabla 2.1: Especificaciones E/S UTA.

La lógica de funcionamiento sigue las siguientes pautas:

- **Compuertas:** No existen compuertas regulables.
- **Filtros:** La señal digital de los filtros indica si están sucios o no.
- **Ventiladores:**
 - La velocidad de los ventiladores se regula mediante una señal 0-10V, dicha velocidad debe ser la misma en cada momento.
 - La señal digital en los ventiladores indica fallo en los mismos.
 - La velocidad debe ser regulable de forma automática o de forma manual:
 - La regulación automática ajusta la velocidad del ventilador en función de la diferencia de temperatura con la consigna.
 - La regulación manual de la velocidad se establece en tres rangos parametrizables: alta, media y baja.
- **Humectador:**
 - Se establece una consigna de humedad.
 - A partir de las mediciones de humedad del aire que entra y sale se activa la humectación o no.
 - El indicador de nivel del humectador indica si éste está lleno o no.
- **Intercambiador de calor:**
 - Debe existir un aviso que indique si el fluido está a una temperatura acorde para alcanzar la consigna deseada.
 - Mientras la UTA esté en funcionamiento, la válvula que lleva el fluido hasta el intercambiador debe estar abierta.
- **E/S de aire:**
 - La temperatura de retorno es la usada como referencia para el control de temperatura del aire.

- El control de temperatura se establece según el modo de funcionamiento: verano (frío) o invierno (calor).
- Las curva de funcionamiento para cada uno de los modos es la de la Figura 2.1:

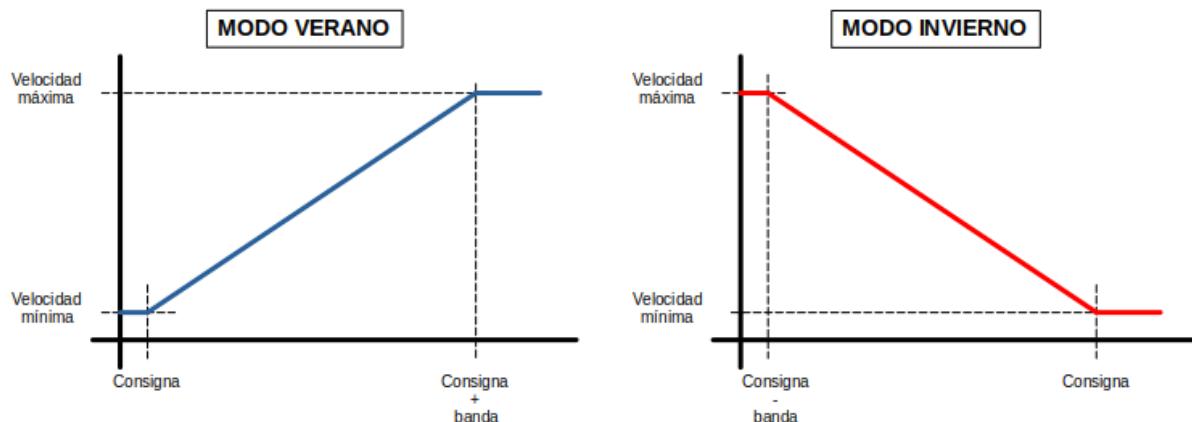


Figura 2.1: Modos de funcionamiento

El Anexo C recoge parte del programa realizado para el funcionamiento de la UTA. Los parámetros necesarios para la configuración del controlador son los de la Tabla 2.2.

Tabla 2.2: Parámetros de configuración de la UTA.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|-------------------------|--------------|---|
| Configuración Equipo | CNF01 | Habilitar módulos de expansión (DIN4/DIN10) |
| | HUM01 | Establecer consigna de humedad en %HR |
| | HUM02 | Establecer banda para control de humedad, en %HR |
| | TMP01 | Establecer consigna de temperatura en °C |
| | TMP02 | Establecer banda de temperatura para la regulación en el modo invierno (°C) |
| | TMP03 | Establecer banda de temperatura para la regulación en el modo verano (°C) |

Tabla 2.2 continua en la página siguiente...

Tabla 2.2 ...continuación de la página anterior.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|--------------|------------------|---|
| Ventiladores | FAN01 | Establecer valor de velocidad baja |
| | FAN02 | Establecer valor de velocidad media |
| | FAN03 | Establecer valor de velocidad alta |
| | FAN04 | Establecer valor de velocidad mínima para la regulación automática |
| | FAN05 | Establecer valor de velocidad máxima para la regulación automática |
| | FAN06 | Establecer banda de velocidad para la regulación automática |
| | FAN07 | Tiempo de retardo de activación de alarma en ventilador de impulsión en segundos |
| | FAN08 | Tiempo de retardo de desactivación de alarma en ventilador de impulsión en segundos |
| | FAN09 | Número de alarmas en el ventilador de impulsión hasta bloqueo |
| | FAN10 | Tiempo de retardo de activación de alarma en ventilador de retorno en segundos |
| | FAN11 | Tiempo de retardo de desactivación de alarma en ventilador de retorno en segundos |
| | FAN12 | Número de alarmas en el ventilador de retorno hasta bloqueo |
| Filtros | FIL01 | Retardo de activación de aviso en filtro de entrada de aire en segundos |
| | FIL02 | Retardo de desactivación de alarma en filtro de entrada de aire en segundos |
| | FIL03 | Retardo de activación de aviso en filtro de impulsión de aire en segundos |

Tabla 2.2 continua en la página siguiente...

Tabla 2.2 ...continuación de la página anterior.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|--------------------|--------------|--|
| | FIL04 | Retardo de desactivación de alarma en filtro de impulsión de aire en segundos |
| | FIL05 | Retardo de activación de aviso en filtro de retorno de aire en segundos |
| | FIL06 | Retardo de desactivación de alarma en filtro de retorno de aire en segundos |
| Entradas digitales | DIG01 | INV(FALSE)/DIR(TRUE): polaridad inversa o directa 0=No utilizado: entrada sin función asociada |
| | DIG02 | 1=ON-OFF: Interruptor de ON/OFF remoto |
| | DIG03 | 2=DI. Ventilador Impulsión: seguridad ventilador impulsión |
| | DIG04 | 3=DI. Ventilador retorno: seguridad ventilador retorno |
| | DIG05 | 4=DI. Nivel Humectador: seguridad nivel humectador |
| | DIG06 | 5=DI. Filtro entrada aire: seguridad filtro entrada aire |
| | DIG07 | 6=DI. Filtro impulsión aire: seguridad filtro impulsión aire |
| | DIG08 | 7=DI. Filtro retorno aire: seguridad filtro retorno aire |
| | DIG09 | |
| | DIG10 | |
| | DIG11 | |
| | DIG12 | |
| | DIG13 | |
| | DIG14 | |
| | DIG15 | |
| | DIG16 | |
| | DIG17 | |
| | DIG18 | |
| | DIG19 | |
| | DIG20 | |
| | DIG21 | |
| | DIG22 | |

Tabla 2.2 continua en la página siguiente...

Tabla 2.2 ...continuación de la página anterior.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|-------------------|------------------|-------------------------------|
| | DIG23 | |
| | DIG24 | |
| | DIG25 | |
| | DIG26 | |
| | DIG27 | |
| | DIG28 | |
| | DIG29 | |
| | DIG30 | |
| | DIG31 | |
| | DIG32 | |
| | DIG33 | |
| | DIG34 | |
| | DIG35 | |
| | DIG36 | |
| | DIG37 | |
| | DIG38 | |
| | DIG39 | |
| | DIG40 | |
| | DIG41 | |
| | DIG42 | |
| | DIG43 | |
| Sondas analógicas | PBS01 | 0=no usado: sonda sin función |
| | PBS02 | 1=Ta impulsión aire |
| | PBS03 | 2=Ta retorno aire |
| | PBS04 | 3=Ta impulsión agua |
| | PBS05 | 4=Ta retorno agua |
| | PBS06 | 5=Ta extracción aire |
| | PBS07 | 6=Ta aire extraído |

Tabla 2.2 continua en la página siguiente...

Tabla 2.2 ...continuación de la página anterior.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|--------------|------------------|---|
| | PBS08 | 7=Ta extracción aire |
| | PBS09 | 8=Humedad de retorno |
| | PBS10 | 9=Humedad exterior |
| | PBS11 | |
| | PBS12 | |
| | PBS13 | |
| | PBS14 | |
| | PBS15 | |
| | PBS16 | |
| | PBS17 | |
| | PBS18 | |
| | PBS19 | |
| | PBS20 | |
| | PBS21 | |
| | PBS22 | |
| | PBS23 | |
| | PBS24 | |
| | PBS25 | |
| | PBS26 | |
| | PBS27 | |
| Rele salida | RLO01 | 0=No usado:relé sin función asociada |
| | RLO02 | 1=Válvula de 3 vías: activación válvula de agua |
| | RLO03 | 2=Humectador: activación humectador |
| | RLO04 | 3=Alarma: activación alarma |
| | RLO05 | |
| | RLO06 | |
| | RLO07 | |
| | RLO08 | |

Tabla 2.2 continua en la página siguiente...

Tabla 2.2 ...continuación de la página anterior.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|-------|--------------|-------------|
| | RLO09 | |
| | RLO10 | |
| | RLO11 | |
| | RLO12 | |
| | RLO13 | |
| | RLO14 | |
| | RLO15 | |
| | RLO16 | |
| | RLO17 | |
| | RLO18 | |
| | RLO19 | |
| | RLO20 | |
| | RLO21 | |
| | RLO22 | |
| | RLO23 | |
| | RLO24 | |
| | RLO25 | |
| | RLO26 | |
| | RLO27 | |
| | RLO28 | |
| | RLO29 | |
| | RLO30 | |
| | RLO31 | |
| | RLO32 | |
| | RLO33 | |
| | RLO34 | |
| | RLO35 | |
| | RLO36 | |

Tabla 2.2 continua en la página siguiente...

Tabla 2.2 ...continuación de la página anterior.

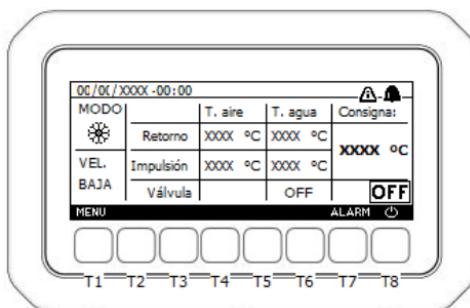
| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|--------------------|--------------|---|
| Salidas analógicas | ANA01 | 0=No utilizado: salida sin función asociada |
| | ANA02 | 1=Vel. Ventilador imp.: salida analógica para regular la velocidad del ventilador de impulsión de aire |
| | ANA03 | |
| | ANA04 | 2=Vel. Ventilador ret.: salida analógica para regular la velocidad del ventilador de retorno de aire |
| | ANA05 | |
| | ANA06 | |
| | ANA07 | |
| | ANA08 | |
| | ANA09 | |
| | ANA10 | |
| | ANA11 | |
| | ANA12 | |
| | ANA13 | |
| | ANA14 | |
| | ANA15 | |

2.2. Diseño Pantalla para el control

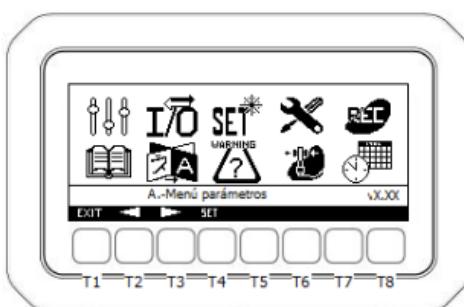
Desde el programa Visoprog, donde se diseña la pantalla, se importan las variables del programa creado con el software ISaGRAF. Para ello, Visoprog tiene dos opciones de importación: directamente desde un proyecto ISaGRAF o desde una hoja excel o fichero csv. El diseño realizado se compone de las ventanas de la Figura 2.2, su funcionamiento se describe en el manual del Tabla E:



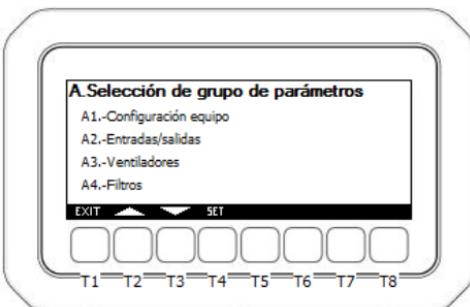
(a) Logo de inicio



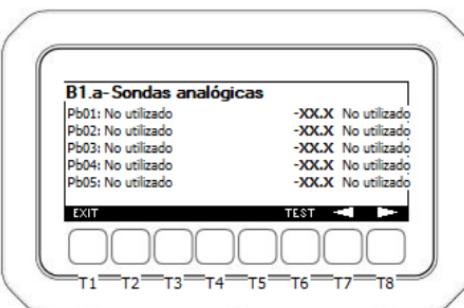
(b) Pantalla principal



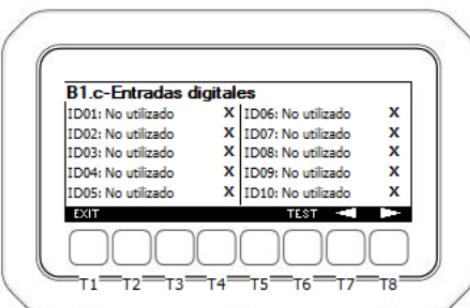
(c) Menú



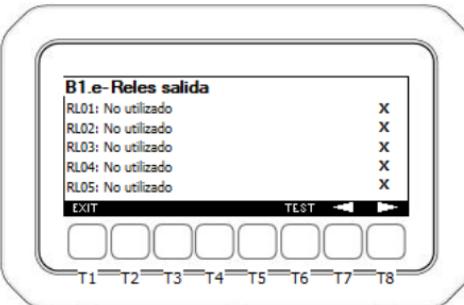
(d) Menú de parámetros



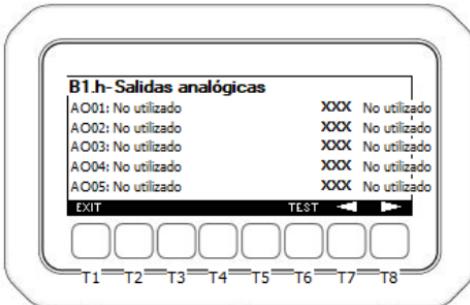
(e) Configuración de sondas analógicas



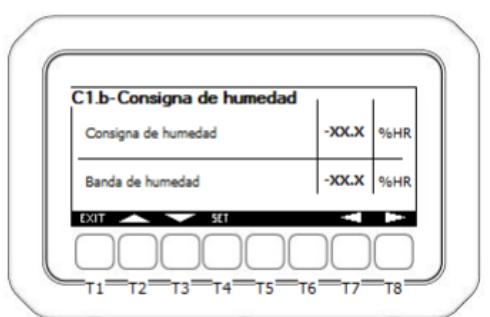
(f) Configuración de entradas digitales



(g) Configuración de relés de salida



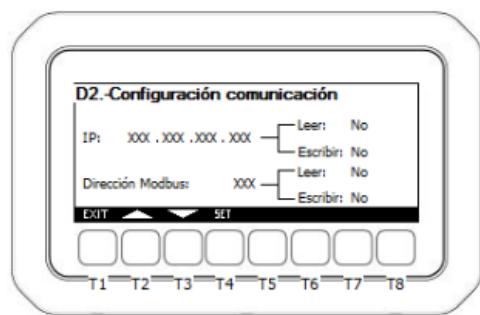
(h) Configuración de sondas analógicas



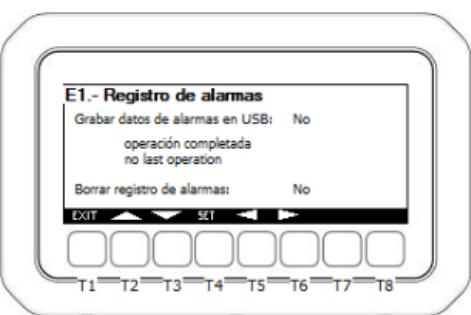
(i) Consigna de humedad



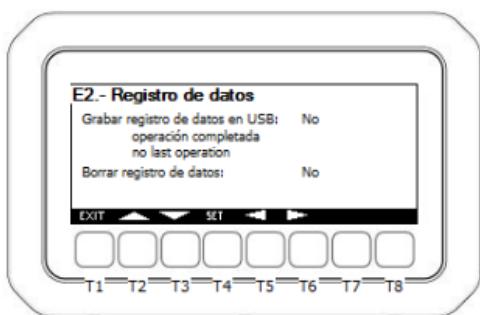
(j) Menú de servicio



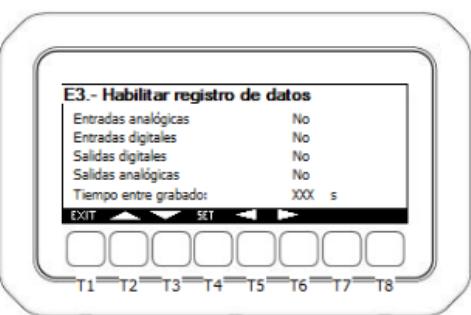
(k) Configuración de comunicación



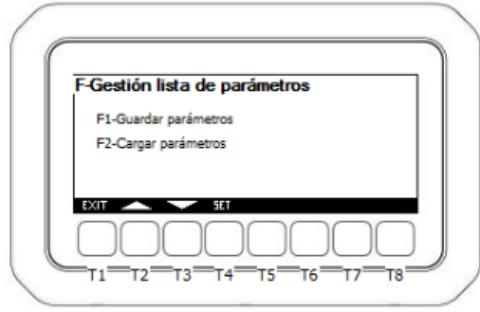
(l) Registro de alarmas



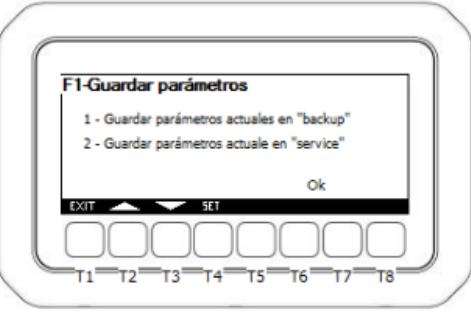
(m) Registro de datos



(n) Habilitar el registro de datos



(ñ) Menú de carga/descarga de parámetros



(o) Guardar parámetros

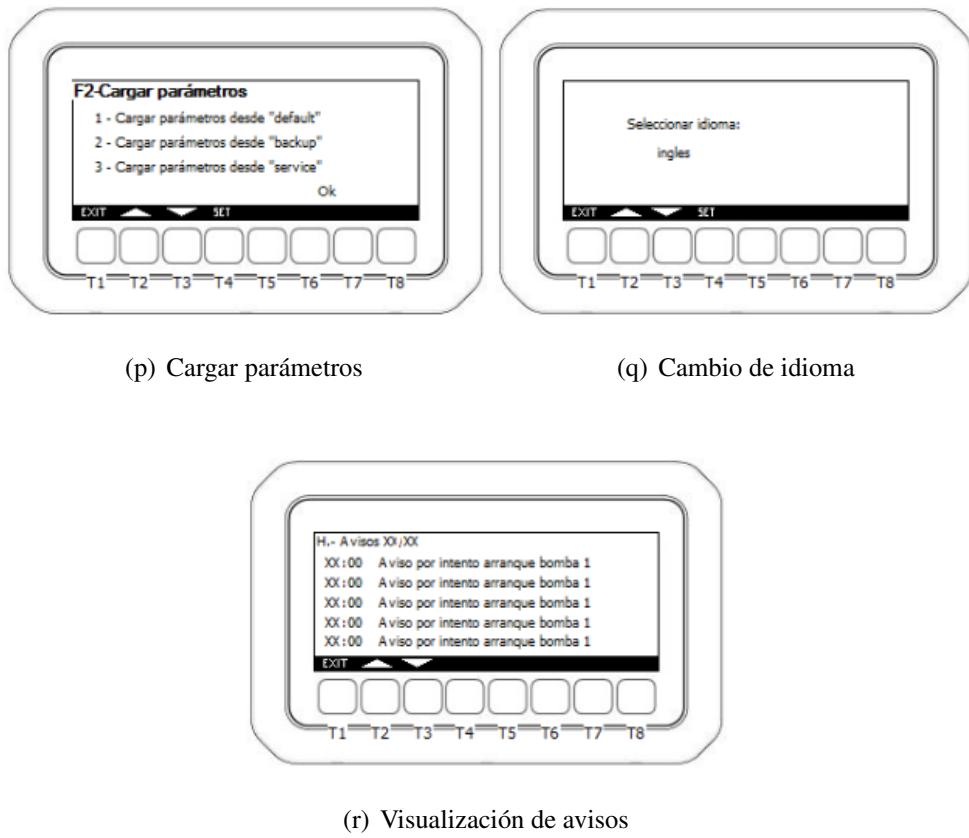


Figura 2.2: Diseño de las ventanas de la pantalla

2.3. Elaboración de librerías en Kiconex

En el capítulo anterior se explicaba que son necesarias librerías que recopilen los registros de cada control para la integración del mismo en la plataforma IoT de kiconex. Los controles que componen la instalación del supermercado son los siguientes:

- **Dixell XW60VS:** para cámaras frigoríficas y obradores.
- **Dixell XR-75:** para murales, semimurales, vitrinas e islas de congelados.
- **Danfoss AK-PC-551:** para la central de refrigeración.
-
- **Keyter Climanager Aire-Aire:** para la bomba de calor.
- **Nuevo control basado en iPro IPG208:** el control programado en el apartado anterior, destinado a la UTA.

Para todos los controles existen ya librerías preparadas en kiconex, excepto para el iPro de la UTA, para el que hay que crearla a partir del programa realizado.

ISaGRAF permite exportar un recopilatorio de variables en formato PDF. La mayoría de lo que aparece en dicho PDF se puede omitir en la librería por tratarse de parámetros por defecto del control, por lo que se han recopilado en el Tabla D los registros que sí hay que tener en cuenta. Para añadir dichos registros a una librería, es necesario completar los siguientes campos de la pestaña librerías en la plataforma IoT:

- Nombre*
- Descripción*
- Categoría*
- Grupo
- Unidad de medida
- Rango
- Registro*
 - Función de lectura
 - Función de escritura
 - Offset
 - Longitud (bits)*
 - Máscara
 - Valor
 - Metadatos

*Campo obligatorio.

Desarrollo del proyecto

La *categoría* clasifica al registro en función de su naturaleza (E/S, Parámetro, Alarma, etc.), y el *grupo* lo clasifica en función de la aplicación (ejemplo: ventiladores, compresores, etc.). Los campos *offset*, *máscara* y *valor* permiten hacer un tratamiento del valor que se lee o que se va a escribir, lo que es útil en situaciones en las que, por ejemplo, solo se quiere escribir un bit concreto de un campo de 8 bits, o cuando se leen datos de temperatura con campos decimales (el valor leído necesita la aplicación de un offset que indique cuántos decimales tiene). El campo *icono* permite tener un ícono asociado al valor de cara a los diagramas de la plataforma. Los *metadatos* asocian valores a un campo de texto o a un ícono. Por ejemplo, el estado de un ventilador se asocia a dos íconos: uno de un ventilador apagado y otro de un ventilador encendido, lo que hace que en la ventana principal se indique el estado de dicho ventilador a través de un ícono en lugar de un valor, lo que lo hace más visual. Esto se aplica también a los diagramas.

El aspecto de la interfaz para introducir los campos anteriores son los de las Figuras 2.3 a 2.7.

Datos del parámetro

GENERAL CATEGORÍA UNIDADES MODBUS METADATOS NOTAS

Icono general Seleccionar

* Nombre

* Descripción

Tags Tags

* Intervalo ^

Visibilidad

← Guardar

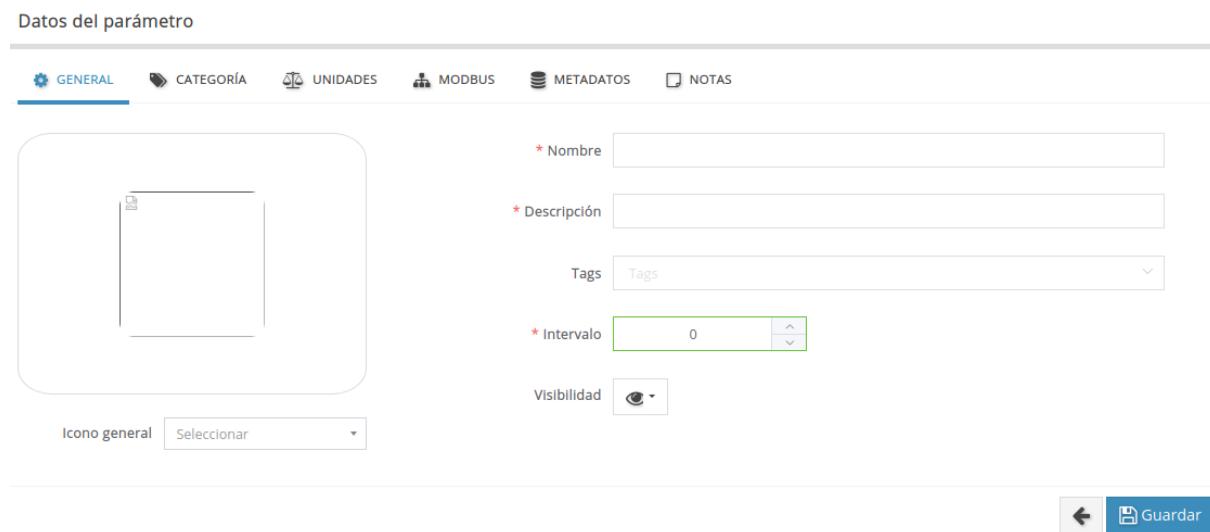


Figura 2.3: Añadir parámetro a librería kiconex - Pestaña general

GENERAL CATEGORÍA UNIDADES MODBUS METADATOS NOTAS

* Categoría: Categoría

Grupo: Grupo

Figura 2.4: Añadir parámetro a librería kiconex - Pestaña categoría

GENERAL CATEGORÍA UNIDADES MODBUS METADATOS NOTAS

U. de medida: Unidad

Rango (De -65535 a 65535): 0 - 1

Figura 2.5: Añadir parámetro a librería kiconex - Pestaña unidades

GENERAL CATEGORÍA UNIDADES MODBUS METADATOS NOTAS

* Registro: Dec 0 ex 0x0

Función de lectura: Seleccionar

Función de escritura: Seleccionar

Offset: 0.25 * Longitud (bits): s16

Máscara:

Valor:

Figura 2.6: Añadir parámetro a librería kiconex - Pestaña modbus

Metadato ×

| | |
|-----------------|--|
| Operador | Igual a |
| * Valor | <input type="text"/> ^ v |
| Etiqueta | <input type="text"/> |
| Icono | Seleccionar |
| Nivel de alarma | Ninguno |

Cancelar Guardar

Figura 2.7: Añadir parámetro a librería kiconex - adición de metadato

2.4. Programación dispositivo wireless

Llegados a este punto, hay que buscar la forma de conectar de forma inalámbrica los muebles de refrigeración: murales de lácteos, vitrinas expositoras, semimurales e islas de congelados. Para ello se ha diseñado un nuevo dispositivo que se conecta de forma cableada al control del equipo y de forma inalámbrica a través de una red WiFi. Ya se describió en el capítulo anterior el hardware a emplear, el ESP32-PoE de Olimex, y se explicó el funcionamiento de una red kiconex. Es por ello que el software del ESP32 debe tener una serie de requisitos:

- Comunicación: debe disponer de una interfaz de configuración de red, tanto de la red Modbus como de la red TCP/IP.
- Esclavo Modbus TCP: debe ser capaz de recibir las tramas Modbus TCP del maestro kibox, entenderlas y procesarlas para extraer cada campo.
- Maestro Modbus RTU: a través de la información extraída de la trama TCP recibida, enviar una nueva trama a través de Modbus RTU.

- Respuesta: es necesario repetir el proceso a la inversa: recibir la respuesta a la trama enviada por Modbus RTU, extraer la información y reenviarla como respuesta a la trama Modbus TCP recibida previamente desde el Maestro kibox.

2.4.1. Librería para interfaz de configuración

Para la interfaz se parte de la librería WiFiManager de Khoi Hoang [14] (consultar su enlace en la bibliografía para más información). Dicha librería ha sido configurada por completo para adaptarla a las necesidades de diseño del kiwi:

1. Interfaz en cualquier modo: accesible tanto en modo AP (para realizar la configuración), como en modo cliente WiFi (para ver y cambiar la configuración realizada).
2. Medio de conexión: permite seleccionar Ethernet o WiFi.
3. Diseño: logo de kiwi y colores de kiconex.
4. Nuevas ventanas con más opciones, como se muestra en la Figura 2.8:

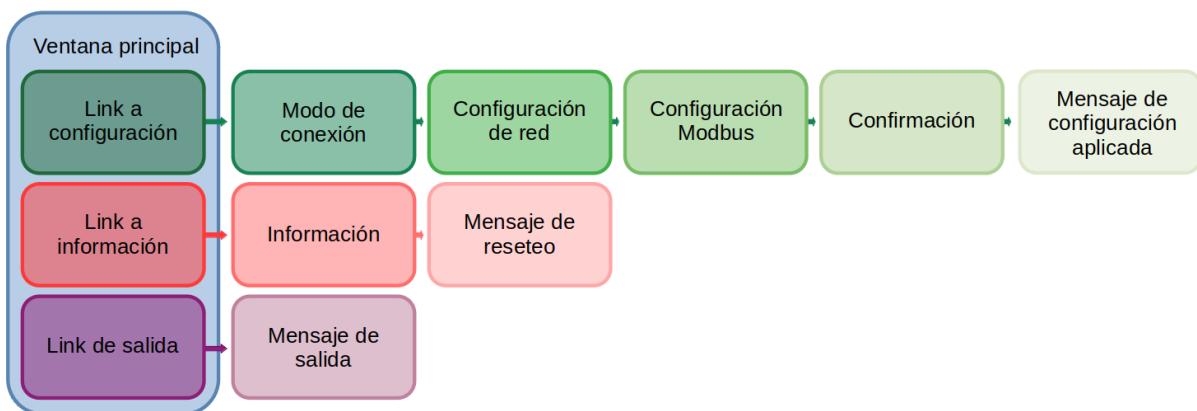


Figura 2.8: Ventanas interfaz kiwi

La nueva librería resultado de aplicar los anteriores cambios se ha bautizado con el nombre de KiWiManager. Su uso dentro del código principal es el siguiente:

- Configuración inicial:

```
#include "./KiWiManager/src/KiWiManager.h"
#include "./KiWiManager/src/KiWiManager.cpp"

#define USE_AVAILABLE_PAGES      false
#define SHOW_AP_DEBUG            true
#define CONFIGURATION_TIMEOUT    90           //time in seconds

// For some unknown reason webserver can only be started once per boot
// up
// so webserver can not be used again in the sketch.

#define WIFI_CONNECT_TIMEOUT     60000L
#define WHILE_LOOP_DELAY          5000L
#define WHILE_LOOP_STEPS          (WIFI_CONNECT_TIMEOUT / ( 3 *
WHILE_LOOP_DELAY )) 

// SSID and PW for Config Portal
const char* HOSTNAME = "kiwigw";
String ssid = "KiWi_" + String(ESP_getChipId(), HEX);
const char* password = "*****";
byte mac[6];

// SSID and PW for your Router
String Router_SSID;
String Router_Pass;

KiWiManager KiWiPortal(HOSTNAME);

void setup()
{
  KiWiPortal.begin();
}
```

■ Acceso al modo AP:

```
KiWiPortal.setDebugOutput(SHOW_AP_DEBUG);
KiWiPortal.showAvailablePages(USE_AVAILABLE_PAGES);
KiWiPortal.setMinimumSignalQuality(15);
```

```

Router_SSID = KiWiPortal.getSSID();
Router_Pass = KiWiPortal.WiFi_Pass();

if (Router_SSID != "") {
    KiWiPortal.setConfigPortalTimeout(CONFIGURATION_TIMEOUT); //If no access point name has been previously entered disable timeout.
#ifdef MBMainDebug
    Serial.println("Stored: SSID = " + Router_SSID + ", Pass = " +
        Router_Pass);
    Serial.print("Timeout ");
    Serial.print(CONFIGURATION_TIMEOUT);
    Serial.println("s");
#endif
}

else{
#ifdef MBMainDebug
    Serial.println("No timeout");
#endif
}

if (!KiWiPortal.startConfigPortal((const char *) ssid.c_str() /*,
    password*/)) {
#ifdef MBMainDebug
    Serial.println("Not connected to WiFi but continuing anyway.");
#endif
} else{
#ifdef MBMainDebug
    Serial.println("WiFi connected in WiFiManager.");
#endif
}
}

```

■ Conexión en modo cliente WiFi:

```

KiWiPortal.setDebugOutput(SHOW_AP_DEBUG);
KiWiPortal.showAvailablePages(USE_AVAILABLE_PAGES);
KiWiPortal.setMinimumSignalQuality(40);

```

Desarrollo del proyecto

```
Router_SSID = KiWiPortal.getSSID();
Router_Pass = KiWiPortal.getPASS();
IPAddress ip = KiWiPortal.getIP();
IPAddress gw = KiWiPortal.getGW();
IPAddress sn = KiWiPortal.getSN();
IPAddress dns1 = KiWiPortal.getDNS1();
IPAddress dns2 = KiWiPortal.getDNS2();

WiFi.mode(WIFI_STA);
WiFi.config(ip,gw,sn,dns1,dns2);
WiFi.begin(Router_SSID.c_str(), Router_Pass.c_str());

while ( ( WiFi.status() != WL_CONNECTED) && (millis() - startedAt <
WIFI_CONNECT_TIMEOUT ) )
{
    #ifdef MBMainDebug
        Serial.print(".");
    #endif
    int i = 0;
    while ((!WiFi.status() || WiFi.status() >= WL_DISCONNECTED) && i
        ++ < WHILE_LOOP_STEPS)
    {
        delay(WHILE_LOOP_DELAY);
    }
}
```

- Conexión en modo cliente Ethernet:

```
KiWiPortal.setDebugOutput(SHOW_AP_DEBUG);
KiWiPortal.showAvailablePages(USE_AVAILABLE_PAGES);
KiWiPortal.setMinimumSignalQuality(15);

configState = KiWiPortal.startEthernetClientConfigPortal((const char
*) ssid.c_str());
```

2.4.2. Librería para maestro Modbus RTU

Para esto se ha empleado la librería Modbus RTU del usuario smarmengol (Samuel Marco) en github. Para usarla dentro del código y poder enviar mensajes Modbus y recibir las respuestas, se emplea el siguiente código:

- Inicialización:

```
#include "../KiWiModbusRTU/KiWiModbusRtu.h"
#include "../KiWiModbusRTU/KiWiModbusRtu.cpp"

Modbus master(u8id, u8serno, u8txenpin);

master->begin(u32speed);           // baud-rate
master->setTimeOut(timeOut);      // Tiempo entre reintentos
```

- Uso:

```
uint8_t rtuCommunication = 1;
uint8_t ret = 0;

modbus_t telegrama[2];
uint8_t w_id = READ_index;

if (rw1 == 1) w_id = WRITE_index;

this->_u8state = 0;

/**
Struct with Modbus RTU parameters
telegrama[0].u8id      slave ID
telegrama[0].u8fct      Modbus Function Code
telegrama[0].u16RegAdd  Start address
telegrama[0].u16CoilsNo Number of registers or coils to read/write
telegrama[0].au16reg     Pointer to memory array
*/
telegrama[rw1].u8id = au16data[ID_index];           // slave ID
telegrama[rw1].u8fct = au16data[FN_index];          // function code
telegrama[rw1].u16RegAdd = au16data[START_index];   // star address
```

```
telegrama[rw1].u16CoilsNo = au16data[LONG_index]; // number of
registers/coils
telegrama[rw1].au16reg = au16data + w_id;           // pointer to
memory array

while (rtuCommunication)
{
    switch (this->_u8state)
    {
        case 0:
            this->_u8state++;
            break;
        case 1:
            master->query(telegrama[rw1]); // send query (only once)
            this->_u8state++;
            break;
        case 2:
            ret = master->poll(); // check message reception
            /**
             When generating the request to the slave, the master is in
             COM_IDLE mode, and after that its status would be
             COM_WAITING
             COM_IDLE=0 ; COM_WAITING=1
            */
            if (master->getState() == COM_IDLE)
            {
                rtuCommunication = 0;
            }
            break;
    }
}
```

2.4.3. Librería para esclavo Modbus TCP

En este caso se ha diseñado por completo una librería cuya función es esperar la recepción de mensajes a través de TCP/IP a la IP del dispositivo, procesar esos mensajes y enviarlos vía Modbus RTU empleando la librería anterior. Es decir se trata de una librería puente, por lo que

se ha bautizado con el nombre de KiWiModbusBridge. Su uso dentro del código principal es el siguiente:

```
// Modbus bridge library
#include "./KiWiModbusBridge/KiWiModbusBridge.h"
#include "./KiWiModbusBridge/KiWiModbusBridge.cpp"

/**
 *  ModBus RTU Configuration
 *  u8id : id = 0 for master, = 1..247 for slaves
 *  u8serno : serial port (0 for Serial)
 *  u8txenpin : 0 for RS-232 and USB-FTDI
 *              any value > 1 para RS-485
 */
uint8_t u8id = 0;
uint8_t u8serno = 0;
uint8_t u8txenpin = 5;

KiWiModbusBridge slave(u8id, u8txenpin, u8txenpin);

long rtu_br;
unsigned int tcp_port;

void setup()
{
    rtu_br=9600;
    tcp_port=502;
    slave.begin(tcp_port, rtu_br);
}

void loop()
{
    slave.run();
}
```

En el Figura 2.9 se describe la lógica de funcionamiento de esta librería.

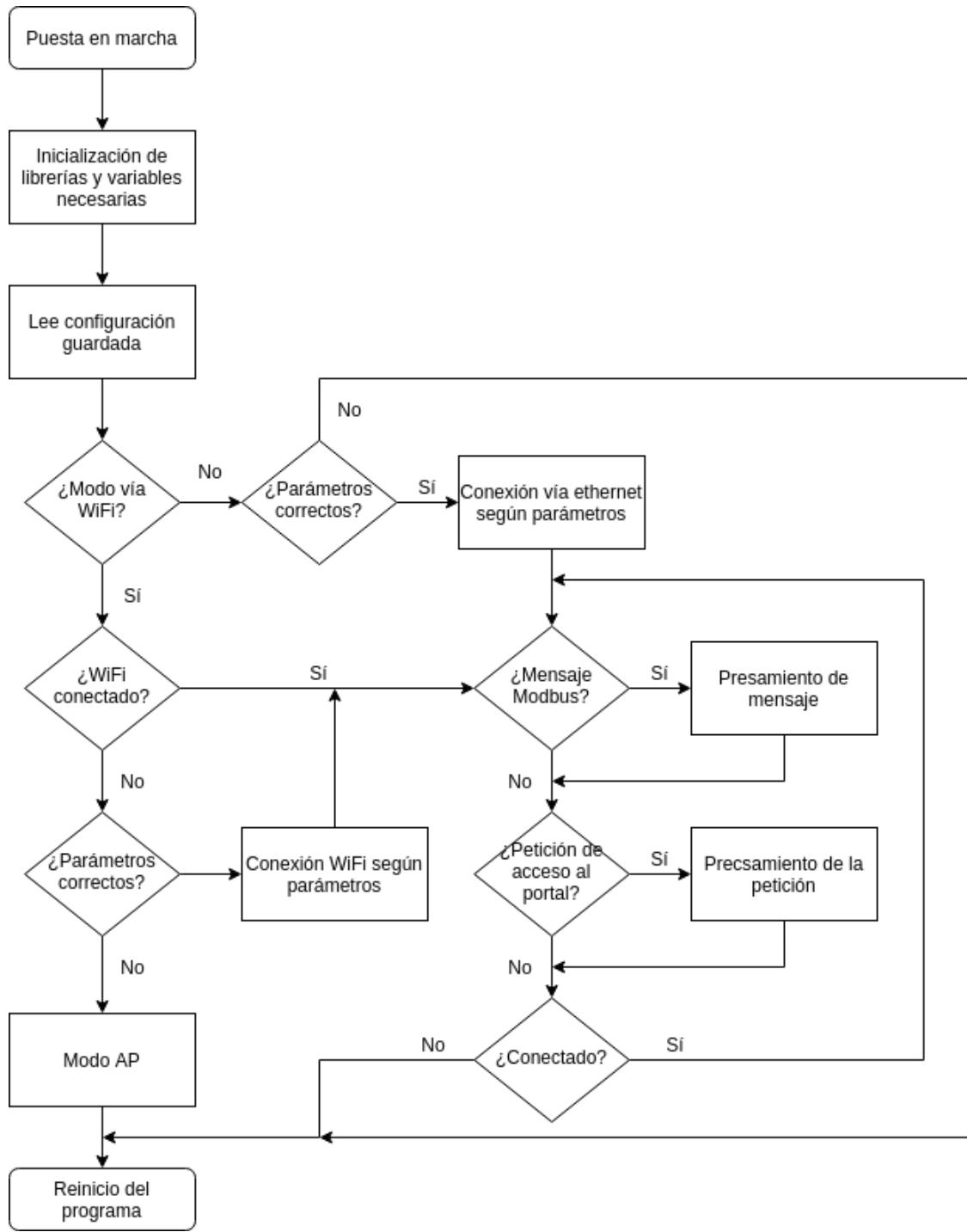


Figura 2.9: Flujo de programa pasarela TCP-RTU

2.4.4. Código principal

El código principal del software de kiwi se encarga de integrar cada una de las librerías anteriores y de gestionar el flujo de programa. Su lógica de funcionamiento se recoge en el

siguiente diagrama:

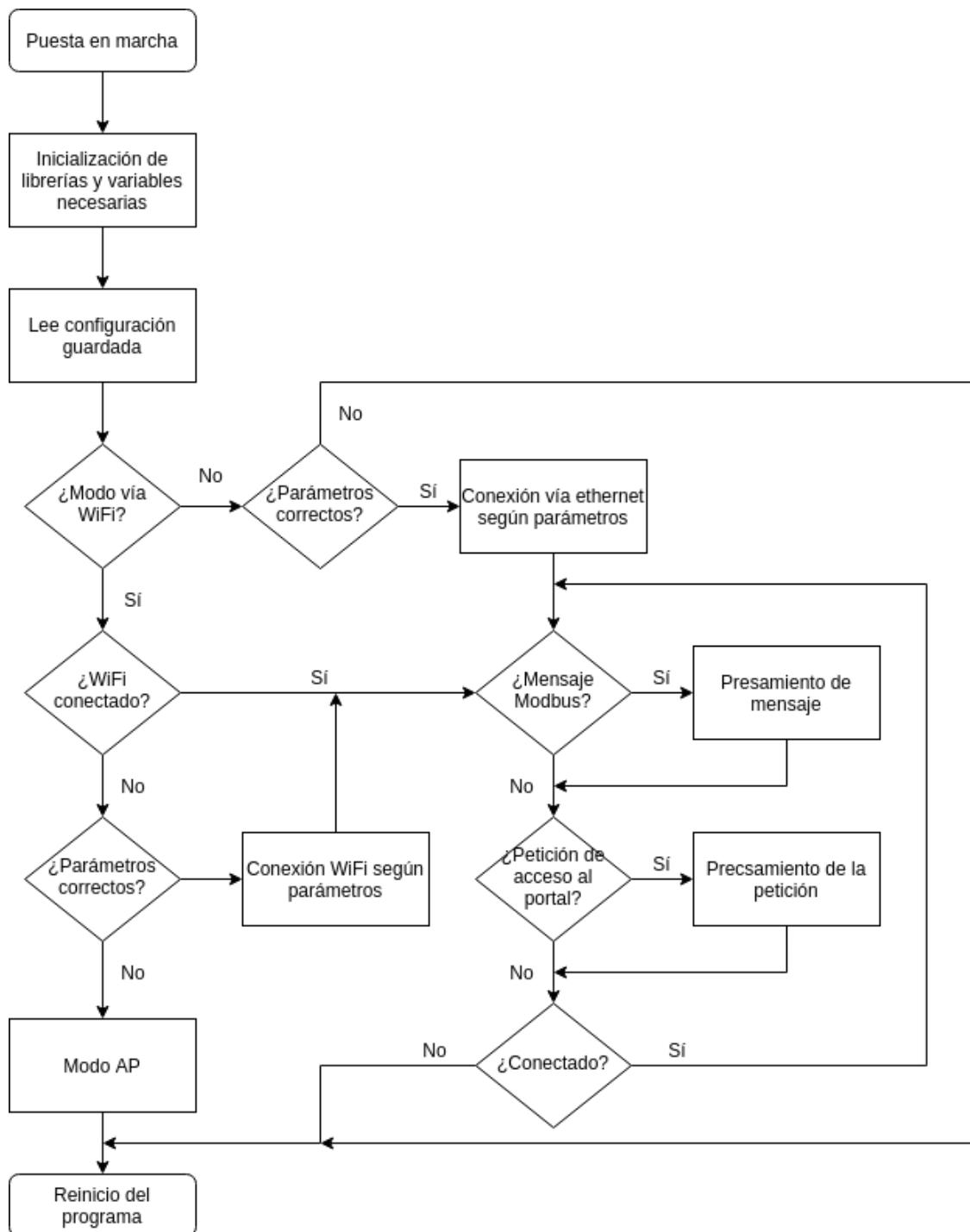


Figura 2.10: Flujo de programa del código principal de kiwi

2.4.5. Integración de todos los elementos del sistema

Ya está todo listo para conectarlo a la red kiconex. Como ya se ha mencionado, los equipos del supermercado se comunican con la plataforma a través del kibox y del kiwi. Por ello, lo primero es crear la instalación del supermercado en la plataforma IoT, utilizando un id único de cada kibox, y después se añaden a dicha instalación cada uno de los equipos:

- Las centrales, las cámaras de refrigeración y los equipos de clima, se conectan de forma cableada al kibox a través de Modbus RTU, por lo que en la plataforma se añaden indicando su librería de registros y el puerto y la dirección Modbus.
- En el caso de los muebles de refrigeración, se conectan de forma cableada con el kiwi a través de Modbus RTU. Dado que la comunicación se realiza a través del mismo kiwi, vía TCP, cada mueble se añade a la instalación indicando la librería de registros del control, la IP del kiwi, y la dirección Modbus del control.

La Figura 2.11 siguiente, indica el esquema de conexión física de cada equipo.

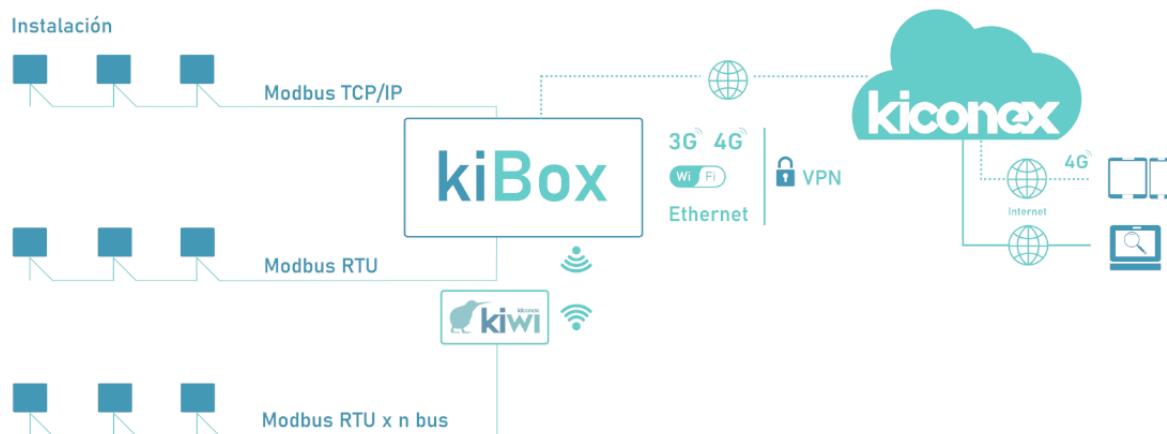


Figura 2.11: Esquema conexión equipos supermercado

La Figura 2.12 y la Figura 2.13 muestran el proceso de creación de una instalación en la plataforma IoT de kiconex y la Figura 2.14 y la Figura 2.15 el proceso de adición de dispositivos a la misma. Se observa como cada instalación necesita un *UUID* único y específico del kibox al que se enlaza, además de una configuración de los posibles puertos de los que disponga dicho modelo de kibox. Por otro lado, el dispositivo necesita que se especifique su configuración

Modbus: TCP o RTU, IP, dirección, puerto, etc. Finalmente, la instalación tiene el aspecto de la Figura 2.16.

GENERAL UBICACIÓN HARDWARE PERMISOS

Arrastra la imagen aquí, péguela o examínala desde el [explorador](#)

* Nombre:

Descripción:

* UUID:

* Muestreo:

Habilitado:

Figura 2.12: Creación instalación en plataforma IoT - Datos generales

GENERAL UBICACIÓN **HARDWARE** PERMISOS

Puertos serie

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| Puerto 1 | Puerto 2 | Puerto 3 | Puerto 4 | Puerto 5 | Puerto 6 | Puerto 7 | Puerto 8 |
|----------|----------|----------|----------|----------|----------|----------|----------|

Función: Modbus RTU - Maestro

Velocidad (bps): 300 bps

Bits de datos: 7

Paridad: Ninguna

Bits de parada: 1

Figura 2.13: Creación instalación en plataforma IoT - Hardware kibox

Desarrollo del proyecto

The screenshot shows the 'GENERAL' tab of a device configuration interface. On the left, there is a placeholder area for dragging and dropping an image or selecting one from a browser. To the right, there are several input fields: 'Nombre' (Name) with a red asterisk, 'Descripción' (Description), 'Muestreo' (Sampling) set to 60, and a 'Habilitado' (Enabled) toggle switch. At the bottom right are 'Guarda' (Save) and 'Cancelar' (Cancel) buttons.

Figura 2.14: Adición de dispositivo a la instalación - Datos generales

The screenshot shows the 'PROTOCOLO' tab of the device configuration interface. It includes a 'Liberaria' (Library) dropdown set to 'kiconex UTA 20PED272 v1.0 r0', a 'Dirección' (Address) input field containing '9' with a dropdown arrow, and a 'MODBUS' section. Below these are protocol selection buttons for 'RTU' (selected) and 'TCP'. The 'Configuración RTU' section contains fields for 'Puerto serie' (Serial port) set to '1', 'Velocidad (bps)' (Baud rate) set to '9600 bps', 'Bits de datos' (Data bits) set to '8', 'Paridad' (Parity) set to 'Ninguna' (None), and 'Bits de parada' (Stop bits) set to '1'. At the bottom right are 'Guarda' (Save) and 'Cancelar' (Cancel) buttons.

Figura 2.15: Adición de dispositivo a la instalación - Datos Modbus dispositivo

ki Supermarket
Kiconex supermarket

CONTROLES **ALARMAS** **ACTIVIDAD** **GRÁFICAS** **DIAGRAMAS** **REGLAS** **DOCUMENTACIÓN** **...**

| <input type="checkbox"/> | Nombre | Estado | Sonda principal | Acciones |
|--------------------------|--|--------|--|----------|
| <input type="checkbox"/> | 01. Cámara BT Cámara de baja temperatura | | Temperatura de cámara -17.1 °c | |
| <input type="checkbox"/> | 02. Cámara MT Cámara de media temperatura | | Temperatura de cámara 1.5 °c | |
| <input type="checkbox"/> | 03. Cámara AT Cámara de alta temperatura | | Temperatura de cámara 16.6 °c | |
| <input type="checkbox"/> | 04. Isla BT Isla de baja temperatura | | Temperatura de cámara -17.4 °c | |
| <input type="checkbox"/> | 05. Vitrina MT Vitrina de media temperatura | | Temperatura de cámara 2.6 °c | |
| <input type="checkbox"/> | 06. Murales MT Murales de media temperatura | | Temperatura de cámara 2.6 °c | |
| <input type="checkbox"/> | 07. Semimurales MT Semimurales de media temperatura | | Temperatura de cámara 1.5 °c | |
| <input type="checkbox"/> | 08. Obrador MT Obrador de media temperatura | | Temperatura de cámara 3.6 °c | |
| <input type="checkbox"/> | 09. Unidad de Tratamiento de Aire Unidad de Tratamiento de Aire | | u12 26.6 °c | |
| <input type="checkbox"/> | 10. Central de Refrigeración Central de Refrigeración | | Temperatura retorno 11.8 °c | |
| <input type="checkbox"/> | 11. Bomba de calor Bomba de calor | | Temperatura retorno 18.2 °c | |

1 - 11 de 11

Figura 2.16: Instalación supermercado creada

Capítulo 3

Pruebas y Resultados

3.1. Pruebas control UTA



Figura 3.1: Simulador de pruebas para iPro

Tras realizar el programa y el diseño de la pantalla para el control de la UTA, estos se cargan en un simulador como el de la Figura 3.1, que se compone de un iPro y una pantalla para pruebas. Dado que el programa se ha simulado previamente en el software ISaGRAF, no se han encontrado problemas de funcionamiento en él, sin embargo, es habitual y así ha sido en este caso, encontrar errores de diseño en la pantalla, que se han pasado por alto durante su creación.

Estos problemas se han corregido desde el software Visoprog y se han vuelto a cargar.

Cuando el resultado de las pruebas es bueno, desde fábrica realizan un cuadro nuevo con iPro, al que se le carga el programa realizado tras comprobar que el iPro funciona correctamente. Este cuadro se envía al cliente, en este caso el instalador al cargo del supermercado.

3.2. Pruebas kiwi

En las primeras pruebas del kiwi, se han descubierto varios problema y/o necesidades:

- La necesidad de borrar la configuración guardada sin necesidad de acceder al portal web de configuración. Para ello se han añadido al programa las líneas necesarias para detectar la pulsación del botón integrado en la placa durante 5 segundos, lo que indica que se quiere resetear el kiwi y borrar la configuración realizada.
- En esa primera versión, no se disponía de la posibilidad de conexión por cable, pero se decidió añadir dicha opción dado que dependiendo del equipo y la instalación puede ser más rentable usar un dispositivo como el kiwi conectado vía ethernet.
- Timeout de mensajes. Cada vez que se recibían mensajes para un dispositivo, kiwi abría un canal para los mensajes destinados a ese dispositivo, pero no volvía a cerrarlo, por lo que se ha añadido un tiempo (timeout) tras el cual, si no se vuelven a recibir mensajes a través de ese canal, se vuelve a cerrar.
- Problema de cambio de modo (AP y cliente). Para la gestión de la conexión de red, WiFi o Ethernet, se han empleado las librerías estándar para el ESP32: WiFi.h, ETH.h, WiFiClient.h. Al parecer estas librerías no gestionan bien el paso de un modo a otro, por lo que el ESP32 parecía dejar de funcionar. La solución ha sido reiniciar el kiwi tras la configuración, para volver a ponerse en marcha desde cero, en el nuevo modo, a partir de los parámetros de configuración guardados en la EEPROM.
- Desconexiones puntuales de la red WiFi. En el kiwi instalado para pruebas en la red de la empresa Intarcon, se detectaron desconexiones contínuas de la red WiFi. Este problema dio muchos dolores de cabeza, ya que el kiwi parecía funcionar sin problemas en otras redes. Para analizar este caso, se conectó un segundo kiwi en una red aislada, empleando

un router de los empleados por kiconex para aquellas instalaciones sin acceso a la red, y se registraron en una hoja las desconexiones de ambos kiwis.

El resultado fue el de la Figura 3.2 y la Figura 3.3, donde se puede ver que el kiwi de la red del router no tuvo ningún problema, mientras que el kiwi de la red de Intarcon tenía desconexiones continuas durante el horario de trabajo: de 8 a 18:30. Fue este último dato el decisivo para saber que el problema estaba en la red de Intarcon, por lo que lo consultamos con el informático al cargo de la misma: nos confirmó que tras unos cambios en la red, había problemas con las DNS, y muchos equipos eran expulsados de la red. Este problema aun persiste, aunque en menor medida, pero el kiwi conectado al router sirve de muestra del buen funcionamiento del mismo.

Seguimiento de errores KiWi

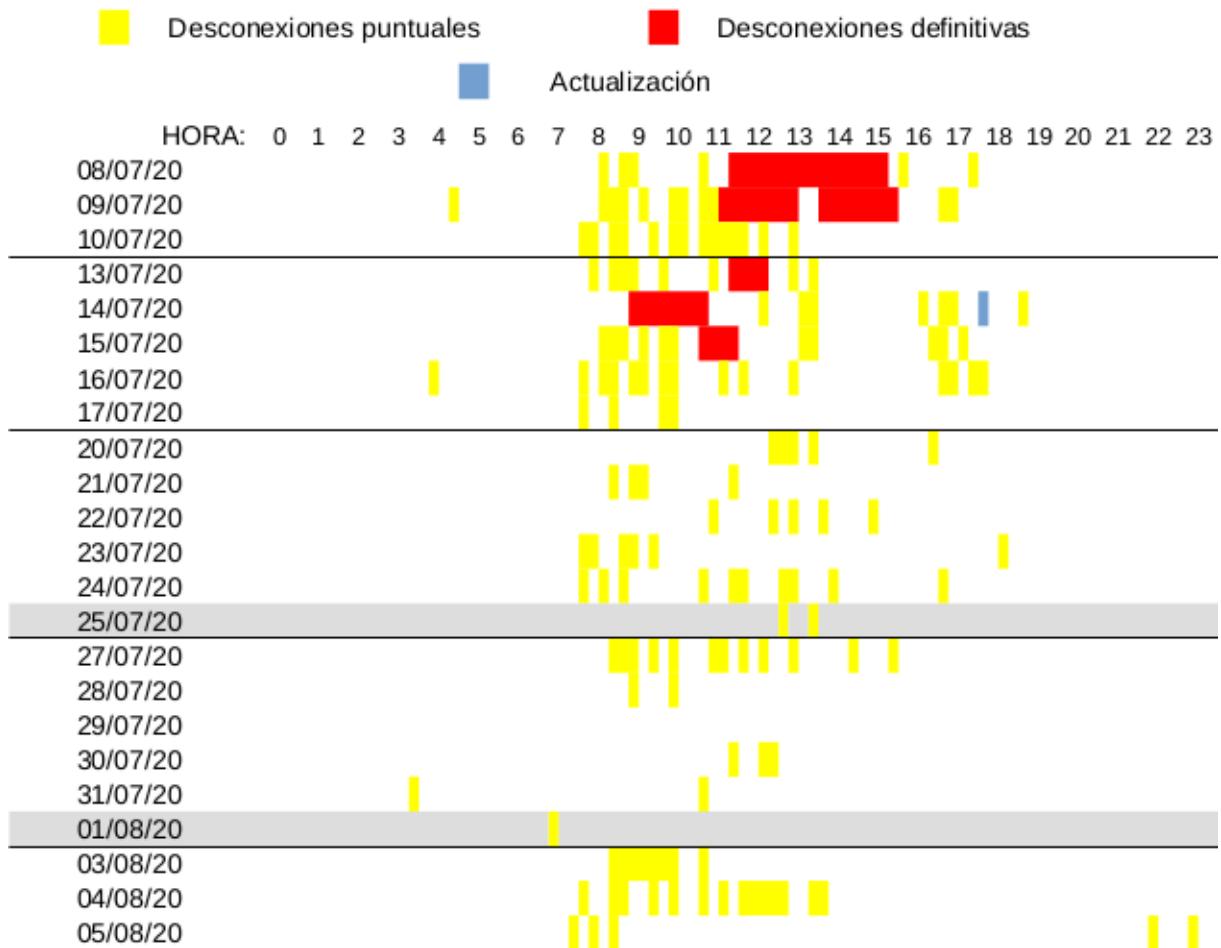


Figura 3.2: Desconexiones kiwi en red de Intarcon

Seguimiento de errores KiWi

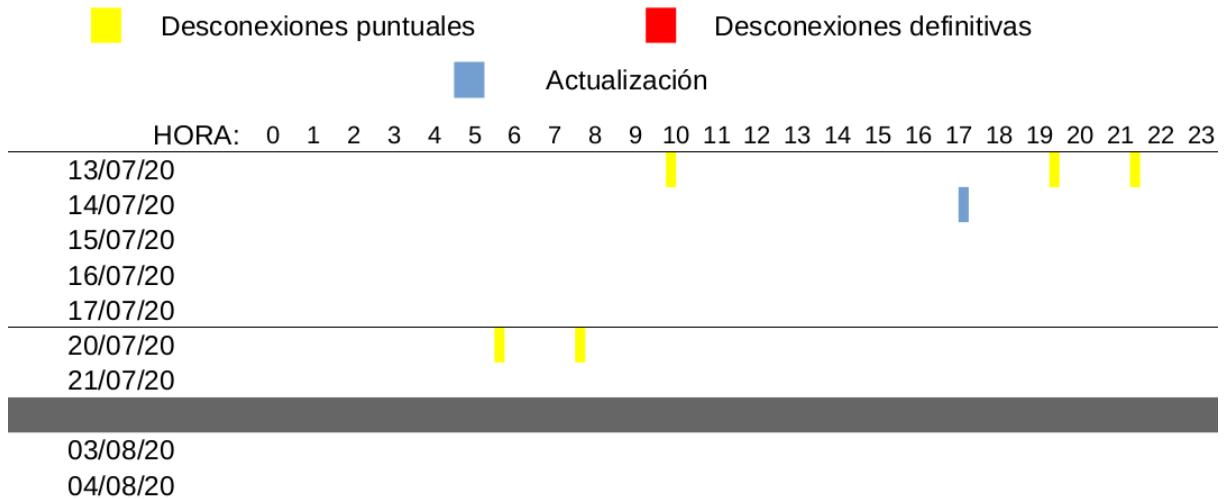


Figura 3.3: Desconexiones kiwi en red de router kiconex

3.3. Puesta en marcha de la instalación

Cuando se conectan todos los dispositivos de la instalación, es momento de su puesta en marcha y de comprobar que todo está correctamente. Se trata de un trámite necesario, ya que aunque normalmente no da problemas, existen ocasiones en las que falla la conexión por causas ajena a kiconex: resistencias de extremo de cable modbus, direcciones Modbus no configuradas en los equipos correctamente, etc. En este caso, el mapa de direcciones Modbus es el de la Tabla 3.1.

| EQUIPO | DIRECCIÓN |
|-------------------------------|-----------|
| Cámara BT | 1 |
| Cámara MT | 2 |
| Cámara AT | 3 |
| Isla BT | 4 |
| Vitrina MT | 5 |
| Murales MT | 6 |
| Semimurales MT | 7 |
| Obrador MT | 8 |
| Unidad de Tratamiento de Aire | 9 |
| Central de Refrigeración | 10 |
| Bomba de calor | 11 |

Tabla 3.1: Direcciones Modbus equipos supermercado.

La Figura 3.4 muestra como todos los elementos de la instalación se han conectado correctamente.

ki Supermarket
Kiconex supermarket

CONTROLES ALARMAS ACTIVIDAD GRÁFICAS DIAGRAMAS REGLAS DOCUMENTACIÓN

| <input type="checkbox"/> | Nombre | Estado | Sonda principal | Acciones |
|--------------------------|--|-----------|--|----------|
| <input type="checkbox"/> | 01. Cámara BT Cámara de baja temperatura | Conectado | Temperatura de cámara -17.1 °c | |
| <input type="checkbox"/> | 02. Cámara MT Cámara de media temperatura | Conectado | Temperatura de cámara 1.5 °c | |
| <input type="checkbox"/> | 03. Cámara AT Cámara de alta temperatura | Conectado | Temperatura de cámara 16.6 °c | |
| <input type="checkbox"/> | 04. Isla BT Isla de baja temperatura | Conectado | Temperatura de cámara -17.4 °c | |
| <input type="checkbox"/> | 05. Vitrina MT Vitrina de media temperatura | Conectado | Temperatura de cámara 2.6 °c | |
| <input type="checkbox"/> | 06. Murales MT Murales de media temperatura | Conectado | Temperatura de cámara 2.6 °c | |
| <input type="checkbox"/> | 07. Semimurales MT Semimurales de media temperatura | Conectado | Temperatura de cámara 1.5 °c | |
| <input type="checkbox"/> | 08. Obrero MT Obrero de media temperatura | Conectado | Temperatura de cámara 3.6 °c | |
| <input type="checkbox"/> | 09. Unidad de Tratamiento de Aire Unidad de Tratamiento de Aire | Conectado | u12 26.6 °c | |
| <input type="checkbox"/> | 10. Central de Refrigeración Central de Refrigeración | Conectado | Temperatura retorno 11.8 °c | |
| <input type="checkbox"/> | 11. Bomba de calor Bomba de calor | Conectado | Temperatura retorno 18.2 °c | |

1 - 11 de 11

Figura 3.4: Instalación supermercado conectada y funcionando

Capítulo 4

Conclusiones y Trabajos Futuros

En este Trabajo Fin de Máster, se desarrolla una red basada en kiconex para un supermercado. Una parte de los equipos del establecimiento se han instalado siguiendo la estructura estándar de una red kiconex, sin embargo, para determinados equipos, se ha innovado a través del desarrollo de un nuevo hardware inalámbrico, bautizado con el nombre de kiwi. Además, para la Unidad de Tratamiento de Aire del supermercado, también se ha programado un control a medida, siguiendo las especificaciones de funcionamiento del cliente. Todo esto ha permitido cumplir con los objetivos que se plantearon al comienzo del documento, obteniendo una instalación que reúne las siguientes características:

- Monitorización de valores en la plataforma IoT: temperaturas, estados dispositivos, alarmas, etc.
- Visualización de valores en el tiempo, en forma de gráficas.
- Control remoto de dispositivos: ON/OFF, reinicio de alarmas, etc.
- Cambios de consigna y configuración de parámetros de forma remota.
- Alertas frente a alarmas.
- Funcionamiento programado a través de reglas.

En el proceso de desarrollo se ha introducido la plataforma IoT de kiconex, donde se crea y visualiza la instalación. Además, se han presentado los entornos de programación de ISaGRAF y Visoprog, mediante los cuales se programa un control como el iPro.

Se han llevado a cabo las pruebas necesarias para verificar el correcto funcionamiento del kiwi, y en este proceso también se ha puesto de manifiesto la necesidad de una correcta configuración de red, haciendo indispensable la presencia de personal con conocimientos de redes en la puesta en marcha de cualquier instalación que incluya un dispositivo como éste.

Se ha verificado el funcionamiento del software de la UTA sobre un simulador físico real, compuesto del mismo control iPro con su pantalla. Esto asegura el correcto funcionamiento cuando se carga el programa en el control final.

Finalmente, como se ha explicado en el capítulo 3, la puesta en marcha ha sido rápida tras configurar en cada elemento del supermercado su correspondiente dirección Modbus. En dicha puesta en marcha, los elementos que afectan a la conectividad con la plataforma IoT son:

- En caso de equipos Modbus RTU:
 - Resistencias de terminación.
 - Conexión de equipos en cadena.
- En caso de equipos Modbus TCP: todos los equipos en el mismo rango de IPs que el kibox.
- Configuración de la red del kibox: apertura de los puertos de salida necesarios.

Bibliografía

- [1] *Plataforma IoT Kiconex.* dirección: <https://app.kiconex.com/>.
- [2] *Productos Kiconex.* dirección: <https://www.kiconex.com/producto/>.
- [3] *Protocolo Modbus.* dirección: <http://www.modbus.org/>.
- [4] *Manual iPro.* dirección: <https://climate.emerson.com/documents/ipro-series-en-4923358.pdf>.
- [5] *Marca Dixell.* dirección: <https://climate.emerson.com/es-es/brands/dixell>.
- [6] *¿Qué es kiconex?* Dirección: <https://www.kiconex.com/que-es-kiconex/>.
- [7] *Web de Espressif sobre su ESP32.* dirección: <https://www.espressif.com/en/products/socs/esp32/overview>.
- [8] *Pantalla VTIPG.* dirección: <https://climate.emerson.com/es-es/shop/1/dixell-electronics-sku-vtipg-hmi-es-es>.
- [9] *Programa Visoprog.* dirección: <https://climate.emerson.com/es-es/shop/1/dixell-electronics-sku-visoprog-es-es>.
- [10] *Empresa Logicbus hablando sobre ISaGRAF.* dirección: <https://www.logicbus.com.mx/isagraf.php>.
- [11] *Página de producto de la placa Olimex ESP32-PoE.* dirección: <https://www.olimex.com/Products/IoT/ESP32/ESP32-POE/open-source-hardware>.
- [12] *Página de producto del conversor UEXT-RS485.* dirección: <https://www.olimex.com/Products/Modules/Interface/MOD-RS485/open-source-hardware>.
- [13] *S. Marco, Librería Modbus RTU para ESP32 en GitHub.* dirección: <https://github.com/smarmengol/Modbus-Master-Slave-for-Arduino>.

BIBLIOGRAFÍA

- [14] K. Hoang, *Librería de la interfaz de conexión WiFi en GitHub.* dirección: https://github.com/khoih-prog/ESP_WiFiManager.

ANEXOS

Anexo A

Aplicación del protocolo Modbus

Fuente: Web oficial de la *Modbus Organization* (modbus.org)



MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3

CONTENTS

| | | |
|------------------------|--|----|
| 1 | Introduction | 2 |
| 1.1 | Scope of this document | 2 |
| 2 | Abbreviations | 2 |
| 3 | Context..... | 3 |
| 4 | General description | 3 |
| 4.1 | Protocol description..... | 3 |
| 4.2 | Data Encoding..... | 5 |
| 4.3 | MODBUS Data model | 6 |
| 4.4 | MODBUS Addressing model | 7 |
| 4.5 | Define MODBUS Transaction | 8 |
| 5 | Function Code Categories | 10 |
| 5.1 | Public Function Code Definition | 11 |
| 6 | Function codes descriptions | 11 |
| 6.1 | 01 (0x01) Read Coils | 11 |
| 6.2 | 02 (0x02) Read Discrete Inputs | 12 |
| 6.3 | 03 (0x03) Read Holding Registers | 15 |
| 6.4 | 04 (0x04) Read Input Registers | 16 |
| 6.5 | 05 (0x05) Write Single Coil | 17 |
| 6.6 | 06 (0x06) Write Single Register | 19 |
| 6.7 | 07 (0x07) Read Exception Status (Serial Line only) | 20 |
| 6.8 | 08 (0x08) Diagnostics (Serial Line only) | 21 |
| 6.8.1 | Sub-function codes supported by the serial line devices | 22 |
| 6.8.2 | Example and state diagram | 24 |
| 6.9 | 11 (0x0B) Get Comm Event Counter (Serial Line only) | 25 |
| 6.10 | 12 (0x0C) Get Comm Event Log (Serial Line only) | 26 |
| 6.11 | 15 (0x0F) Write Multiple Coils..... | 29 |
| 6.12 | 16 (0x10) Write Multiple registers | 30 |
| 6.13 | 17 (0x11) Report Server ID (Serial Line only) | 31 |
| 6.14 | 20 (0x14) Read File Record | 32 |
| 6.15 | 21 (0x15) Write File Record | 34 |
| 6.16 | 22 (0x16) Mask Write Register | 36 |
| 6.17 | 23 (0x17) Read/Write Multiple registers | 38 |
| 6.18 | 24 (0x18) Read FIFO Queue | 40 |
| 6.19 | 43 (0x2B) Encapsulated Interface Transport | 41 |
| 6.20 | 43 / 13 (0x2B / 0x0D) CANopen General Reference Request and Response PDU | 42 |
| 6.21 | 43 / 14 (0x2B / 0x0E) Read Device Identification | 43 |
| 7 | MODBUS Exception Responses | 47 |
| Annex A (Informative): | MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES | 50 |
| Annex B (Informative): | CANOPEN GENERAL REFERENCE COMMAND | 50 |

1 Introduction

1.1 Scope of this document

MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model, which provides client/server communication between devices connected on different types of buses or networks.

The industry's serial de facto standard since 1979, MODBUS continues to enable millions of automation devices to communicate. Today, support for the simple and elegant structure of MODBUS continues to grow. The Internet community can access MODBUS at a reserved system port 502 on the TCP/IP stack.

MODBUS is a request/reply protocol and offers services specified by **function codes**. MODBUS function codes are elements of MODBUS request/reply PDUs. The objective of this document is to describe the function codes used within the framework of MODBUS transactions.

MODBUS is an application layer messaging protocol for client/server communication between devices connected on different types of buses or networks.

It is currently implemented using:

- TCP/IP over Ethernet. See MODBUS Messaging Implementation Guide V1.0a.
- Asynchronous serial transmission over a variety of media (wire : EIA/TIA-232-E, EIA-422, EIA/TIA-485-A; fiber, radio, etc.)
- MODBUS PLUS, a high speed token passing network.

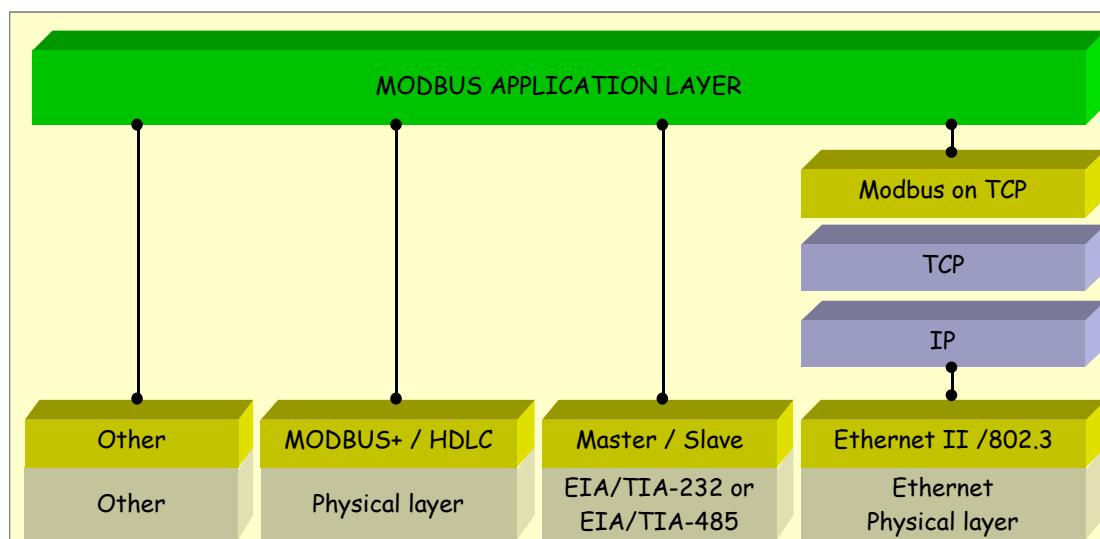


Figure 1: MODBUS communication stack

References

1. RFC 791, Internet Protocol, Sep81 DARPA

2 Abbreviations

| | |
|-------------|---------------------------------|
| ADU | Application Data Unit |
| HDLC | High level Data Link Control |
| HMI | Human Machine Interface |
| IETF | Internet Engineering Task Force |
| I/O | Input/Output |
| IP | Internet Protocol |
| MAC | Media Access Control |
| MB | MODBUS Protocol |

MBAP MODBUS Application Protocol
PDU Protocol Data Unit
PLC Programmable Logic Controller
TCP Transmission Control Protocol

3 Context

The MODBUS protocol allows an easy communication within all types of network architectures.

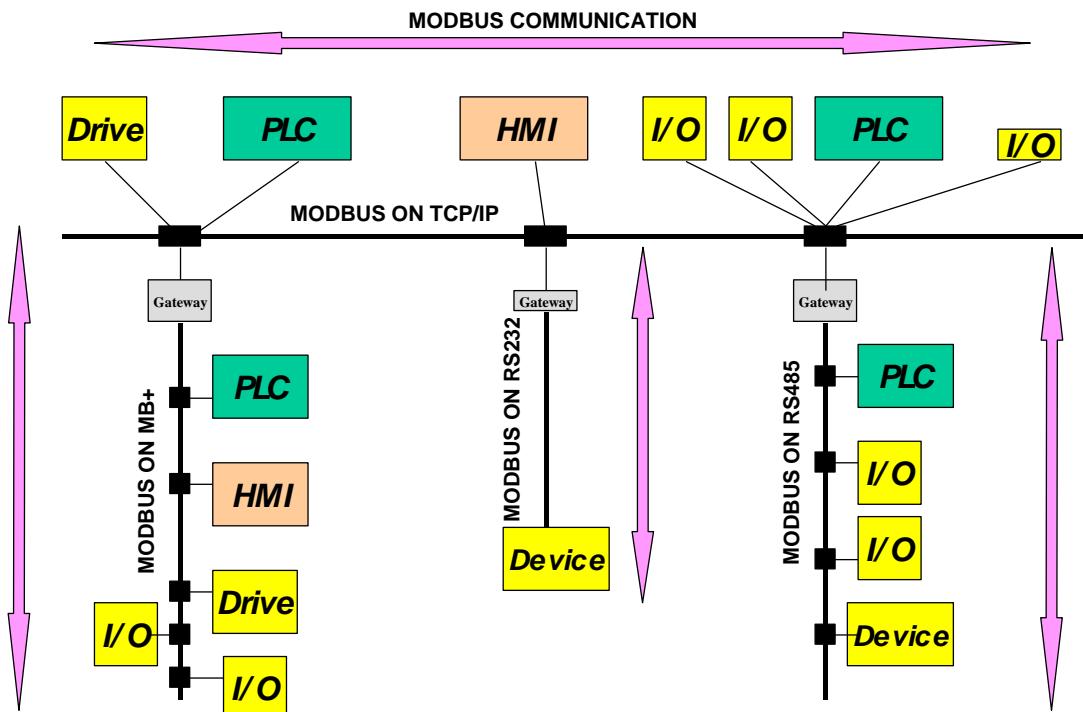


Figure 2: Example of MODBUS Network Architecture

Every type of devices (PLC, HMI, Control Panel, Driver, Motion control, I/O Device...) can use MODBUS protocol to initiate a remote operation.

The same communication can be done as well on serial line as on an Ethernet TCP/IP networks. Gateways allow a communication between several types of buses or network using the MODBUS protocol.

4 General description

4.1 Protocol description

The MODBUS protocol defines a simple protocol data unit (**PDU**) independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or network can introduce some additional fields on the application data unit (**ADU**).

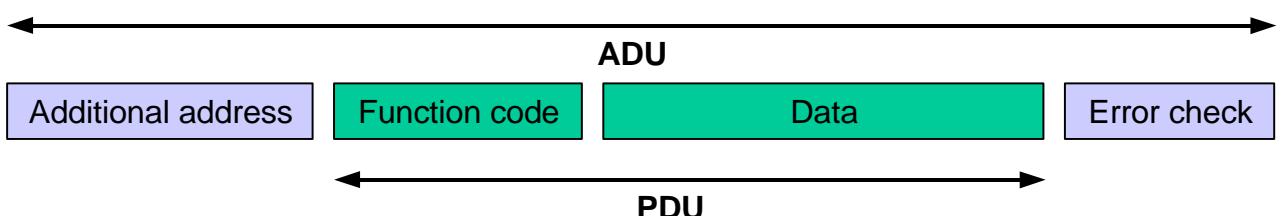


Figure 3: General MODBUS frame

The MODBUS application data unit is built by the client that initiates a MODBUS transaction. The function indicates to the server what kind of action to perform. The MODBUS application protocol establishes the format of a request initiated by a client.

The function code field of a MODBUS data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 – 255 is reserved and used for exception responses). When a message is sent from a Client to a Server device the function code field tells the server what kind of action to perform. Function code "0" is not valid.

Sub-function codes are added to some function codes to define multiple actions.

The data field of messages sent from a client to server devices contains additional information that the server uses to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

The data field may be nonexistent (of zero length) in certain kinds of requests, in this case the server does not require any additional information. The function code alone specifies the action.

If no error occurs related to the MODBUS function requested in a properly received MODBUS ADU the data field of a response from a server to a client contains the data requested. If an error related to the MODBUS function requested occurs, the field contains an exception code that the server application can use to determine the next action to be taken.

For example a client can read the ON / OFF states of a group of discrete outputs or inputs or it can read/write the data contents of a group of registers.

When the server responds to the client, it uses the function code field to indicate either a normal (error-free) response or that some kind of error occurred (called an exception response). For a normal response, the server simply echoes to the request the original function code.

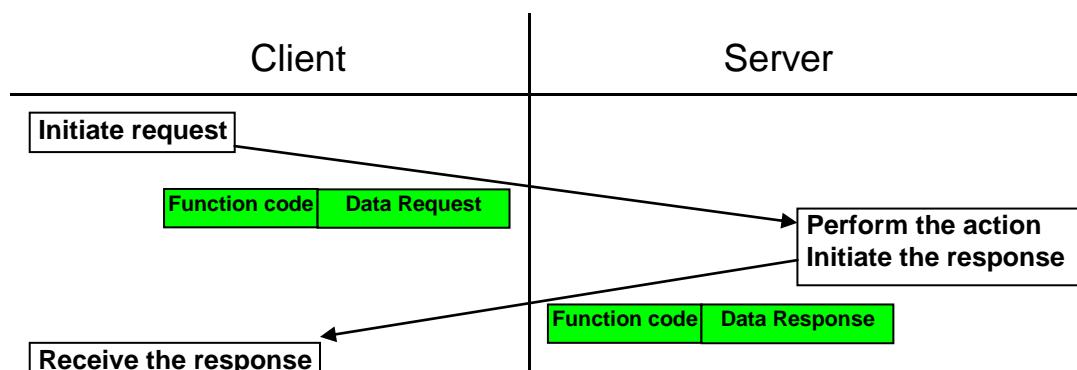


Figure 4: MODBUS transaction (error free)

For an exception response, the server returns a code that is equivalent to the original function code from the request PDU with its most significant bit set to logic 1.

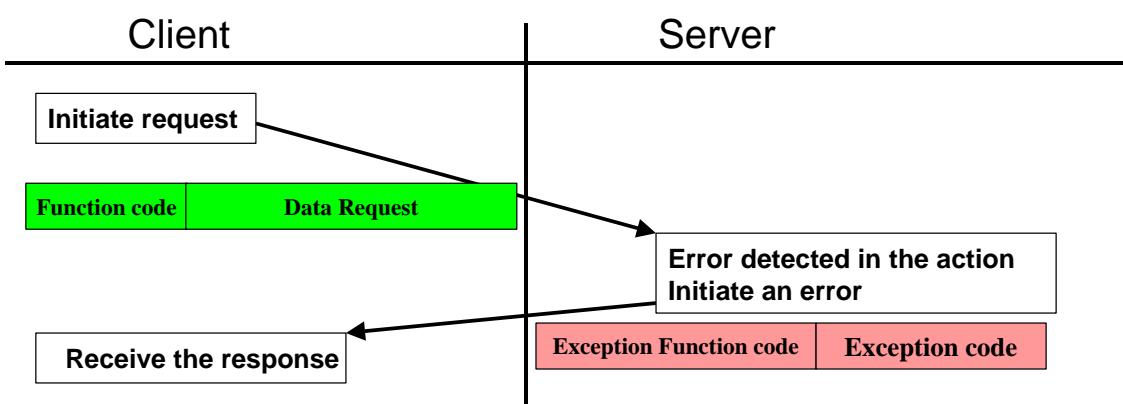


Figure 5: MODBUS transaction (exception response)

 **Note:** It is desirable to manage a time out in order not to indefinitely wait for an answer which will perhaps never arrive.

The size of the MODBUS PDU is limited by the size constraint inherited from the first MODBUS implementation on Serial Line network (max. RS485 ADU = 256 bytes).

Therefore:

MODBUS PDU for serial line communication = 256 - Server address (1 byte) - CRC (2 bytes) = 253 bytes.

Consequently:

RS232 / RS485 ADU = 253 bytes + Server address (1 byte) + CRC (2 bytes) = 256 bytes.

TCP MODBUS ADU = 253 bytes + MBAP (7 bytes) = 260 bytes.

The MODBUS protocol defines three PDUs. They are :

- MODBUS Request PDU, mb_req_pdu
- MODBUS Response PDU, mb_rsp_pdu
- MODBUS Exception Response PDU, mb_excep_rsp_pdu

The mb_req_pdu is defined as:

```
mb_req_pdu = {function_code, request_data},    where
            function_code = [1 byte] MODBUS function code,
            request_data = [n bytes] This field is function code dependent and usually
                                contains information such as variable references,
                                variable counts, data offsets, sub-function codes etc.
```

The mb_rsp_pdu is defined as:

```
mb_rsp_pdu = {function_code, response_data},    where
            function_code = [1 byte] MODBUS function code
            response_data = [n bytes] This field is function code dependent and usually
                                contains information such as variable references,
                                variable counts, data offsets, sub-function codes, etc.
```

The mb_excep_rsp_pdu is defined as:

```
mb_excep_rsp_pdu = {exception-function_code, request_data},    where
            exception-function_code = [1 byte] MODBUS function code + 0x80
            exception_code = [1 byte] MODBUS Exception Code Defined in table
                                "MODBUS Exception Codes" (see section 7 ).
```

4.2 Data Encoding

- MODBUS uses a 'big-Endian' representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example

| Register size | value |
|---------------|--------|
| 16 - bits | 0x1234 |

the first byte sent is 0x12 then 0x34

 **Note:** For more details, see [1].

4.3 MODBUS Data model

MODBUS bases its data model on a series of tables that have distinguishing characteristics. The four primary tables are:

| Primary tables | Object type | Type of | Comments |
|-------------------|-------------|------------|---|
| Discretes Input | Single bit | Read-Only | This type of data can be provided by an I/O system. |
| Coils | Single bit | Read-Write | This type of data can be alterable by an application program. |
| Input Registers | 16-bit word | Read-Only | This type of data can be provided by an I/O system |
| Holding Registers | 16-bit word | Read-Write | This type of data can be alterable by an application program. |

The distinctions between inputs and outputs, and between bit-addressable and word-addressable data items, do not imply any application behavior. It is perfectly acceptable, and very common, to regard all four tables as overlaying one another, if this is the most natural interpretation on the target machine in question.

For each of the primary tables, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span multiple consecutive data items up to a data size limit which is dependent on the transaction function code.

It's obvious that all the data handled via MODBUS (bits, registers) must be located in device application memory. But physical address in memory should not be confused with data reference. The only requirement is to link data reference with physical address.

MODBUS logical reference numbers, which are used in MODBUS functions, are unsigned integer indices starting at zero.

- **Implementation examples of MODBUS model**

The examples below show two ways of organizing the data in device. There are different organizations possible, but not all are described in this document. Each device can have its own organization of the data according to its application

Example 1 : Device having 4 separate blocks

The example below shows data organization in a device having digital and analog, inputs and outputs. Each block is separate because data from different blocks have no correlation. Each block is thus accessible with different MODBUS functions.

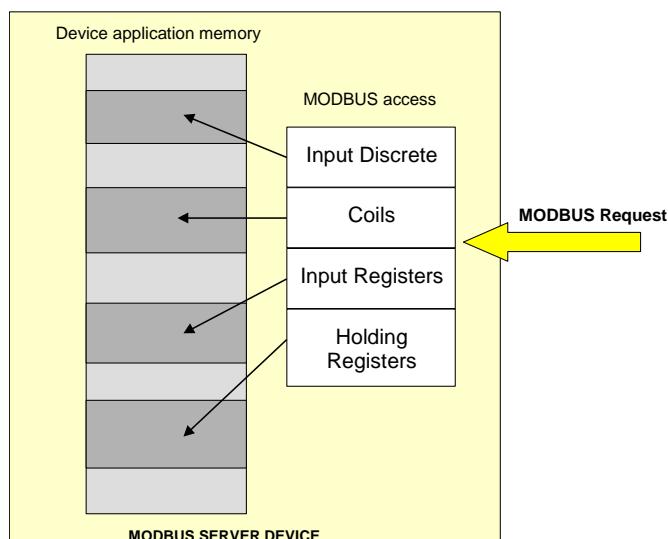


Figure 6 MODBUS Data Model with separate block

Example 2: Device having only 1 block

In this example, the device has only 1 data block. The same data can be reached via several MODBUS functions, either via a 16 bit access or via an access bit.

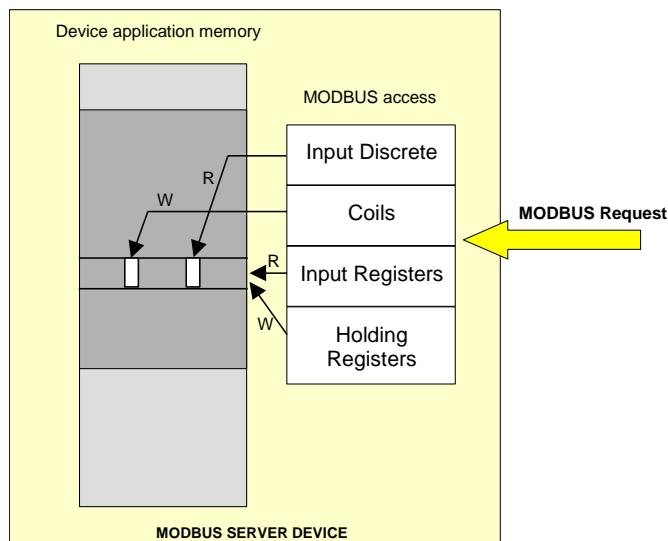


Figure 7 MODBUS Data Model with only 1 block

4.4 MODBUS Addressing model

The MODBUS application protocol defines precisely PDU addressing rules.

In a MODBUS PDU each data is addressed from 0 to 65535.

It also defines clearly a MODBUS data model composed of 4 blocks that comprises several elements numbered from 1 to n.

In the MODBUS data Model each element within a data block is numbered from 1 to n.

Afterwards the MODBUS data model has to be bound to the device application (IEC-61131 object, or other application model).

The pre-mapping between the MODBUS data model and the device application is totally vendor device specific.

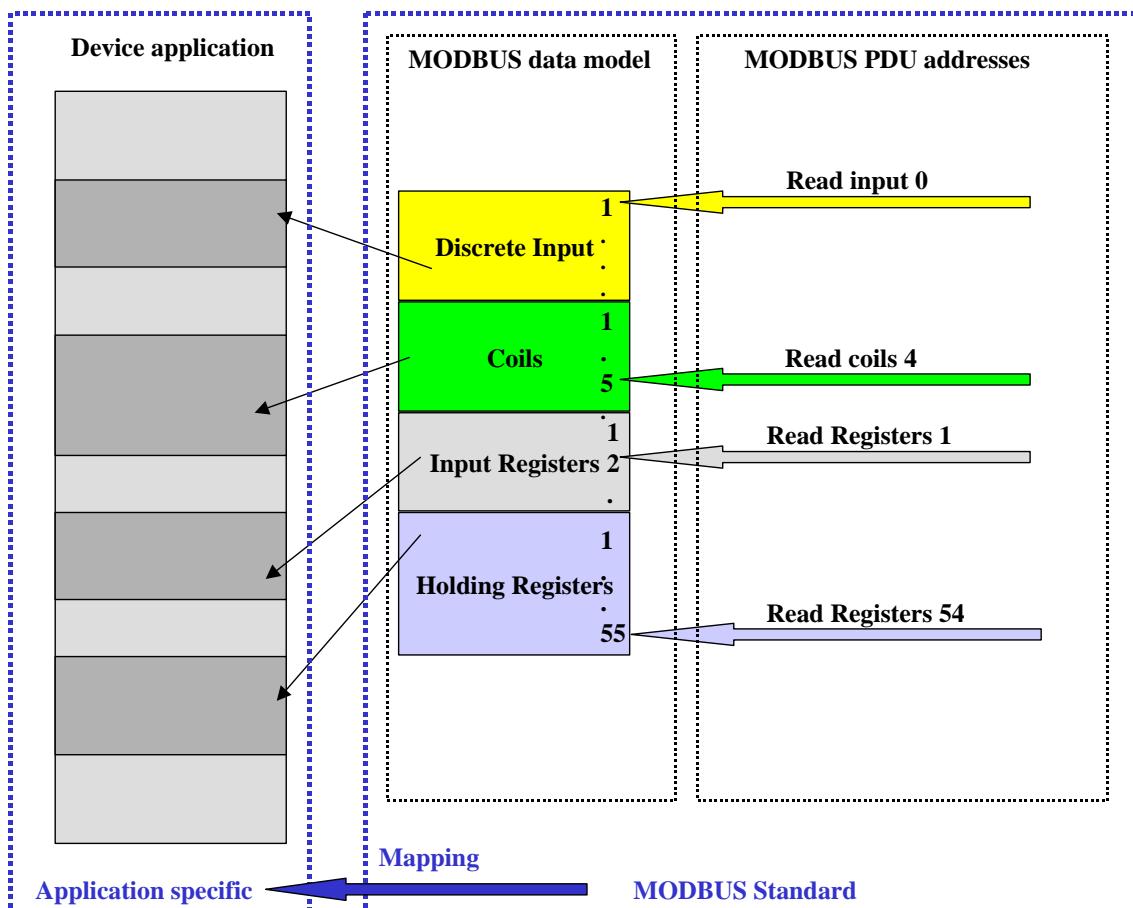


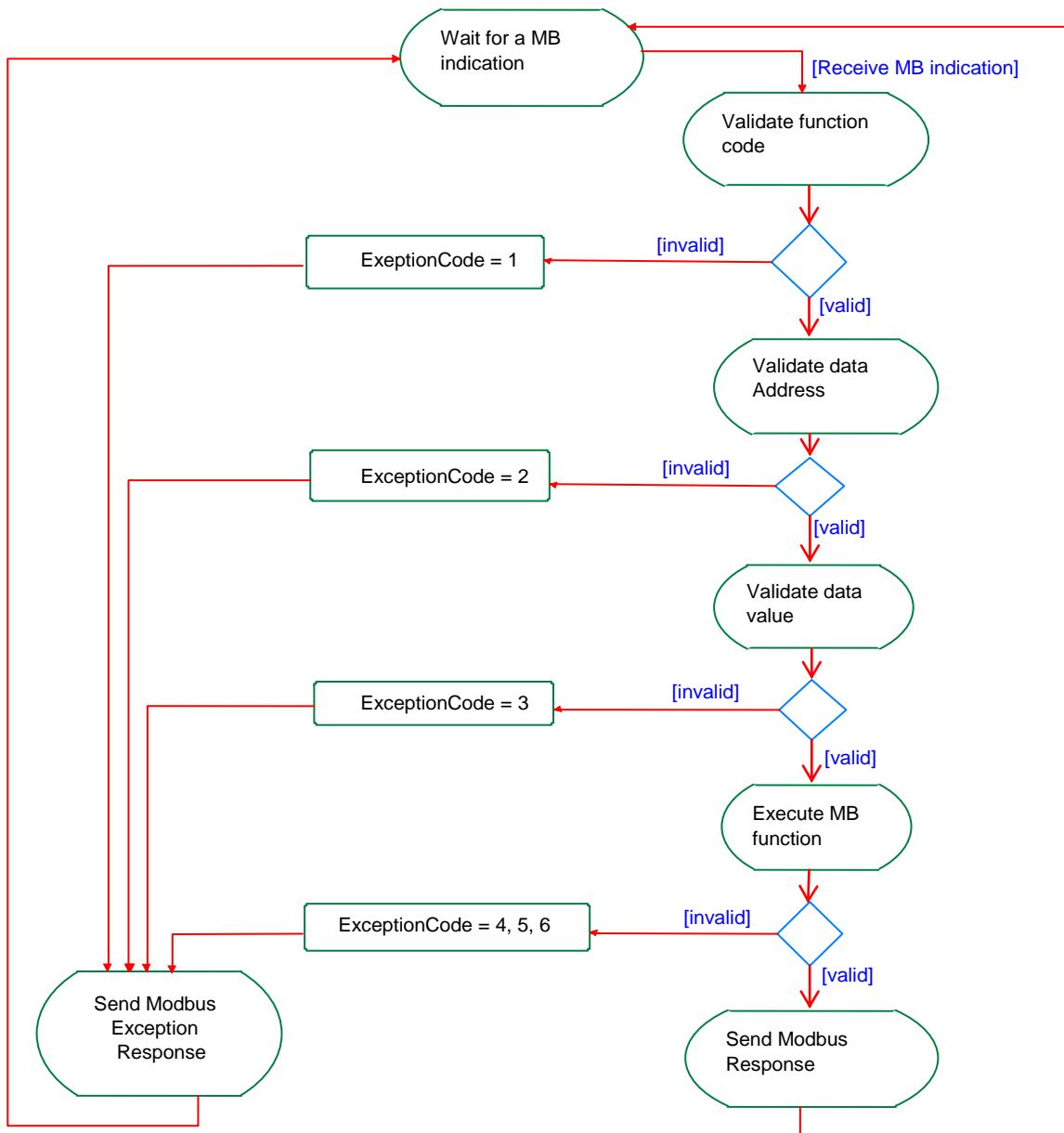
Figure 8 MODBUS Addressing model

The previous figure shows that a MODBUS data numbered X is addressed in the MODBUS PDU X-1.

4.5 Define MODBUS Transaction

The following state diagram describes the generic processing of a MODBUS transaction in server side.

V

**Figure 9 MODBUS Transaction state diagram**

Once the request has been processed by a server, a MODBUS response using the adequate MODBUS server transaction is built.

Depending on the result of the processing two types of response are built :

- A positive MODBUS response :
 - the response function code = the request function code
- A MODBUS Exception response (see section 7) :
 - the objective is to provide to the client relevant information concerning the error detected during the processing ;
 - the exception function code = the request function code + 0x80 ;
 - an exception code is provided to indicate the reason of the error.

5 Function Code Categories

There are three categories of MODBUS Functions codes. They are :

Public Function Codes

- Are well defined function codes ,
- guaranteed to be unique,
- validated by the MODBUS.org community,
- publicly documented
- have available conformance test,
- includes both defined public assigned function codes as well as unassigned function codes reserved for future use.

User-Defined Function Codes

- there are two ranges of user-defined function codes, i.e. 65 to 72 and from 100 to 110 decimal.
- user can select and implement a function code that is not supported by the specification.
- there is no guarantee that the use of the selected function code will be unique
- if the user wants to re-position the functionality as a public function code, he must initiate an RFC to introduce the change into the public category and to have a new public function code assigned.
- MODBUS Organization, Inc expressly reserves the right to develop the proposed RFC.

Reserved Function Codes

- Function Codes currently used by some companies for legacy products and that are not available for public use.
- Informative Note: The reader is asked refer to Annex A (Informative) MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES.

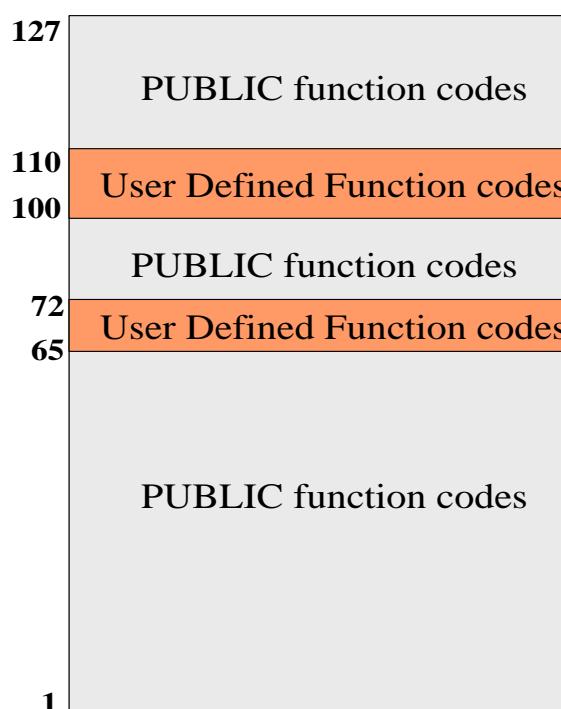


Figure 10 MODBUS Function Code Categories

5.1 Public Function Code Definition

| | | | Function Codes | | | | |
|-------------|--------------------|---|----------------------------------|----------|----------|---------|--|
| | | | code | Sub code | (hex) | Section | |
| Data Access | Bit access | Physical Discrete Inputs | Read Discrete Inputs | 02 | | 02 6.2 | |
| | | Internal Bits Or Physical coils | Read Coils | 01 | | 01 6.1 | |
| | | | Write Single Coil | 05 | | 05 6.5 | |
| | | | Write Multiple Coils | 15 | | 0F 6.11 | |
| | 16 bits access | Physical Input Registers | Read Input Register | 04 | | 04 6.4 | |
| | | | Read Holding Registers | 03 | | 03 6.3 | |
| | | Internal Registers Or Physical Output Registers | Write Single Register | 06 | | 06 6.6 | |
| | | | Write Multiple Registers | 16 | | 10 6.12 | |
| | | | Read/Write Multiple Registers | 23 | | 17 6.17 | |
| | | | Mask Write Register | 22 | | 16 6.16 | |
| | | | Read FIFO queue | 24 | | 18 6.18 | |
| | File record access | Read File record | 20 | | 14 | 6.14 | |
| | | Write File record | 21 | | 15 | 6.15 | |
| Diagnostics | | | Read Exception status | 07 | | 07 6.7 | |
| | | | Diagnostic | 08 | 00-18,20 | 08 6.8 | |
| | | | Get Com event counter | 11 | | OB 6.9 | |
| | | | Get Com Event Log | 12 | | 0C 6.10 | |
| | | | Report Server ID | 17 | | 11 6.13 | |
| | | | Read device Identification | 43 | 14 | 2B 6.21 | |
| Other | | | Encapsulated Interface Transport | 43 | 13,14 | 2B 6.19 | |
| | | | CANopen General Reference | 43 | 13 | 2B 6.20 | |

6 Function codes descriptions

6.1 01 (0x01) Read Coils

This function code is used to read from 1 to 2000 contiguous status of coils in a remote device. The Request PDU specifies the starting address, i.e. the address of the first coil specified, and the number of coils. In the PDU Coils are addressed starting at zero. Therefore coils numbered 1-16 are addressed as 0-15.

The coils in the response message are packed as one coil per bit of the data field. Status is indicated as 1= ON and 0= OFF. The LSB of the first data byte contains the output addressed in the query. The other coils follow toward the high order end of this byte, and from low order to high order in subsequent bytes.

If the returned output quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request

| | | |
|-------------------|---------|-------------------|
| Function code | 1 Byte | 0x01 |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of coils | 2 Bytes | 1 to 2000 (0x7D0) |

Response

| | | |
|---------------|--------|--------------|
| Function code | 1 Byte | 0x01 |
| Byte count | 1 Byte | N* |
| Coil Status | n Byte | n = N or N+1 |

* $N = \text{Quantity of Outputs} / 8$, if the remainder is different of 0 $\Rightarrow N = N+1$
Error

| | | |
|----------------|--------|----------------------|
| Function code | 1 Byte | Function code + 0x80 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to read discrete outputs 20–38:

| Request | | Response | |
|------------------------|-------|----------------------|-------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 01 | Function | 01 |
| Starting Address Hi | 00 | Byte Count | 03 |
| Starting Address Lo | 13 | Outputs status 27-20 | CD |
| Quantity of Outputs Hi | 00 | Outputs status 35-28 | 6B |
| Quantity of Outputs Lo | 13 | Outputs status 38-36 | 05 |

The status of outputs 27–20 is shown as the byte value CD hex, or binary 1100 1101. Output 27 is the MSB of this byte, and output 20 is the LSB.

By convention, bits within a byte are shown with the MSB to the left, and the LSB to the right. Thus the outputs in the first byte are ‘27 through 20’, from left to right. The next byte has outputs ‘35 through 28’, left to right. As the bits are transmitted serially, they flow from LSB to MSB: 20 . . . 27, 28 . . . 35, and so on.

In the last data byte, the status of outputs 38–36 is shown as the byte value 05 hex, or binary 0000 0101. Output 38 is in the sixth bit position from the left, and output 36 is the LSB of this byte. The five remaining high order bits are zero filled.



Note: The five remaining bits (toward the high order end) are zero filled.

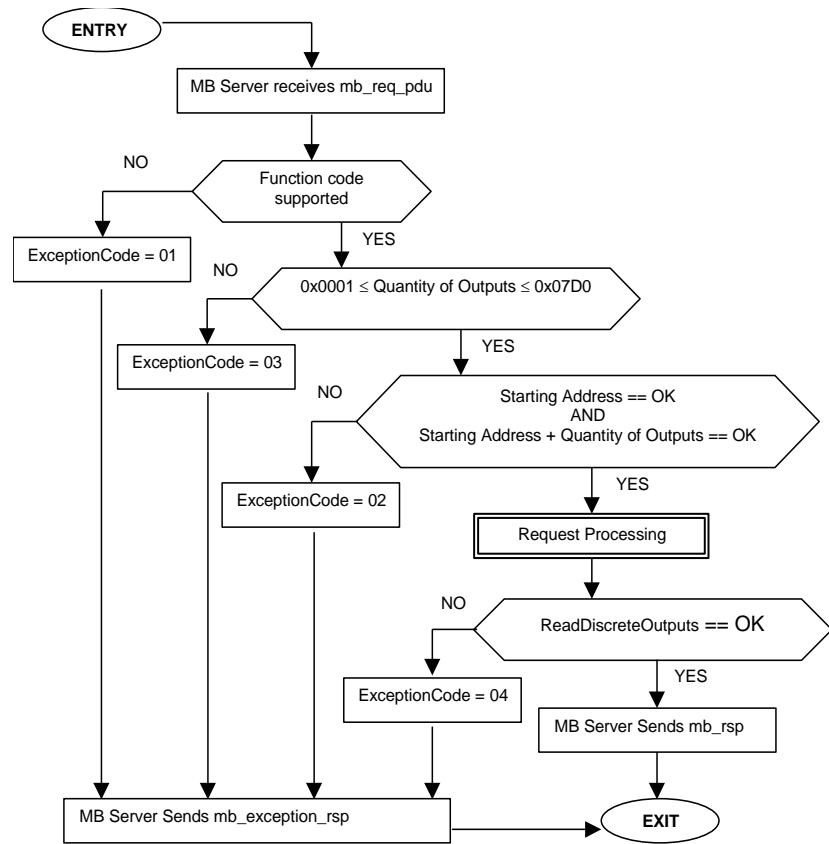


Figure 11: Read Coils state diagram

6.2 02 (0x02) Read Discrete Inputs

This function code is used to read from 1 to 2000 contiguous status of discrete inputs in a remote device. The Request PDU specifies the starting address, i.e. the address of the first

input specified, and the number of inputs. In the PDU Discrete Inputs are addressed starting at zero. Therefore Discrete inputs numbered 1-16 are addressed as 0-15.

The discrete inputs in the response message are packed as one input per bit of the data field. Status is indicated as 1= ON; 0= OFF. The LSB of the first data byte contains the input addressed in the query. The other inputs follow toward the high order end of this byte, and from low order to high order in subsequent bytes.

If the returned input quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request

| | | |
|--------------------|---------|-------------------|
| Function code | 1 Byte | 0x02 |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Inputs | 2 Bytes | 1 to 2000 (0x7D0) |

Response

| | | |
|---------------|--------------------|-------------|
| Function code | 1 Byte | 0x02 |
| Byte count | 1 Byte | N* |
| Input Status | N* x 1 Byte | |

*N = Quantity of Inputs / 8 if the remainder is different of 0 $\Rightarrow N = N+1$

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x82 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to read discrete inputs 197 – 218:

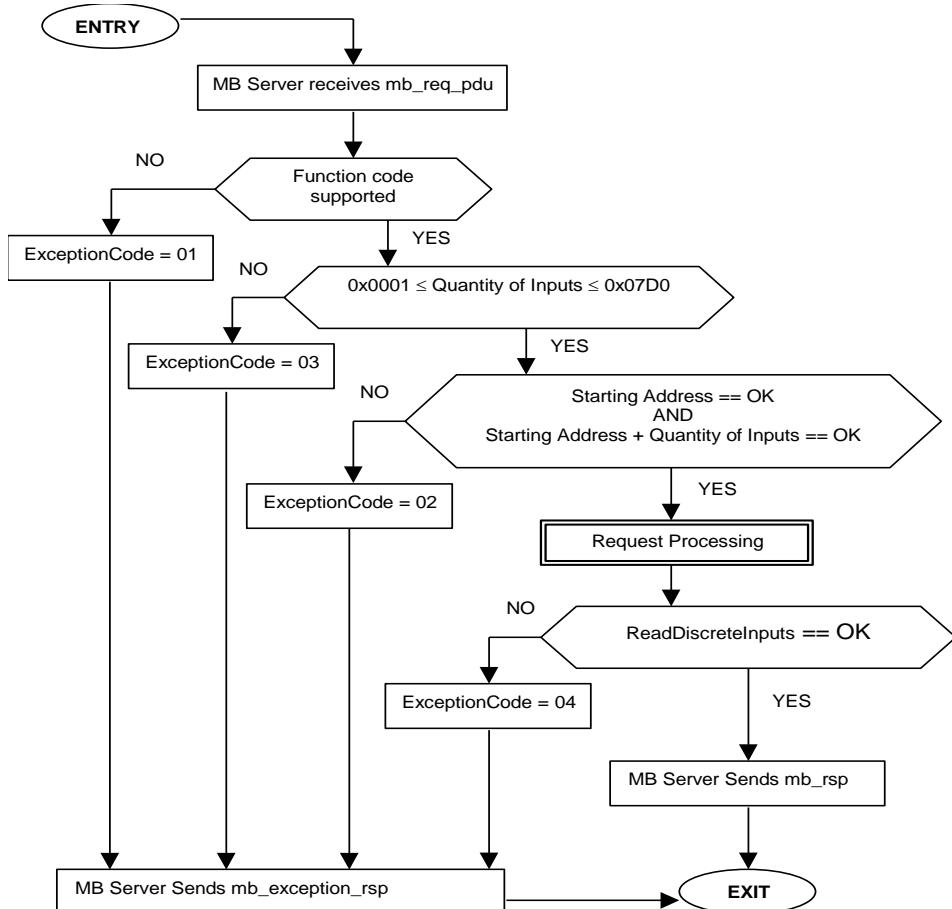
| Request | | Response | |
|-----------------------|-----------|-----------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 02 | Function | 02 |
| Starting Address Hi | 00 | Byte Count | 03 |
| Starting Address Lo | C4 | Inputs Status 204-197 | AC |
| Quantity of Inputs Hi | 00 | Inputs Status 212-205 | DB |
| Quantity of Inputs Lo | 16 | Inputs Status 218-213 | 35 |

The status of discrete inputs 204–197 is shown as the byte value AC hex, or binary 1010 1100. Input 204 is the MSB of this byte, and input 197 is the LSB.

The status of discrete inputs 218–213 is shown as the byte value 35 hex, or binary 0011 0101. Input 218 is in the third bit position from the left, and input 213 is the LSB.



Note: The two remaining bits (toward the high order end) are zero filled.

**Figure 12:** Read Discrete Inputs state diagram

6.3 03 (0x03) Read Holding Registers

This function code is used to read the contents of a contiguous block of holding registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore registers numbered 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request

| | | |
|-----------------------|---------|------------------|
| Function code | 1 Byte | 0x03 |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Registers | 2 Bytes | 1 to 125 (0x7D) |

Response

| | | |
|----------------|---------------------|---------------|
| Function code | 1 Byte | 0x03 |
| Byte count | 1 Byte | 2 x N* |
| Register value | N* x 2 Bytes | |

*N = Quantity of Registers

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x83 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to read registers 108 – 110:

| Request | | Response | |
|---------------------|-----------|-------------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 03 | Function | 03 |
| Starting Address Hi | 00 | Byte Count | 06 |
| Starting Address Lo | 6B | Register value Hi (108) | 02 |
| No. of Registers Hi | 00 | Register value Lo (108) | 2B |
| No. of Registers Lo | 03 | Register value Hi (109) | 00 |
| | | Register value Lo (109) | 00 |
| | | Register value Hi (110) | 00 |
| | | Register value Lo (110) | 64 |

The contents of register 108 are shown as the two byte values of 02 2B hex, or 555 decimal. The contents of registers 109–110 are 00 00 and 00 64 hex, or 0 and 100 decimal, respectively.

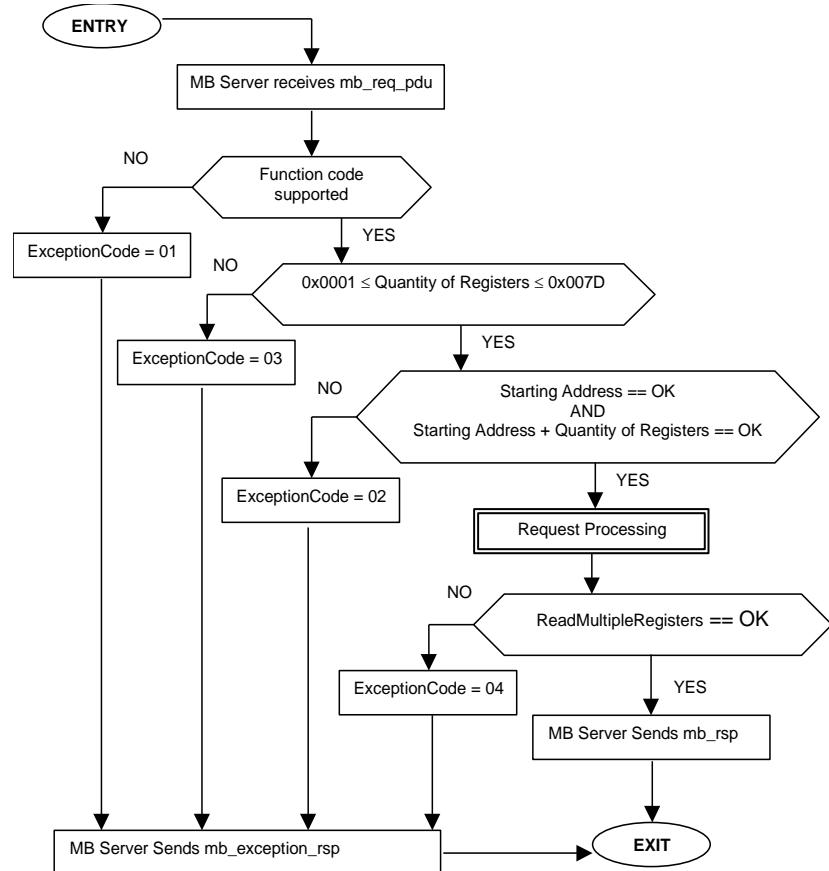


Figure 13: Read Holding Registers state diagram

6.4 04 (0x04) Read Input Registers

This function code is used to read from 1 to 125 contiguous input registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore input registers numbered 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request

| | | |
|-----------------------------|---------|------------------|
| Function code | 1 Byte | 0x04 |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Input Registers | 2 Bytes | 0x0001 to 0x007D |

Response

| | | |
|-----------------|--------------|-------------|
| Function code | 1 Byte | 0x04 |
| Byte count | 1 Byte | 2 x N* |
| Input Registers | N* x 2 Bytes | |

*N = Quantity of Input Registers

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x84 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to read input register 9:

| Request | Response | | |
|---------------------------|-----------|-----------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 04 | Function | 04 |
| Starting Address Hi | 00 | Byte Count | 02 |
| Starting Address Lo | 08 | Input Reg. 9 Hi | 00 |
| Quantity of Input Reg. Hi | 00 | Input Reg. 9 Lo | 0A |

| | | |
|---------------------------|----|--|
| Quantity of Input Reg. Lo | 01 | |
|---------------------------|----|--|

The contents of input register 9 are shown as the two byte values of 00 0A hex, or 10 decimal.

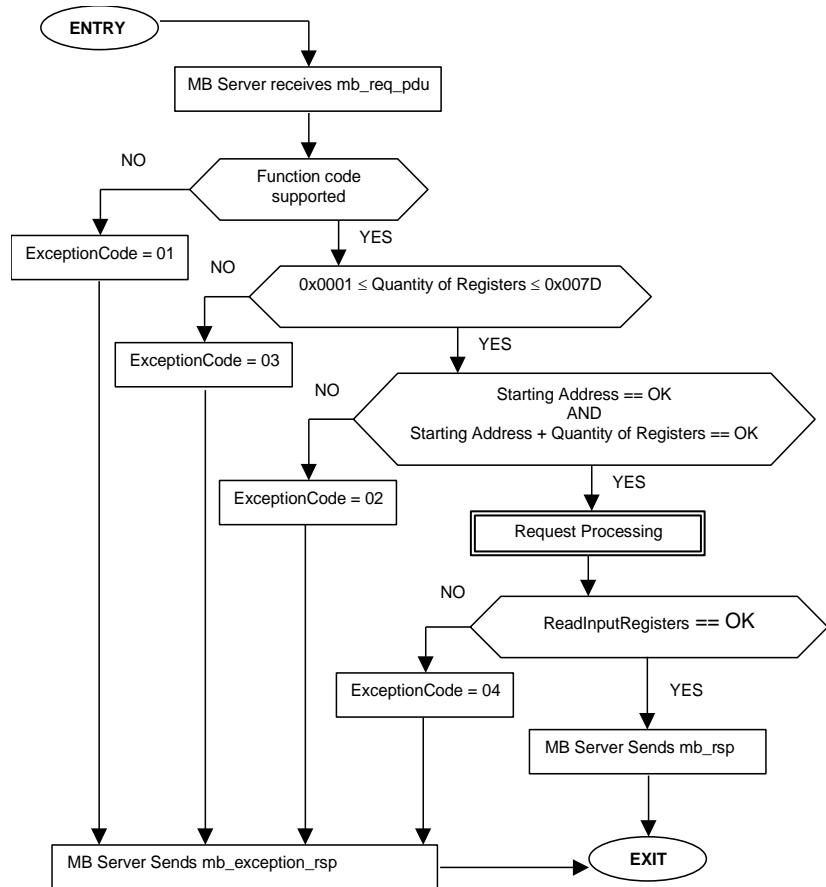


Figure 14: Read Input Registers state diagram

6.5 05 (0x05) Write Single Coil

This function code is used to write a single output to either ON or OFF in a remote device. The requested ON/OFF state is specified by a constant in the request data field. A value of FF 00 hex requests the output to be ON. A value of 00 00 requests it to be OFF. All other values are illegal and will not affect the output.

The Request PDU specifies the address of the coil to be forced. Coils are addressed starting at zero. Therefore coil numbered 1 is addressed as 0. The requested ON/OFF state is specified by a constant in the Coil Value field. A value of 0XFF00 requests the coil to be ON. A value of 0X0000 requests the coil to be off. All other values are illegal and will not affect the coil.

The normal response is an echo of the request, returned after the coil state has been written.

Request

| | | |
|----------------|---------|------------------|
| Function code | 1 Byte | 0x05 |
| Output Address | 2 Bytes | 0x0000 to 0xFFFF |
| Output Value | 2 Bytes | 0x0000 or 0xFF00 |

Response

| | | |
|----------------|---------|------------------|
| Function code | 1 Byte | 0x05 |
| Output Address | 2 Bytes | 0x0000 to 0xFFFF |

| | | |
|--------------|---------|------------------|
| Output Value | 2 Bytes | 0x0000 or 0xFF00 |
|--------------|---------|------------------|

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x85 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to write Coil 173 ON:

| Request | | Response | |
|-------------------|-----------|-------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 05 | Function | 05 |
| Output Address Hi | 00 | Output Address Hi | 00 |
| Output Address Lo | AC | Output Address Lo | AC |
| Output Value Hi | FF | Output Value Hi | FF |
| Output Value Lo | 00 | Output Value Lo | 00 |

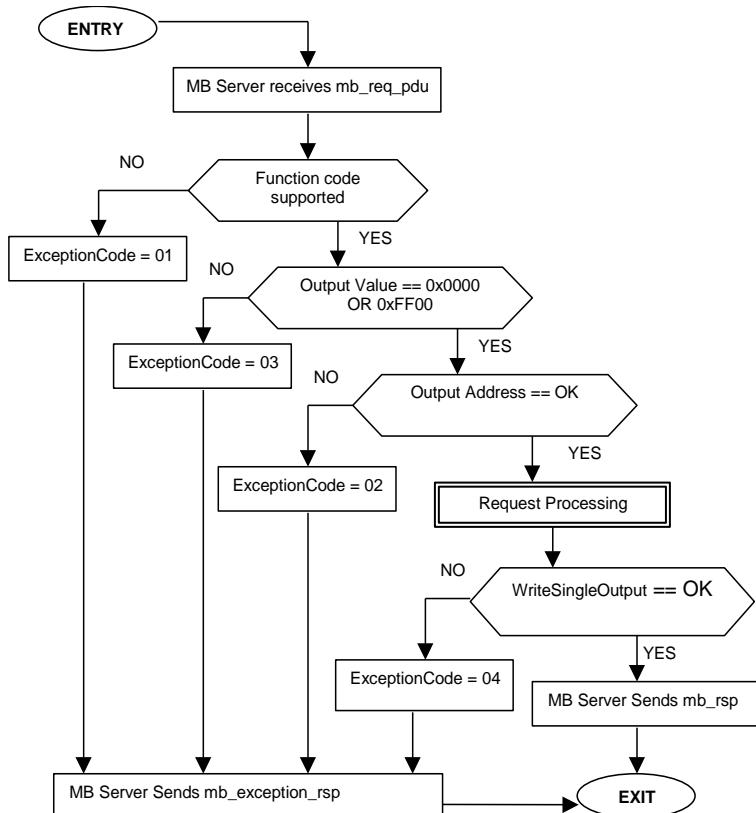


Figure 15: Write Single Output state diagram

6.6 06 (0x06) Write Single Register

This function code is used to write a single holding register in a remote device.

The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register numbered 1 is addressed as 0.

The normal response is an echo of the request, returned after the register contents have been written.

Request

| | | |
|------------------|---------|------------------|
| Function code | 1 Byte | 0x06 |
| Register Address | 2 Bytes | 0x0000 to 0xFFFF |
| Register Value | 2 Bytes | 0x0000 to 0xFFFF |

Response

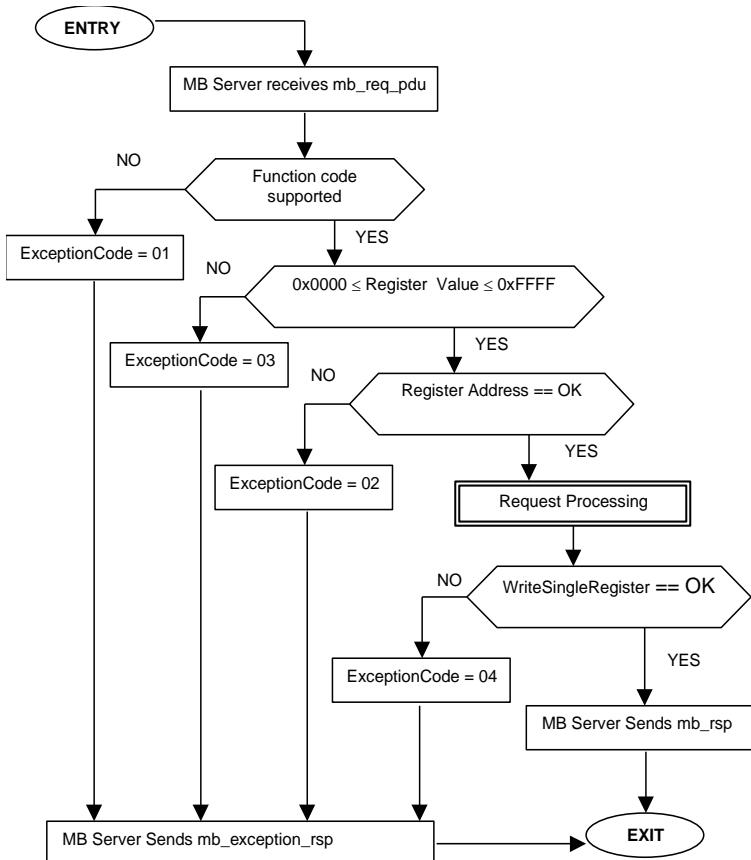
| | | |
|------------------|---------|------------------|
| Function code | 1 Byte | 0x06 |
| Register Address | 2 Bytes | 0x0000 to 0xFFFF |
| Register Value | 2 Bytes | 0x0000 to 0xFFFF |

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x86 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to write register 2 to 00 03 hex:

| Request | Response | | |
|---------------------|-----------|---------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 06 | Function | 06 |
| Register Address Hi | 00 | Register Address Hi | 00 |
| Register Address Lo | 01 | Register Address Lo | 01 |
| Register Value Hi | 00 | Register Value Hi | 00 |
| Register Value Lo | 03 | Register Value Lo | 03 |

**Figure 16:** Write Single Register state diagram

6.7 07 (0x07) Read Exception Status (Serial Line only)

This function code is used to read the contents of eight Exception Status outputs in a remote device.

The function provides a simple method for accessing this information, because the Exception Output references are known (no output reference is needed in the function).

The normal response contains the status of the eight Exception Status outputs. The outputs are packed into one data byte, with one bit per output. The status of the lowest output reference is contained in the least significant bit of the byte.

The contents of the eight Exception Status outputs are device specific.

Request

| | | |
|---------------|--------|-------------|
| Function code | 1 Byte | 0x07 |
|---------------|--------|-------------|

Response

| | | |
|---------------|--------|--------------|
| Function code | 1 Byte | 0x07 |
| Output Data | 1 Byte | 0x00 to 0xFF |

Error

| | | |
|----------------|--------|-------------|
| Error code | 1 Byte | 0x87 |
| Exception code | 1 Byte | 01 or 04 |

Here is an example of a request to read the exception status:

| Request | | Response | |
|------------|-----------|-------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 07 | Function | 07 |
| | | Output Data | 6D |

In this example, the output data is 6D hex (0110 1101 binary). Left to right, the outputs are OFF–ON–ON–OFF–ON–ON–ON–OFF–ON. The status is shown from the highest to the lowest addressed output.

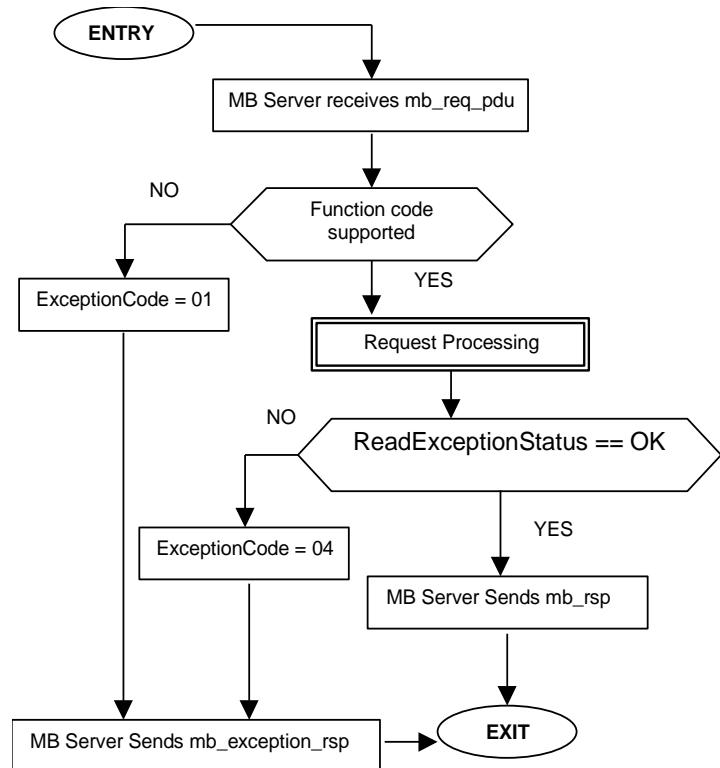


Figure 17: Read Exception Status state diagram

6.8 08 (0x08) Diagnostics (Serial Line only)

MODBUS function code 08 provides a series of tests for checking the communication system between a client device and a server, or for checking various internal error conditions within a server.

The function uses a two-byte sub-function code field in the query to define the type of test to be performed. The server echoes both the function code and sub-function code in a normal response. Some of the diagnostics cause data to be returned from the remote device in the data field of a normal response.

In general, issuing a diagnostic function to a remote device does not affect the running of the user program in the remote device. User logic, like discrete and registers, is not accessed by the diagnostics. Certain functions can optionally reset error counters in the remote device.

A server device can, however, be forced into ‘Listen Only Mode’ in which it will monitor the messages on the communications system but not respond to them. This can affect the outcome of your application program if it depends upon any further exchange of data with the remote device. Generally, the mode is forced to remove a malfunctioning remote device from the communications system.

The following diagnostic functions are dedicated to serial line devices.

The normal response to the Return Query Data request is to loopback the same data. The function code and sub-function codes are also echoed.

Request

| | | |
|---------------|-------------|------|
| Function code | 1 Byte | 0x08 |
| Sub-function | 2 Bytes | |
| Data | N x 2 Bytes | |

Response

| | | |
|---------------|-------------|-------------|
| Function code | 1 Byte | 0x08 |
| Sub-function | 2 Bytes | |
| Data | N x 2 Bytes | |

Error

| | | |
|----------------|--------|----------------|
| Error code | 1 Byte | 0x88 |
| Exception code | 1 Byte | 01 or 03 or 04 |

6.8.1 Sub-function codes supported by the serial line devices

Here the list of sub-function codes supported by the serial line devices. Each sub-function code is then listed with an example of the data field contents that would apply for that diagnostic.

| Sub-function code | Name | |
|-------------------|--------------|--|
| Hex | Dec | |
| 00 | 00 | Return Query Data |
| 01 | 01 | Restart Communications Option |
| 02 | 02 | Return Diagnostic Register |
| 03 | 03 | Change ASCII Input Delimiter |
| 04 | 04 | Force Listen Only Mode |
| 05.. 09 | | RESERVED |
| 0A | 10 | Clear Counters and Diagnostic Register |
| 0B | 11 | Return Bus Message Count |
| 0C | 12 | Return Bus Communication Error Count |
| 0D | 13 | Return Bus Exception Error Count |
| 0E | 14 | Return Server Message Count |
| 0F | 15 | Return Server No Response Count |
| 10 | 16 | Return Server NAK Count |
| 11 | 17 | Return Server Busy Count |
| 12 | 18 | Return Bus Character Overrun Count |
| 13 | 19 | RESERVED |
| 14 | 20 | Clear Overrun Counter and Flag |
| N.A. | 21 ... 65535 | RESERVED |

00 Return Query Data

The data passed in the request data field is to be returned (looped back) in the response. The entire response message should be identical to the request.

| Sub-function | Data Field (Request) | Data Field (Response) |
|--------------|----------------------|-----------------------|
| 00 00 | Any | Echo Request Data |

01 Restart Communications Option

The remote device serial line port must be initialized and restarted, and all of its communications event counters are cleared. If the port is currently in Listen Only Mode, no response is returned. This function is the only one that brings the port out of Listen Only Mode. If the port is not currently in Listen Only Mode, a normal response is returned. This occurs before the restart is executed.

When the remote device receives the request, it attempts a restart and executes its power-up confidence tests. Successful completion of the tests will bring the port online.

A request data field contents of FF 00 hex causes the port's Communications Event Log to be cleared also. Contents of 00 00 leave the log as it was prior to the restart.

| Sub-function | Data Field (Request) | Data Field (Response) |
|--------------|----------------------|-----------------------|
| 00 01 | 00 00 | Echo Request Data |
| 00 01 | FF 00 | Echo Request Data |

02 Return Diagnostic Register

The contents of the remote device's 16-bit diagnostic register are returned in the response.

| Sub-function | Data Field (Request) | Data Field (Response) |
|--------------|----------------------|------------------------------|
| 00 02 | 00 00 | Diagnostic Register Contents |

03 Change ASCII Input Delimiter

The character 'CHAR' passed in the request data field becomes the end of message delimiter for future messages (replacing the default LF character). This function is useful in cases of a Line Feed is not required at the end of ASCII messages.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 03 | CHAR 00 | Echo Request Data |

04 Force Listen Only Mode

Forces the addressed remote device to its Listen Only Mode for MODBUS communications. This isolates it from the other devices on the network, allowing them to continue communicating without interruption from the addressed remote device. No response is returned.

When the remote device enters its Listen Only Mode, all active communication controls are turned off. The Ready watchdog timer is allowed to expire, locking the controls off. While the device is in this mode, any MODBUS messages addressed to it or broadcast are monitored, but no actions will be taken and no responses will be sent.

The only function that will be processed after the mode is entered will be the Restart Communications Option function (function code 8, sub-function 1).

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 04 | 00 00 | No Response Returned |

10 (0A Hex) Clear Counters and Diagnostic Register

The goal is to clear all counters and the diagnostic register. Counters are also cleared upon power-up.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 0A | 00 00 | Echo Request Data |

11 (0B Hex) Return Bus Message Count

The response data field returns the quantity of messages that the remote device has detected on the communications system since its last restart, clear counters operation, or power-up.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 0B | 00 00 | Total Message Count |

12 (0C Hex) Return Bus Communication Error Count

The response data field returns the quantity of CRC errors encountered by the remote device since its last restart, clear counters operation, or power-up.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 0C | 00 00 | CRC Error Count |

13 (0D Hex) Return Bus Exception Error Count

The response data field returns the quantity of MODBUS exception responses returned by the remote device since its last restart, clear counters operation, or power-up.

Exception responses are described and listed in section 7 .

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 0D | 00 00 | Exception Error Count |

14 (0E Hex) Return Server Message Count

The response data field returns the quantity of messages addressed to the remote device, or broadcast, that the remote device has processed since its last restart, clear counters operation, or power-up.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 0E | 00 00 | Server Message Count |

15 (0F Hex) Return Server No Response Count

The response data field returns the quantity of messages addressed to the remote device for which it has returned no response (neither a normal response nor an exception response), since its last restart, clear counters operation, or power-up.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 0F | 00 00 | Server No Response Count |

16 (10 Hex) Return Server NAK Count

The response data field returns the quantity of messages addressed to the remote device for which it returned a Negative Acknowledge (NAK) exception response, since its last restart, clear counters operation, or power-up. Exception responses are described and listed in section 7 .

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 10 | 00 00 | Server NAK Count |

17 (11 Hex) Return Server Busy Count

The response data field returns the quantity of messages addressed to the remote device for which it returned a Server Device Busy exception response, since its last restart, clear counters operation, or power-up.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 11 | 00 00 | Server Device Busy Count |

18 (12 Hex) Return Bus Character Overrun Count

The response data field returns the quantity of messages addressed to the remote device that it could not handle due to a character overrun condition, since its last restart, clear counters operation, or power-up. A character overrun is caused by data characters arriving at the port faster than they can be stored, or by the loss of a character due to a hardware malfunction.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|--------------------------------|
| 00 12 | 00 00 | Server Character Overrun Count |

20 (14 Hex) Clear Overrun Counter and Flag

Clears the overrun error counter and reset the error flag.

| Sub-function | Data Field (Request) | Data Field (Response) |
|---------------------|-----------------------------|------------------------------|
| 00 14 | 00 00 | Echo Request Data |

6.8.2 Example and state diagram

Here is an example of a request to remote device to Return Query Data. This uses a sub-function code of zero (00 00 hex in the two-byte field). The data to be returned is sent in the two-byte data field (A5 37 hex).

| Request | | Response | |
|-------------------|-------|-------------------|-------|
| <i>Field Name</i> | (Hex) | <i>Field Name</i> | (Hex) |
| Function | 08 | Function | 08 |
| Sub-function Hi | 00 | Sub-function Hi | 00 |
| Sub-function Lo | 00 | Sub-function Lo | 00 |
| Data Hi | A5 | Data Hi | A5 |
| Data Lo | 37 | Data Lo | 37 |

The data fields in responses to other kinds of queries could contain error counts or other data requested by the sub-function code.

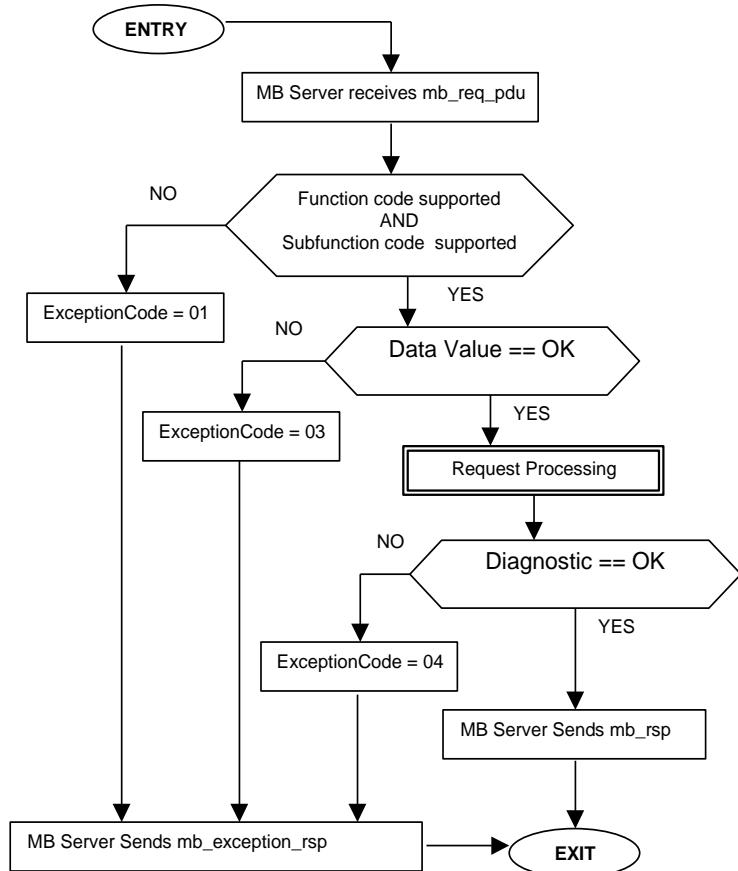


Figure 18: Diagnostic state diagram

6.9 11 (0x0B) Get Comm Event Counter (Serial Line only)

This function code is used to get a status word and an event count from the remote device's communication event counter.

By fetching the current count before and after a series of messages, a client can determine whether the messages were handled normally by the remote device.

The device's event counter is incremented once for each successful message completion. It is not incremented for exception responses, poll commands, or fetch event counter commands.

The event counter can be reset by means of the Diagnostics function (code 08), with a sub-function of Restart Communications Option (code 00 01) or Clear Counters and Diagnostic Register (code 00 0A).

The normal response contains a two-byte status word, and a two-byte event count. The status word will be all ones (FF FF hex) if a previously-issued program command is still being processed by the remote device (a busy condition exists). Otherwise, the status word will be all zeros.

Request

| | | |
|---------------|--------|-------------|
| Function code | 1 Byte | 0x0B |
|---------------|--------|-------------|

Response

| | | |
|---------------|---------|------------------|
| Function code | 1 Byte | 0x0B |
| Status | 2 Bytes | 0x0000 to 0xFFFF |
| Event Count | 2 Bytes | 0x0000 to 0xFFFF |

Error

| | | |
|----------------|--------|-------------|
| Error code | 1 Byte | 0x8B |
| Exception code | 1 Byte | 01 or 04 |

Here is an example of a request to get the communications event counter in remote device:

| Request | Response |
|---------|----------|
|---------|----------|

| Field Name | (Hex) | Field Name | (Hex) |
|------------|-------|----------------|-------|
| Function | 0B | Function | 0B |
| | | Status Hi | FF |
| | | Status Lo | FF |
| | | Event Count Hi | 01 |
| | | Event Count Lo | 08 |

In this example, the status word is FF FF hex, indicating that a program function is still in progress in the remote device. The event count shows that 264 (01 08 hex) events have been counted by the device.

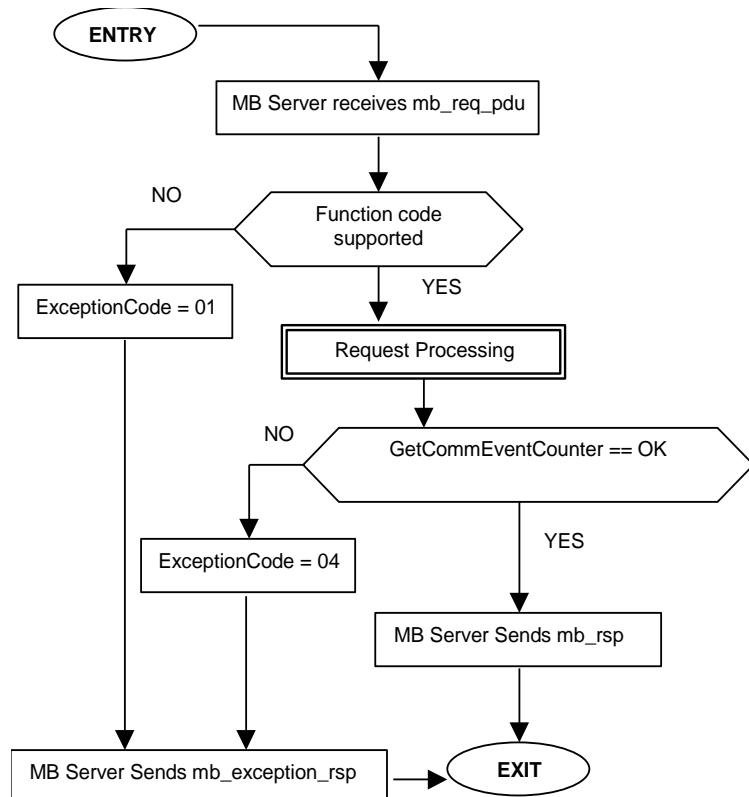


Figure 19: Get Comm Event Counter state diagram

6.10 12 (0x0C) Get Comm Event Log (Serial Line only)

This function code is used to get a status word, event count, message count, and a field of event bytes from the remote device.

The status word and event counts are identical to that returned by the Get Communications Event Counter function (11, 0B hex).

The message counter contains the quantity of messages processed by the remote device since its last restart, clear counters operation, or power-up. This count is identical to that returned by the Diagnostic function (code 08), sub-function Return Bus Message Count (code 11, 0B hex).

The event bytes field contains 0-64 bytes, with each byte corresponding to the status of one MODBUS send or receive operation for the remote device. The remote device enters the events into the field in chronological order. Byte 0 is the most recent event. Each new byte flushes the oldest byte from the field.

The normal response contains a two-byte status word field, a two-byte event count field, a two-byte message count field, and a field containing 0-64 bytes of events. A byte count field defines the total length of the data in these four fields.

Request

| | | |
|---------------|--------|-------------|
| Function code | 1 Byte | 0x0C |
|---------------|--------|-------------|

Response

| | | |
|---------------|----------------|------------------|
| Function code | 1 Byte | 0x0C |
| Byte Count | 1 Byte | N* |
| Status | 2 Bytes | 0x0000 to 0xFFFF |
| Event Count | 2 Bytes | 0x0000 to 0xFFFF |
| Message Count | 2 Bytes | 0x0000 to 0xFFFF |
| Events | (N-6) x 1 Byte | |

*N = Quantity of Events + 3 x 2 Bytes, (Length of Status, Event Count and Message Count)

Error

| | | |
|----------------|--------|-------------|
| Error code | 1 Byte | 0x8C |
| Exception code | 1 Byte | 01 or 04 |

Here is an example of a request to get the communications event log in remote device:

| Request | | Response | |
|------------|-----------|------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 0C | Function | 0C |
| | | Byte Count | 08 |
| | | Status Hi | 00 |
| | | Status Lo | 00 |
| | | Event Count Hi | 01 |
| | | Event Count Lo | 08 |
| | | Message Count Hi | 01 |
| | | Message Count Lo | 21 |
| | | Event 0 | 20 |
| | | Event 1 | 00 |

In this example, the status word is 00 00 hex, indicating that the remote device is not processing a program function. The event count shows that 264 (01 08 hex) events have been counted by the remote device. The message count shows that 289 (01 21 hex) messages have been processed.

The most recent communications event is shown in the Event 0 byte. Its content (20 hex) show that the remote device has most recently entered the Listen Only Mode.

The previous event is shown in the Event 1 byte. Its contents (00 hex) show that the remote device received a Communications Restart.

The layout of the response's event bytes is described below.

What the Event Bytes Contain

An event byte returned by the Get Communications Event Log function can be any one of four types. The type is defined by bit 7 (the high-order bit) in each byte. It may be further defined by bit 6. This is explained below.

- **Remote device MODBUS Receive Event**

The remote device stores this type of event byte when a query message is received. It is stored before the remote device processes the message. This event is defined by bit 7 set to logic '1'. The other bits will be set to a logic '1' if the corresponding condition is TRUE. The bit layout is:

| Bit | Contents |
|-----|---------------------|
| 0 | Not Used |
| 1 | Communication Error |
| 2 | Not Used |
| 3 | Not Used |
| 4 | Character Overrun |

- 5 Currently in Listen Only Mode
- 6 Broadcast Received
- 7 1

- **Remote device MODBUS Send Event**

The remote device stores this type of event byte when it finishes processing a request message. It is stored if the remote device returned a normal or exception response, or no response. This event is defined by bit 7 set to a logic '0', with bit 6 set to a '1'. The other bits will be set to a logic '1' if the corresponding condition is TRUE. The bit layout is:

| Bit | Contents |
|------------|--|
| 0 | Read Exception Sent (Exception Codes 1-3) |
| 1 | Server Abort Exception Sent (Exception Code 4) |
| 2 | Server Busy Exception Sent (Exception Codes 5-6) |
| 3 | Server Program NAK Exception Sent (Exception Code 7) |
| 4 | Write Timeout Error Occurred |
| 5 | Currently in Listen Only Mode |
| 6 | 1 |
| 7 | 0 |

- **Remote device Entered Listen Only Mode**

The remote device stores this type of event byte when it enters the Listen Only Mode. The event is defined by a content of 04 hex.

- **Remote device Initiated Communication Restart**

The remote device stores this type of event byte when its communications port is restarted. The remote device can be restarted by the Diagnostics function (code 08), with sub-function Restart Communications Option (code 00 01).

That function also places the remote device into a 'Continue on Error' or 'Stop on Error' mode. If the remote device is placed into 'Continue on Error' mode, the event byte is added to the existing event log. If the remote device is placed into 'Stop on Error' mode, the byte is added to the log and the rest of the log is cleared to zeros.

The event is defined by a content of zero.

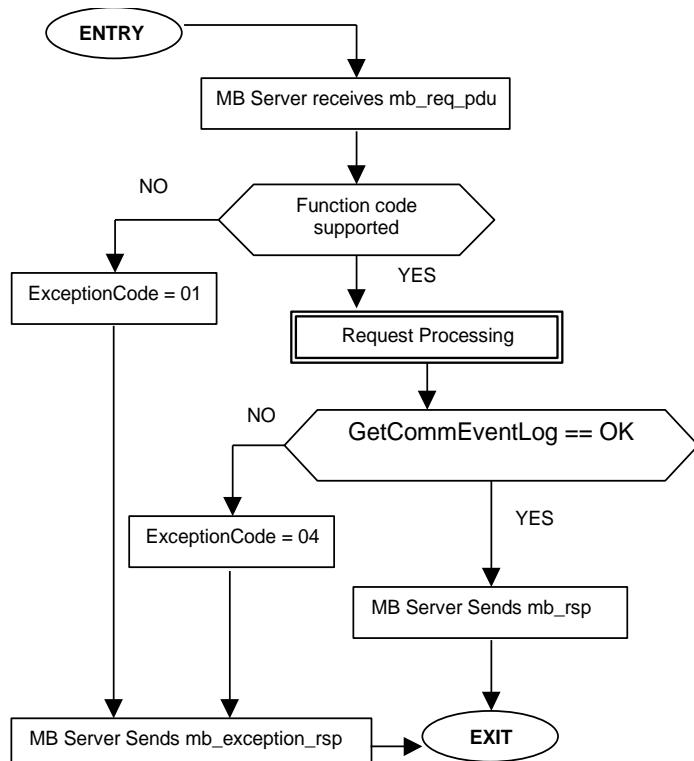


Figure 20: Get Comm Event Log state diagram

6.11 15 (0x0F) Write Multiple Coils

This function code is used to force each coil in a sequence of coils to either ON or OFF in a remote device. The Request PDU specifies the coil references to be forced. Coils are addressed starting at zero. Therefore coil numbered 1 is addressed as 0.

The requested ON/OFF states are specified by contents of the request data field. A logical '1' in a bit position of the field requests the corresponding output to be ON. A logical '0' requests it to be OFF.

The normal response returns the function code, starting address, and quantity of coils forced.

Request PDU

| | | |
|---------------------|--------------------|------------------|
| Function code | 1 Byte | 0x0F |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Outputs | 2 Bytes | 0x0001 to 0x07B0 |
| Byte Count | 1 Byte | N* |
| Outputs Value | N* x 1 Byte | |

*N = Quantity of Outputs / 8, if the remainder is different of 0 \Rightarrow N = N+1

Response PDU

| | | |
|---------------------|---------|------------------|
| Function code | 1 Byte | 0x0F |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Outputs | 2 Bytes | 0x0001 to 0x07B0 |

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x8F |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to write a series of 10 coils starting at coil 20:

The request data contents are two bytes: CD 01 hex (1100 1101 0000 0001 binary). The binary bits correspond to the outputs in the following way:

Bit: 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1
Output: 27 26 25 24 23 22 21 20 - - - - - - 29 28

The first byte transmitted (CD hex) addresses outputs 27-20, with the least significant bit addressing the lowest output (20) in this set.

The next byte transmitted (01 hex) addresses outputs 29-28, with the least significant bit addressing the lowest output (28) in this set. Unused bits in the last data byte should be zero-filled.

| Request | | Response | |
|------------------------|-----------|------------------------|-----------|
| <i>Field Name</i> | (Hex) | <i>Field Name</i> | (Hex) |
| Function | 0F | Function | 0F |
| Starting Address Hi | 00 | Starting Address Hi | 00 |
| Starting Address Lo | 13 | Starting Address Lo | 13 |
| Quantity of Outputs Hi | 00 | Quantity of Outputs Hi | 00 |
| Quantity of Outputs Lo | 0A | Quantity of Outputs Lo | 0A |
| Byte Count | 02 | | |
| Outputs Value Hi | CD | | |
| Outputs Value Lo | 01 | | |

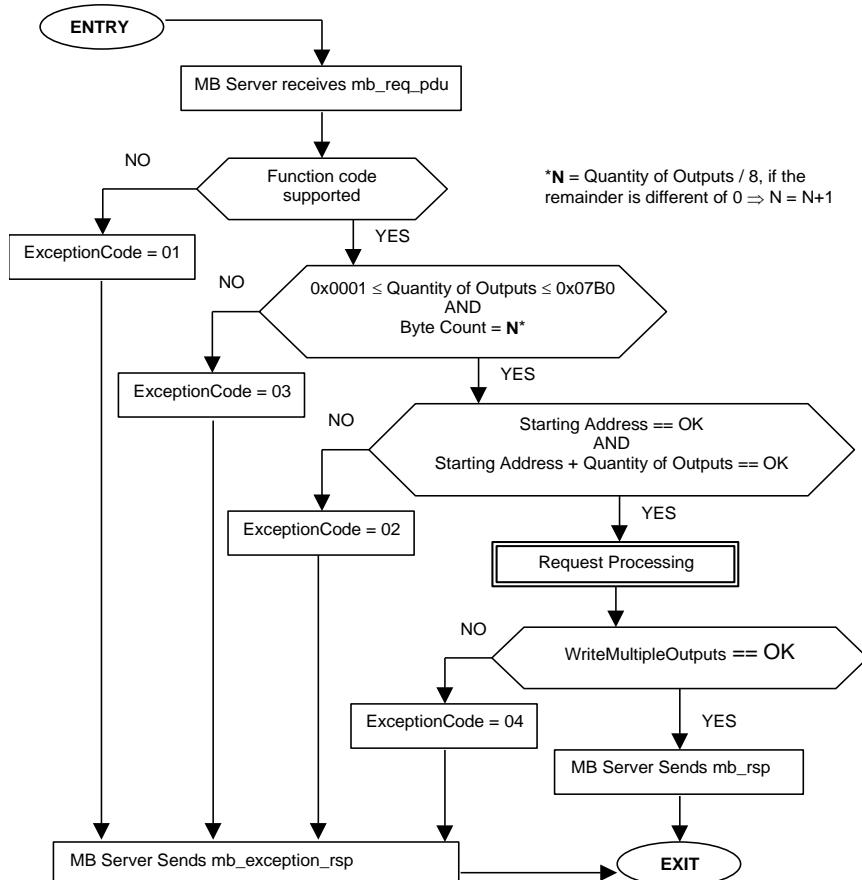


Figure 21: Write Multiple Outputs state diagram

6.12 16 (0x10) Write Multiple registers

This function code is used to write a block of contiguous registers (1 to 123 registers) in a remote device.

The requested written values are specified in the request data field. Data is packed as two bytes per register.

The normal response returns the function code, starting address, and quantity of registers written.

Request

| | | |
|-----------------------|---------------------|------------------|
| Function code | 1 Byte | 0x10 |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Registers | 2 Bytes | 0x0001 to 0x007B |
| Byte Count | 1 Byte | 2 x N* |
| Registers Value | N* x 2 Bytes | value |

*N = Quantity of Registers

Response

| | | |
|-----------------------|---------|------------------|
| Function code | 1 Byte | 0x10 |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of Registers | 2 Bytes | 1 to 123 (0x7B) |

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x90 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to write two registers starting at 2 to 00 0A and 01 02 hex:

| Request | Response | | |
|---------------------|-----------|---------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 10 | Function | 10 |
| Starting Address Hi | 00 | Starting Address Hi | 00 |
| Starting Address Lo | 01 | Starting Address Lo | 01 |

| | | | |
|--------------------------|----|--------------------------|----|
| Quantity of Registers Hi | 00 | Quantity of Registers Hi | 00 |
| Quantity of Registers Lo | 02 | Quantity of Registers Lo | 02 |
| Byte Count | 04 | | |
| Registers Value Hi | 00 | | |
| Registers Value Lo | 0A | | |
| Registers Value Hi | 01 | | |
| Registers Value Lo | 02 | | |

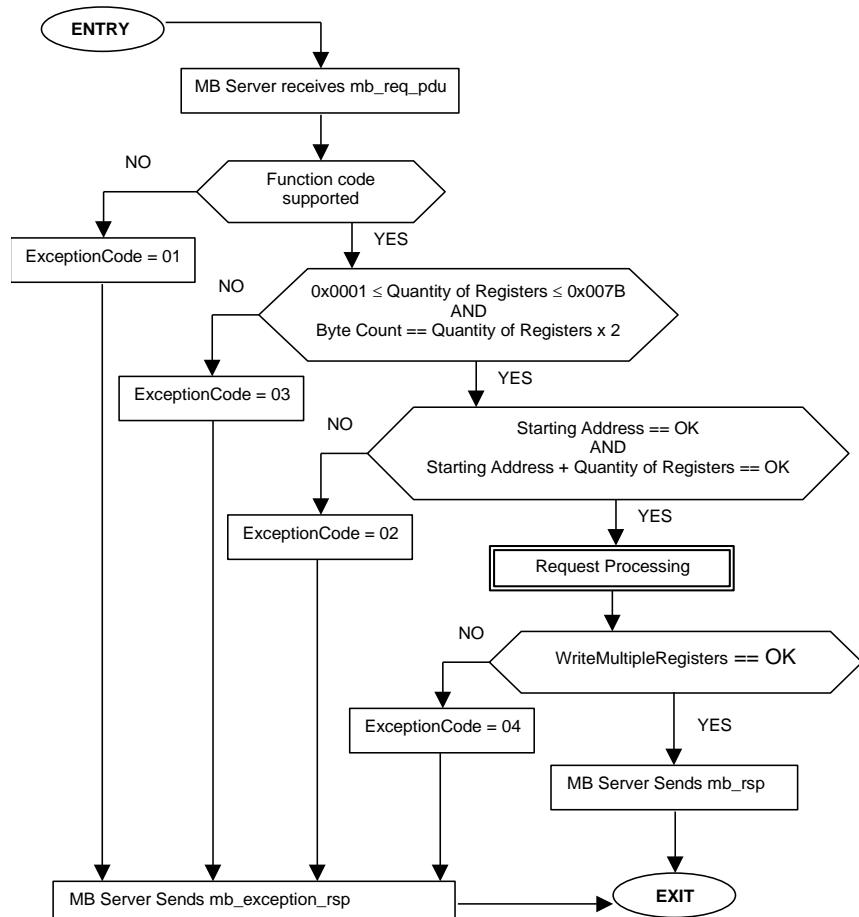


Figure 22: Write Multiple Registers state diagram

6.13 17 (0x11) Report Server ID (Serial Line only)

This function code is used to read the description of the type, the current status, and other information specific to a remote device.

The format of a normal response is shown in the following example. The data contents are specific to each type of device.

Request

| | | |
|---------------|--------|------|
| Function code | 1 Byte | 0x11 |
|---------------|--------|------|

Response

| | | |
|----------------------|-----------------|-----------------------|
| Function code | 1 Byte | 0x11 |
| Byte Count | 1 Byte | |
| Server ID | device specific | |
| Run Indicator Status | 1 Byte | 0x00 = OFF, 0xFF = ON |
| Additional Data | | |

Error

| | | |
|----------------|--------|----------|
| Error code | 1 Byte | 0x91 |
| Exception code | 1 Byte | 01 or 04 |

Here is an example of a request to report the ID and status:

| Request | | Response | |
|------------|-------|----------------------|-----------------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 11 | Function | 11 |
| | | Byte Count | Device Specific |
| | | Server ID | Device Specific |
| | | Run Indicator Status | 0x00 or 0xFF |
| | | Additional Data | Device Specific |

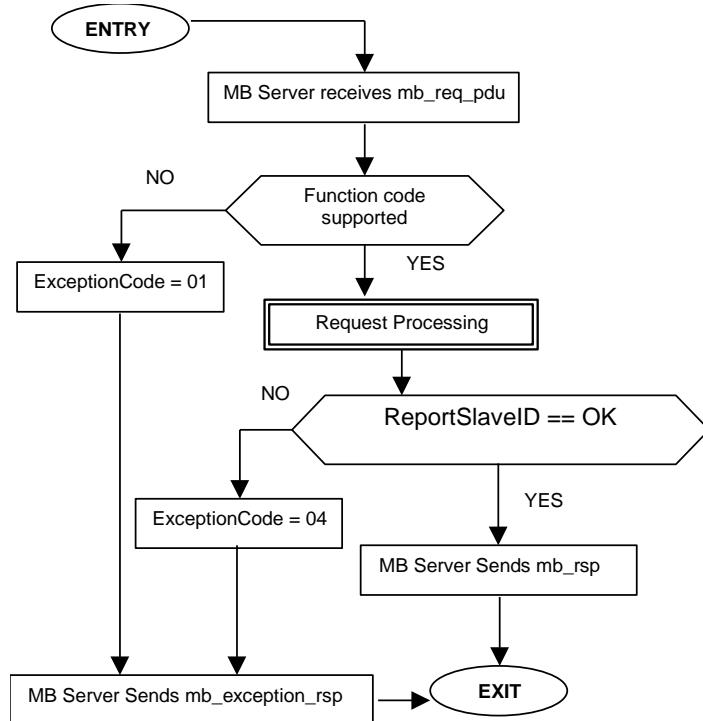


Figure 23: Report server ID state diagram

6.14 20 (0x14) Read File Record

This function code is used to perform a file record read. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of registers.

A file is an organization of records. Each file contains 10000 records, addressed 0000 to 9999 decimal or 0X0000 to 0X270F. For example, record 12 is addressed as 12.

The function can read multiple groups of references. The groups can be separating (non-contiguous), but the references within each group must be sequential.

Each group is defined in a separate 'sub-request' field that contains 7 bytes:

The reference type: 1 byte (must be specified as 6)

The File number: 2 bytes

The starting record number within the file: 2 bytes

The length of the record to be read: 2 bytes.

The quantity of registers to be read, combined with all other fields in the expected response, must not exceed the allowable length of the MODBUS PDU : 253 bytes.

The normal response is a series of 'sub-responses', one for each 'sub-request'. The byte count field is the total combined count of bytes in all 'sub-responses'. In addition, each 'sub-response' contains a field that shows its own byte count.

Request

| | | |
|----------------------------|---------|--------------------|
| Function code | 1 Byte | 0x14 |
| Byte Count | 1 Byte | 0x07 to 0xF5 bytes |
| Sub-Req. x, Reference Type | 1 Byte | 06 |
| Sub-Req. x, File Number | 2 Bytes | 0x0001 to 0xFFFF |

| | | |
|---------------------------|---------|------------------|
| Sub-Req. x, Record Number | 2 Bytes | 0x0000 to 0x270F |
| Sub-Req. x, Record Length | 2 Bytes | N |
| Sub-Req. x+1, ... | | |

Response

| | | |
|-------------------------------|--------------------|--------------|
| Function code | 1 Byte | 0x14 |
| Resp. data Length | 1 Byte | 0x07 to 0xF5 |
| Sub-Req. x, File Resp. length | 1 Byte | 0x07 to 0xF5 |
| Sub-Req. x, Reference Type | 1 Byte | 6 |
| Sub-Req. x, Record Data | N x 2 Bytes | |
| Sub-Req. x+1, ... | | |

Error

| | | |
|----------------|--------|----------------------------|
| Error code | 1 Byte | 0x94 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 or 08 |

While it is allowed for the File Number to be in the range 1 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if the File Number is greater than 10 (0xA).

Here is an example of a request to read two groups of references from remote device:

- Group 1 consists of two registers from file 4, starting at register 1 (address 0001).
- Group 2 consists of two registers from file 3, starting at register 9 (address 0009).

| Request | | Response | |
|------------------------------|-------|-------------------------------|-------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 14 | Function | 14 |
| Byte Count | 0E | Resp. Data length | 0C |
| Sub-Req. 1, Ref. Type | 06 | Sub-Req. 1, File resp. length | 05 |
| Sub-Req. 1, File Number Hi | 00 | Sub-Req. 1, Ref. Type | 06 |
| Sub-Req. 1, File Number Lo | 04 | Sub-Req. 1, Register.Data Hi | 0D |
| Sub-Req. 1, Record number Hi | 00 | Sub-Req. 1, Register.DataLo | FE |
| Sub-Req. 1, Record number Lo | 01 | Sub-Req. 1, Register.Data Hi | 00 |
| Sub-Req. 1, Record Length Hi | 00 | Sub-Req. 1, Register.DataLo | 20 |
| Sub-Req. 1, Record Length Lo | 02 | Sub-Req. 1, File resp. length | 05 |
| Sub-Req. 2, Ref. Type | 06 | Sub-Req. 2, Ref. Type | 06 |
| Sub-Req. 2, File Number Hi | 00 | Sub-Req. 2, Register.Data H | 33 |
| Sub-Req. 2, File Number Lo | 03 | Sub-Req. 2, Register.DataLo | CD |
| Sub-Req. 2, Record number Hi | 00 | Sub-Req. 2, Register.Data Hi | 00 |
| Sub-Req. 2, Record number Lo | 09 | Sub-Req. 2, Register.DataLo | 40 |
| Sub-Req. 2, Record Length Hi | 00 | | |
| Sub-Req. 2, Record Length Lo | 02 | | |

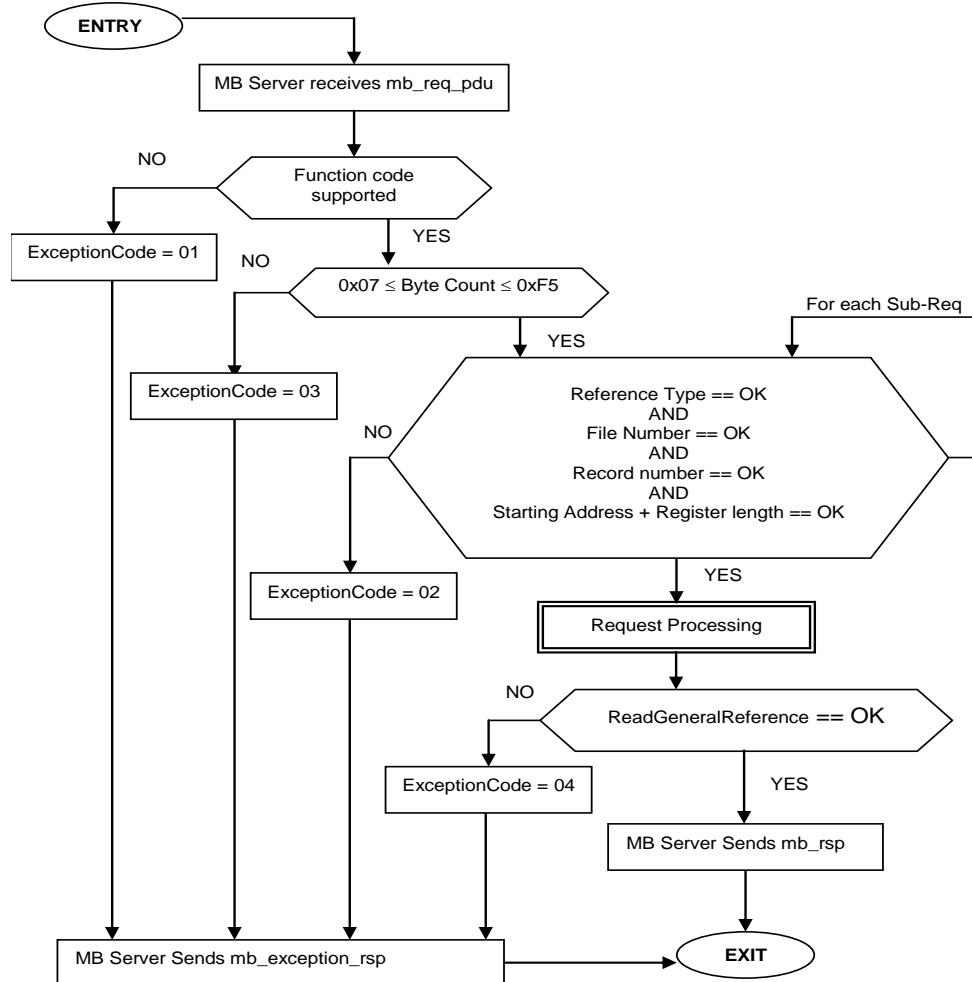


Figure 24: Read File Record state diagram

6.15 21 (0x15) Write File Record

This function code is used to perform a file record write. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of the number of 16-bit words.

A file is an organization of records. Each file contains 10000 records, addressed 0000 to 9999 decimal or 0X0000 to 0X270F. For example, record 12 is addressed as 12.

The function can write multiple groups of references. The groups can be separate, i.e. non-contiguous, but the references within each group must be sequential.

Each group is defined in a separate 'sub-request' field that contains 7 bytes plus the data:

- The reference type: 1 byte (must be specified as 6)
- The file number: 2 bytes
- The starting record number within the file: 2 bytes
- The length of the record to be written: 2 bytes
- The data to be written: 2 bytes per register.

The quantity of registers to be written, combined with all other fields in the request, must not exceed the allowable length of the MODBUS PDU : 253bytes.

The normal response is an echo of the request.

Request

| | | |
|----------------------------|---------|------------------|
| Function code | 1 Byte | 0x15 |
| Request data length | 1 Byte | 0x09 to 0xFB |
| Sub-Req. x, Reference Type | 1 Byte | 06 |
| Sub-Req. x, File Number | 2 Bytes | 0x0001 to 0xFFFF |
| Sub-Req. x, Record Number | 2 Bytes | 0x0000 to 0x270F |

| | | |
|---------------------------|--------------------|----------|
| Sub-Req. x, Record length | 2 Bytes | N |
| Sub-Req. x, Record data | N x 2 Bytes | |
| Sub-Req. x+1, ... | | |

Response

| | | |
|----------------------------|--------------------|------------------|
| Function code | 1 Byte | 0x15 |
| Response Data length | 1 Byte | 0x09 to 0xFB |
| Sub-Req. x, Reference Type | 1 Byte | 06 |
| Sub-Req. x, File Number | 2 Bytes | 0x0001 to 0xFFFF |
| Sub-Req. x, Record number | 2 Bytes | 0x0000 to 0x270F |
| Sub-Req. x, Record length | 2 Bytes | N |
| Sub-Req. x, Record Data | N x 2 Bytes | |
| Sub-Req. x+1, ... | | |

Error

| | | |
|----------------|--------|----------------------------|
| Error code | 1 Byte | 0x95 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 or 08 |

While it is allowed for the File Number to be in the range 1 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if the File Number is greater than 10 (0x0A).

Here is an example of a request to write one group of references into remote device:

- The group consists of three registers in file 4, starting at register 7 (address 0007).

| Request | | | |
|------------------------------|-----------|------------------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 15 | Function | 15 |
| Request Data length | 0D | Request Data length | 0D |
| Sub-Req. 1, Ref. Type | 06 | Sub-Req. 1, Ref. Type | 06 |
| Sub-Req. 1, File Number Hi | 00 | Sub-Req. 1, File Number Hi | 00 |
| Sub-Req. 1, File Number Lo | 04 | Sub-Req. 1, File Number Lo | 04 |
| Sub-Req. 1, Record number Hi | 00 | Sub-Req. 1, Record number Hi | 00 |
| Sub-Req. 1, Record number Lo | 07 | Sub-Req. 1, Record number Lo | 07 |
| Sub-Req. 1, Record length Hi | 00 | Sub-Req. 1, Record length Hi | 00 |
| Sub-Req. 1, Record length Lo | 03 | Sub-Req. 1, Record length Lo | 03 |
| Sub-Req. 1, Register Data Hi | 06 | Sub-Req. 1, Register Data Hi | 06 |
| Sub-Req. 1, Register Data Lo | AF | Sub-Req. 1, Register Data Lo | AF |
| Sub-Req. 1, Register Data Hi | 04 | Sub-Req. 1, Register Data Hi | 04 |
| Sub-Req. 1, Register Data Lo | BE | Sub-Req. 1, Register Data Lo | BE |
| Sub-Req. 1, Register Data Hi | 10 | Sub-Req. 1, Register Data Hi | 10 |
| Sub-Req. 1, Register Data Lo | 0D | Sub-Req. 1, Register Data Lo | 0D |

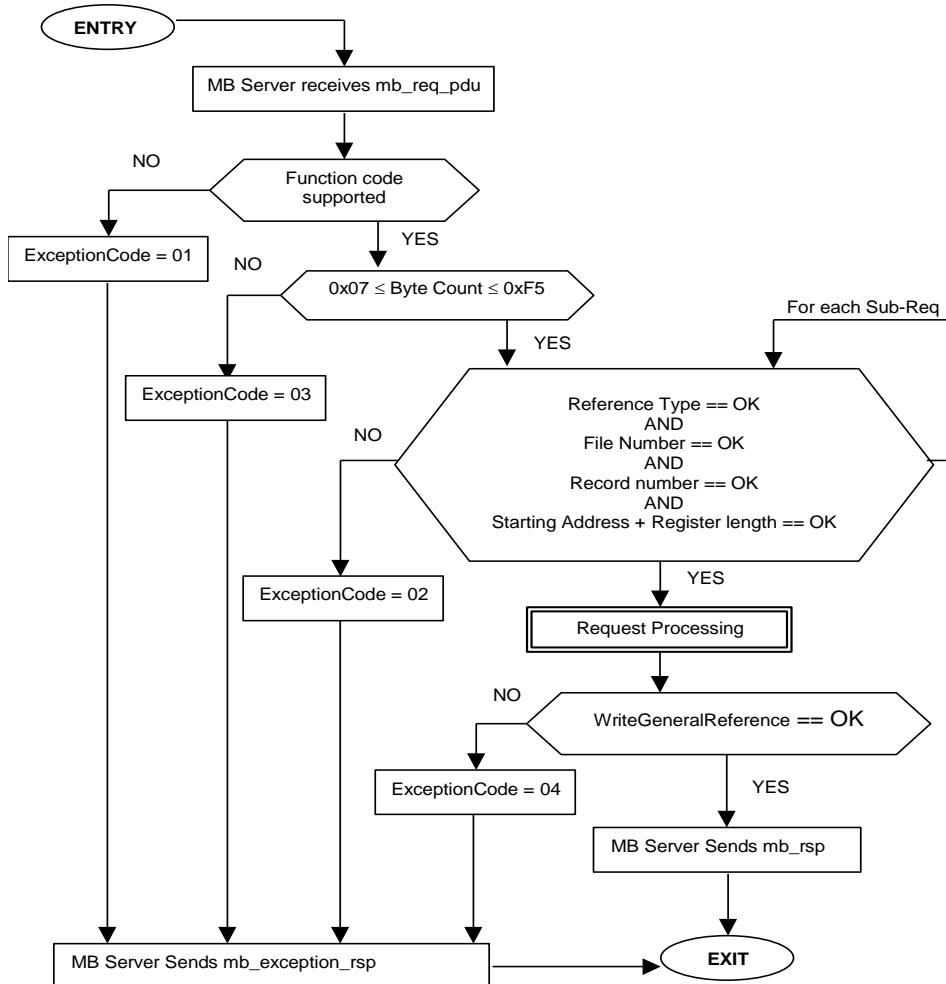


Figure 25: Write File Record state diagram

6.16 22 (0x16) Mask Write Register

This function code is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents. The function can be used to set or clear individual bits in the register.

The request specifies the holding register to be written, the data to be used as the AND mask, and the data to be used as the OR mask. Registers are addressed starting at zero. Therefore registers 1-16 are addressed as 0-15.

The function's algorithm is:

$$\text{Result} = (\text{Current Contents AND And_Mask}) \text{ OR } (\text{Or_Mask AND (NOT And_Mask)})$$

For example:

| | Hex | Binary |
|-------------------|-----|-----------|
| Current Contents= | 12 | 0001 0010 |
| And_Mask = | F2 | 1111 0010 |
| Or_Mask = | 25 | 0010 0101 |
| (NOT And_Mask)= | 0D | 0000 1101 |
| Result = | 17 | 0001 0111 |



Note:

- If the Or_Mask value is zero, the result is simply the logical ANDing of the current contents and And_Mask. If the And_Mask value is zero, the result is equal to the Or_Mask value.
- The contents of the register can be read with the Read Holding Registers function (function code 03). They could, however, be changed subsequently as the controller scans its user logic program.

The normal response is an echo of the request. The response is returned after the register has been written.

Request

| | | |
|-------------------|---------|------------------|
| Function code | 1 Byte | 0x16 |
| Reference Address | 2 Bytes | 0x0000 to 0xFFFF |
| And_Mask | 2 Bytes | 0x0000 to 0xFFFF |
| Or_Mask | 2 Bytes | 0x0000 to 0xFFFF |

Response

| | | |
|-------------------|---------|------------------|
| Function code | 1 Byte | 0x16 |
| Reference Address | 2 Bytes | 0x0000 to 0xFFFF |
| And_Mask | 2 Bytes | 0x0000 to 0xFFFF |
| Or_Mask | 2 Bytes | 0x0000 to 0xFFFF |

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x96 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a Mask Write to register 5 in remote device, using the above mask values.

| Request | | Response | |
|----------------------|-----------|----------------------|-----------|
| <i>Field Name</i> | (Hex) | <i>Field Name</i> | (Hex) |
| Function | 16 | Function | 16 |
| Reference address Hi | 00 | Reference address Hi | 00 |
| Reference address Lo | 04 | Reference address Lo | 04 |
| And_Mask Hi | 00 | And_Mask Hi | 00 |
| And_Mask Lo | F2 | And_Mask Lo | F2 |
| Or_Mask Hi | 00 | Or_Mask Hi | 00 |
| Or_Mask Lo | 25 | Or_Mask Lo | 25 |

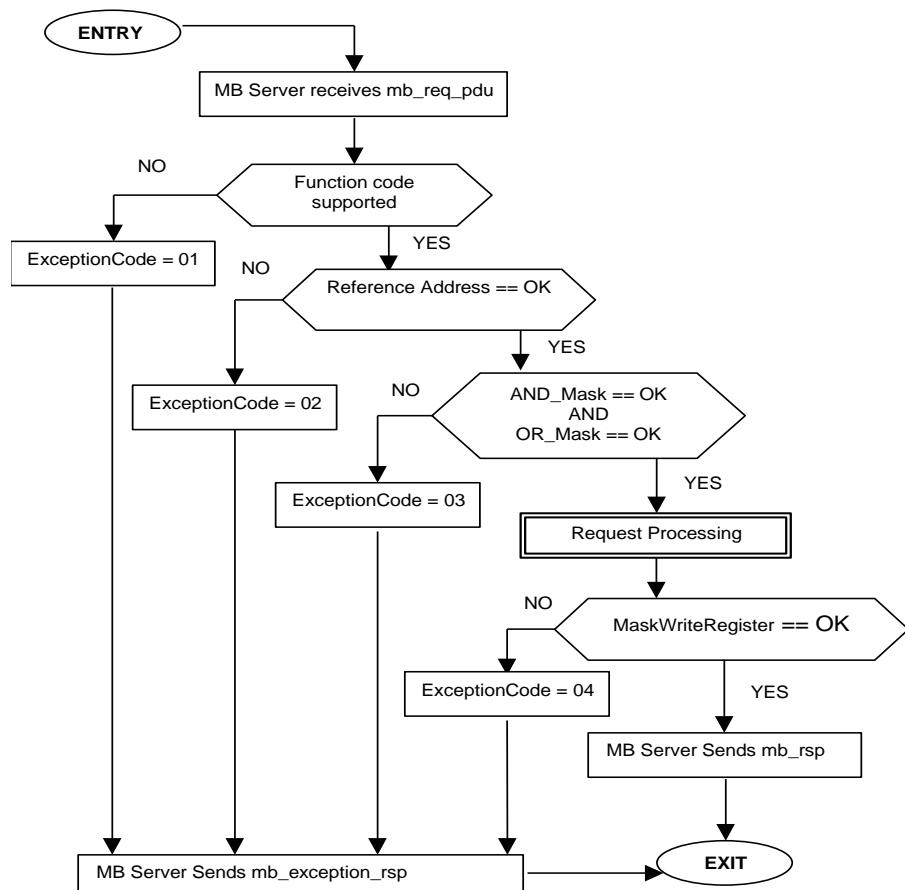


Figure 26: Mask Write Holding Register state diagram

6.17 23 (0x17) Read/Write Multiple registers

This function code performs a combination of one read operation and one write operation in a single MODBUS transaction. The write operation is performed before the read.

Holding registers are addressed starting at zero. Therefore holding registers 1-16 are addressed in the PDU as 0-15.

The request specifies the starting address and number of holding registers to be read as well as the starting address, number of holding registers, and the data to be written. The byte count specifies the number of bytes to follow in the write data field.

The normal response contains the data from the group of registers that were read. The byte count field specifies the quantity of bytes to follow in the read data field.

Request

| | | |
|------------------------|---------------------|------------------|
| Function code | 1 Byte | 0x17 |
| Read Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity to Read | 2 Bytes | 0x0001 to 0x007D |
| Write Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity to Write | 2 Bytes | 0x0001 to 0X0079 |
| Write Byte Count | 1 Byte | 2 x N* |
| Write Registers Value | N* x 2 Bytes | |

***N** = Quantity to Write

Response

| | | |
|----------------------|----------------------|----------------|
| Function code | 1 Byte | 0x17 |
| Byte Count | 1 Byte | 2 x N** |
| Read Registers value | N** x 2 Bytes | |

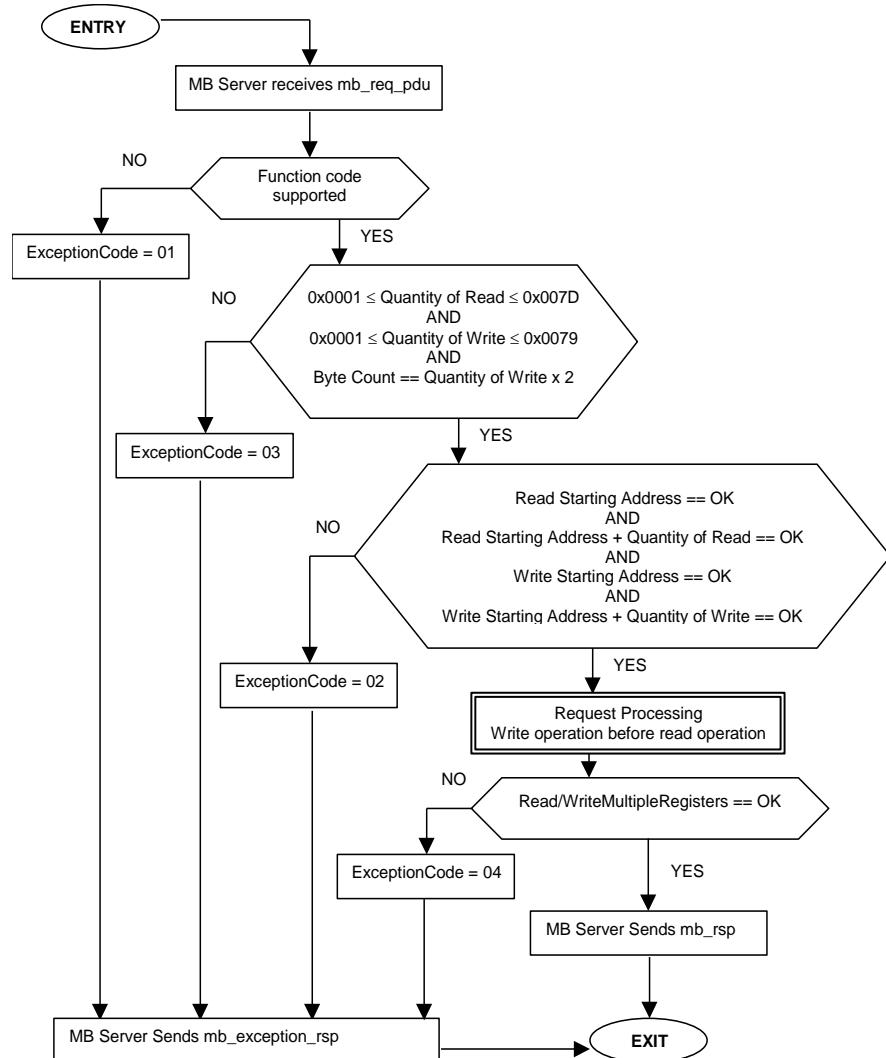
***N** = Quantity to Read

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x97 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of a request to read six registers starting at register 4, and to write three registers starting at register 15:

| Request | | Response | |
|---------------------------|-----------|-------------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 17 | Function | 17 |
| Read Starting Address Hi | 00 | Byte Count | 0C |
| Read Starting Address Lo | 03 | Read Registers value Hi | 00 |
| Quantity to Read Hi | 00 | Read Registers value Lo | FE |
| Quantity to Read Lo | 06 | Read Registers value Hi | 0A |
| Write Starting Address Hi | 00 | Read Registers value Lo | CD |
| Write Starting address Lo | 0E | Read Registers value Hi | 00 |
| Quantity to Write Hi | 00 | Read Registers value Lo | 01 |
| Quantity to Write Lo | 03 | Read Registers value Hi | 00 |
| Write Byte Count | 06 | Read Registers value Lo | 03 |
| Write Registers Value Hi | 00 | Read Registers value Hi | 00 |
| Write Registers Value Lo | FF | Read Registers value Lo | 0D |
| Write Registers Value Hi | 00 | Read Registers value Hi | 00 |
| Write Registers Value Lo | FF | Read Registers value Lo | FF |
| Write Registers Value Hi | 00 | | |
| Write Registers Value Lo | FF | | |



6.18 24 (0x18) Read FIFO Queue

This function code allows to read the contents of a First-In-First-Out (FIFO) queue of register in a remote device. The function returns a count of the registers in the queue, followed by the queued data. Up to 32 registers can be read: the count, plus up to 31 queued data registers. The queue count register is returned first, followed by the queued data registers.

The function reads the queue contents, but does not clear them.

In a normal response, the byte count shows the quantity of bytes to follow, including the queue count bytes and value register bytes (but not including the error check field).

The queue count is the quantity of data registers in the queue (not including the count register).

If the queue count exceeds 31, an exception response is returned with an error code of 03 (Illegal Data Value).

Request

| | | |
|----------------------|---------|------------------|
| Function code | 1 Byte | 0x18 |
| FIFO Pointer Address | 2 Bytes | 0x0000 to 0xFFFF |

Response

| | | |
|---------------------|----------------------|-------------|
| Function code | 1 Byte | 0x18 |
| Byte Count | 2 Bytes | |
| FIFO Count | 2 Bytes | ≤ 31 |
| FIFO Value Register | $N^* \times 2$ Bytes | |

*N = FIFO Count

Error

| | | |
|----------------|--------|----------------------|
| Error code | 1 Byte | 0x98 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

Here is an example of Read FIFO Queue request to remote device. The request is to read the queue starting at the pointer register 1246 (0x04DE):

| Request | | Response | |
|-------------------------|-----------|------------------------|-----------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 18 | Function | 18 |
| FIFO Pointer Address Hi | 04 | Byte Count Hi | 00 |
| FIFO Pointer Address Lo | DE | Byte Count Lo | 06 |
| | | FIFO Count Hi | 00 |
| | | FIFO Count Lo | 02 |
| | | FIFO Value Register Hi | 01 |
| | | FIFO Value Register Lo | B8 |
| | | FIFO Value Register Hi | 12 |
| | | FIFO Value Register Lo | 84 |

In this example, the FIFO pointer register (1246 in the request) is returned with a queue count of 2. The two data registers follow the queue count. These are:

1247 (contents 440 decimal -- 0x01B8); and 1248 (contents 4740 -- 0x1284).

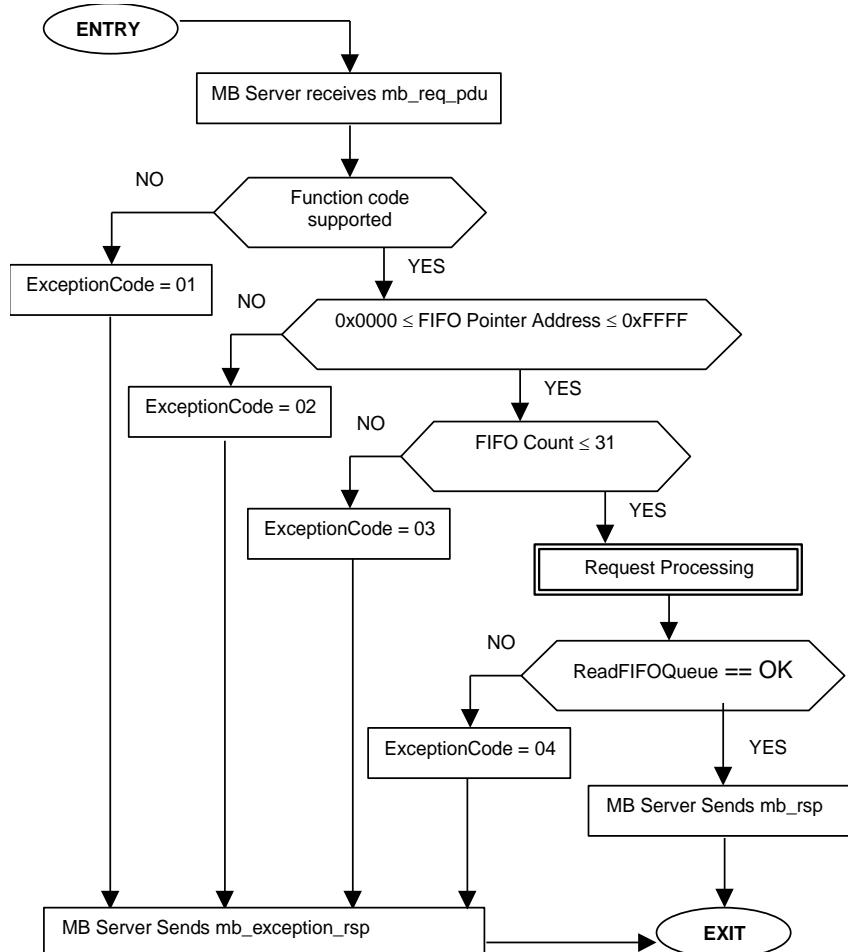


Figure 28: Read FIFO Queue state diagram

6.19 43 (0x2B) Encapsulated Interface Transport

Informative Note: The user is asked to refer to Annex A (Informative) MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES.

Function Code 43 and its MEI Type 14 for Device Identification is one of two Encapsulated Interface Transport currently available in this Specification. The following function codes and MEI Types shall not be part of this published Specification and these function codes and MEI Types are specifically reserved: 43/0-12 and 43/15-255.

The MODBUS Encapsulated Interface (MEI)Transport is a mechanism for tunneling service requests and method invocations, as well as their returns, inside MODBUS PDUs.

The primary feature of the MEI Transport is the encapsulation of method invocations or service requests that are part of a defined interface as well as method invocation returns or service responses.

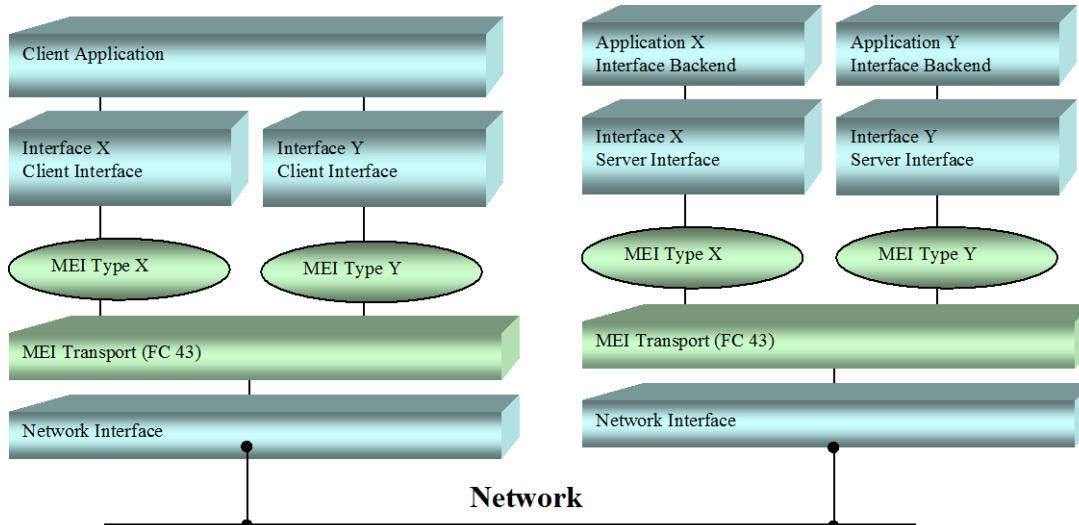


Figure 29: MODBUS encapsulated Interface Transport

The **Network Interface** can be any communication stack used to send MODBUS PDUs, such as TCP/IP, or serial line.

A **MEI Type** is a MODBUS Assigned Number and therefore will be unique, the value between 0 to 255 are Reserved according to Annex A (Informative) except for MEI Type 13 and MEI Type 14.

The MEI Type is used by MEI Transport implementations to dispatch a method invocation to the indicated interface.

Since the MEI Transport service is interface agnostic, any specific behavior or policy required by the interface must be provided by the interface, e.g. MEI transaction processing, MEI interface error handling, etc.

Request

| | | |
|------------------------|---------|--------------|
| Function code | 1 Byte | 0x2B |
| MEI Type* | 1 Byte | 0x0D or 0x0E |
| MEI type specific data | n Bytes | |

* MEI = MODBUS Encapsulated Interface

Response

| | | |
|------------------------|---------|-----------------------------|
| Function code | 1 Byte | 0x2B |
| MEI Type | 1 byte | echo of MEI Type in Request |
| MEI type specific data | n Bytes | |

Error

| | | |
|----------------|--------|------------------------------|
| Function code | 1 Byte | 0xAB : Fc 0x2B + 0x80 |
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |

As an example see Read device identification request.

6.20 43 / 13 (0x2B / 0x0D) CANopen General Reference Request and Response PDU

The CANopen General reference Command is an encapsulation of the services that will be used to access (read from or write to) the entries of a CAN-Open Device Object Dictionary as well as controlling and monitoring the CANopen system, and devices.

The MEI Type 13 (0x0D) is a MODBUS Assigned Number licensed to CiA for the CANopen General Reference.

The system is intended to work within the limitations of existing MODBUS networks. Therefore, the information needed to query or modify the object dictionaries in the system is

mapped into the format of a MODBUS message. The PDU will have the 253 Byte limitation in both the Request and the Response message.

Informative: Please refer to Annex B for a reference to a specification that provides information on MEI Type 13.

6.21 43 / 14 (0x2B / 0x0E) Read Device Identification

This function code allows reading the identification and additional information relative to the physical and functional description of a remote device, only.

The Read Device Identification interface is modeled as an address space composed of a set of addressable data elements. The data elements are called objects and an object Id identifies them.

The interface consists of 3 categories of objects :

- Basic Device Identification. All objects of this category are mandatory : VendorName, Product code, and revision number.
- Regular Device Identification. In addition to Basic data objects, the device provides additional and optional identification and description data objects. All of the objects of this category are defined in the standard but their implementation is optional .
- Extended Device Identification. In addition to regular data objects, the device provides additional and optional identification and description private data about the physical device itself. All of these data are device dependent.

| Object Id | Object Name / Description | Type | M/O | category |
|-----------|--|------------------|------------------|----------|
| 0x00 | VendorName | ASCII String | Mandatory | Basic |
| 0x01 | ProductCode | ASCII String | Mandatory | |
| 0x02 | MajorMinorRevision | ASCII String | Mandatory | |
| 0x03 | VendorUrl | ASCII String | Optional | Regular |
| 0x04 | ProductName | ASCII String | Optional | |
| 0x05 | ModelName | ASCII String | Optional | |
| 0x06 | UserApplicationName | ASCII String | Optional | |
| 0x07 | Reserved | | Optional | |
| ... | | | | |
| 0x7F | | | | |
| 0x80 | <i>Private objects may be optionally defined.</i> | device dependant | Optional | Extended |
| ... | <i>The range [0x80 – 0xFF] is Product dependant.</i> | | | |
| 0xFF | | | | |

Request

| | | |
|---------------------|--------|-------------------|
| Function code | 1 Byte | 0x2B |
| MEI Type* | 1 Byte | 0x0E |
| Read Device ID code | 1 Byte | 01 / 02 / 03 / 04 |
| Object Id | 1 Byte | 0x00 to 0xFF |

* MEI = MODBUS Encapsulated Interface

Response

| | | |
|---------------------|---------------|--|
| Function code | 1 Byte | 0x2B |
| MEI Type | 1 byte | 0xE |
| Read Device ID code | 1 Byte | 01 / 02 / 03 / 04 |
| Conformity level | 1 Byte | 0x01 or 0x02 or 0x03 or 0x81 or 0x82 or 0x83 |
| More Follows | 1 Byte | 00 / FF |
| Next Object Id | 1 Byte | Object ID number |
| Number of objects | 1 Byte | |
| List Of | | |
| Object ID | 1 Byte | |
| Object length | 1 Byte | |
| Object Value | Object length | Depending on the object ID |

Error

| | | |
|---------------|--------|------------------------------|
| Function code | 1 Byte | 0xAB : Fc 0x2B + 0x80 |
|---------------|--------|------------------------------|

| | | |
|----------------|--------|----------------------|
| Exception code | 1 Byte | 01 or 02 or 03 or 04 |
|----------------|--------|----------------------|

Request parameters description :

A MODBUS Encapsulated Interface assigned number 14 identifies the Read identification request.

The parameter " Read Device ID code " allows to define four access types :

- 01: request to get the basic device identification (stream access)
- 02: request to get the regular device identification (stream access)
- 03: request to get the extended device identification (stream access)
- 04: request to get one specific identification object (individual access)

An exception code 03 is sent back in the response if the Read device ID code is illegal.

In case of a response that does not fit into a single response, several transactions (request/response) must be done. The Object Id byte gives the identification of the first object to obtain. For the first transaction, the client must set the Object Id to 0 to obtain the beginning of the device identification data. For the following transactions, the client must set the Object Id to the value returned by the server in its previous response.

Remark : An object is indivisible, therefore any object must have a size consistent with the size of transaction response.

If the Object Id does not match any known object, the server responds as if object 0 were pointed out (restart at the beginning).

In case of an individual access: ReadDevId code 04, the Object Id in the request gives the identification of the object to obtain, and if the Object Id doesn't match to any known object, the server returns an exception response with exception code = 02 (Illegal data address).

If the server device is asked for a description level (readDevice Code) higher than its conformity level, It must respond in accordance with its actual conformity level.

Response parameter description :

| | |
|-------------------|---|
| Function code : | Function code 43 (decimal) 0x2B (hex) |
| MEI Type | 14 (0x0E) MEI Type assigned number for Device Identification Interface |
| ReadDevId code : | Same as request ReadDevId code : 01, 02, 03 or 04 |
| Conformity Level | Identification conformity level of the device and type of supported access 0x01: basic identification (stream access only) 0x02: regular identification (stream access only) 0x03: extended identification (stream access only) 0x81: basic identification (stream access and individual access) 0x82: regular identification (stream access and individual access) 0x83: extended identification (stream access and individual access) |
| More Follows | In case of ReadDevId codes 01, 02 or 03 (stream access), If the identification data doesn't fit into a single response, several request/response transactions may be required. 0x00 : no more Object are available 0xFF : other identification Object are available and further MODBUS transactions are required In case of ReadDevId code 04 (individual access), this field must be set to 00. |
| Next Object Id | If "MoreFollows = FF", identification of the next Object to be asked for. If "MoreFollows = 00", must be set to 00 (useless) |
| Number Of Objects | Number of identification Object returned in the response (for an individual access, Number Of Objects = 1) |
| Object0.Id | Identification of the first Object returned in the PDU (stream access) or the requested Object (individual access) |

| | |
|----------------|---|
| Object0.Length | Length of the first Object in byte |
| Object0.Value | Value of the first Object (Object0.Length bytes) |
| ... | |
| ObjectN.Id | Identification of the last Object (within the response) |
| ObjectN.Length | Length of the last Object in byte |
| ObjectN.Value | Value of the last Object (ObjectN.Length bytes) |

Example of a Read Device Identification request for "Basic device identification" : In this example all information are sent in one response PDU.

| Request | | Response | |
|-------------------|--------------|-------------------|----------------------------------|
| <i>Field Name</i> | <i>Value</i> | <i>Field Name</i> | <i>Value</i> |
| Function | 2B | Function | 2B |
| MEI Type | 0E | MEI Type | 0E |
| Read Dev Id code | 01 | Read Dev Id Code | 01 |
| Object Id | 00 | Conformity Level | 01 |
| | | More Follows | 00 |
| | | NextObjectId | 00 |
| | | Number Of Objects | 03 |
| | | Object Id | 00 |
| | | Object Length | 16 |
| | | Object Value | " Company identification" |
| | | Object Id | 01 |
| | | Object Length | 0D |
| | | Object Value | " Product code XX" |
| | | Object Id | 02 |
| | | Object Length | 05 |
| | | Object Value | "V2.11" |

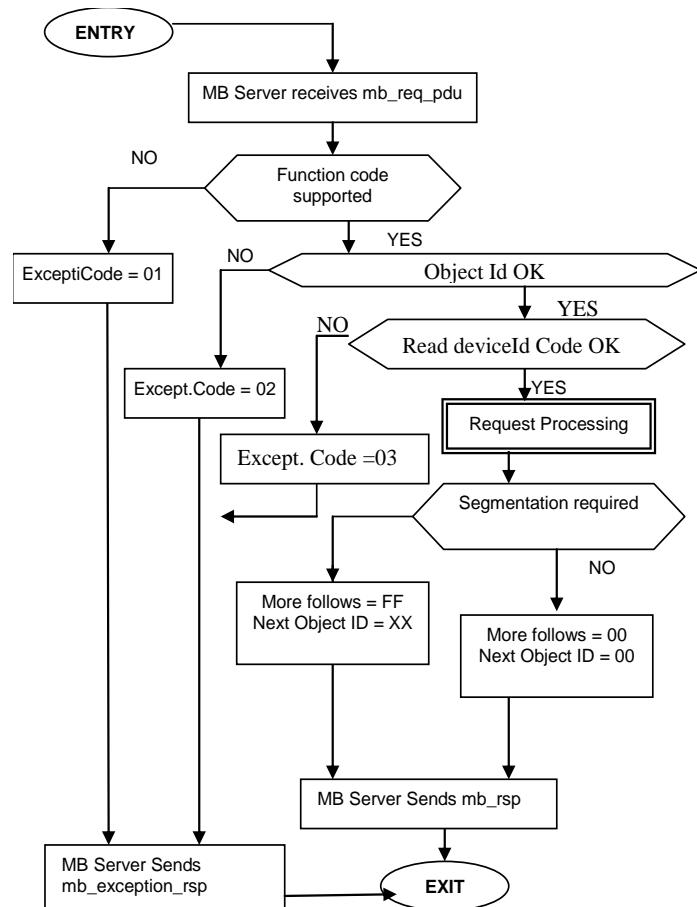
In case of a device that required several transactions to send the response the following transactions is intiated.

First transaction :

| Request | | Response | |
|-------------------|--------------|-------------------|---|
| <i>Field Name</i> | <i>Value</i> | <i>Field Name</i> | <i>Value</i> |
| Function | 2B | Function | 2B |
| MEI Type | 0E | MEI Type | 0E |
| Read Dev Id code | 01 | Read Dev Id Code | 01 |
| Object Id | 00 | Conformity Level | 01 |
| | | More Follows | FF |
| | | NextObjectId | 02 |
| | | Number Of Objects | 03 |
| | | Object Id | 00 |
| | | Object Length | 16 |
| | | Object Value | " Company identification" |
| | | Object Id | 01 |
| | | Object Length | 1C |
| | | Object Value | " Product code XXXXXXXXXXXXXXXX" |

Second transaction :

| Request | | Response | |
|-------------------|--------------|-------------------|----------------|
| <i>Field Name</i> | <i>Value</i> | <i>Field Name</i> | <i>Value</i> |
| Function | 2B | Function | 2B |
| MEI Type | 0E | MEI Type | 0E |
| Read Dev Id code | 01 | Read Dev Id Code | 01 |
| Object Id | 02 | Conformity Level | 01 |
| | | More Follows | 00 |
| | | NextObjectId | 00 |
| | | Number Of Objects | 03 |
| | | Object Id | 02 |
| | | Object Length | 05 |
| | | Object Value | "V2.11" |

**Figure 30: Read Device Identification state diagram**

7 MODBUS Exception Responses

When a client device sends a request to a server device it expects a normal response. One of four possible events can occur from the client's query:

- If the server device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the server does not receive the request due to a communication error, no response is returned. The client program will eventually process a timeout condition for the request.
- If the server receives the request, but detects a communication error (parity, LRC, CRC, ...), no response is returned. The client program will eventually process a timeout condition for the request.
- If the server receives the request without a communication error, but cannot handle it (for example, if the request is to read a non-existent output or register), the server will return an exception response informing the client of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function Code Field: In a normal response, the server echoes the function code of the original request in the function code field of the response. All function codes have a most-significant bit (MSB) of 0 (their values are all below 80 hexadecimal). In an exception response, the server sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response.

With the function code's MSB set, the client's application program can recognize the exception response and can examine the data field for the exception code.

Data Field: In a normal response, the server may return data or statistics in the data field (any information that was requested in the request). In an exception response, the server returns an exception code in the data field. This defines the server condition that caused the exception.

Example of a client request and server exception response

| Request | | Response | |
|------------------------|-------|----------------|-------|
| Field Name | (Hex) | Field Name | (Hex) |
| Function | 01 | Function | 81 |
| Starting Address Hi | 04 | Exception Code | 02 |
| Starting Address Lo | A1 | | |
| Quantity of Outputs Hi | 00 | | |
| Quantity of Outputs Lo | 01 | | |

In this example, the client addresses a request to server device. The function code (01) is for a Read Output Status operation. It requests the status of the output at address 1185 (04A1 hex). Note that only that one output is to be read, as specified by the number of outputs field (0001).

If the output address is non-existent in the server device, the server will return the exception response with the exception code shown (02). This specifies an illegal data address for the server.

A listing of exception codes begins on the next page.

| MODBUS Exception Codes | | |
|------------------------|-----------------------|---|
| Code | Name | Meaning |
| 01 | ILLEGAL FUNCTION | The function code received in the query is not an allowable action for the server. This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values. |
| 02 | ILLEGAL DATA ADDRESS | The data address received in the query is not an allowable address for the server. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, the PDU addresses the first register as 0, and the last one as 99. If a request is submitted with a starting register address of 96 and a quantity of registers of 4, then this request will successfully operate (address-wise at least) on registers 96, 97, 98, 99. If a request is submitted with a starting register address of 96 and a quantity of registers of 5, then this request will fail with Exception Code 0x02 "Illegal Data Address" since it attempts to operate on registers 96, 97, 98, 99 and 100, and there is no register with address 100. |
| 03 | ILLEGAL DATA VALUE | A value contained in the query data field is not an allowable value for server. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register. |
| 04 | SERVER DEVICE FAILURE | An unrecoverable error occurred while the server was attempting to perform the requested action. |
| 05 | ACKNOWLEDGE | Specialized use in conjunction with programming commands. The server has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed. |
| 06 | SERVER DEVICE BUSY | Specialized use in conjunction with programming commands. The server is engaged in processing a long-duration program command. The client should retransmit the message later when the server is free. |
| 08 | MEMORY PARITY ERROR | Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server attempted to read record file, but detected a parity error in the memory. The client can retry the request, but service may be required |

| | | |
|-----------|---|--|
| | | on the server device. |
| 0A | GATEWAY PATH UNAVAILABLE | Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded. |
| 0B | GATEWAY TARGET DEVICE FAILED TO RESPOND | Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network. |

Annex A (Informative): MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES

The following function codes and subcodes shall not be part of this published Specification and these function codes and subcodes are specifically reserved. The format is function code/subcode or just function code where all the subcodes (0-255) are reserved: 8/19; 8/21-65535, 9, 10, 13, 14, 41, 42, 90, 91, 125, 126 and 127.

Function Code 43 and its MEI Type 14 for Device Identification and MEI Type 13 for CANopen General Reference Request and Response PDU are the currently available Encapsulated Interface Transports in this Specification.

The following function codes and MEI Types shall not be part of this published Specification and these function codes and MEI Types are specifically reserved: 43/0-12 and 43/15-255. In this Specification, a User Defined Function code having the same or similar result as the Encapsulated Interface Transport is not supported.

MODBUS is a registered trademark of Schneider Automation Inc.

Annex B (Informative): CANOPEN GENERAL REFERENCE COMMAND

Please refer to the MODBUS website or the CiA (CAN in Automation) website for a copy and terms of use that cover Function Code 43 MEI Type 13.

Anexo B

Las Unidades de Tratamiento de Aire

Una UTA es un equipo encargado de tratar el aire atendiendo a distintos aspectos de la climatización:

- Ventilación
- Filtrado
- Control de temperatura
- Control de humedad

Su funcionamiento da la posibilidad de regular el caudal de ventilación en función de la medición del CO₂ y de las condiciones térmicas del local, así como recuperar parte de la energía térmica del aire que se expulsa al exterior.

Las partes de las que se compone una UTA son:

- Compuertas o tomas de aire para la entrada y salida del aire.
- Zona de mezcla del aire de retorno y del aire exterior para permitir recuperar energía mediante lo que se conoce como *free-cooling*. La recuperación puede ser de distintos tipos:

- Recuperación aire-agua: el aire extraído transfiere su calor al agua a través de un intercambiador. El mismo agua transfiere ese calor al aire introducido desde el exterior, el cual fluye por otro conducto. Este sistema es más salubre al impedir que se mezclen ambas corrientes de aire, pero tiene menor rendimiento.
 - Recuperación mediante placas aire-aire: a través de un cubo metálico pasan ambas corrientes de aire a modo de flujo cruzado. El calor se transfiere a través del contacto del aire con las placas metálicas de los canales.
-
- Intercambiador de calor, por donde fluye agua a la temperatura deseada (frío o calor). La temperatura del agua se transfiere al aire cuando éste pasa a través del intercambiador.
 - Filtros, que pueden ser de distintos tipos en función de las especificaciones de filtrado.
 - Humectador: el paso del aire seco y caliente por un panel mojado hace que se evapore parte de su agua y se produzca aire húmedo y frío. Dispone de una bandeja de almacenamiento y recogida del agua.
 - Zona de ventilación con ventilador y motor.

La siguiente figura ilustra a modo de ejemplo todos los elementos mencionados:

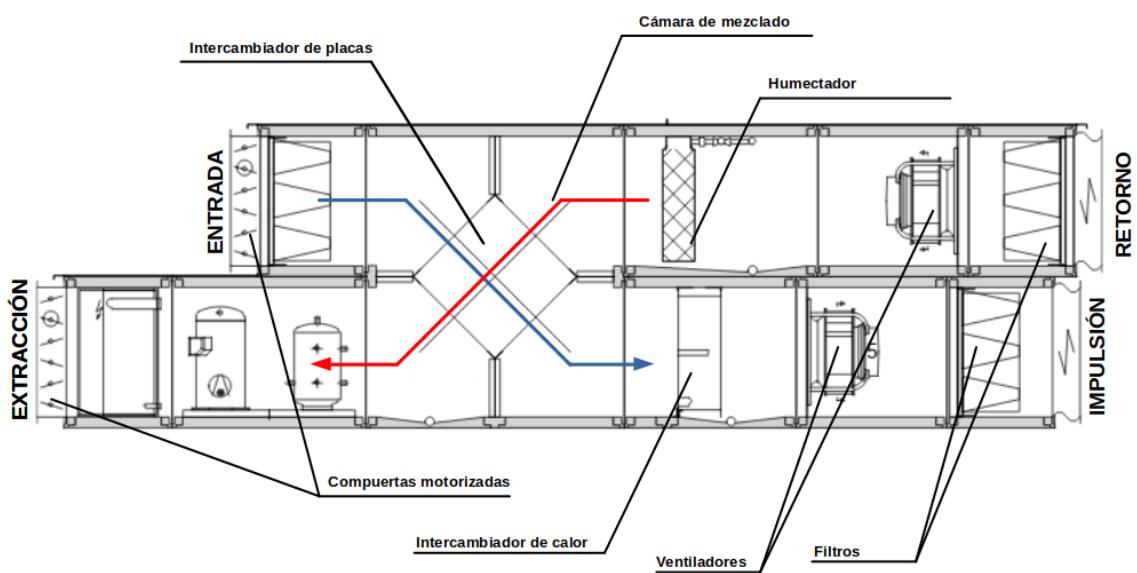


Figura B.1: Plano de partes de una UTA

Anexo C

Programa Unidad de Tratamiento de Aire

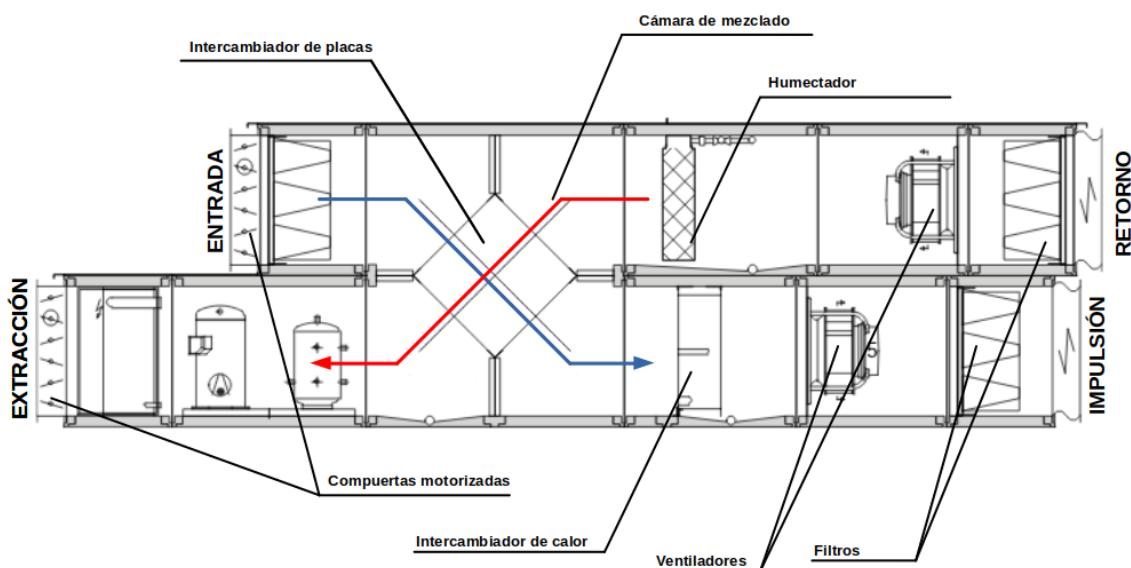


Figura C.1: Plano de partes de una UTA

El programa para el control de la UTA se ha realizado el software ISaGRAF, que emplea los lenguajes de estándar IEC61131:

- (LD) Escalera
- (FBD) Diagrama a Bloques

- (SFC) Tabla de Funciones Secuenciales
- (ST) Texto Estructurado
- (IL) Lista de Instrucciones
- (FC) Diagrama de flujo

El programa se compone de una parte de configuración de entradas, salidas, parámetros, alarmas y warnings, y otra parte dedicada a la programación del funcionamiento de la UTA.

En este anexo se muestran las variables configuradas (Tabla C.1), y la lógica de funcionamiento programada.

| GRUPO | VARIABLE | DESCRIPCIÓN |
|---------------------|-------------------------|--|
| Entradas analógicas | Pb_T_IMP_AIRE | Sonda de temperatura de impulsión de aire |
| | Pb_T_RET_AIRE | Sonda de temperatura de retorno de aire |
| | Pb_T_IMP_AGUA | Sonda de temperatura de impulsión de agua |
| | Pb_T_RET_AGUA | Sonda de temperatura de retorno de agua |
| | Pb_T_EXTRAC_AIRE | Sonda de temperatura de extracción de aire |
| | Pb_T_EXT_AIRE | Sonda de temperatura de aire exterior |
| | Pb_H_RET | Sensor de humedad de aire de retorno |
| | Pb_H_EXT | Sensor de humedad de aire exterior |
| Entradas digitales | DI_ONOFF | Interruptor de ON/OFF remoto |
| | DI_VENT_IMP | Seguridad ventilador de impulsión |
| | DI_VENT_RET | Seguridad ventilador de retorno |
| | DI_NIVEL_HUM | Indicador nivel humectador lleno |

Tabla C.1 continua en la página siguiente...

Tabla C.1 ...continuación de la página anterior.

| GRUPO | PARÁMETRO | DESCRIPCIÓN |
|--------------------|---------------------------------|--|
| Salidas digitales | RELE_VALVE3V | Válvula de recirculación de agua desde central |
| | RELE_HUMECT | ON/OFF humectador |
| | RELE_ALARM | ON/OFF alarma |
| Salidas analógicas | AO_VENT_IMP | Regulación velocidad ventilador impulsión |
| | AO_VENT_RET | Regulación velocidad ventilador retorno |
| Alarmas | ALARM_VENT_IMPULSION | Alarma en ventilador de impulsión |
| | ALARM_VENT_RETORNO | Alarma en ventilador de retorno |
| Warnings | WARNING_VENT_IMPULSION | Aviso de fallo en ventilador de impulsión |
| | WARNING_VENT_RETORNO | Aviso de fallo en ventilador de retorno |
| | WARNING_FILTER_ENTRADA | Aviso de filtro de entrada sucio |
| | WARNING_FILTER_IMPULSION | Aviso de filtro de impulsión sucio |
| | WARNING_FILTER_RETORNO | Aviso de filtro de retorno sucio |
| | WARNING_H_FLUID_TMP | Aviso de temperatura alta en fluido para alcanzar consigna |
| | WARNING_L_FLUID_TMP | Aviso de temperatura baja en fluido para alcanzar consigna |

Tabla C.1: Variables configuración control

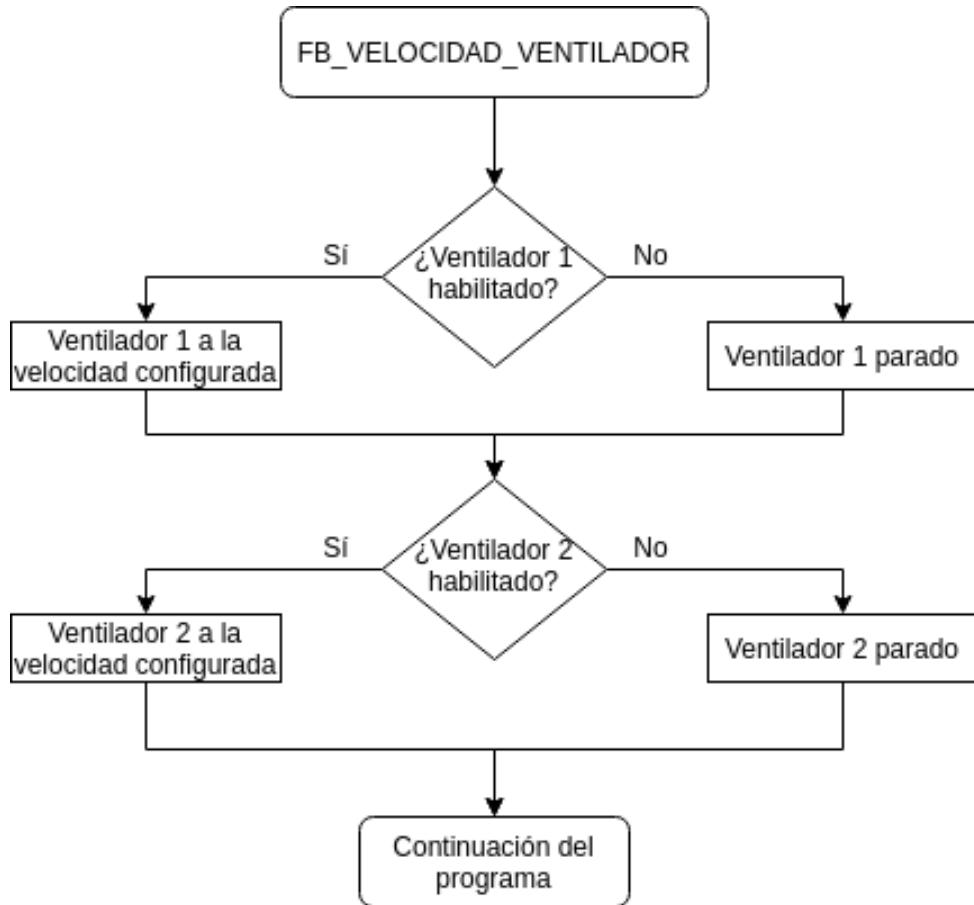


Figura C.2: Flujo de programa para gestión de ventiladores

La Figura C.2 anterior muestra el flujo de programa para la función encargada de analizar si los ventiladores están habilitados y sin fallos, para enviarles la velocidad deseada. Esta función será llamada desde el módulo encargado de decidir qué velocidad es la deseada en los ventiladores (Figura C.3): la velocidad puede ser baja, media, alta o el resultado del control automático de la Figura C.4.

Por último, el humectador se controla según la lógica de la Figura C.5.

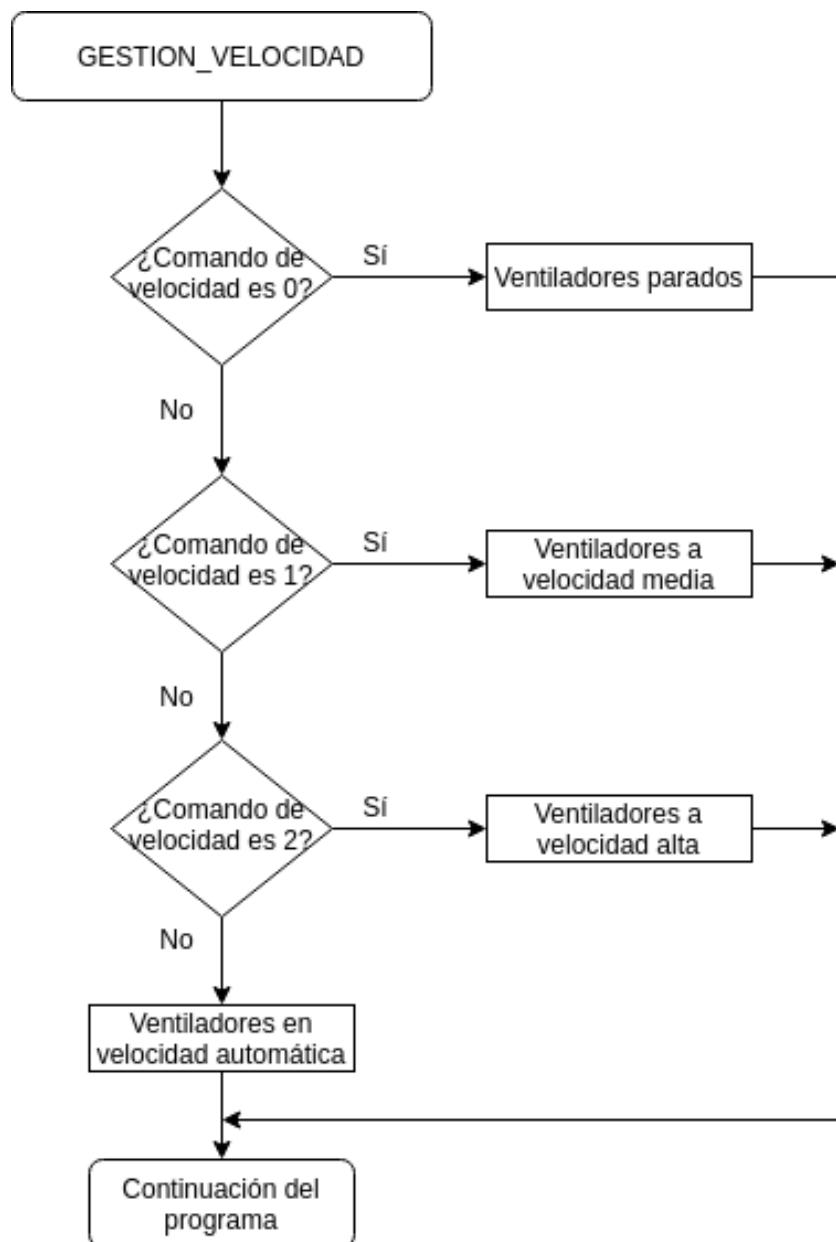


Figura C.3: Flujo de programa de gestión de la velocidad

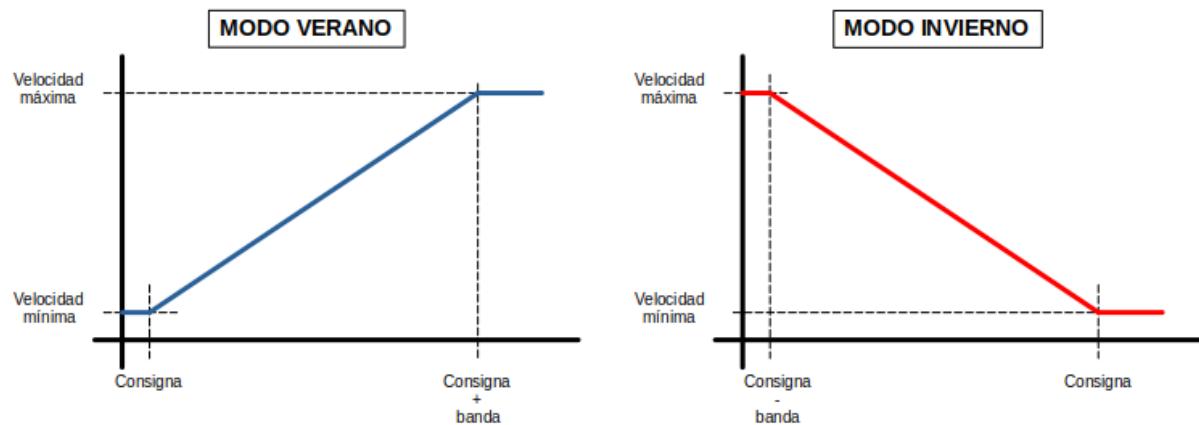


Figura C.4: Curvas de control automático de la velocidad en ventiladores, en función de la temperatura

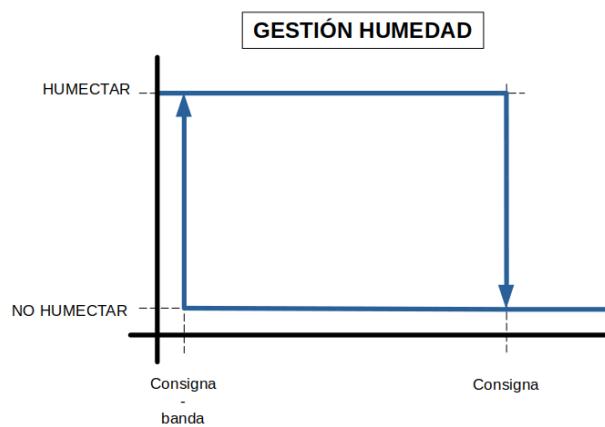


Figura C.5: Curva de control del humectador

Anexo D

Registros Control Unidad de Tratamiento de Aire

Este anexo recoge los parámetros necesarios para el control de la Unidad de Tratamiento de Aire programada en el proyecto:

Pb_T_IMP_AIRE

(* temperatura impulsión de aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1000

Pb_T_RET_AIRE

(* temperatura retorno de aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1001

Pb_T_IMP_AGUA

(* temperatura impulsión fluido *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1002

Pb_T_RET_AGUA

(* temperatura retorno fluido *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1003

Pb_T_EXTRAC_AIRE

(* temperatura extracción de aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1004

Pb_T_EXT_AIRE

(* temperatura aire exterior *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1005

Pb_H_RET

(* humedad exterior *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1006

Pb_H_EXT

(* humedad retorno *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1007

AO_VENT_IMP

(* Ventilador impulsión de aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1080

AO_VENT_RET

(* Ventilador retorno aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1081

DI_ONOFF

(* On-OFF remoto *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1100

DI_VENT_IMP

(* Seguridad ventilador imp. aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1101

DI_VENT_RET

(* Seguridad ventilador ret. aire *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1102

DI_NIVEL_HUM

(* Sensor nivel humectador *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1103

DI_FILT_ENT

(* Filtro de entrada de aire sucio *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1104

DI_FILT_IMP

(* Filtro de impulsión de aire sucio *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1105

DI_FILT_RET

(* Filtro de retorno de aire sucio *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1106

RELE_VALVE3V

(* Conmutación válvula *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1200

RELE_HUMECT

(* Activación/Desactivación de la humectación *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1201

RELE_COMP_IMPUL

(* Apertura/Cierre Compuerta de Impulsión *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1202

RELE_COMP_RET

(* Apertura/Cierre Compuerta de retorno *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1203

RELE_ALARM

(* Activación / Desactivación Alarma *)

Grupo: ENTRADAS_SALIDAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1204

ONOFF_GEN

Grupo: ESTADOS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1402

PAR_SET

(* Consigna *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1508

PAR_SET_HUM

(* Consigna de diferencia de humedad con el exterior *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1509

PAR_VEL_BAJA

(* Configuración de velocidad baja *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 150B

PAR_VEL_MEDIA

(* Configuración de velocidad media *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 150C

PAR_VEL_ALTA

(* Configuración de velocidad alta *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 150D

PAR_VEL_MIN

(* Velocidad mínima en los ventiladores *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 150E

PAR_VEL_MAX

(* Velocidad máxima en los ventiladores *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 150F

PAR_BANDA_VEL

(* Ancho de banda para velocidad máxima *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1510

PAR_BAND_INV

(* Temperatura de banda en modo invierno *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1512

PAR_BAND_VER

(* Temperatura de banda en modo verano *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1513

PAR_BAND_HUM

(* Banda humedad *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1514

PAR_TEMP_FIL1

(* Temporización para alarma filtro entrada aire(no usada) *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1515

PAR_TEMP_FIL2

(* Temporización para alarma filtro impulsión aire(no usada) *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1516

PAR_TEMP_FIL3

(* Temporización para alarma filtro impulsión aire(no usada) *)

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1517

PAR_TOFF_AL_FAN1

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1518

PAR_TON_AL_FAN1

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1519

PAR_NUM_INTENTOS_FAN1

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 151A

PAR_TOFF_AL_FAN2

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 151B

PAR_TON_AL_FAN2

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 151C

PAR_NUM_INTENTOS_FAN2

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 151D

PAR_TON_AL_FIL1

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 151E

PAR_TON_AL_FIL2

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 151F

PAR_TON_AL_FIL3

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1520

PAR_TOFF_AL_FIL1

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1521

PAR_TOFF_AL_FIL2

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1522

PAR_TOFF_AL_FIL3

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1523

PAR_NUM_INTENTOS_FIL1

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1524

PAR_NUM_INTENTOS_FIL2

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1525

PAR_NUM_INTENTOS_FIL3

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1526

SONDAS

Dimensión: [1..27]

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1600

ENTRADAS_DIGITALES

Dimensión: [1..43]

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1630

POLARIDAD_DI

Dimensión: [1..43]

Grupo: PARAMETROS_CONFIGURACION

Tipo: BOOL

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1660

RELE_SALIDA

Dimensión: [1..36]

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 1690

SALIDAS_ANALOG

Dimensión: [1..15]

Grupo: PARAMETROS_CONFIGURACION

Tipo: DINT

Atributo: Libre

Sentido: Memory

No volátil: SI

Dirección: 16C0

ALARM_VENT_IMPULSION

Grupo: ALARMAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1703

ALARM_VENT_RETORNO

Grupo: ALARMAS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1704

WARNING_VENT_IMPULSION

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A06

WARNING_VENT_RETORNO

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A07

WARNING_FILTRO_ENTRADA

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A08

WARNING_FILTRO_IMPULSION

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A09

WARNING_FILTRO_RETORNO

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A0A

WARNING_H_FLUID_TMP

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A0B

WARNING_L_FLUID_TMP

Grupo: WARNINGS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1A0C

KB_ONOFF

Grupo: COMANDOS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1C00

RESET_ALARM_MANUAL

Grupo: COMANDOS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1C01

KB_VEL

(* Selección de velocidad: 0,1,2,3(AUTO) *)

Grupo: COMANDOS

Tipo: DINT

Atributo: Libre

Sentido: Memory

Dirección: 1C06

KB_MODO

(* Modo verano/invierno *)

Grupo: COMANDOS

Tipo: BOOL

Atributo: Libre

Sentido: Memory

Dirección: 1C07

Anexo E

Manual funcionamiento Unidad de Tratamiento de Aire

Contenido

| | |
|---|----|
| Contenido | 1 |
| 1.- Descripción control. Pantalla bienvenida..... | 2 |
| 2.- Pantalla principal: Visualización variables principales | 3 |
| 3.- Pantalla menú principal: selección de las diferentes subsecciones | 3 |
| 3.1.- A.-Menú parámetros  | 4 |
| 3.2.- B.-Menú entradas salidas  | 11 |
| 3.3.- C.-Menú consignas  | 12 |
| 3.4.- D.-Menú servicio  | 13 |
| 3.5.- E.- Registro de datos  | 14 |
| 3.6.- F.- Gestión lista de parámetros  | 14 |
| 3.7.- G.- Idiomas  | 16 |
| 3.8.- H.- Avisos  | 16 |
| 3.9.- J.- Configuración fecha y hora  | 16 |
| 4.- ALARMAS Y AVISOS | 17 |
| 4.1.- Listado de alarmas. | 17 |
| 4.2.- Listado de avisos (warnings) | 17 |
| 5.- FUNCIONES ESPECIALES | 18 |
| 5.1.- Gestión de bombas | 18 |
| 5.1.1- Gestión manual | 18 |
| 5.1.2- Gestión automática | 18 |
| 5.2.- Gestión del suelo radiante..... | 18 |
| 6.- Habilitar módulo de expansión..... | 19 |

1.- Descripción control. Pantalla bienvenida

El software Kiconex_UTA_20PED272 está concebido para realizar la gestión de una UTA, incluyendo la regulación de velocidad de los ventiladores y el control de temperatura y humedad. Entre sus características, se encuentra lo siguiente:

- Gestión de 3 rangos de velocidad y un modo automático. Dentro de esta gestión es posible:
 - Selección manual de velocidad: baja, media, alta.
 - Control automático de la velocidad en función de la temperatura.
 - Gestión de alarmas y alertas en los ventiladores.
- Gestión de un humectador. Esta gestión tiene en cuenta:
 - Humedad de retorno.
 - Nivel del humectador.
- Gestión de válvula de tres vías. En esta gestión se tiene en cuenta:
 - Modo de funcionamiento: verano/invierno
 - Temperatura de retorno
 - Consigna de temperatura

Control escalable:

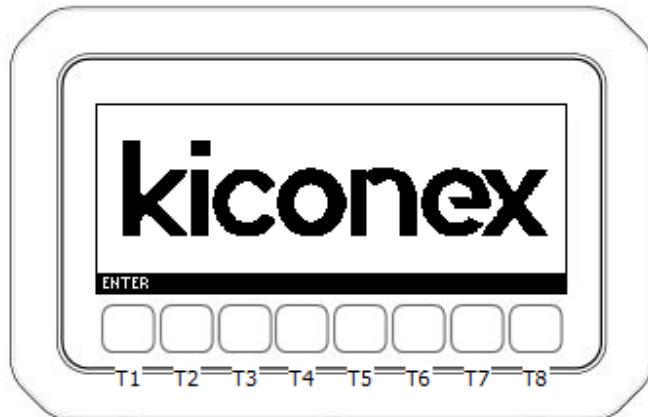
- Versión 4DIN (8 relés, 6 sondas, 11 entradas digitales y 4 salidas analógicas)
- Versión 10DIN (15 relés, 10 sondas, 20 entradas digitales y 6 salidas analógicas)
- Módulo expansión 4DIN(6 relés extra, 7 sondas extras, 3 entradas digitales extra y 3 salidas analógicas extra)
- Módulo expansión 10DIN(15 relés extra, 10 sondas extras, 20 entradas digitales extra y 6 salidas analógicas extra)

Por ejemplo Versión 10DIN+módulo expansión 4DIN serían: 21 relés, 17 sondas, 23 entradas digitales y 9 salidas analógicas

El control dispone de un web server interno al cual se puede conectar mediante cable Ethernet (para módulo DIN4 es necesario un adaptador). Desde dicha página web se puede gestionar:

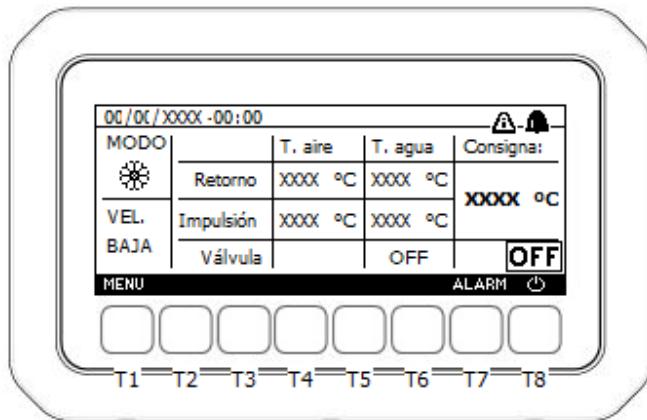
- Estado actual control (sondas, relés, entradas digitales, estados,...etc)
- Gestión alarmas e histórico alarmas.
- Datalog. Registro de datos de funcionamiento.
- Configuración de parámetros (parámetros individuales, archivos backup,...etc)
- Representación gráfica de los datos de funcionamiento.

Pantalla de bienvenida personalizada.



T1 **ENTER:** Pulsar el botón ENTER para ir a la pantalla principal del control o esperar 10s tras los cuales saltará automáticamente a la pantalla principal del control.

2.- Pantalla principal: Visualización variables principales



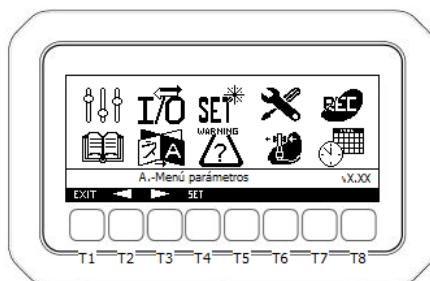
En esta pantalla se pueden visualizar las principales variables del control.

Por un lado, se muestra el modo de funcionamiento (**INVIERNO/VERANO**). También se muestra la temperatura de consigna establecida y las temperaturas de impulsión y retorno tanto de aire como de agua, además del estado de la válvula que gestiona dicha agua. Arriba a la izquierda se puede ver la fecha y hora del control, y a la derecha se puede ver si existen avisos o alarmas.

BOTONES

- T1** **MENU:** Acceso a la pantalla menú principal desde donde se encuentran los diversos apartados del control (ver Sección 3).
- T7** **ALARM:** Acceso a la sección de alarmas para visualizar las alarmas activas o resetearlas.
- T8** **ON-OFF:** Apagado o encendido del equipo. Para apagar o encender el equipo es necesario pulsar el botón por más de 3segundos. El equipo arrancara siempre y cuando no esté su entrada digital de on-off remoto activa.

3.- Pantalla menú principal: selección de las diferentes subsecciones



Este es el menú principal del equipo. Desde aquí es posible acceder a múltiples submenús del sistema, que permiten al usuario configurar tanto el comportamiento del control, como otros parámetros de comunicación y servicio.

Con las teclas "T2" y "T3" es posible desplazarse dentro de este menú. Al desplazarse por esta sección, es posible ver el nombre de cada sección al situarse encima de cada ícono. Para acceder a cualquier sección, basta con pulsar "T4" (SET) una vez que el cursor esté situado encima de la imagen correspondiente.

- **A.- Menú parámetros.** En esta sección es posible cambiar la configuración de las entradas y salidas, así como el funcionamiento del equipo.
- **B.-Menú entradas-salidas.** Aquí es posible visualizar el estado de las entradas y salidas del equipo, así como probarlas de manera manual mediante el modo "TEST I/O".
- **C.-Menú consignas.** Submenú sin función.
- **D.-Menú servicio.** En esta sección es posible cambiar los parámetros de comunicación del equipo (IP, Modbus).
- **E.-Registro de datos.** Aquí se configura el registro de datos del equipo. Solo si existen gráficas en el webserver.
- **F.-Gestión lista de parámetros.** Acceso al menú para carga/descarga de parámetros.
- **G.-Idiomas.** Acceso al menú para modificar el idioma del display.
- **H.-Avisos.** Acceso a la sección de avisos (warning), el control nos informa sobre posibles problemas del equipo.
- Submenú sin función
- **J.-Configuración fecha y hora.** Configuración del reloj interno del control.

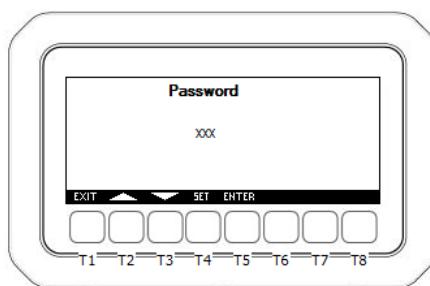
En esta pantalla también es posible ver la versión de software del equipo (abajo a la derecha).

NOTA: Las listas de parámetros, pantallas, página web, librerías kiconex, etc., deberán ser compatibles con la versión de software en uso. En los manuales aparece la versión a la que está referida el documento.

Botones:

- T1** **EXIT:** Volver a la pantalla principal.
T2 : elemento previo. Permite desplazar el cursor hacia atrás en el menú.
T3 : elemento previo. Permite desplazar el cursor hacia delante en el menú.
T4 **SET:** Acceder al submenú seleccionado.

3.1.- A.-Menú parámetros

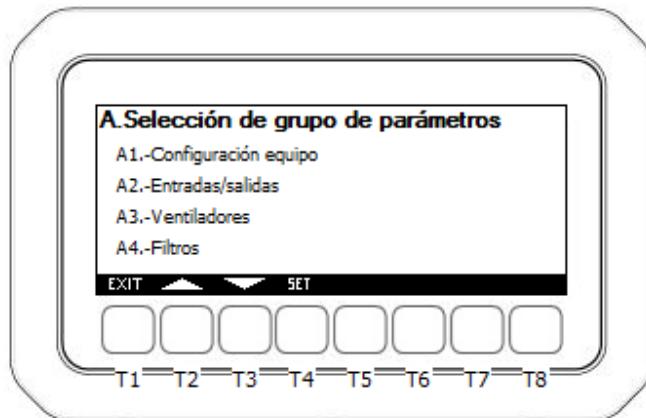


Esta pantalla aparecerá tras seleccionar el submenú "A.- Menú parámetros". Para acceder al menú, es necesario pulsar set para modificar la contraseña (Password). Tras elegir el valor correspondiente, pulsar "T1" (ENTER). La contraseña de este control es "**0**".

Botones:

- T1** **EXIT:** Volver al menú anterior.
T2 **ARRIBA:** Aumentar en 1 el valor.
T3 **ABAJO:** Disminuir en un el valor.
T4 **SET:** Activar la variable y fijar el valor.
T5 **ENTER:** Verificar la contraseña.

Una vez se haya introducido la contraseña correcta, aparecerá la siguiente pantalla:



Las secciones disponibles serán:

- **A1.- Configuración equipo:** Gestión del módulo de expansión.
- **A2.- Entradas/salidas:** Configuración de las entradas y salidas del equipo.
- **A3.- Ventiladores:** Configuración de velocidad y alarmas de ventiladores.
- **A4.- Filtros:** Configuración general de las alarmas en filtros de aire.

En la siguiente página aparece un listado detallado de las variables de configuración del equipo, así como los distintos valores que pueden tomar.

MENÚ CONFIGURACIÓN

Parámetros de funcionamiento del equipo

| CATEGORIA | PARAMETRO | PANTALLA | NOMBRE | DESCRIPCION |
|---------------------------|--------------|--|---|---|
| A1.- Configuración Equipo | CNF01 | A1.a-Configuración equipo | Habilitar módulo expansión | Habilitar módulos de expansión (DIN4/DIN10) |
| | HUM01 | A1.a-Configuración equipo | Consigna de humedad | Establecer consigna de humedad en %HR |
| | HUM02 | A1.a-Configuración equipo | Banda de humedad | Establecer banda para control de humedad, en %HR |
| | TMP01 | A1.b-Configuración equipo | Consigna de temperatura | Establecer consigna de temperatura en °C |
| | TMP02 | A1.b-Configuración equipo | Banda de temperatura en modo invierno | Establecer banda de temperatura para la regulación en el modo invierno (°C) |
| | TMP03 | A1.b-Configuración equipo | Banda de temperatura en modo verano | Establecer banda de temperatura para la regulación en el modo verano (°C) |
| A3.- Ventiladores | FAN01 | A3.a-Configuración manual ventiladores | Valor velocidad baja | Establecer valor de velocidad baja. |
| | FAN02 | | Valor velocidad media | Establecer valor de velocidad media. |
| | FAN03 | | Valor velocidad alta | Establecer valor de velocidad alta. |
| | FAN04 | | Valor velocidad mínima | Establecer valor de velocidad mínima para la regulación automática. |
| | FAN05 | | Valor velocidad máxima | Establecer valor de velocidad máxima para la regulación automática. |
| | FAN06 | | Banda de velocidad | Establecer band de velocidad para la regulación automática. |
| | FAN07 | | Retardo de activación de alarma en ventilador de impulsión | Tiempo de retardo de activación de alarma en ventilador de impulsión en segundos |
| | FAN08 | | Retardo de desactivación de alarma en ventilador de impulsión | Tiempo de retardo de desactivación de alarma en ventilador de impulsión en segundos |
| | FAN11 | | Alarmas hasta bloqueo | Número de alarmas en el ventilador de impulsión hasta bloqueo |
| | FAN09 | | Retardo de activación de alarma en ventilador de retorno | Tiempo de retardo de activación de alarma en ventilador de retorno en segundos |
| | FAN10 | | Retardo de desactivación de alarma en ventilador de retorno | Tiempo de retardo de desactivación de alarma en ventilador de retorno en segundos |
| | FAN12 | | Alarmas hasta bloqueo | Número de alarmas en el ventilador de retorno hasta bloqueo |
| A4.- Filtros | FIL01 | | Retardo de activación de aviso en filtro de entrada de aire | Retardo de activación de aviso en filtro de entrada de aire en segundos |
| | FIL02 | | Retardo de desactivación de alarma en filtro de entrada de aire | Retardo de desactivación de alarma en filtro de entrada de aire en segundos |

| | | | | |
|--|--------------|--|--|--|
| | FIL03 | A4.a- Alarms filtro impulsión aire | Retardo de activación de aviso en filtro de impulsión de aire | Retardo de activación de aviso en filtro de impulsión de aire en segundos |
| | FIL04 | | Retardo de desactivación de alarma en filtro de entrada de aire | Retardo de desactivación de alarma en filtro de impulsión de aire en segundos |
| | FIL05 | A4.a- Alarms filtro retorno aire | Retardo de activación de aviso en filtro de retorno de aire | Retardo de activación de aviso en filtro de retorno de aire en segundos |
| | FIL06 | | Retardo de desactivación de alarma en filtro de retorno de aire | Retardo de desactivación de alarma en filtro de retorno de aire en segundos |

Parámetros de configuración de las entradas/salidas

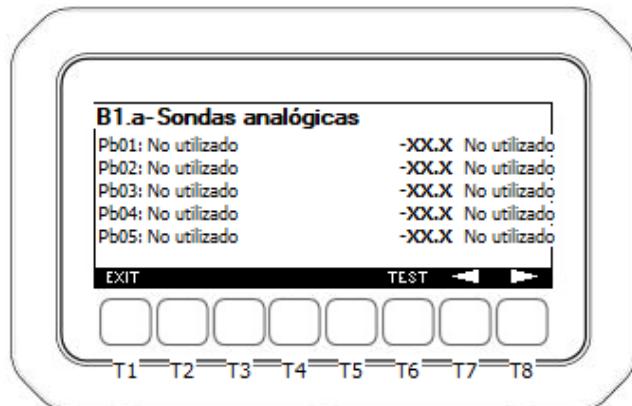
| CATEGORIA | PARAMET. | NOMBRE | DESCRIPCION |
|---------------------------|--------------|---|---|
| A2.c.- Entradas digitales | DIG01 | Conf.entrada digital 1 | INV(FALSE)/DIR(TRUE) : polaridad inversa o directa 0=No utilizado : entrada sin función asociada |
| | DIG02 | Conf.entrada digital 2 | 1=ON-OFF : Interruptor de ON/OFF remoto |
| | DIG03 | Conf.entrada digital 3 | 2=DI. Ventilador Impulsión : seguridad ventilador impulsión |
| | DIG04 | Conf.entrada digital 4 | 3=DI. Ventilador retorno : seguridad ventilador retorno |
| | DIG05 | Conf.entrada digital 5 | 4=DI. Nivel Humectador : seguridad nivel humectador |
| | DIG06 | Conf.entrada digital 6 | 5=DI. Filtro entrada aire : seguridad filtro entrada aire |
| | DIG07 | Conf.entrada digital 7 | 6=DI. Filtro impulsión aire : seguridad filtro impulsión aire |
| | DIG08 | Conf.entrada digital 8 | 7=DI. Filtro retorno aire : seguridad filtro retorno aire |
| | DIG09 | Conf.entrada digital 9 | |
| | DIG10 | Conf.entrada digital 10 | |
| | DIG11 | Conf.entrada digital 11 | |
| | DIG12 | Conf.entrada digital 12 | |
| | DIG13 | Conf.entrada digital 13 | |
| | DIG14 | Conf.entrada digital 14 | |
| | DIG15 | Conf.entrada digital 15 | |
| | DIG16 | Conf.entrada digital 16 | |
| | DIG17 | Conf.entrada digital 17 | |
| | DIG18 | Conf.entrada digital 18 | |
| | DIG19 | Conf.entrada digital 19 | |
| | DIG20 | Conf.entrada digital 20 | |
| | DIG21 | Conf.entrada digital 01 módulo expans. DIN4 | |
| | DIG22 | Conf.entrada digital 02 módulo expans. DIN4 | |
| | DIG23 | Conf.entrada digital 03 módulo expans. DIN4 | |
| | DIG24 | Conf.entrada digital 01 módulo expans. DIN10 | |
| | DIG25 | Conf.entrada digital 02 módulo expans. DIN10 | |
| | DIG26 | Conf.entrada digital 03 módulo expans. DIN10 | |
| | DIG27 | Conf.entrada digital 04 módulo expans. DIN10 | |
| | DIG28 | Conf.entrada digital 05 módulo expans. DIN10 | |
| | DIG29 | Conf.entrada digital 06 módulo expans. DIN10 | |
| | DIG30 | Conf.entrada digital 07 módulo expans. DIN10 | |
| | DIG31 | Conf. entrada digital 08 | |

| | | | |
|--------------------------|--|--|--|
| | | módulo expans. DIN10 | |
| DIG32 | Conf. entrada digital 09 módulo expans. DIN10 | | |
| DIG33 | Conf. entrada digital 10 módulo expans. DIN10 | | |
| DIG34 | Conf. entrada digital 11 módulo expans. DIN10 | | |
| DIG35 | Conf. entrada digital 12 módulo expans. DIN10 | | |
| DIG36 | Conf. entrada digital 13 módulo expans. DIN10 | | |
| DIG37 | Conf. entrada digital 14 módulo expans. DIN10 | | |
| DIG38 | Conf. entrada digital 15 módulo expans. DIN10 | | |
| DIG39 | Conf. entrada digital 16 módulo expans. DIN10 | | |
| DIG40 | Conf. entrada digital 17 módulo expans. DIN10 | | |
| DIG41 | Conf. entrada digital 18 módulo expans. DIN10 | | |
| DIG42 | Conf. entrada digital 19 módulo expans. DIN10 | | |
| DIG43 | Conf. entrada digital 20 módulo expans. DIN10 | | |
| A2.a.- Sondas analógicas | PBS01 | Configuración sonda 1 | 0=no usado: sonda sin función |
| | PBS02 | Configuración sonda 2 | 1=T^a impulsión aire |
| | PBS03 | Configuración sonda 3 | 2=T^a retorno aire |
| | PBS04 | Configuración sonda 4 | 3=T^a impulsión agua |
| | PBS05 | Configuración sonda 5 | 4=T^a retorno agua |
| | PBS06 | Configuración sonda 6 | 5=T^a extracción aire |
| | PBS07 | Configuración sonda 7 | 6=T^a aire extraído |
| | PBS08 | Configuración sonda 8 | 7=T^a extracción aire |
| | PBS09 | Configuración sonda 9 | 8=Humedad de retorno |
| | PBS10 | Configuración sonda 10 | 9=Humedad exterior |
| | PBS11 | Configuración sonda 1 módulo expan. DIN4 | |
| | PBS12 | Configuración sonda 2 módulo expan. DIN4 | |
| | PBS13 | Configuración sonda 3 módulo expan. DIN4 | |
| | PBS14 | Configuración sonda 4 módulo expan. DIN4 | |
| | PBS15 | Configuración sonda 5 módulo expan. DIN4 | |
| | PBS16 | Configuración sonda 6 módulo expan. DIN4 | |
| | PBS17 | Configuración sonda 7 módulo expan. DIN4 | |
| | PBS18 | Configuración sonda 1 módulo expan. DIN10 | |
| | PBS19 | Configuración sonda 2 módulo expan. DIN10 | |
| | PBS20 | Configuración sonda 3 módulo expan. DIN10 | |
| | PBS21 | Configuración sonda 4 módulo expan. DIN10 | |
| | PBS22 | Configuración sonda 5 módulo expan. DIN10 | |
| | PBS23 | Configuración sonda 6 módulo expan. DIN10 | |

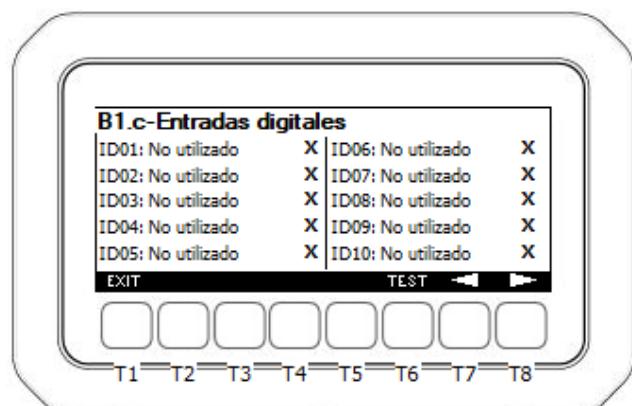
| | | | |
|------------------|--------------|---|--|
| | PBS24 | Configuración sonda 7 Módulo expan. DIN10 | |
| | PBS25 | Configuración sonda 8 módulo expan. DIN10 | |
| | PBS26 | Configuración sonda 9 módulo expan. DIN10 | |
| | PBS27 | Configuración sonda 10 módulo expan. DIN10 | |
| A2.e.- salida | RLO01 | Config. Relé salida 1 | 0=No usado: relé sin función asociada |
| | RLO02 | Config. Relé salida 2 | 1=Válvula de 3 vías: activación válvula de agua |
| | RLO03 | Config. Relé salida 3 | 2=Humectador: activación humectador |
| | RLO04 | Config. Relé salida 4 | 3=Alarma: activación alarma |
| | RLO05 | Config. Relé salida 5 | |
| | RLO06 | Config. Relé salida 6 | |
| | RLO07 | Config. Relé salida 7 | |
| | RLO08 | Config. Relé salida 8 | |
| | RLO09 | Config. Relé salida 9 | |
| | RLO10 | Config. Relé salida 10 | |
| | RLO11 | Config. Relé salida 11 | |
| | RLO12 | Config. Relé salida 12 | |
| | RLO13 | Config. Relé salida 13 | |
| | RLO14 | Config. Relé salida 14 | |
| | RLO15 | Config. Relé salida 15 | |
| | RLO16 | Config. Relé salida 1 módulo expan.DIN4 | |
| | RLO17 | Config. Relé salida 2 módulo expan.DIN4 | |
| | RLO18 | Config. Relé salida 3 módulo expan.DIN4 | |
| | RLO19 | Config. Relé salida 4 módulo expan.DIN4 | |
| | RLO20 | Config. Relé salida 5 módulo expan.DIN4 | |
| | RLO21 | Config. Relé salida 6 módulo expan.DIN4 | |
| | RLO22 | Config. Relé salida 1 módulo expan.DIN10 | |
| | RLO23 | Config. Relé salida 2 módulo expan.DIN10 | |
| | RLO24 | Config. Relé salida 3 módulo expan.DIN10 | |
| | RLO25 | Config. Relé salida 4 módulo expan.DIN10 | |
| | RLO26 | Config. Relé salida 5 módulo expan.DIN10 | |
| | RLO27 | Config. Relé salida 6 módulo expan.DIN10 | |
| | RLO28 | Config. Relé salida 7 módulo expan.DIN10 | |
| | RLO29 | Config. Relé salida 8 módulo expan.DIN10 | |
| | RLO30 | Config. Relé salida 9 módulo expan.DIN10 | |
| | RLO31 | Config. Relé salida 10 módulo expan.DIN10 | |
| | RLO32 | Config. Relé salida 11 módulo expan.DIN10 | |
| | RLO33 | Config. Relé salida 12 módulo expan.DIN10 | |
| | RLO34 | Config. Relé salida 13 módulo expan.DIN10 | |

| | | | |
|---------------------------|--------------|--|--|
| | RLO35 | Config. Relé salida 14 módulo expan.DIN10 | |
| | RLO36 | Config. Relé salida 15 módulo expan.DIN10 | |
| A2.h.- salidas analógicas | ANA01 | Config. Salida ana.1 | 0=No utilizado: salida sin función asociada 1=Vel. Ventilador imp.: salida analógica para regular velocidad ventilador de impulsión de aire 2=Vel. Ventilador ret.: salida analógica para regular velocidad ventilador de retorno de aire |
| | ANA02 | Config. Salida ana. 2 | |
| | ANA03 | Config. Salida ana. 3 | |
| | ANA04 | Config. Salida ana. 4 | |
| | ANA05 | Config. Salida ana. 5 | |
| | ANA06 | Config. Salida ana. 6 | |
| | ANA07 | Config. Salida ana. 1 módulo expan. DIN4 | |
| | ANA08 | Config. Salida ana. 2 módulo expan. DIN4 | |
| | ANA09 | Config. Salida ana. 3 módulo expan.DIN4 | |
| | ANA10 | Config. Salida ana. 1 módulo expan.DIN10 | |
| | ANA11 | Config. Salida ana. 2 módulo expan.DIN10 | |
| | ANA12 | Config. Salida ana. 3 módulo expan.DIN10 | |
| | ANA13 | Config. Salida ana. 4 módulo expan.DIN10 | |
| | ANA14 | Config. Salida ana. 5 módulo expan.DIN10 | |
| | ANA15 | Config. Salida ana. 6 módulo expan.DIN10 | |

3.2.- B.-Menú entradas salidas I/O



En este submenú es posible comprobar el estado de todas las entradas y salidas del control. Por ejemplo, en la imagen superior se indicaría la función asignada a cada sonda, así como el valor registrado.

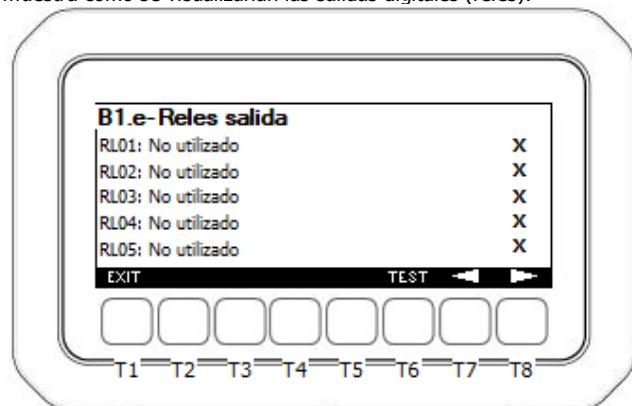


En las entradas digitales, además del estado de dicha entrada, se mostraría también la polaridad de la misma.

0: Contacto no activo

1: Contacto activo

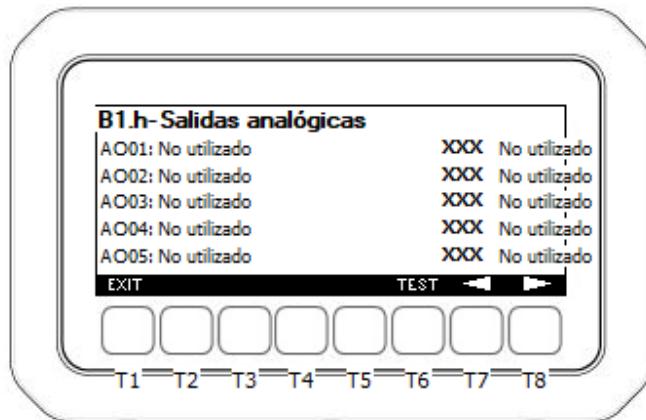
En la siguiente imagen se muestra como se visualizarían las salidas digitales (relés):



0: Contacto no activo

1: Contacto activo

Por último, se mostraría el estado de las salidas analógicas en %:



Activación modo TEST.

Esta pantalla solo aparecerá si el control se encuentra apagado (OFF). Si está apagado, aparecerá una tecla "TEST" (T6), con la cual se podrá acceder a un nuevo menú. Aquí se puede modificar el estado de las entradas y salidas para comprobar el funcionamiento del equipo.

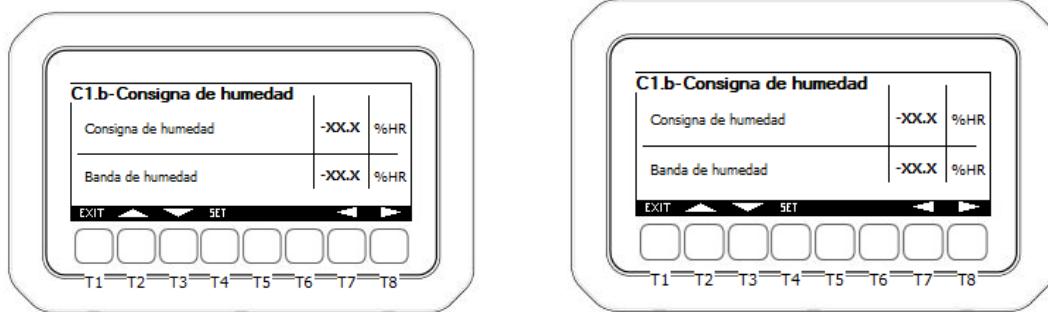


Es posible forzar las salidas de relé, y fijar el valor deseado en las salidas analógicas. Bastaría con desplazarse hasta el parámetro que se desea modificar, pulsar SET, y establecer el valor deseado. Una vez hecho esto, pulsando de nuevo SET se fijaría dicho valor.

Una vez pulsada la tecla "EXIT", todos los valores volverían a su estado de partida.

3.3.- C.-Menú consignas SET*

En este menú es posible establecer el modo de funcionamiento (invierno/verano), así como cambiar las consignas y bandas de humedad y temperatura.

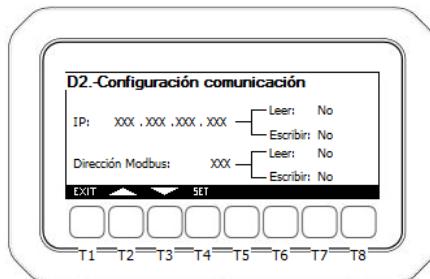


3.4.- D.-Menú servicio

Desde este menú es posible cambiar los parámetros de comunicación del equipo.



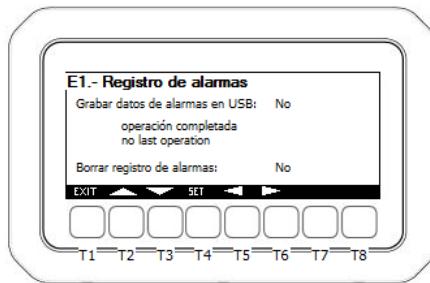
Al entrar en "D1.- Configuración comunicación" se tendría una pantalla similar a la siguiente:



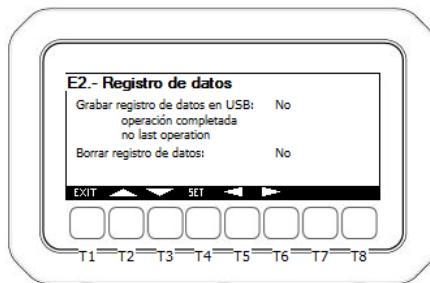
Es posible ver y cambiar tanto la IP del equipo como la dirección Modbus del mismo. Es recomendable que, una vez hechos los cambios, se proceda al reinicio del dispositivo.

Antes de cambiar cualquier valor es necesario leer el actual. Para ello, desplazarse sobre "leer" utilizando las flechas, y cambiar el valor "NO" a "SÍ". Tras unos segundos, aparecerá la IP o dirección Modbus actual del equipo. Hecho esto, ya se podrá cambiar. Para ello, desplazarse hacia el valor que se desea modificar, y mediante la tecla "SET" y las flechas, cambiarlo. Una vez esté en el valor deseado, desplazarse hacia "escribir", y cambiar aquí el valor de "NO" a "SÍ".

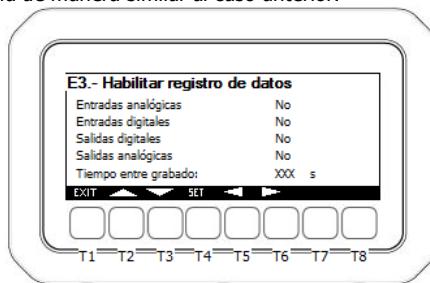
3.5.- E.- Registro de datos



En esta sección se pueden exportar los registros de alarmas a una memoria USB externa. Para ello, únicamente habrá que insertar el USB en el puerto correspondiente, y cambiar el valor de la variable “**grabar datos de alarmas en USB**” a “**SI**”. También es posible eliminar el histórico de alarmas borrando el registro correspondiente.



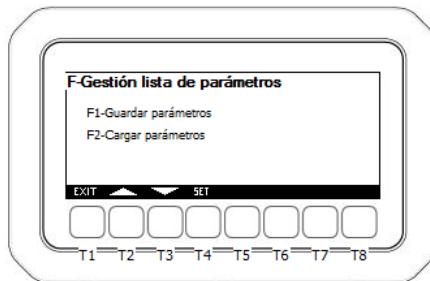
En esta pantalla se puede exportar el registro de datos recogidos por el control a una memoria USB, además de borrar los datos almacenados. Todo esto se haría de manera similar al caso anterior.



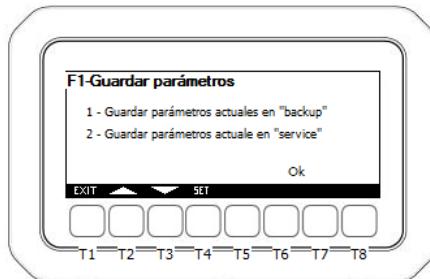
Por último, en esta pantalla es posible elegir qué variables se van a registrar. Para ello, habrá que cambiar a “**SI**” aquellas variables que queramos registrar. El tiempo entre grabados también es configurable.

3.6.- F.- Gestión lista de parámetros

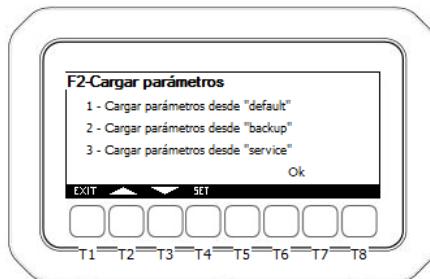
En esta sección es posible guardar o cargar las listas de parámetros con la configuración del equipo.



Al guardar los parámetros (F1), la configuración actual del dispositivo se copia en los archivos de memoria.



Cuando se cargan parámetros desde memoria, se realiza una copia de estos y se actualizan los del control.

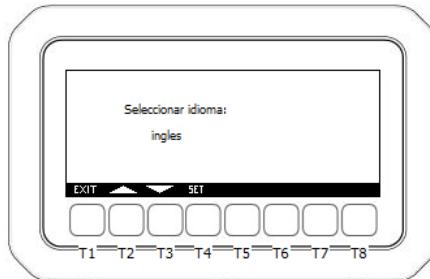


El control dispone de tres ficheros para gestionar los parámetros. Cada uno tiene una función distinta:

- **Por defecto (default):** Estos parámetros son los recomendados por el fabricante. Solo son de lectura. Para cargarlos, basta con seleccionarlo y pulsar "SET"
- **Copia actual (backup):** En este fichero se guardan los parámetros actuales. De esta manera, en caso de realizar una mala configuración, es posible volver a un punto "seguro" de configuración. El fichero "**backup**" es copiado cada 24h, así siempre está disponible una copia de seguridad.
- **Servicio (service):** Estos parámetros obvian alarmas y temporizaciones de seguridad, pensando que el equipo está siendo manejado por personal altamente cualificado.

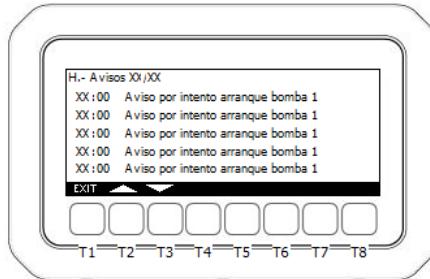
Por último, es posible conocer el estado de la lectura/escritura de los parámetros en el texto situado abajo a la derecha.

3.7.- G.- Idiomas



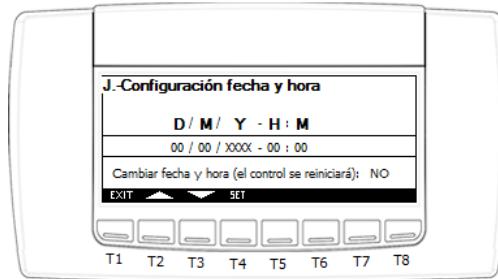
En esta sección es posible modificar el idioma de la pantalla. Solo hace falta navegar por la lista hasta encontrar el idioma deseado.

3.8.- H.- Avisos



En esta sección es posible visualizar los avisos del control. Estos avisos son meramente informativos, y no provocarán ningún tipo de fallo en el control, pero es importante ya que se registran pequeños fallos del control.

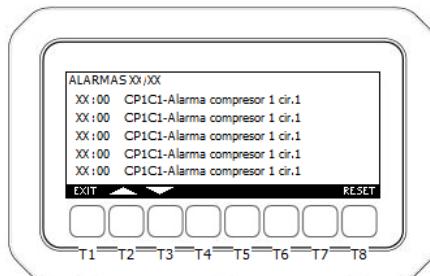
3.9.- J.- Configuración fecha y hora



Aquí es posible modificar la fecha y hora interna del control, en caso de que no corresponda con la actual. Al realizar los cambios correspondientes, el control se reiniciará.

4.- ALARMAS Y AVISOS

4.1.- Listado de alarmas.



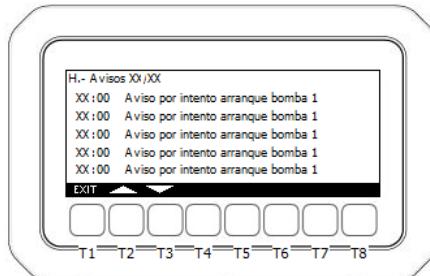
En caso de producirse una alarma, es posible ver el estado de las mismas si se accede desde la pantalla principal a través de la tecla "**ALARM**".

El listado y significado de cada código queda resumido en la siguiente tabla:

ALARMAS

| Código | Descripción | Efecto | Reset |
|--------|--------------------------------|-----------------------------|---|
| PBE1 | Alarma sonda X desconectada | Lectura de valor errónea | Automático. Al reconectarse la sonda correspondiente. |
| PBE2 | | | |
| PBE3 | | | |
| ... | | | |
| PSR1 | Alarma bombas suelo radiante 1 | Detiene el grupo de bombas. | Manual, pulsar reinicio de alarmas. |
| PFC1 | Alarma bombas fancoils 1 | | |
| ... | | | |

4.2.- Listado de avisos (warnings)



En esta pantalla se podrán ver los avisos de funcionamiento no deseado del equipo. Estos avisos no afectan al funcionamiento del control.

AVISOS

| Descripción | Causa |
|-------------|--|
| Bomba X | Detectada señal positiva en la entrada de seguridad del grupo de bombas correspondiente. |

5.- FUNCIONES ESPECIALES

5.1.- Gestión de ventiladores

Es posible gestionar el funcionamiento de los ventiladores, tanto de manera manual como de manera automática.

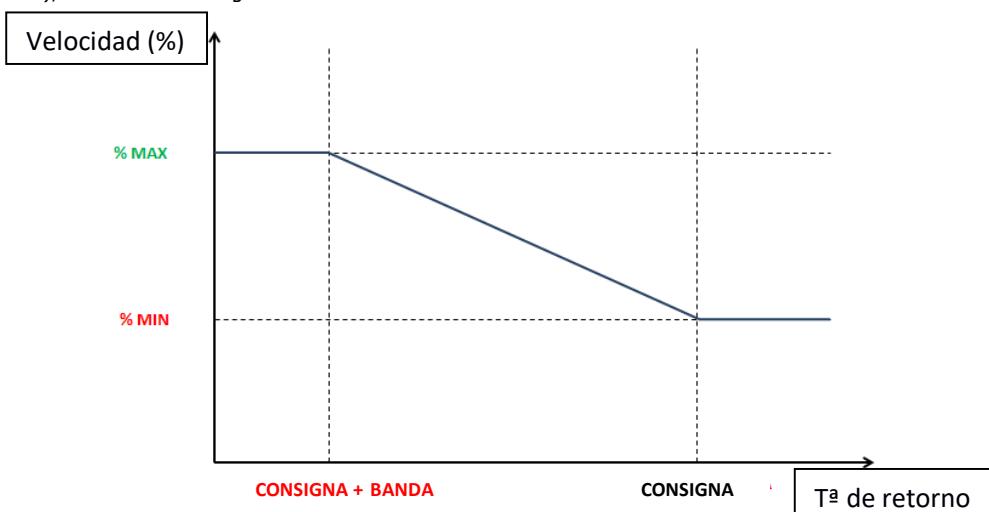
5.1.1- Gestión manual de la velocidad

Para realizar la gestión manual de los ventiladores, basta con modificar el comando “**KB_MODO**” de la pantalla de consignas, usando los valor 0 (Velocidad Baja), 1(Velocidad Media) ó 2(Velocidad Alta). Las velocidades características de cada rango se pueden establecer desde la ventana de configuración de ventiladores, a través de los parámetros “**FAN01**”, “**FAN02**” y “**FAN03**”. También es posible realizar la gestión manual mediante un sistema de supervisión externo (como Kiconex).

5.1.2- Gestión automática de la velocidad

También es posible gestionar los ventiladores cambiando “**KB_MODO**” a 3 (Modo Automático). Para la regulación se puede cambiar la velocidad máxima y la velocidad mínima, a través de los parámetros “**FAN04**” y “**FAN05**” de la ventana de configuración de ventiladores.

La regulación se realiza midiendo la temperatura del aire de retorno y comparando su valor con la consigna de temperatura establecida “**TMP01**” y con la banda de velocidad “**FAN06**” (que se configura en la misma ventana de configuración de ventiladores), estableciendo la siguiente relación:



5.2.- Gestión de Temperatura

Es posible controlar la recirculación de agua a la temperatura deseada a través de una válvula de 3 vías. Para ello, se dispone de tres parámetros totalmente modificables, que permiten ajustar la regulación. La lógica es la siguiente:

- Se establece una temperatura de consigna (**TMP01**).
- Se establecen una diferencia de temperatura (banda) por encima de la consigna para el modo invierno (**TMP02**) y por debajo de la consigna para el modo verano (**TMP03**).
- Se mide la temperatura de retorno de aire, y se controla la válvula de la siguiente manera:
 - Se abre la válvula para permitir el paso de agua hasta alcanzar consigna.
 - Se cierra la válvula tras alcanzar consigna, hasta volver a la banda de temperatura establecida.

5.3.- Gestión de Humedad

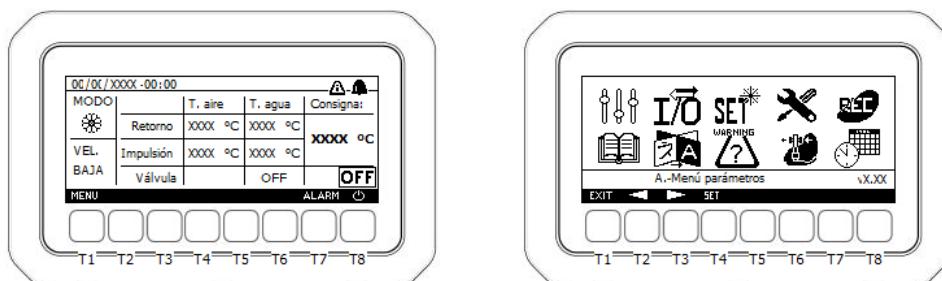
Es posible controlar la recirculación de humedad a través de un humectador. Para ello, se dispone de dos parámetros totalmente modificables, que permiten ajustar la regulación. La lógica es similar a la regulación de temperatura:

- Se establece una humedad de consigna (**HUM01**).
- Se establecen una diferencia de humedad (banda) (**HUM02**).
- Se mide la humedad de retorno de aire, y se controla el humectador de la siguiente manera:
 - Se activa el humectador hasta alcanzar consigna.
 - Se desactiva el humectador tras alcanzar consigna, hasta volver a la banda de humedad establecida.

5.4.- Gestión de Filtros

Es posible la detección de filtros sucios a través de una serie de avisos. Existen avisos para los filtros de impulsión, entrada y retorno de aire.

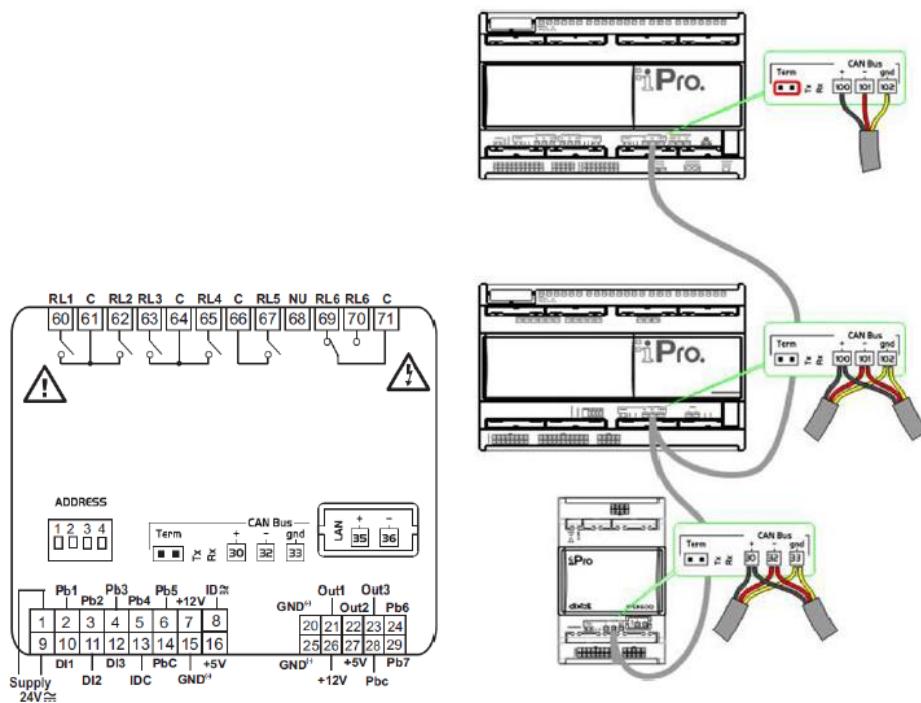
En la esquina superior derecha de la pantalla principal se puede ver la existencia de avisos activos y en el menú principal, en la ventana de warnings se pueden ver los avisos de todo tipo, incluidos los filtros.



6.- Habilitar módulo de expansión

Si el control se quedase pequeño, es posible añadir varios módulos de expansión al mismo. De esta manera se consigue ampliar el número de entradas y salidas disponibles en el control. Existen distintas expansiones en función de las necesidades de control.

Para habilitar este módulo, es necesario ajustar el parámetro "**CNF01**" en "**SI**". La conexión entre el control principal y los módulos de expansión se realiza mediante un cable de 3 hilos conectado al conector "**CAN BUS**" de cada control. Par que el módulo funcione correctamente, se debe asignar la dirección "**DIP**" a 1:



Cuando se configuran las ampliaciones, hay que tener en cuenta que el módulo principal debe tener la dirección 1 y los demás módulos direcciones a partir de 2 para que funcione correctamente.

La combinación de entradas y salidas es la siguiente según la combinación de módulos:

| | IPG208 (DIN4) | IPG215 (DIN10) | IPG215+IPX206 (DIN10+exp.DIN4) | IPG215+IPX215 (DIN10+exp.DIN10) | IPG215+IPX206+IPG215 (DIN10+exp.DIN4+exp.DIN10) |
|----------------------------------|---------------|----------------|--------------------------------|---------------------------------|---|
| Entra.analógicas (sondas) | 6 | 10 | 17 | 20 | 27 |
| Salidas digitales (relés) | 8 | 15 | 21 | 30 | 36 |
| Entradas digitales | 11 | 20 | 23 | 40 | 43 |
| Salidas analógicas | 3 | 6 | 9 | 12 | 15 |