INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

Siemens Costa Rica

Monitoreo de variables utilizando un sistema maestro-esclavo con PLC´s Siemens por medio del protocolo Modbus ASCII y transmisión inalámbrica.

Informe de Proyecto de graduación para optar por el grado De Bachiller en Ingeniería Electrónica

Carlos Robles Cordero

CARTAGO – junio de 2002

Resumen:

La empresa Siemens de Costa Rica se encarga de automatizar sistemas de control, por ejemplo acueductos. La principal característica de estos sistemas es la considerable distancia entre los tanques de almacenamiento y los pozos de bombeo de agua. Por esto se hace necesaria la implementación de telemetrías, comunicación inalámbrica entres estaciones

Para muchas aplicaciones se exige la utilización de un protocolo de comunicación abierto. De aquí surge este proyecto, desarrollar una comunicación entre plc´s por medio del protocolo Modbus ASCII utilizando comunicación inalámbrica.

El principal objetivo es el de realizar un programa que implemente las funciones necesarias del protocolo garantizando la transferencia de datos entre el maestro y los esclavos. Además de debió desarrollar un programa de fácil modificación de manera que pueda ser utilizado en futuras aplicaciones.

El sistema se desarrolló diseñando cada una de las rutinas, primero se diseñó la comunicación del maestro con un esclavo. Luego se implementó una función y se comprobó que el esclavo la reciba. Seguidamente se programó la rutina en el esclavo que interpreta la solicitud y genera una respuesta, para después ampliar la solicitud a varios esclavos. El manejo del protocolo se completó implementando el resto de las funciones.

Finalmente se realizaron aplicaciones en diferentes telemetrías para verificar la funcionalidad del sistema.

Palabra claves: Modbus ASCII; protocolo abierto; telemetría; Siemens; PLC´s; Programador lógico programable; comunicación; Inalámbrico.

Abstract:

One of the business units of Siemens Costa Rica takes care of implementing different control systems, like in aqueducts. Such applications are characterized by the considerable distance between storage tanks and water wells; this makes wireless communications a must for telemetry transmissions.

For many applications an open protocol is demanded, that's why this project came to be, to use a wireless communication protocol such as modbus ASCII, to transfer data between PLCs

The main objective is to develop a program that can implement the main functions codes of the protocol to guarantee a proper data transfer on a master slave PLC system. It's also important that the final program can be easily modifiable so it can be used on future projects without having to make many changes.

The system was developed by designing each and every routine, first, the master slave communication routine. Then a simple function code was implemented and verified that the slave PLC received it. Afterwards a routine was programmed on the slave so that it allows the slave PLC to acknowledge a query and generate a proper response. This routine was extended to allow the polling of different slaves. Finally the rest of the functions codes were implememented.

To verify that the system works, several different telemtrys were implement.

Keywords: Modbus ASCII; open protocol; telemetry; PLC´s; Siemens; programmable logic controllers; communication; wireless.

Dedicatoria

Deseo dedicar este trabajo a todas las persona que me ayudaron a dar este pequeño paso en mi carrera: por supuesto a mis padres por su soporte y paciencia asi como a mi novia por estar siempre a mi lado.

GRACIAS

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo 1. INTRODUCCIÓN

1.1. Descripción de la empresa

Siemens, AG tiene su base en Alemania de la cual es filial Siemens Costa Rica. Esta es la empresa en el campo de la electrotecnia con mayor trayectoria en el mundo, con más de 159 años de experiencia en el desarrollo de soluciones para la industria y el campo de la aplicación de la ingeniería eléctrica y electrónica en general.

En Costa Rica, Siemens cuenta con más de 40 años de existencia, en los cuales se ha destacado como empresa líder en los campos de servicio en los sectores de telecomunicaciones pública y privada, técnica médica, generación y transmisión de energía y equipo eléctrico industrial.

Siemens de Costa Rica cuenta además con fábrica propia de tableros, departamento de montaje y servicio, departamento de diseño y una bodega con un stock superior a los 2000 artículos.

En la industria, las aplicaciones desarrolladas por el departamento de automatización de Siemens Costa Rica abarcan los más variados procesos, desde máquinas de conteo hasta control y monitoreo de líneas de proceso completas.

En el campo del monitoreo, control y los sistemas de adquisición de datos (SCADA) Siemens se ha visto involucrada con proyectos de la más variada complejidad. Estos van desde telemetría punto a punto de una estación de bombeo hacia un tanque de captación; hasta sistemas de monitoreo, control y adquisición de datos de centrales de generación hidroeléctrica, pasando por sistemas de monitoreo y control de hoteles y edificios.

La figura 1.1 muestra el organigrama del departamento donde se desarrollará el proyecto.

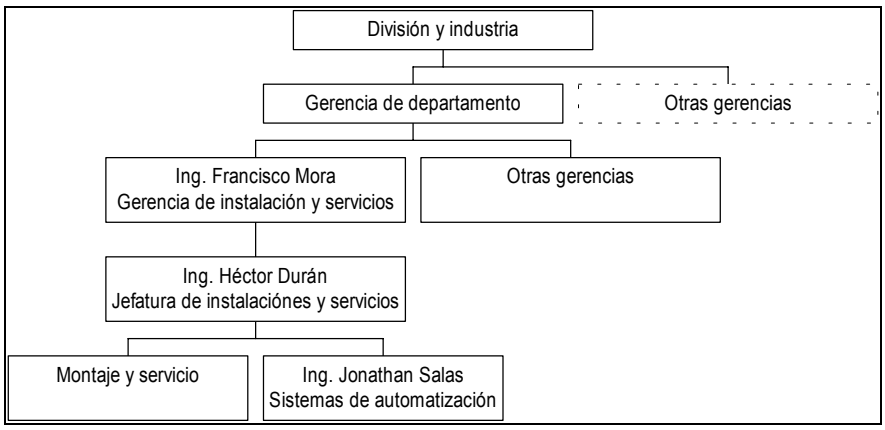

Figura 1.1 Organigrama de la sección donde se desarrollará el proyecto

1.2. Descripción detallada del problema a resolver y sus efectos

Muchos de los proyectos que se realizan en Siemens consisten en el sensado y manejo de información proveniente de sistemas tales como tanques de captación de agua, sistemas de rebombeo, etc. Estos monitoreos se realizan por medio de controladores lógicos programables (PLC´s) que cumplen la función de un esclavo y la información de estos es enviada a un PLC maestro.

La mayoría de los clientes que utilizan estas aplicaciones son en el área de acueductos, tanto públicos como privados. El problema es que por lo general los sistemas que se están sensando y que se desean enviar al maestro, se encuentran separados por una distancia considerable o que el terreno que separa una estación de otra es muy irregular, eliminando la posibilidad de realizar algún tipo de cableado. La figura 1.2 muestra la aplicación típica. Se tienen dos bombas que se desean controlar, sensando el estado de tanques, se determina si se requieren llenar activando las bombas o si el estado es *el correcto. Las distancias son grandes y el terreno irregular. Por lo tanto, la casa de control debe supervisar y controlar el sistema con transmisión por radio.

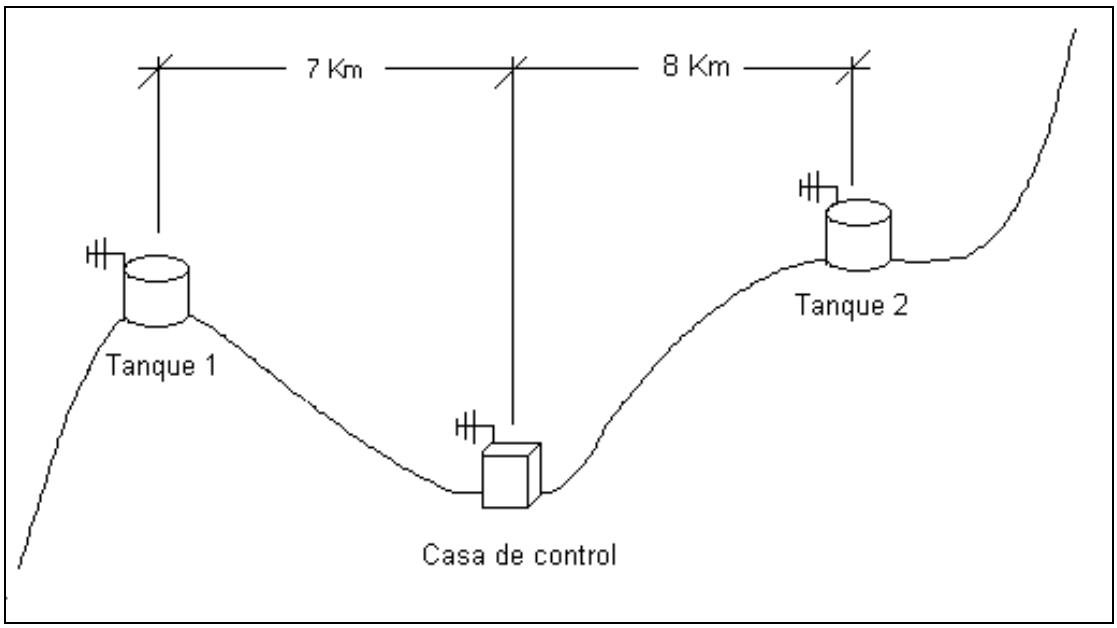

Figura 1.2 Aplicación típica para el sistema a desarrollar

Se debe diseñar una aplicación que permita comunicar los diferentes esclavos con su respectivo maestro, para que asi, se pueda procesar la información y manipular las diferentes estaciones desde un solo punto por medio de un sistema inalámbrico. Siemens cuenta con un protocolo de comunicación propio para realizar este tipo de aplicaciones, sin embargo no es un protocolo abierto.

Muchos de los proyectos desarrollados son por medio de licitación, en donde el cliente da una serie de especificaciones con respecto a las características del sistema. Recientemente se ha estado solicitando en algunos casos que para la comunicación se utilice un protocolo abierto y en otros se especifica el protocolo Modbus, el cual también es abierto . Para dicho protocolo, Siemens no tiene aplicaciones desarrolladas.

De aquí surge la necesidad de desarrollar una aplicación que permita comunicar toda clase de datos utilizando PLC´s y comunicación inalámbrica en protocolo Modbus. En vista de que los proyectos consisten en monitoreo y control de una serie de estaciones, la aplicación debe ser capaz de manejar, con un PLC´s maestro, las diferentes estaciones, realizando una comunicación tanto de transmisión como de recepción de diferentes tipos de información.

Para Siemens, el desarrollo de esta aplicación tiene una alta prioridad debido a que de esta depende el desarrollo de varios proyectos y además la empresa brinda el servicio de asesoría técnica sobre proyectos ya realizados, lo cual requiere del conocimiento de los resultados de la aplicación a realizar. En vista de esto, la rentabilidad del proyecto es muy alta ya que los materiales utilizados son los que se utilizan actualmente en otros trabajos, lo que reduce los costos de inversión. Además, el desarrollo de este tipo de protocolo le va a brindar a Siemens la oportunidad de concursar en futuras licitaciones ampliando su cantidad de servicios brindados y el número de clientes.

1.3. Objetivos

1.3.1. Objetivo General

Implementar una aplicación que permita la comunicación entre un PLC maestro con dos esclavos utilizando comunicación inalámbrica y el envío de datos utilizando el protocolo Modbus ASCII

1.3.2. Objetivos específicos

- Realizar una documentación sobre las principales características del protocolo Modbus ASCII.
- Configurar los puerto serie del maestro y el esclavo para poder comunicarse con el radio.
- Recibir una capacitación sobre el funcionamiento de los radios.
- Configurar los radios para que puedan enviar y recibir datos inalámbricamente.
- Diseñar un procedimiento en el maestro que permita calcular la dirección de un PLC específico.
- Diseñar un procedimiento en el maestro que genere una solicitud para un esclavo específico.

- Diseñar un procedimiento que permita convertir un conjunto de datos según el protocolo Modbus ASCII.
- Diseñar un procedimiento que permita a los PLC´s transferir los datos a su respectivo radio vía puerto serie.
- Diseñar un procedimiento que permita a los PLC´s recibir los datos provenientes de su respectivo radio.
- Diseñar un procedimiento en el maestro que permita tomar la respuesta de un esclavo y tomar decisiones según los datos recibidos.
- Diseñar un procedimiento que compruebe la dirección enviada por el maestro y tome las decisiones correspondientes.
- Diseñar un procedimiento en el esclavo que interprete la solicitud hecha por el maestro y tome las medidas correspondientes.
- Diseñar un procedimiento en el esclavo que maneje las entradas y salidas del PLC según la petición del maestro.
- Diseñar un procedimiento en el esclavo que genere una respuesta para el maestro según la acción ejecutada.
- Realizar pruebas al sistema enviando diferentes solicitudes.

Capítulo 2. ANTECEDENTES

2.1. Estudio del problema a resolver

El proyecto a realizar tiene como fin el ser aplicado a los diferentes sistemas que utilice Siemens en el área de la telemetría. Por lo tanto, la aplicación a desarrollar debe ser capaz de manipular datos de diferentes clases para poder satisfacer cualquier futuro proyecto.

La captación de señales por parte de sensores va a quedar determinada por las características del proyecto a realizar, por lo tanto, esta etapa (desarrollo de sensores e implementación) no es tratada en este trabajo. El sistema a desarrollar debe cumplir las siguientes especificaciones:

- Un PLC esclavo debe recibir una señal proveniente de sensores: Se tienen diferentes entradas en el PLC, el sistema debe tomar los valores lógicos de estas entradas y almacenarlas en memoria.
- Deben manipular entradas analógicas de manera que se tome el valor y se almacene en memoria.
- Debe convertir estas señales en formato Modbus ASCII: Para la transmisión de datos se debe utilizar el protocolo Modbus ASCII (ver apéndice para detalles sobre las especificaciones del protocolo). La entrada capturada, ya sea analógica o discreta, se debe convertir al formato especificado por el protocolo de manera que se pueda transmitir y la recepción en el maestro sea válida.
- El esclavo debe escribir o leer una dirección de memoria según la petición hecha por el maestro: El maestro envía una solicitud al esclavo en un paquete Modbus ASCII, dentro de este paquete se especifica un código que le indica al esclavo si se va a escribir o leer una dirección de memoria y cual es dicha localidad.
- Establecer una comunicación por medio de radios entre el maestro y los diferentes esclavos: Para las aplicaciones más comunes (telemetrías) se utiliza comunicación inalámbrica, por lo tanto la comunicación entre el

maestro y los diferentes esclavos debe llevarse a cabo por medio de radio módem, estos se comunican con el PLC por medio de puerto serie.

- El PLC maestro debe realizar un Polling con los diferentes esclavos: Como el sistema consiste en el manejo de varios esclavos, el maestro debe ser capaz de interrogar a cada uno. Además tiene que saber cuándo un esclavo no esta respondiendo para enviar una cantidad específica de réplicas y en el caso de no obtener respuesta, interrogar al siguiente esclavo para no estancarse con uno solo.
- El PLC debe recibir información proveniente de un esclavo: Se debe recibir un paquete de información y ser capaz de interpretarlo (viene en formato Modbus ASCII), además debe tomar las decisiones correspondientes según la información procesada.

2.2. Requerimientos de la empresa

La empresa requiere de un sistema que se pueda aplicar a múltiples aplicaciones, típicamente telemetrías donde se controlan acueductos.

Debido a la variedad de dichas aplicaciones y para cubrir futuros trabajos, se debe diseñar un sistema capaz de manipular cualquier eventual comunicación entre el maestro y una cantidad configurable de esclavos.

Es decir, el maestro puede enviar una solicitud de escritura sobre una localidad de memoria, puede hacer una lectura de una localidad, puede leer un bloque de entradas o puede leer un bloque de salidas. El esclavo, sin importar la aplicación, debe tener previsto la acción para cada una de las solicitudes hechas por el maestro.

Además el sistema debe ser de fácil manipulación, o sea, el programa del maestro debe ser sencillamente modificable para agregarle o quitarle la cantidad de esclavos que se van a manejar así como las solicitudes que se hacen a cada uno y las acciones a tomar según la respuesta obtenida.

2.3. Solución propuesta

Analizando el sistema completo se puede generar un diagrama de bloques indicando las entradas y salidas, tomando tanto los maestro como los esclavos como un solo bloque. La figura 2.1 muestra este diagrama.

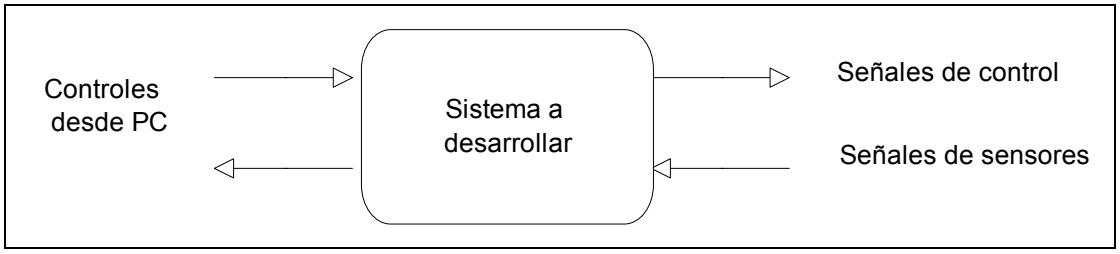

Figura 2.1 Diagrama de bloques del sistema a desarrollar

Para profundizar mas en la explicación del sistema se debe desarrollar un diagrama de bloques de segundo nivel; la figura 2.2 lo muestra, y se evidencia el hecho de que los maestros y los esclavos están físicamente separados. Las características de los esclavos son básicamente las mismas entre ellos, estos deberán ser capaces de:

a) Recibir una o varias señales de diferentes sistemas de sensado.
b) Enviar información a diferentes actuadores.
c) Recibir una solicitud de información proveniente del maestro en formato MODBUS ASCII.
d) Interpretar la solicitud del maestro y tomar las medidas necesarias.
e) Enviar una respuesta al maestro en formato MODBUS ASCII
f) Comunicarse con el radio modem para comunicarse con el maestro.

Para el caso del maestro se va a tener un PLC el cual debe ser capaz de:

a) Generar una petición de información para un esclavo en formato MODBUS ASCII.
b) Revisar todos los esclavos por medio de un sistema tipo polling.

c) Recibir una señal proveniente de un esclavo.

d) Identificar el esclavo que esta enviando una respuesta.

e) Interpretar la información recibida.

g) Comunicarse con el radio modem para interactuar con el esclavo.

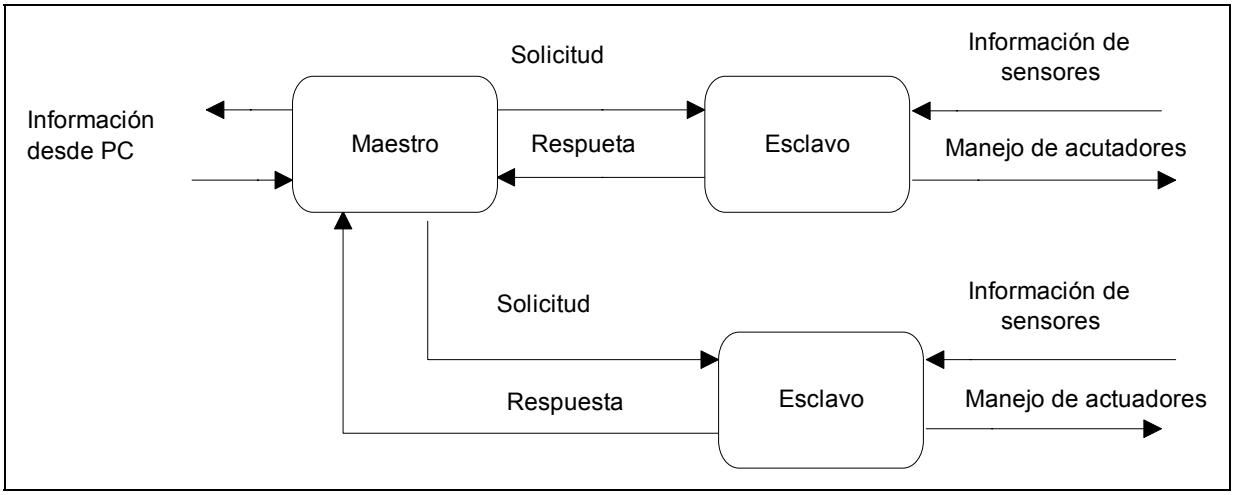

Figura 2.2 Diagrama de segundo nivel para el sistema a desarrollar

En el proyecto se van a tener que desarrollar tanto partes de software como de hardware ya sea para los esclavos como para los maestros. A continuación se va a dar la solución propuesta para satisfacer el sistema

Descripción de los elementos del Hardware

Maestro:

La figura 2.3 muestra el diagrama con la solución para que el maestro cumpla con las especificaciones ya mencionadas. Cada uno de los dispositivos se comentan a continuación:

PLC:

Se trabajará con un PLC Siemens de la familia SIMATIC S7-200 específicamente el CPU 216. Este cuenta con la característica de que tiene dos puertos de comunicación serie, uno para comunicarse con el transmisor y el otro para comunicarse con la computadora.

Radio transmisor:

Este dispositivo es un radio módem. Se utilizó uno marca Dataradio. Este tiene como entrada de datos un puerto serie que trabaja con el estándar RS232. El PLC maestro le envía los datos a este por medio del puerto serie.

Cable convertidor:

El PLC tiene un puerto serie que trabaja en estándar RS485 (protocolo PPI: point to point interface) el cual es un formato de comunicción por medio de corriente. El radio transmisor maneja un puerto serie en RS232 por lo tanto se necesita un cable convertidor para que los dos dispositivos se puedan comunicar.

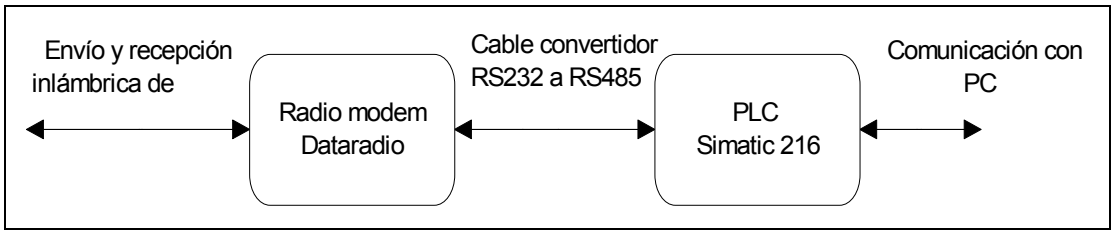

Figura 2.3 Partes del hardware que consisten el maestro

Esclavo:

En cuanto a lo que hardware se refiere, los dos esclavos que se utilizaron cuentan con equipo parecido al del maestro. El radio transmisor es el mismo del maestro, sin embargo, el PLC requiere de un único puerto serie por lo tanto se puede utilizar el PLC SIMATIC S7-222. El ingreso de la información desde sensores y las salidas a los actuadores se realizan por los módulos de entrada y salida convencionales del PLC, tanto digitales como analógicas.

La figura 2.4 muestra la configuración para el caso de un esclavo, se muestra solamente uno debido a que en cuanto a hardware ambos tienen las mismas características.

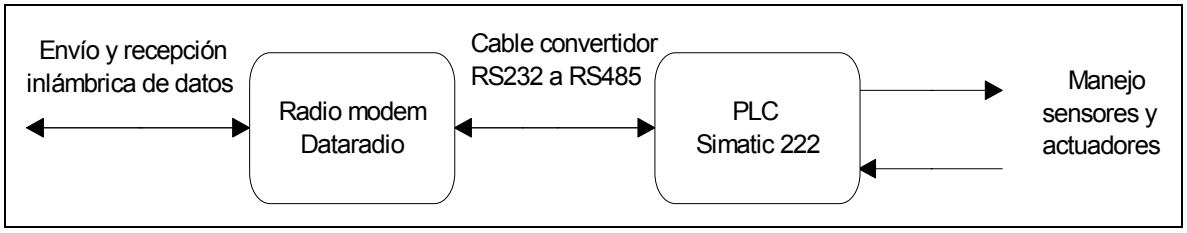

Figura 2.4 Partes del hardware que consisten el esclavo

Descripción de los elementos del Software:

Maestro:

La figura 2.5 muestra las características del maestro en lo que a software se refiere. Para la parte del esclavo, se explica únicamente uno ya que ambos deben ser capaces de interpretar cualquier tipo de información por lo tanto ambos se comportan de la misma forma. Cada una de las partes se explica a continuación:

Generación de dirección:

Como el sistema que se va a desarrollar debe controlar una cantidad determinada de esclavos (dos en este caso con posibilidad de expandirlos), se va a utilizar un sistema de tipo “polling” el cual atiende a un esclavo y cuando termina atiene a otro. El generador de dirección determina el número correspondiente al esclavo que se va a atender, esto implica que este procedimiento debe ser capaz de saber si ya se revisó la memoria del esclavo y se le enviaron las respuestas correspondientes según la respuesta del mismo.

Una vez finalizada la atención de ese esclavo se debe cambiar la dirección para atender el siguiente. También se tiene que generar un replica en caso de que no haya recepción, una vez pasadas las replicas (dos en esta aplicación), se debe cambiar de esclavo para que el sistema no se quede atendiendo uno sólo que puede ser que este dañado y nunca vaya a responder.

Generador de solicitud:

Este procedimiento debe calcular el tipo de solicitud que se está enviando al esclavo. El maestro es el que inicia una comunicación, por lo tanto este debe enviar las ordenes al esclavo y el segundo envía respuestas.

Se pueden dar dos clases de comunicación, una solicitud de lectura, en la cual el procedimiento determina la localidad de memoria en el esclavo que va a ser leída (entrada, salida o alguna localidad de memoria). Otra opción es que se quiera escribir sobre una localidad de memoria, en este caso, el paquete de información especifica la dirección a escribir y el valor que va a adoptar.

Convertidor a Modbus ASCII

El fin del proyecto es el de desarrollar una aplicación que transmita en formato Modbus ASCII, este es un protocolo que se utiliza para comunicar instrumentos configurados en modo maestro esclavo, existen dos tipos de comunicación Modbus, RTU y ASCII; para el proyecto se utilizó el ASCII en el cual, los ocho bits de información se envían como dos caracteres ASCII. Existe toda una documentación de la forma en que se debe mandar un paquete de información para que cumpla con el protocolo (ver apéndice para detalles del protocolo). Este procedimiento se encarga de tomar las solicitudes del maestro y darle el formato necesario para que el esclavo sea capaz de interpretar una solicitud de lectura o una de escritura y además, poder decodificar los bits de información.

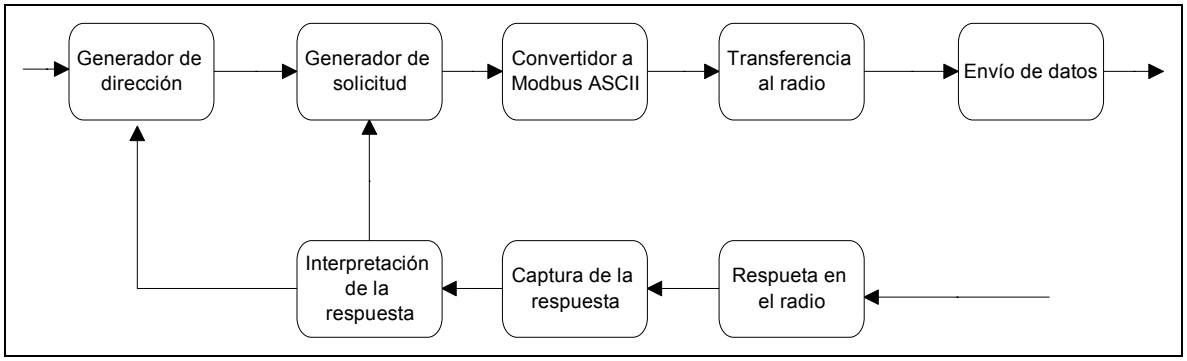

Figura 2.5 Partes de software que componen el maestro del sistema

Transferencia al radio:

Una vez que la información se encuentra codificada en formato Modbus ASCII, se debe utilizar el puerto serie del PLC para transferir la información al

radio modem. En esta etapa lo que se va a controlar es toda la configuración del maestro para que se pueda entender con el radio. Es importante hacer notar que el PLC trabaja en formato RS485 y el radio en RS232, sin embargo, se cuenta con un cable convertidor que realiza la respectiva comunicación.

Envío de datos:

Esta es la etapa que se encarga de manejar el radio y enviar los datos que le llegan. Para esta parte se va a recibir una capacitación por parte de la compañía distribuidora de los mismos. Dicha capacitación se va a encargar de suministrar los conocimientos para la configuración y manejo de información en el radio.

Respuesta en el radio:

La otra parte del funcionamiento del maestro es la recepción de datos provenientes de un esclavo específico. El maestro tiene que saber cuál PLC se está consultando para que al momento de recibir una respuesta, se sepa de donde proviene.

Este procedimiento consiste en la configuración del radio para que sea capaz de capturar información.

Captura de la respuesta:

Este procedimiento se encarga de la comunicación entre el radio y el PLC. Se debe configurar el puerto serie de ambos para que cuando se reciba información el maestro pueda tomar la información del puerto y pasarla al procedimiento que se encarga de la interpretación de esta.

Interpretación de la respuesta:

La información que llega viene en protocolo Modbus ASCII, por lo tanto se debe decodificar la información para saber qué tipo de información llegó. Además este procedimiento determina qué hacer según la información recibida. Se puede dar el caso de que llegue como respuesta información sobre una localidad de

memoria consultada previamente por el maestro o se puede recibir la confirmación de que una orden fue ejecutada por el esclavo.

Esclavo:

La figura 2.6 muestra las características de lo que corresponde al esclavo. Ambos esclavos se comportan de manera similar.

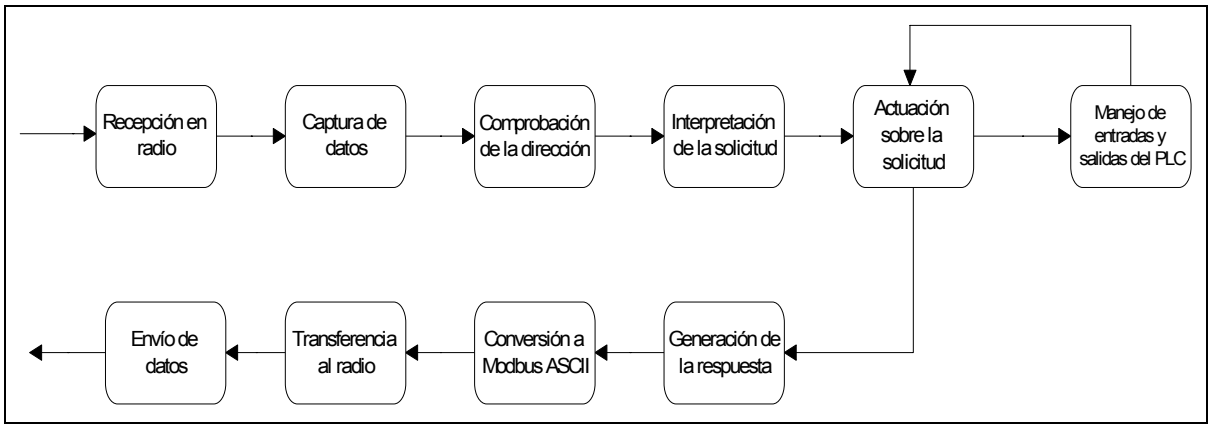

Figura 2.6 Partes de software que componen el esclavo del sistema

Recepción en radio:

Esta etapa se encarga de la configuración del radio que maneja el esclavo. Al igual que el maestro, este tiene que recibir y transmitir por lo tanto la configuración de este es el mismo que el caso del maestro.

Captura de datos:

Este procedimiento manipula el puerto serie del PLC esclavo para capturar la información proveniente del puerto serie del radio y enviar la información al procedimiento que verifica la dirección.

Comprobación de la dirección:

El maestro determina la dirección del esclavo a atender y lo envía en el paquete de información Modbus por medio del radio. Como la transferencia es inalámbrica, al momento del envío, todos los radios de los esclavos van a capturar el paquete de información. Por lo tanto se debe tener una etapa que revise la

dirección enviada por el maestro para que el esclavo específico sepa que se le está solicitando algo y los demás no actúen sobre dicha solicitud.

Interpretación de la solicitud:

Una vez verificada la dirección se debe saber qué esta solicitando el maestro. Este procedimiento se encarga de tomar el paquete de información en formato Modbus y verificar si lo que se recibió fue una solicitud de lectura o escritura. Si es una orden de lectura, se recibe la localidad de memoria por revisar y la información en esta se envía al procedimiento que se encarga de la actuación de la respuesta. Si por el contrario, se recibe una orden de escritura, se actualiza la localidad de memoria específica.

Actuación sobre la respuesta:

Una vez que se ha determinado si lo que se desea hacer es consultar una dirección o escribirla, entonces este procedimiento se encarga de manipular las entradas y salidas del PLC, ya sea para verificar una dirección o para controlar una salida. Además es el que recibe la respuesta del manejador de entradas y salidas del PLC, para indicarle al generador de respuesta que ya se ha actuado sobre la petición.

Manejo de las entradas y salidas del PLC

Este procedimiento es el que se encarga de la configuración de las entradas y salidas del PLC. Cuando se manipula una entrada se le envía el valor de la localidad de memoria al procedimiento anterior para que este se lo envíe al generador de respuesta. Este procedimiento varía dependiendo de la aplicación, pero típicamente, se activa una salida según un bit de control, el maestro le envía una orden al esclavo para que escriba en dicho bit de control para que el esclavo lo interprete y actúe sobre la salida.

Generación de la respuesta:

Una vez que se ha actuado sobre la solicitud del maestro se debe generar una respuesta indicando ya sea los datos contenidos en una localidad de memoria específica o una comprobación de orden ejecutada. Este procedimiento toma esta respuesta y se la envía al procediendo que convierte estos datos al protocolo Modbus ASCII.

Conversión a Modbus ASCII:

Este procedimiento toma los datos de la respuesta y los convierte en formato Modbus ASCII para luego enviarlo al procedimiento que transfiere los datos al radio.

Transferencia al radio:

Al igual que el maestro, este procedimiento se encarga de manipular el puerto serie y enviar los datos al radio para su envío al maestro.

Envío de datos:

Este procedimiento se encarga del manejo del radio para la transferencia de datos hacia el maestro.

Capítulo 3. PROCEDIMIENTO METODOLOGICO

1.Documentar las principales características del protocolo Modbus ASCII. (3 semanas)

   a) Características generales del protocolo
   b) Forma del paquete de información
   c) Funciones de cada parte del paquete de información
   d) Forma de transmitir el protocolo
   e) Ventajas en la utilización del protocolo
   f) Diferencias entre el Modbus ASCII y el Modbus RTU

2. Preparación del puerto serie del PLC maestro para comunicarse con el radio. (1 semana)
3. Preparación de los puertos serie de los esclavos para que se puedan comunicar con el radio (1 semana)
4. Capacitación sobre el funcionamiento de los radios (1 semana)
5. Manipulación de los radios para que se puedan comunicar con sus respectivos PLC´s (2 semanas)
   a) Configuración de los radios
   b) Realización de rutinas de prueba en el PLC para comprobar la comunicación.
6. Manipulación de los radios para que se pueda comunicar entre ellos inalámbricamente. (1 semana)
7. Programación de la rutina que determina la dirección del esclavo (1 semana)
8. Programación de la rutina que genera la solicitud que se va a enviar al esclavo. (2 semanas)
   a) Generación de una solicitud de lectura en una localidad de memoria.
   b) Generación de una solicitud de ejecución sobre una localidad de memoria
   c) Pruebas

9. Programación de la rutina que convierte una solicitud en formato Modbus ASCII. (1 semana)
    a) Programación de la rutina.
    b) Pruebas.
10. Programación de la rutina que envía el paquete de información al respectivo radio (1 semana)
    a) programación de la rutina
    b) Pruebas
11. Programación de la rutina que recibe el paquete de información del respectivo radio. (1 semana)
    a) Programación de la rutina
    b) Pruebas
12. Programar la rutina que toma el paquete recibido del esclavo y lo interpreta. (2 semanas)
    a) Captura del paquete
    b) Decodificación de Modbus ASCII
    c) Generación de la orden de escritura en localidad de memoria si es el caso.
    d) Comprobación de la dirección en memoria a la que se desea acceder si es el caso.
    e) Pruebas.
13. Programación en el esclavo de la rutina que comprueba la dirección enviada por el maestro. (1 semana)
    a) Programación de la rutina
    b) Pruebas
13. Programación de la rutina que actúa sobre la solicitud enviada al esclavo. (1 semanas)
    a) Lectura de una localidad de memoria.
    b) Escritura sobre una localidad de memoria.
    c) Pruebas

14. Programación de la rutina que maneja las entradas y salidas del PLC esclavo. (1 semana)
    a) Programación de la rutina.
    b) Pruebas.
15. Programación de la rutina que genera una respuesta según la acción ejecutada sobre el esclavo. (1 semana)
    a) Generación de la respuesta al maestro.
    b) Conversión de la información a Modbus ASCII
    c) Pruebas
16. Realización de pruebas finales del sistema completo. (1 semana)
17. Diseño de aplicaciones utilizando el protoclo.
    a) Telemetría de un maestro y un esclavo, Acueducto de Hojancha, Nicoya.
    b) Cliente: Acueductos y Alcantarillados (A y A)
    c) Telemetría de un maestro y un esclavo, Acueducto de Maiquietia, Desamparados. Cliente: A y A.
    d) Telemetría de un maestro y un esclavo, Acueducto de Atenas, cliente: Pabel
    e) Telemetría de un maestro y un esclavo, Acueducto de Guadalupe, cliente: A y A.
    f) Telemetría de un maestro y cuatro esclavos, Acueducto de Cruz, Guanacaste. Cliente: A y A.

Capítulo 4. DESCRIPCIÓN DEL HARDWARE UTILIZADO

No hizo falta diseñar el hardware pues la empresa cuenta con todo lo necesario. A continuación se enumeran los dispositivos utilizados y sus principales características:

- PLC SIMATIC S7200 CPU 222:
  Este PLC se utilizó como esclavo.
  - Cuenta con un puerto serie RS-485
  - Tiene 8 entradas y cinco salidas.
  - Alimentación del PLC, AC de 85 V a 264 V.
  - Alimentación relays DC de 20.4 a 28.8 V

- PLC SIMATIC S7-200 CPU 216
  Este PLC se utilizó como maestro.
  - Cuenta con un puerto serie RS-485
  - Tiene 24 entradas y 16 salidas.
  - Alimentación del PLC, AC de 85 V a 264 V.
  - Alimentación relays DC de 20.4 a 28.8 V.

- Cable convertidor: El puerto serie de los PLC´s trabajan en RS-485 y los radios trabajan en RS-232. Para realizar una adecuada comunicación, se utilizó un cable convertidor de RS-485 a RS-232 denominado cable PC/PPI, este cable se encarga de la conversión. Es un cable especial al cual se le puede ajustar la tasa de transferencia de datos entre un puerto y otro. También se le puede ajustar diferentes modos de comunicación, todo esto configurable a través de unos dip switch que se manejan según la aplicación.

- Radio Modem: Se necesita un radio para cada PLC, para conseguir la comunicación inalámbrica deseada. La comunicación del radio con el PLC se realiza por puerto serie. La velocidad de transmisión se configura para que sea compatible con el PLC, esta velocidad, para las aplicaciones que se desarrollan en la empresa, se utiliza en 9600 bps. Es importante destacar que el cable convertidor no se puede conectar directo al radio modem pues no habrá comunicación. El radio tiene una salida hembra al igual que el cable, se necesita un convertidor macho-macho para conectar el cable; el convertidor debe tener un crossover para que los dispositivos se comuniquen.

Capítulo 5. DESCRIPCIÓN DEL SOFTWARE UTILIZADO

El programa es la parte fundamental del proyecto pues con él se programa el PLC para que cumpla con todas las especificaciones ya mencionadas.

El software involucrado en el desarrollo de la aplicación es el software STEP 7 MicroWin 32 V3.1.0.31. Cuenta con las siguientes características:

- Permite realizar la programación por medio de subrutinas, lo cual facilita la comprensión durante el desarrollo de una aplicación. Se puede hacer uso de interrupciones las cuales se asocian a un evento determinado.
- Se pueden ver la variables en tiempo real, es decir, se selecciona un grupo de variables y el programa despliega el estado en que se encuentra dicha variable en ese momento.
- Se pueden realizar diversas operaciones matemáticas: suma, resta, multiplicación, división. Todo esto se puede realizar tanto para bytes como para enteros (palabras ó palabras dobles), ya sea en coma fija o en coma flotante.
- Permite la comparación de valores del tipo mayor que, menor que, igual que, tanto en bytes, palabras o palabras, dobles
- Tiene diferente tipos de contadores que se pueden utilizar para llevar un conteo de eventos.
- Permite la creación de ciclo por medio de operandos *FOR* que da la opción de repetir una orden una cantidad determinada de veces. También permite saltos dentro del programa y salidas condicionales de una rutina específica.
- Se pueden hacer conversiones de valores ASCII a hexadecimal y viceversa, rotación de bytes, palabras o palabras dobles, tanto a la izquierda como a la derecha.

- Tiene una opción para ver referencias cruzadas, la cual permite ver donde está localizada cada una de las variables dentro de las diferentes rutinas de un programa.
- Se puede trabajar con direccionamiento simbólico, lo cual permite ponerle un nombre a una variable dada y utilizar el nombre en lugar de la dirección absoluta de dicha variable, facilitando la comprensión a la hora de leer el programa.

Capítulo 6. ANÁLISIS Y RESULTADOS

6.1. Explicación del diseño

El diseño implementado consiste básicamente en el desarrollo de una aplicación con PLC, por lo tanto el fuerte del diseño son los diferentes módulos que componen el programa tanto para el maestro como para el esclavo. A continuación se detalla cada uno de dichos módulos y su respectivo funcionamiento.

6.1.1. Diseño del maestro

El programa del maestro se compone de una rutina que se encarga de hacer llamados a diversas subrutinas. Tiene cuatro subrutinas que son llamadas cuando se dan ciertas condiciones detalladas mas adelante. Finalmente cuenta con una interrupción que se activa con una condición que también será comentada en su momento.

6.1.1.1. Rutina principal

Este es el programa principal que el PLC recorre permanentemente mientras no esta atendiendo una subrutina o una interrupción.

También da las condiciones necesarias para que se haga un salto a una subrutina. Por las características de una interrupción, esta no se llama desde el principal sino que cuando se da la condición, la ejecución del programa se suspende para atender dicha interrupción.

La ejecución de esta sección es la siguiente:

El PLC cuenta con una marca especial que se activa únicamente cuando se esta recorriendo el primer ciclo, es decir cuando el PLC pasa del modo de stop a run, esto sucede cuando se suspende el flujo de corriente, por selección manual o

cuando el PLC encuentra un error de ejecución. Al activarse este bit, se hace el llamado a la subrutina que se encarga de la inicialización de ciertas variables.

Otra marca especial es una que se activa cada medio segundo. Se tiene una variable llamada *esclavo* que indica cuál esclavo se está atendiendo. La marca especial junto con la variable dan las condiciones para que se determine cuándo y cuál esclavo se va a atender. Es decir, cada medio segundo se envía una solicitud a un esclavo el cual lo determina la variable *esclavo*.

Dentro de la iniciación se pueden especificar las características de funcionamiento del puerto serie tanto para la recepción como para la transmisión de datos. Dentro de las especificaciones se le puede indicar el carácter que define el final en la recepción de un mensaje. Al llegar este carácter se activa una marca especial indicando el final de la recepción.

Al activarse este bit de final de recepción, se hace el llamado a la subrutina que activa el bit de final de mensaje y realiza otras funciones. Este bit de final de mensaje recibido es el que hace el llamado a la última subrutina, la cual manipula el mensaje recibido en formato Modbus ASCII y lo convierte a valores decimales para su interpretación.

Se puede dar el caso de que el maestro envíe una solicitud y no reciba respuesta, para esto el programa debe enviarla de nuevo. Sin embargo, no puede quedarse repitiendo una solicitud a un mismo esclavo porque cabe la posibilidad de que este se encuentre dañado y no vaya a responder nunca, dejando a un lado la atención de los demás esclavos que pueden estar trabajando normalmente.

Para esto se tiene un byte de control llamado *intentos*. Cada vez que se da una transmisión y no se recibe respuesta se incrementa, al darse dos intentos se verifica el estado de *esclavo* y se cambia, de manera que se active el siguiente para que no se de la situación ya explicada.

La figura 6.1 muestra el diagrama de flujo para el funcionamiento de esta rutina principal.

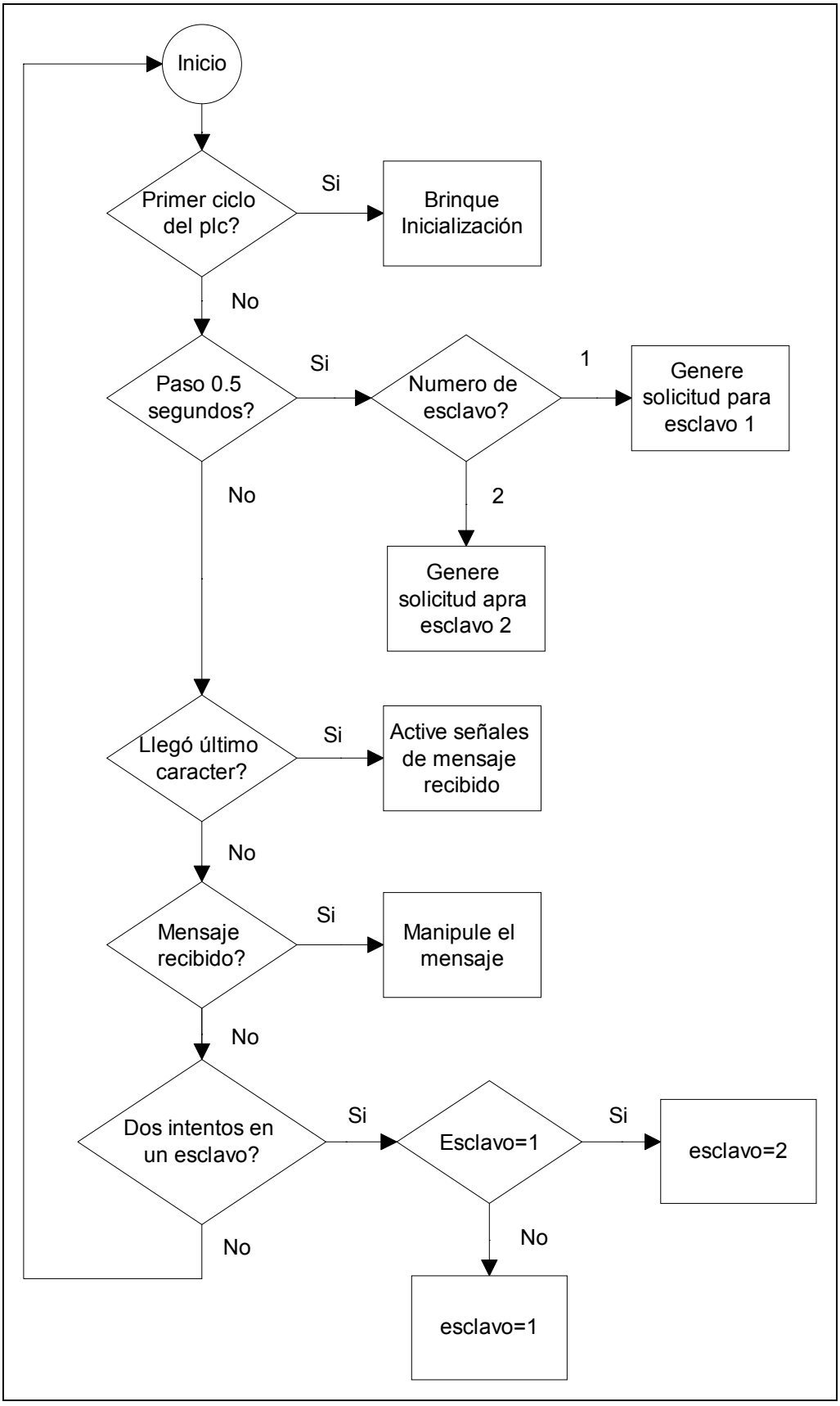

Figura 6.1 Diagrama de flujo para la rutina principal

6.1.1.2. Subrutina 0 (Inicialización)

La figura 6.2 muestra el diagrama de flujo para la rutina que se encarga de la inicialización del programa.

Esta subrutina, como ya se mencionó, se ejecuta únicamente la primera vez que se corre el PLC, es decir cuando se pasa de stop a run. En ella se da la velocidad de transmisión del puerto serie.

Las marcas internas del PLC no se borran cuando se pone en stop, esto puede ser un inconveniente pues si el PLC se apaga, una marca puede quedar activa y al iniciar se espera que este inactiva para genera las condiciones necesaria para un adecuado funcionamiento. En este procedimiento se borran las marcas utilizadas de manera que no se dé este problema.

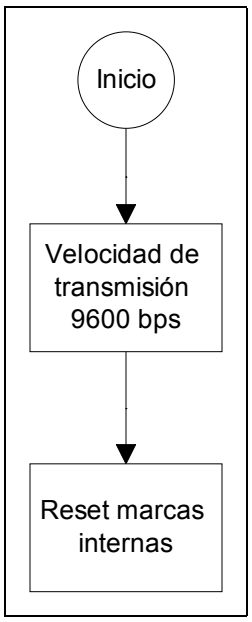

Figura 6.2 Diagrama de flujo para la rutina 0 (inicialización)

6.1.1.3. Subrutina 1 (Envío de solicitud esclavo 1)

Esta subrutina se encarga de la transmisión de una solicitud para el esclavo número uno. Lo primero que hace es una “rutina tonta”. Esta es una parte muy importante para el adecuado funcionamiento de todo el programa. La figura 6.3 muestra el diagrama de flujo para este procedimiento.

Como ya se mencionó, se puede dar el caso de que el esclavo no responda por lo tanto se tiene que volver a enviar la solicitud. Bajo condiciones normales el funcionamiento de la comunicación es el siguiente:

a) El maestro transmite una solicitud (función XMT).
b) Al enviar el último carácter, se activa una interrupción.
c) La interrupción activa la recepción. (función RCV)
d) El esclavo recibe la solicitud y envía respuesta.
e) El maestro recibe la respuesta, al recibir el último carácter finaliza la recepción.
f) Al finalizar la recepción, atiende la respuesta y envía otra solicitud.

El problema que se da es que cuando no se recibe respuesta no hay carácter final que termine la recepción, entonces cualquier intento de mandar otra solicitud se ve truncada pues el puerto serie está esperando respuesta. Esta “rutina tonta” se encarga de activar la recepción pero con parámetros de inicialización incorrectos. En este caso el programa finaliza la recepción por errores, entonces se deja libre el puerto serie para realizar la transmisión.

Estos pasos se ejecutan únicamente en el procedimiento que se encarga de enviar una solicitud, es decir, cuando se espera que el puerto este libre, por lo tanto, si esta libre no pasa nada y si esta ocupado se va a finalizar por medio este procedimiento.

Una vez que se tiene seguridad de que el puerto esta libre, se incrementa la variable *intentos* que indica la cantidad de intentos que se han hecho sin recibir respuesta.

Luego se define el cuerpo del mensaje a transmitir. En esta parte se le da a cada localidad de memoria (a partir de la localidad VB39) el valor ASCII de cada número según el formato mostrado en la tabla 6.1:

Tabla 6.1 Valores ASCII y localidad en que se almacena una solicitud para el Esclavo 1

Localidd	Valor ASCII	Comentario
VB39	-------	Cantidad de caracteres a transmitir
VB40	58	Inicio de mensaje (según protolo Modbus ASCII)
VB41	55	Dirección del esclavo a la que va dirigida la solicitud (7C hexadecimal)
VB42	67	
VB43	48	Código de función (03 hexadecimal: leer registro en memoria)
VB44	51	
VB45	48	Dirección del registro a leer parte alta (01 hexadecimal)
VB46	49	
VB47	57	Dirección del registro a leer parte baja (90 hexadecimal)
VB48	48	
VB49	48	Cantidad de registro a leer parte alta (00 hexadecimal)
VB50	48	
VB51	48	Cantidad de registro a leer parte baja (01 hexadecimal)
VB52	49	

Después se cuentan la cantidad de caracteres a enviar en el mensaje. Este dato se debe especificar a la hora de utilizar la función XMT. Se debe poner la dirección inicial de envío de datos, donde la primera localidad es la cantidad de caracteres a enviar y después siguen las localidades a transmitir (ver tabla 6.1).

Para realizar dicho conteo se coloca un puntero sobre el primer carácter del mensaje y se verifica con el valor 10h (valor ASCII para indica el final de un mensaje según el protocolo Modbus ASCII, ver apéndice para mas detalles), si es igual activa el final del conteo. Si no es ese valor, incrementa un contador que es la primera localidad del mensaje correspondiente a la cantidad de caracteres a enviar.

Al recibir el carácter final se activa un indicador que habilita la transmisión del paquete de información y se lleva a cabo la transmisión de la información a partir de la localidad VB40, junto a esto, se le asocia la interrupción 0 al evento 9. Este evento sucede cuando se termina de transmitir el último carácter.

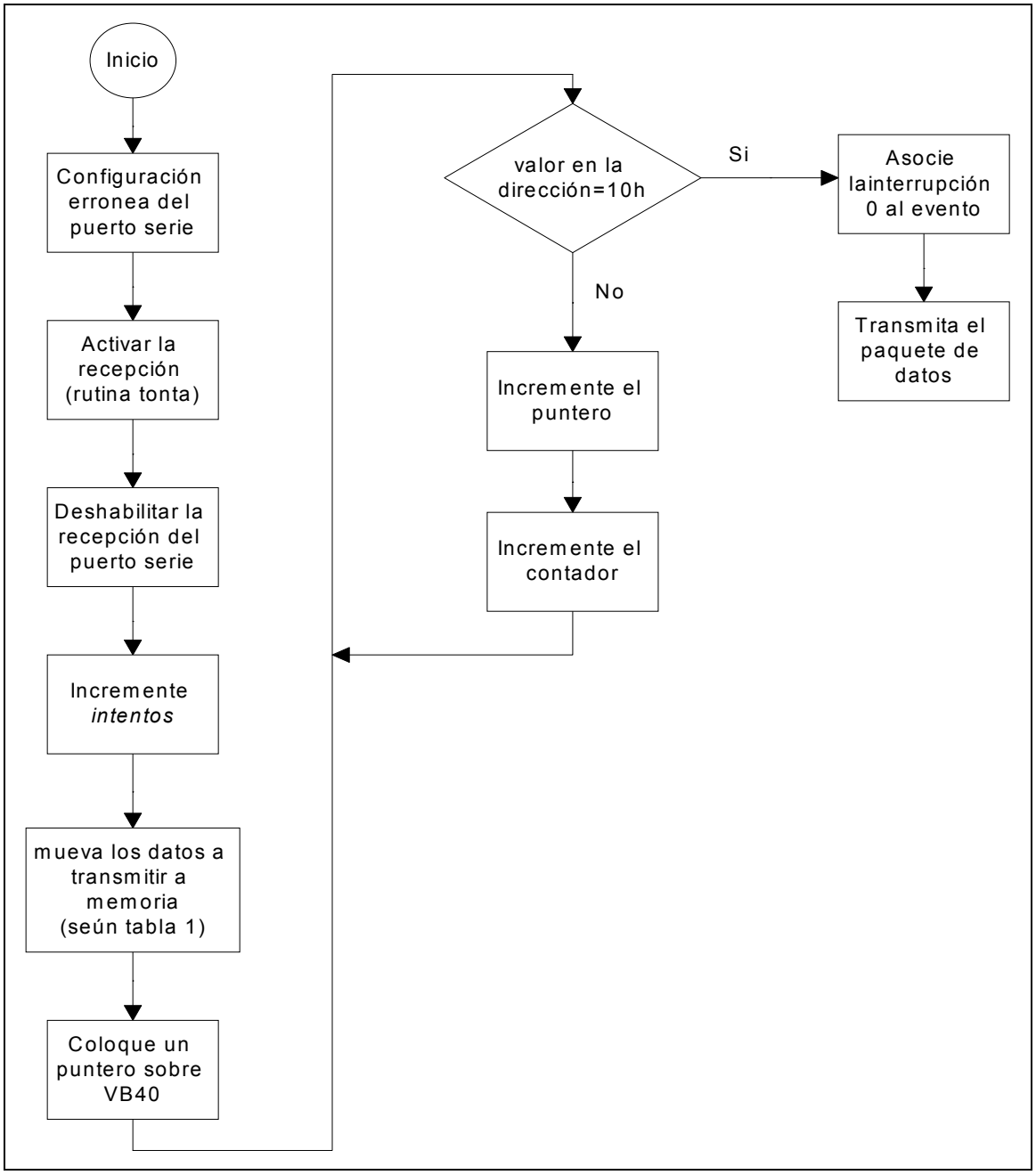

Figura 6.3 Diagrama de flujo para la rutina 1 (Solicitud para el esclavo 1)

6.1.1.4. Interrupción 0 (Transmisión finalizada)

Una vez que la subrutina de envío de solicitud termina de transmitir el mensaje al respectivo esclavo se genera una interrupción que se encarga de activar la recepción para la respuesta del esclavo.

Primero desasocia la interrupción al evento para evitar que se genere cuando no se desea.

Luego se configura el puerto serie para la recepción con las siguientes características:

- Carácter que inicia la recepción: 58h
- Carácter que finaliza la recepción: 10h
- Máxima cantidad de caracteres a recibir: 40
- Activa la recepción.
- Ignorar condiciones de paro

Finalmente, activa la función de recepción del puerto serie. La figura 6.4 muestra el diagrama de flujo que explica la lógica de esta subrutina. El paquete que se reciba se va a almacenar a partir de la dirección VB2, donde VB2 va a guardar la cantidad de caracteres que se recibieron.

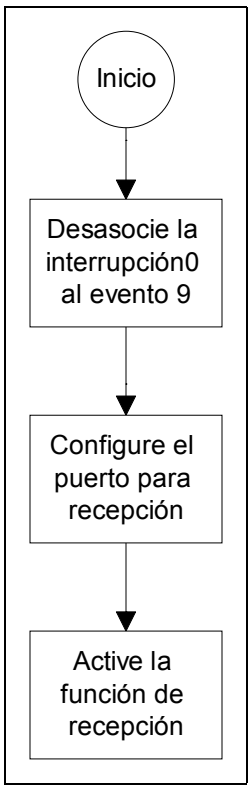

Figura 6.4 Diagrama de flujo para la interrupción 0 (Activar recepción)

6.1.1.5. Subrutina 3 (Carácter final recibido)

Como ya se mencionó entre los parámetros de programación del puerto serie para la recepción, se configura un carácter que va a indicar el final de la transmisión. En este caso es un 10h que junto con un 13h conforman el par de valores que se reciben para finalizar un mensaje según el protocolo Modbus ASCII. Cuando este carácter llega, se activa un bit que indica este hecho. Al activarse se ingresa a esta rutina, la figura 6.5 muestra la lógica de funcionamiento.

Lo primero que hace es poner la variable *intentos* en un valor de cero. Esto porque esta variable controla la cantidad de transmisiones sin respuesta realizadas por el maestro. Como ya se recibió respuesta, ya no se tienen que hacer más intentos con dicho esclavo.

Luego activa un indicador de que se recibió un mensaje esta variable se llama *llego_colon*.

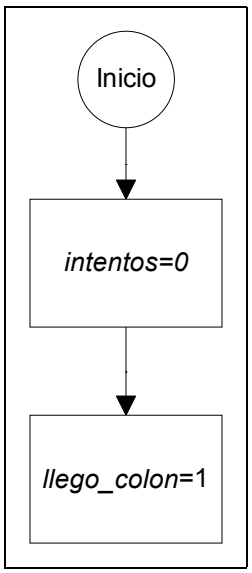

Figura 6.5 Diagrama de flujo para la subrutina 3 (Mensaje recibido)

6.1.1.6. Subrutina 2 (Manipulación del mensaje recibido)

Este es uno de los procedimientos más complejos del programa, por lo tanto, para su explicación se utilizará además del diagrama de flujo figuras mostrando el movimiento de valores antes y después de una instrucción.

Se tiene una variable llamada *contador* que se utiliza para asignar valores y se desea hacer sólo una vez.

En esta parte se verifica el esclavo que se esta atendiendo y se cambia por el siguiente para que la próxima solicitud sea enviada a otro esclavo.

Luego, también con *contador=0,* se colocan AC1 apuntando a VB3 al igual que AC3 y AC2 se hace que apunte a VB4. La figura 6.6 muestra un dibujo del bloque de memoria y como quedan inicializados los punteros.

Esta es la parte de inicialización, inmediatamente después de esta se incrementa la variable *contador* y se realiza un procedimiento que se va a repetir hasta que la condición *Pr_chr_fin* sea verdadera. Esta condición esta controlada por el valor que tenga la variable que esta siendo apuntada por AC1. Si se tiene un 13h (primer carácter final según Modbus ASCII), se activa la variable y se suspende la manipulación del bloque de memoria porque ya se llegó al final.

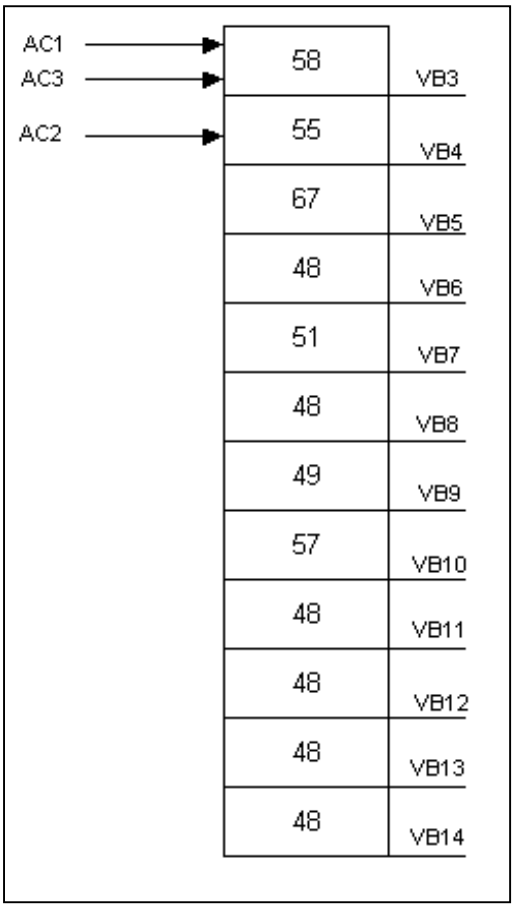

Figura 6.6 Muestra del bloque de memoria que almacena los datos recibidos

Mientras el valor que apunta AC1 no sea 13h se van a realizar los siguientes pasos:

Se convierte el valor apuntado por AC1 de ASCII a decimal

Se convierte el calor apuntado por AC2 de ASCII a decimal

La figura 6.7 (a) muestra la forma en que se observa el bloque de memoria después de estos pasos, los valores decimales se muestran en binario.

Como los datos que se reciben tienen decenas y unidades, cada dos localidades de memoria componen un valor decimal (00 a FF ó 00 a 255).

AC1 siempre va a apuntar a la decena, por lo tanto, el siguiente paso es desplazar el valor apuntado por AC1 cuatro espacios a la izquierda. La figura 6.7 (b) muestra el resultado de esa acción.

Seguidamente se suma el valor apuntado por AC1 con el de AC2 para formar el valor decimal completo y luego borra el contenido de AC2. La figura 6.7 (c) ilustra esta parte.

Como se tomaron dos localidades de memoria y se dejó una sola, queda un espacio sin utilizar. Para optimizar el uso de la memoria, se toma el valor calculado (apuntado por AC1) y se almacena donde apunta AC3. En este momento no se aprecia esta parte pues AC1 y AC3 apuntan al mismo lugar pero la figura 6.8 es una muestra del siguiente manejo del bloque de memoria donde se repiten los pasos anteriores.

Finalmente, se incrementa AC1 en dos posiciones al igual que AC2 y AC3 se incrementa en una posición. La figura 6.7 (d) muestra la forma en que queda la memoria para el próximo ciclo.

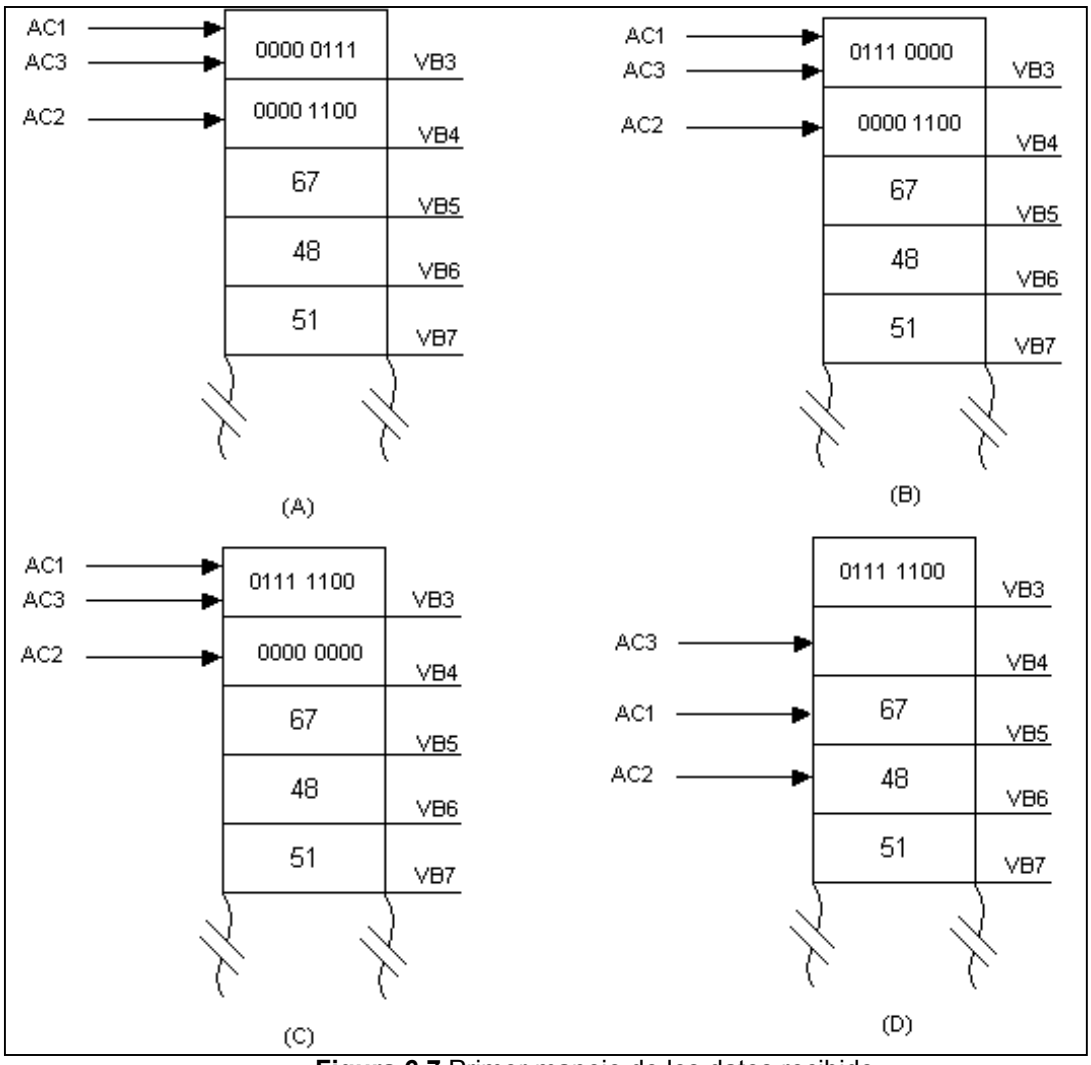

Figura 6.7 Primer manejo de los datos recibido

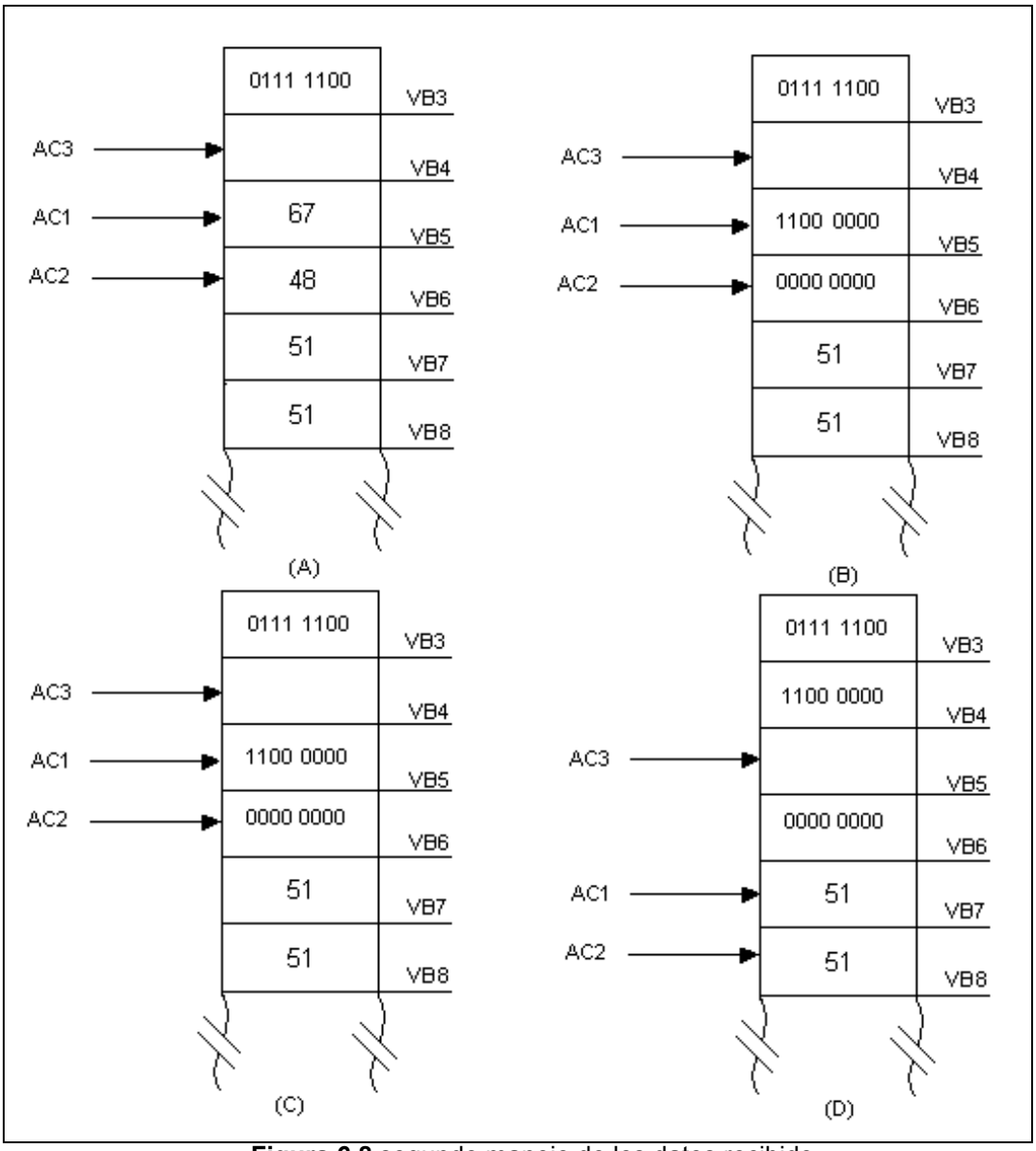

Figura 6.8 segundo manejo de los datos recibido

Una vez que AC1 apunta a 13h se desactiva la variable *ptr_chr_fn* y si esta condición es valida, inmediatamente se evalúa el valor apuntado por AC2 si es 10h, se ha recibido el final del mensaje y se desactiva *llego_colon.* Con esto se deshabilita la condición que permite el ingreso a la subrutina.

Como la mayoría de las aplicaciones consisten en realizar una acción según un valor recibido, se implementó un sistema que pone en un byte de salida del maestro un byte de entradas proveniente de un esclavo. El programa verifica el esclavo activo, si es el esclavo uno (dirección 124 decimal) transfiere el dato

recibido al byte QB0; si es el esclavo 2 (dirección 200) coloca el byte recibido en el byte QB1. La figura 6.9 muestra el diagrama de flujo para esta subrutina.

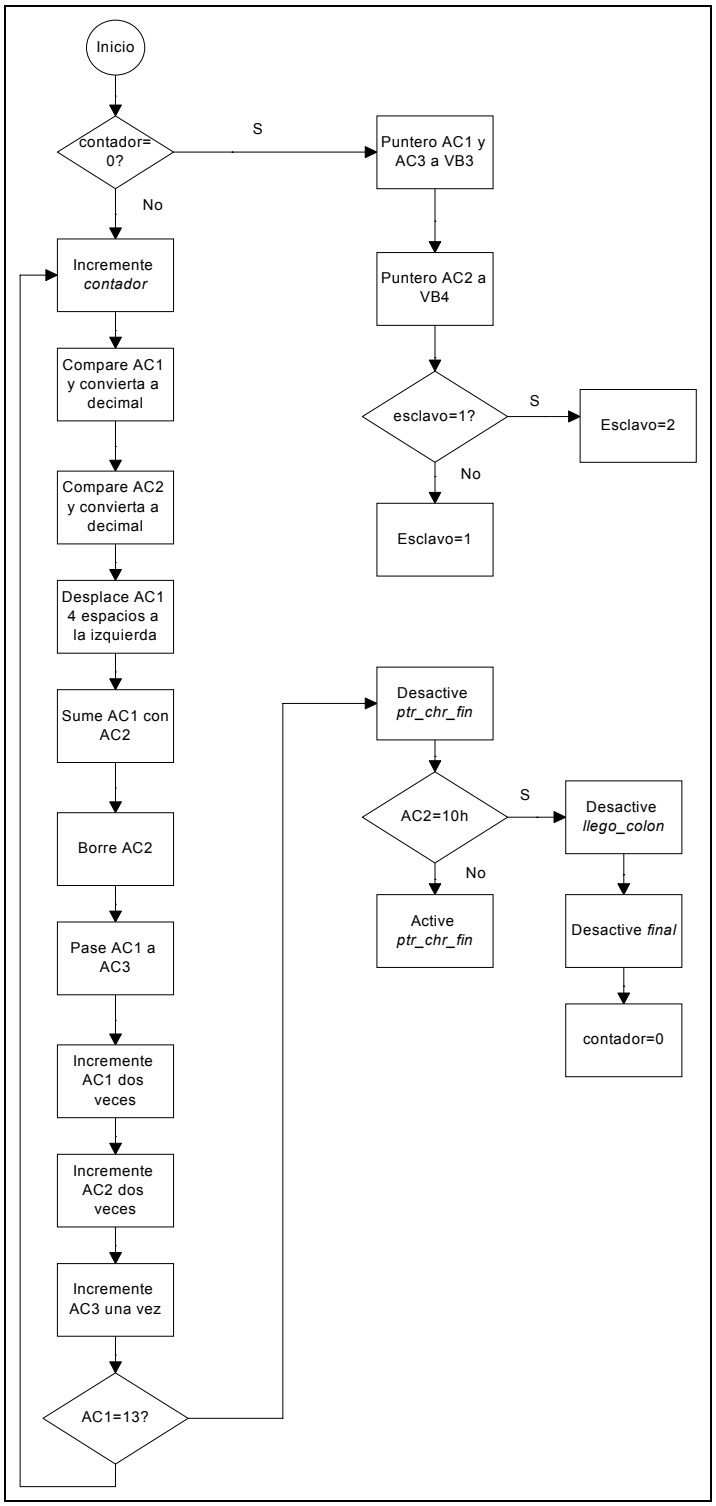

Figura 6.9 Diagrama de flujo para la subrutina 2 (manejo del mensaje recibido)

6.1.1.7. Subrutina 7 (Envío de solicitud esclavo 2):

Esta subrutina es igual a la subrutina 1 lo único que cambia es la dirección del esclavo. Esto porque para la aplicación se está haciendo la misma consulta a ambos esclavo. Cuando se desee manejar de diferente forma un esclavo de otro (para alguna aplicación específica), esta es la parte del programa que se debe cambiar, en cuanto a la solicitud hecha para un esclavo dado.

Por lo tanto la figura 6.3 también muestra el procedimiento para esta subrutina solamente que la tabla 6.1 quedaría modificada, en cuanto a la dirección del esclavo, como se muestra en la tabla 6.2.

Tabla 6.2 Valores ASCII y localidad en que almacena una solicitud para el Esclavo 2

Localidd	Valor ASCII	Comentario
VB39	-------	Cantidad de caracteres a transmitir
VB40	58	Inicio de mensaje (según protolo Modbus ASCII)
VB41	57	Dirección del esclavo a la que va dirigida la solicitud (C8 hexadecimal)
VB42	67	
VB43	48	Código de función (03 hexadecimal: leer registro en memoria)
VB44	51	
VB45	48	Dirección del registro a leer parte alta (01 hexadecimal)
VB46	49	
VB47	57	Dirección del registro a leer parte baja (90 hexadecimal)
VB48	48	
VB49	48	Cantidad de registro a leer parte alta (00 hexadecimal)
VB50	48	
VB51	48	Cantidad de registro a leer parte baja (01 hexadecimal)
VB52	49	

6.1.2. Diseño del esclavo

La ventaja del sistema desarrollado es que no importa cuántos esclavos tenga la aplicación, todos cuentan con los mismos elementos. Lo único que cambia es la dirección del mismo. Por lo tanto los cambios son prácticamente nulos. Es por esto que se desarrolla la explicación de un solo esclavo.

El programa que maneja el esclavo 1 se compone de un programa principal que hace llamados a las demás subrutinas, ocho en total. Además contiene dos interrupciones que son activadas según condiciones detalladas más adelante. El manejo del esclavo 2 es muy parecido ya que, como se mencionó, todos los

esclavos son capaces de interpretar cualquier solicitud aunque la aplicación no las requiera todas. Esto para simplificar futuros proyectos, debido a que, en lo que a comunicación respecta, sólo se modifica el maestro para hacer una solicitud específica y el esclavo esta en capacidad de interpretarla.

6.1.2.1. Diseño de la rutina principal

La rutina principal es la que hace el llamado a las diferentes subrutinas según diversas condiciones.

Lo primero que hace es utilizar la marca especial que sólo se activa cuando se pasa de stop a run para inicializar ciertas variables. Luego consulta si se recibió un mensaje y si es así brinca a la rutina que manipula el mensaje.

Si se terminó de manipular el mensaje se va a activar la variable *final*. Si esta condición esta dada, se verifica el valor del código de función recibido, según este valor se va a atender la solicitud en una subrutina diferente. Se pueden manejar cuatro códigos de función los cuales se consideran suficientes para implementar cualquier aplicación futura; código 1 (leer bytes de salida), código 2 (leer byte de entrada), código 3 (leer bloque de memoria), código 16 (modificar bloque de memoria).

Finalmente y para darle una aplicación específica al proyecto, permanentemente se está ingresando a una subrutina que hace una lectura de un byte para ser enviado. La figura 6.10 muestra el diagrama de flujo para esta parte del programa.

6.1.2.2. Subrutina 0 (inicialización)

Lo primero que se hace es inicializar el puerto serie para la recepción con la siguiente configuración:

- Carácter que inicia la recepción: 58h
- Carácter que finaliza la recepción: 10h
- Máxima cantidad de caracteres a recibir: 40

- Activa la recepción.

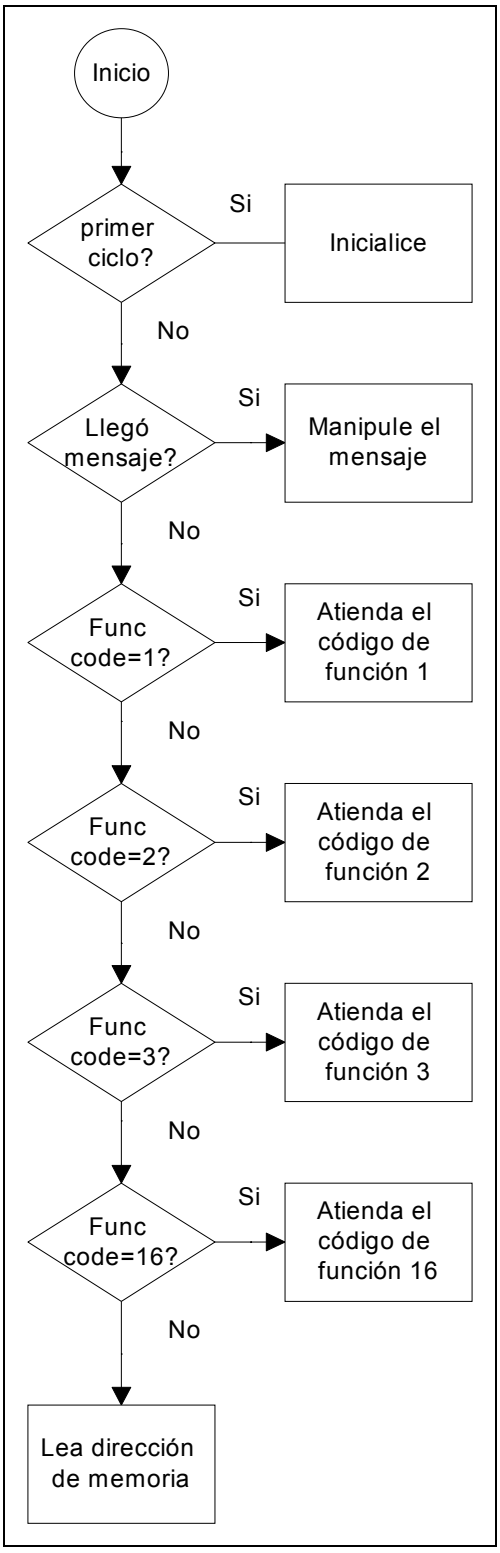

Figura 6.10 Diagrama de flujo para el programa principal

Luego se ajusta la velocidad de transmisión a 9600 bps. Seguidamente se inicializan algunas variables para que no se den errores al encender el PLC.

Como el esclavo no puede iniciar una comunicación, siempre tiene que esperar por la solicitud del maestro. Por lo tanto una transmisión nunca va a estar antes de una recepción lo cual facilita la programación y se evita utilizar la "rutina tonta" que se utilizó en el programa del maestro para deshabilitar el puerto de recepción en el caso de error y que se desee repetir una transmisión. A raíz de esto, el esclavo debe estar listo para recibir desde su encendido, por lo tanto, en esta parte de inicialización se activa la recepción de mensajes. La figura 6.11 muestra el diagrama de flujo de esta etapa de inicialización.

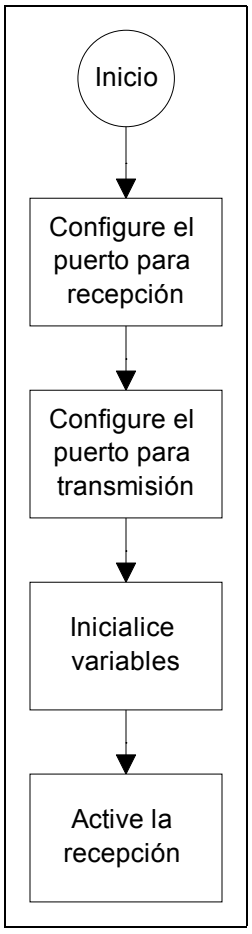

Figura 6.11 Diagrama de flujo para la subrutina 0 (inicialización)

6.1.2.3. Interrupción 0 (mensaje recibido)

Esta interrupción se activa cuando se recibe el final de un mensaje. Este final se detecta por medio del carácter indicado en la configuración de la recepción. Al darse esa condición se da el evento 23 el cual se ha asociado a la interrupción 0.

Lo primero que se hace es desasociar la interrupción con el evento, como medida preventiva evitando que se generen interrupciones falsas. Luego se activa la variable *llego_colon* la cual es la condición para manipular el mensaje recibido.

Como se el esclavo debe generar una respuesta, sin importar la solicitud debe contener un eco de la dirección y un eco del código de función. En esta etapa se copian los valores recibidos, correspondientes a estas variables, en la tabla que se va a enviar como respuesta. Se hace en este momento porque todavía están en ASCII y apenas se salga de esta rutina van a ser modificados a decimal. La figura 6.12 muestra el diagrama de flujo de este procedimiento.

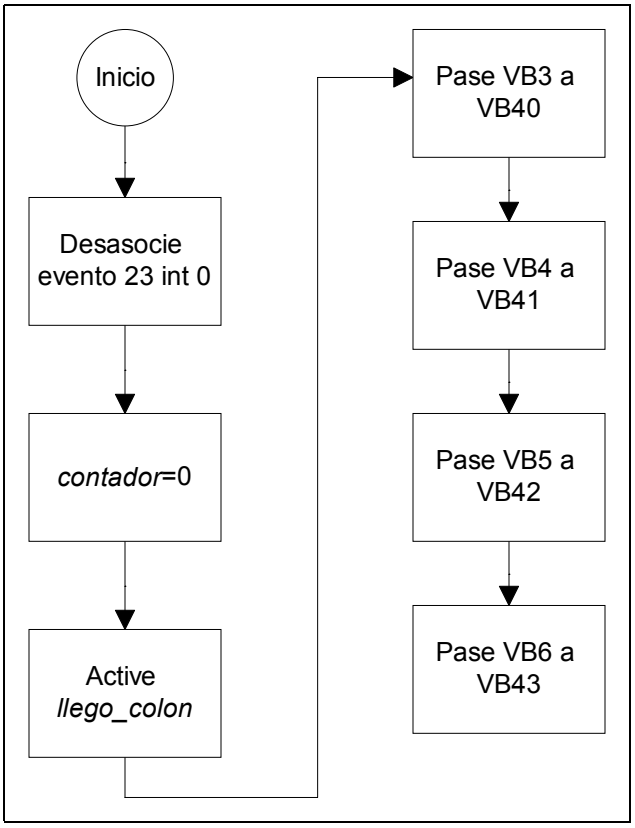

Figura 6.12 Diagrama de flujo para la interrupción 0 (mensaje recibido)

6.1.2.4. Subrutina 4 (manipulación del mensaje)

Esta rutina se activa una vez recibido un mensaje, se ingresa mientras la variable *llego_colon* este activa. Se encarga de tomar un mensaje y convertirlo de ASCII a decimal.

Como se especifica en la documentación de Modbus ASCII (ver apéndice), el esclavo debe generar una respuesta al maestro. Si la solicitud es una lectura de memoria, la respuesta es el valor de la localidad de memoria solicitada. Si es una escritura lo que se debe hacer, la respuesta es un eco de la solicitud. Esto se da para el código de función 16h. Al principio de esta subrutina se verifica el código de función para ver si es 16h, si es así, se realiza un eco de todo el mensaje antes de que se modifique de ASCII a decimal.

Luego empieza la manipulación del mensaje, primero se colocan los punteros sobre las localidades VB3 y VB4. El manejo de los datos es el mismo que se le da a un mensaje recibido en el maestro, por lo tanto las figuras 6.7 y 6.8 sirven como ilustración. La figura 6.13 muestra el diagrama de flujo para esta subrutina.

Una vez que se ha convertido la dirección del esclavo a decimal, se compara con la del esclavo para constatar si la solicitud va dirigida a dicho esclavo. Si no es la dirección, entonces se deshabilita la variable *llego_colon* para que no se siga manejando el mensaje ya que no tiene sentido pues la solicitud no es para él. Finalmente se activa la recepción para dejar al esclavo listo para otra transmisión del maestro.

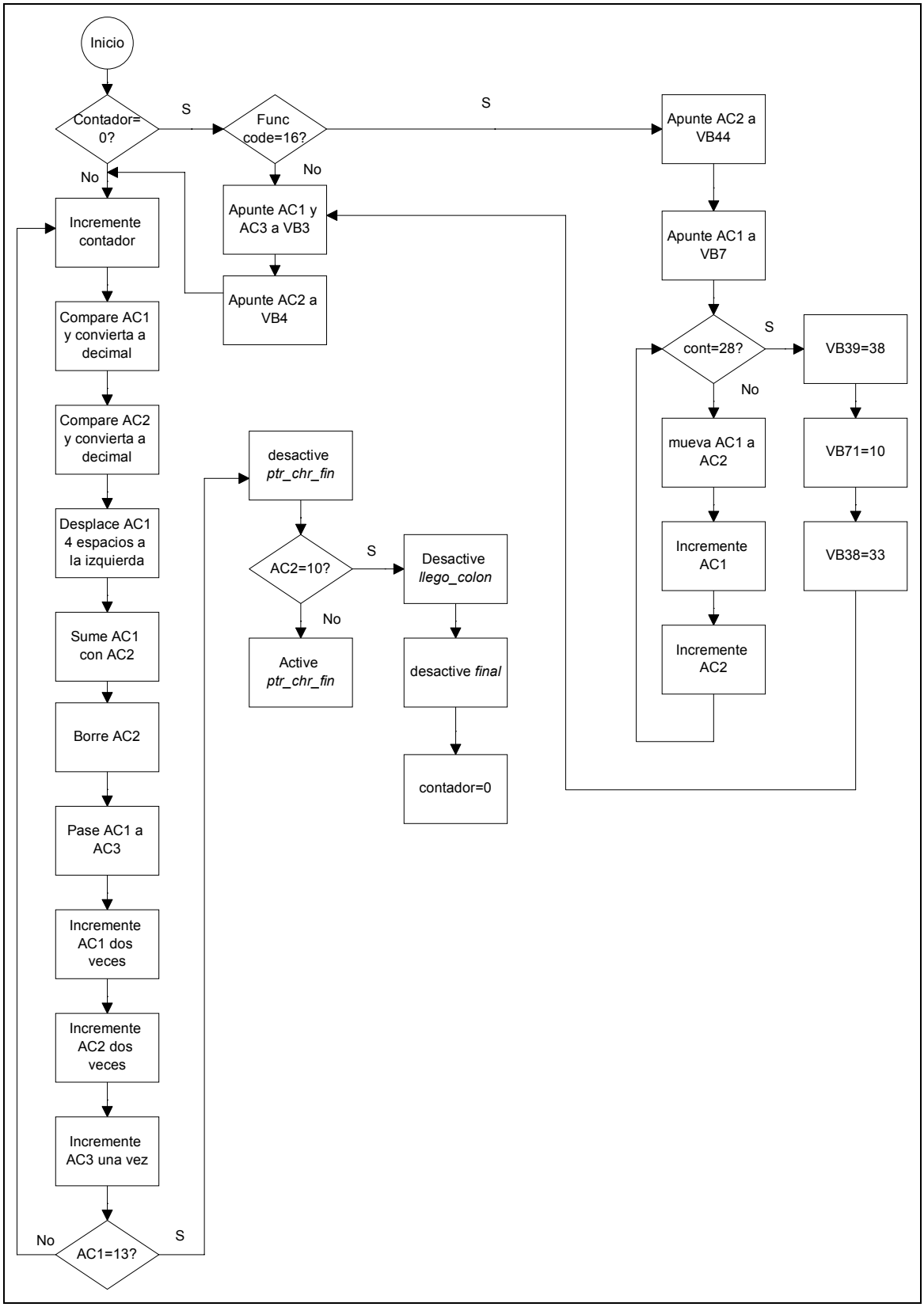

Figura 6.13 Diagrama de flujo para la subrutina 4 (manipulación del mensaje)

6.1.2.5. Subrutina 2 (Código de función 1: leer byte de salidas)

Esta rutina se encarga de leer un grupo de salidas. Una limitante del sistema es que el protocolo Modbus especifica para este código la lectura de una bobina de salida (un bit) y se le indica, en la solicitud, la cantidad de bits por leer. El PLC no permite direccionar bits sólo bytes, palabras y palabras doble. Como se recibe un valor dentro de una localidad de memoria, la única forma de colocar ese valor como dirección es utilizando un puntero pero no se puede llegar a un único bit. Por lo tanto, se implementó para este código la lectura de un byte de salida. Se indica el byte de inicio y la cantidad de bytes a leer.

En varios procedimientos se utiliza una variable llamada *primer_ciclo*. Esta se utiliza para inicializar valores, es decir, esta activa cuando se entra al procedimiento, se asignan valores y luego se desactiva la variable.

Lo primero que se hace es colocar los punteros AC1 y AC2 a QB0. Esto con el fin de colocarlos en el byte inicial (AC1) y en el byte final (AC2). La idea es de incrementar AC1 tantas veces como lo indica la dirección inicial e incrementar AC2 tantas como especifica el valor final recibido en el mensaje de solicitud. Luego coloca el puntero AC3 en la dirección VB44 este es el valor en la tabla que luego se va a convertir a Modbus ASCII para enviar la respuesta.

Seguidamente se tiene una dirección con valor inicial cero la cual se compara con VB6 (Dirección inicial a leer según solicitud), si no es igual se incrementa el puntero AC1 y el mismo contador de comparación. Así, se va a incrementar AC1 tantas veces como el mensaje recibido lo indique.

De igual forma, AC2 se incrementa tantas veces como el valor VB8 (Cantidad de bytes a leer) lo indique y controlado por la variable VB31 la cual inicia en cero y termina cuando VB31=VB8.

Hasta aquí la inicialización. Lo que sigue es borrar VB31 y VB30 para futuros ingresos al procedimiento. Hasta este punto lo que se tiene es AC1 en una localidad inicial y AC2 en una final. Mientras AC1 sea menor que AC2, se guarda el valor apuntado por AC1 en la localidad apuntada por AC3 para luego incrementar AC1 una vez. Como Se tiene un valor con decenas y unidades y se va

a convertir a Modbus ASCII, cada localidad de memoria va a requerir dos localidades (dos caracteres ASCII), por esto se incrementa AC3 dos veces, para dejar un espacio libre para la futura conversión.

Lo que sigue es la conversión de los datos capturados en decimal a ASCII para su envío. Al igual que con el maestro, este procedimiento es complejo por lo tanto se apoya su explicación con figuras sobre el movimiento en memoria de los punteros y variables. Primero se colocan los punteros AC1 en VB44 y AC2 en VB45. La figura 6.14(a) muestra la forma en que se ve la memoria al inicio de la conversión.

Luego se toma el valor apuntado por AC1 y se desplaza cuatro espacios a la izquierda guardando el resultado en AC2. La figura 6.14(b) muestra en binario el resultado de esta operación. Como el resultado de la operación se almacena en AC2, AC1 permanece sin variación.

Con esto se tiene en AC2 uno de los dos dígitos que componen el valor, solamente que se encuentra desplazado a la izquierda. Lo que se hace ahora es desplazar a la derecha AC2. De igual forma, se desplaza AC1 a la derecha para dejar sólo el valor deseado. La figura 6.14(c) muestra el resultado de estas dos operaciones.

Finalmente se toman los valores manipulados y se convierten a ASCII, esto lo hace la subrutina 3. Una vez convertidos, se incrementan los punteros AC1 y AC2 dos veces. La figura 6.14(d) enseña la forma en que quedan los valores y los punteros para la siguiente conversión.

La localidad VB35 contiene la cantidad de caracteres que se van a transmitir, además se incrementa dos veces cada vez que se manipula un par de localidades de memoria. La variable *contador* contiene la cantidad de datos a manejar, por lo tanto el procedimiento anterior se repite mientras VB35 sea menor que *contador*.

Finalizado el manejo de la tabla, se tienen todos los caracteres en ASCII, faltan colocar los indicadores de final de mensaje (13h y 10h) y el inicio de mensaje (58h).

Una vez colocados estos valores, se tiene que incrementar el valor de VB35 para que se envíe la tabla completa para esto se le suma 7 correspondientes a los tres caracteres de control y los cuatro que se hicieron como eco al inicio de la recepción (dirección del esclavo y código de función). Con esto se tiene le mensaje completo y listo para transmitir.

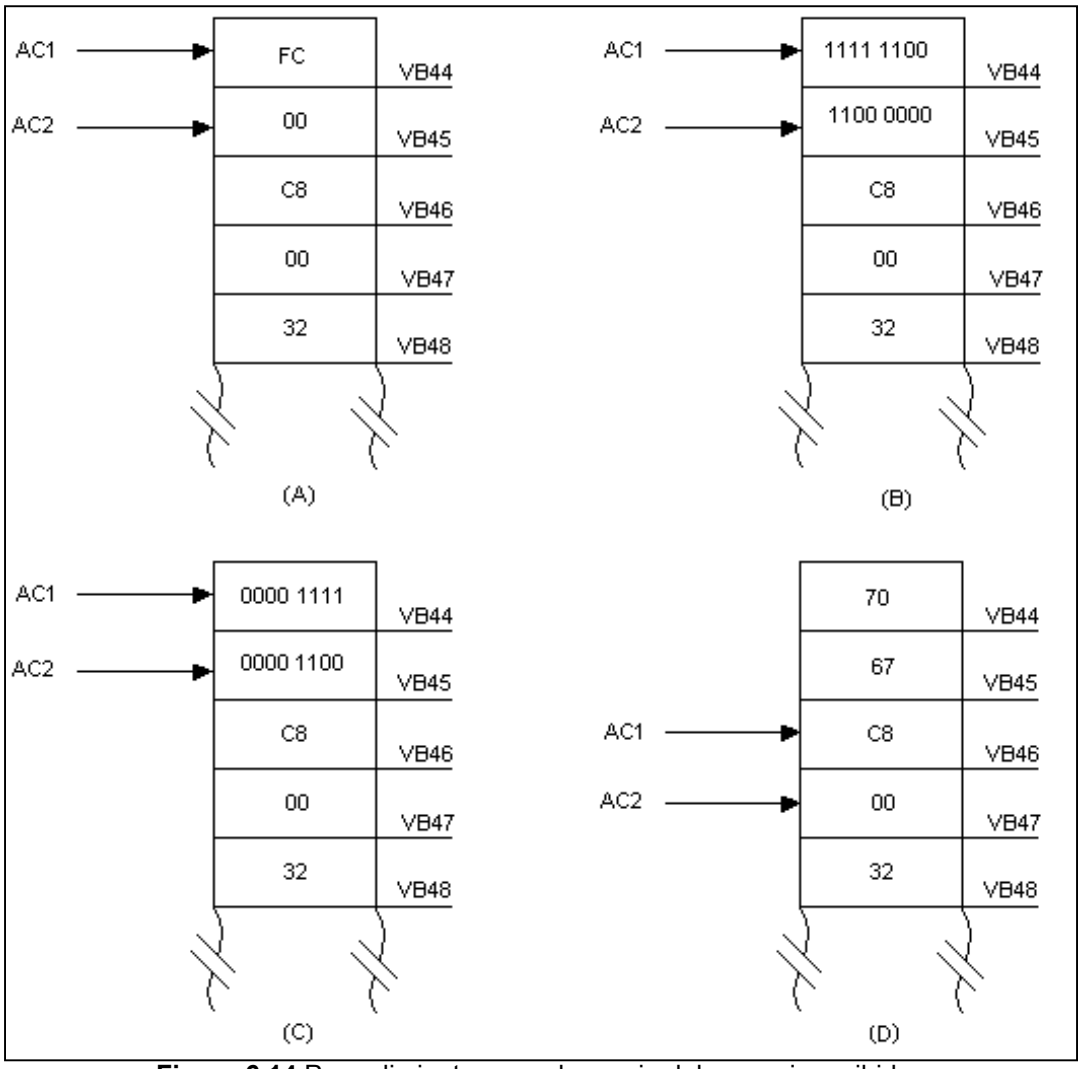

Figura 6.14 Procedimiento para el manejo del mensaje recibido

Lo último que se hace es la transmisión. Se asocia la interrupción 1 al evento 9. Este evento corresponde al final de la transmisión. La figura 6.15 muestra el diagrama de flujo para el funcionamiento de este procedimiento.

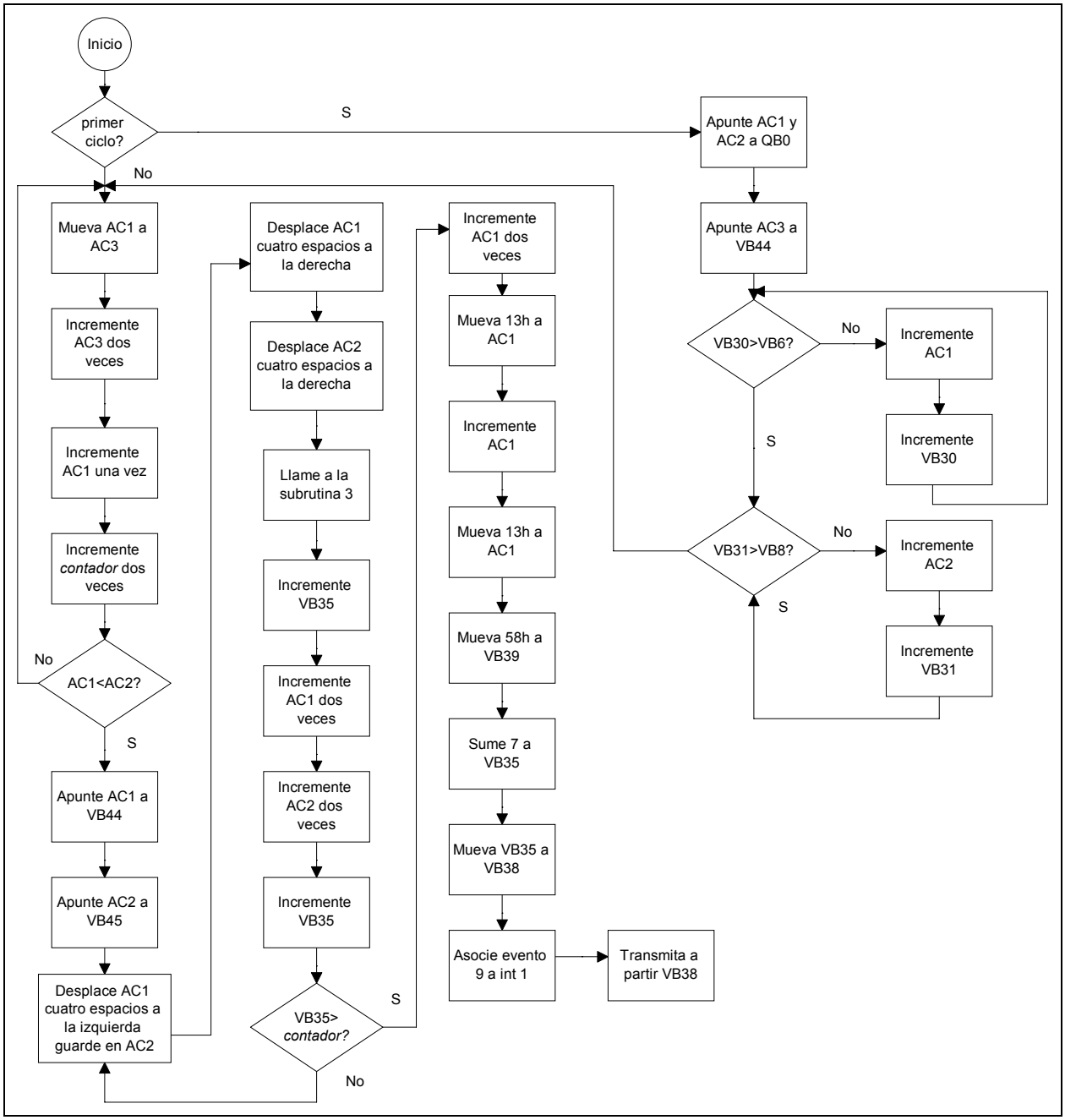

Figura 6.15 Diagrama de flujo para la subrutina 2 (código de función 1)

6.1.2.6. Subrutina 1 (Código de función 2: leer byte de entradas)

Esta subrutina es parecida a la anterior pues consiste en leer un byte de entradas. Por lo tanto no se profundiza en explicaciones ya que el manejo de los datos es exactamente igual, lo único que cambia es la inicialización, donde se colocan los punteros AC1 y AC2 sobre IB0 en vez de QB0 para el caso anterior.

6.1.2.7. Subrutina 5 (Código de función 3: leer byte en memoria)

Esta rutina se encarga de leer un grupo de bytes en memoria. Se recibe la dirección inicial y la cantidad de bytes a leer. Lo primero que se hace es una inicialización, Se colocan los punteros AC1 sobre VB44. AC2 sobre VB45 y AC3 en VB0.

Después de colocados los punteros se ingresa a un ciclo *for*. La posición a la que se desea empezar a leer en memoria está determinada por la localidad VB6 (dato recibido en la solicitud del maestro). El ciclo incrementa AC3 hasta que el contador de control VW34 sea igual que VW5, cuando esto se da, AC3 va a estar apuntando a la dirección inicial.

Después corresponde la lectura de los bytes en memoria tantos como VB8 lo indique (cantidad de bytes a leer según solicitud recibida). Se ingresa a un ciclo *for* que se repite hasta que el contador de control VW34 sea igual a VW7. Mientras la condición no sea válida se va a realizar lo siguiente:

Se mueve el contenido de AC3 a AC1 (AC1 apunta a la tabla de envío de la respuesta).

Luego se manipula el mensaje. Se desplaza AC1 cuatro espacios a la izquierda y se guarda el resultado en AC2. Después se desplaza tanto AC1 como AC2 cuatro espacios a la derecha para obtener el valor decimal descompuesto en unidades y centenas (cada uno en un localidad de memoria diferente).

Seguidamente, se hace el llamado a la subrutina 3 para convertir el valor decimal en ASCII, para finalmente, incrementar AC1 y AC2 dos veces así como AC3 una vez.

Una vez fuera del ciclo, corresponde agregar los caracteres de control del mensaje (inicio y final). Se mueve un 13h a donde esta apuntando AC1 y un 10h donde esta apuntando AC2. También, se guarda un 58h en la localidad VB39.

Finalmente, se suma 7 a VB38 (cantidad de datos a transmitir) y se realiza la transmisión de la respuesta. La figura 6.16 muestra el diagrama de flujo para esta subrutina.

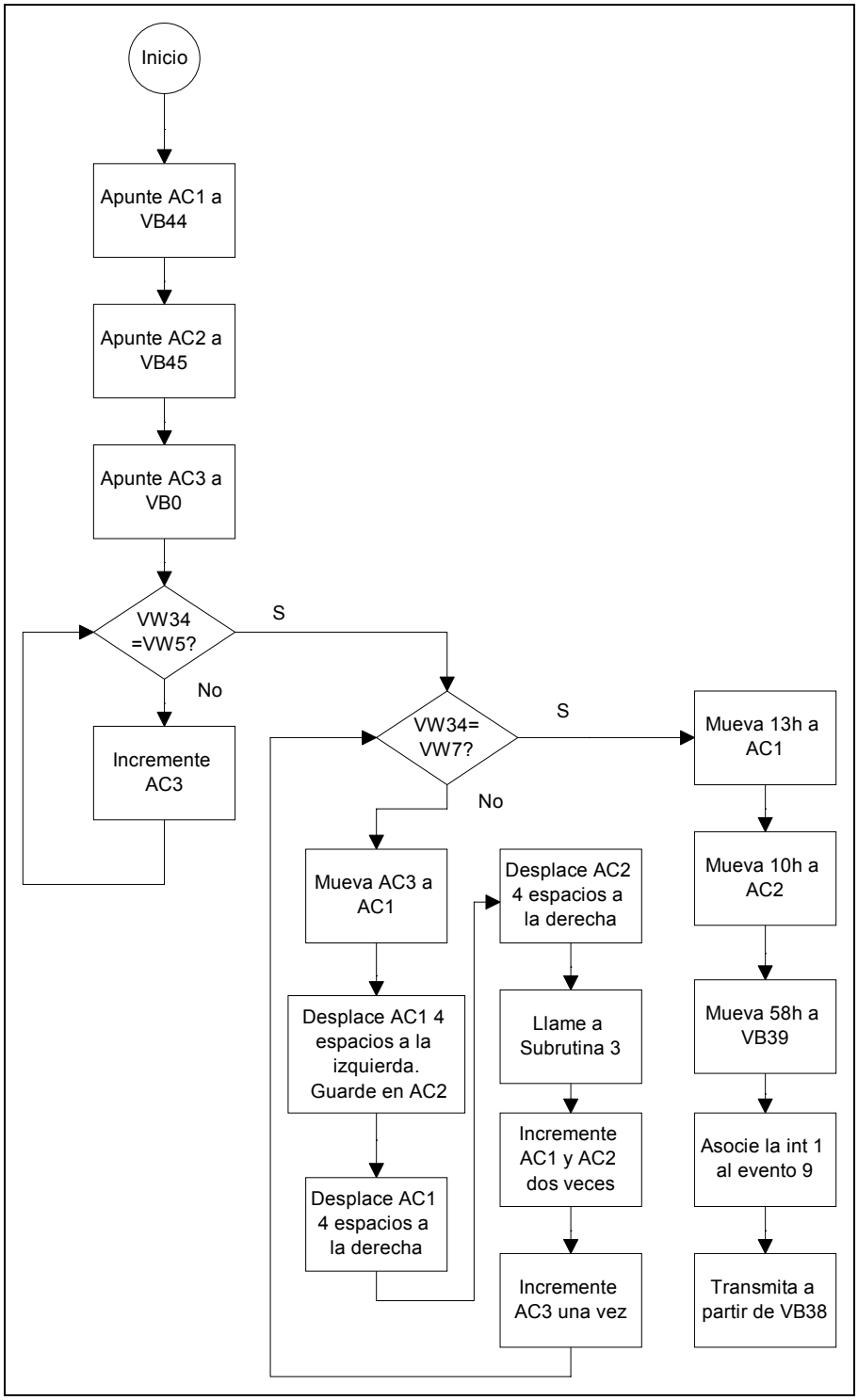

Figura 6.16 Diagrama de flujo para la subrutina 5 (código de función 3)

6.1.2.8. Subrutina 6 (Código de función 16: escribir byte en memoria)

Esta subrutina recibe una localidad inicial en memoria para escribir, la cantidad de bytes a escribir y luego la información que va a ser escrita en cada dirección. La figura 6.17 muestra el diagrama de flujo con la lógica de operación de esta subrutina.

Lo primero que se hace es colocar AC1 apuntando a VB0 y AC2 apuntando a VB10.

VB6 contiene la dirección inicial a la que se va a empezar a escribir (según la solicitud recibida). Se ingresa a un ciclo *for* que se repite hasta que el contador de control VW36 sea igual a VW5. Mientras se esta en el ciclo, se va a incrementar AC1. Cuando se termina el ciclo se tienen AC1 apuntando a la dirección inicial de escritura.

De igual forma, VB8 contiene la cantidad de bytes que se van a escribir. Se realiza un ciclo *for* controlado por VW36. Mientras no sea igual a VW7, se va a pasar el contenido apuntado por AC2 (valor a escribir según mensaje recibido) en la localidad apuntada por AC1.

Como el mensaje ya fue convertido a decimal en un procedimiento anterior, al pasar de una localidad a otra, se tiene el valor como se desea, por lo tanto no se requiere de mayor conversión. La tabla de respuesta ya fue llenada en otra subrutina ya que es sólo un eco de la solicitud enviada por el maestro. Lo que seguiría es enviar dicha tabla.

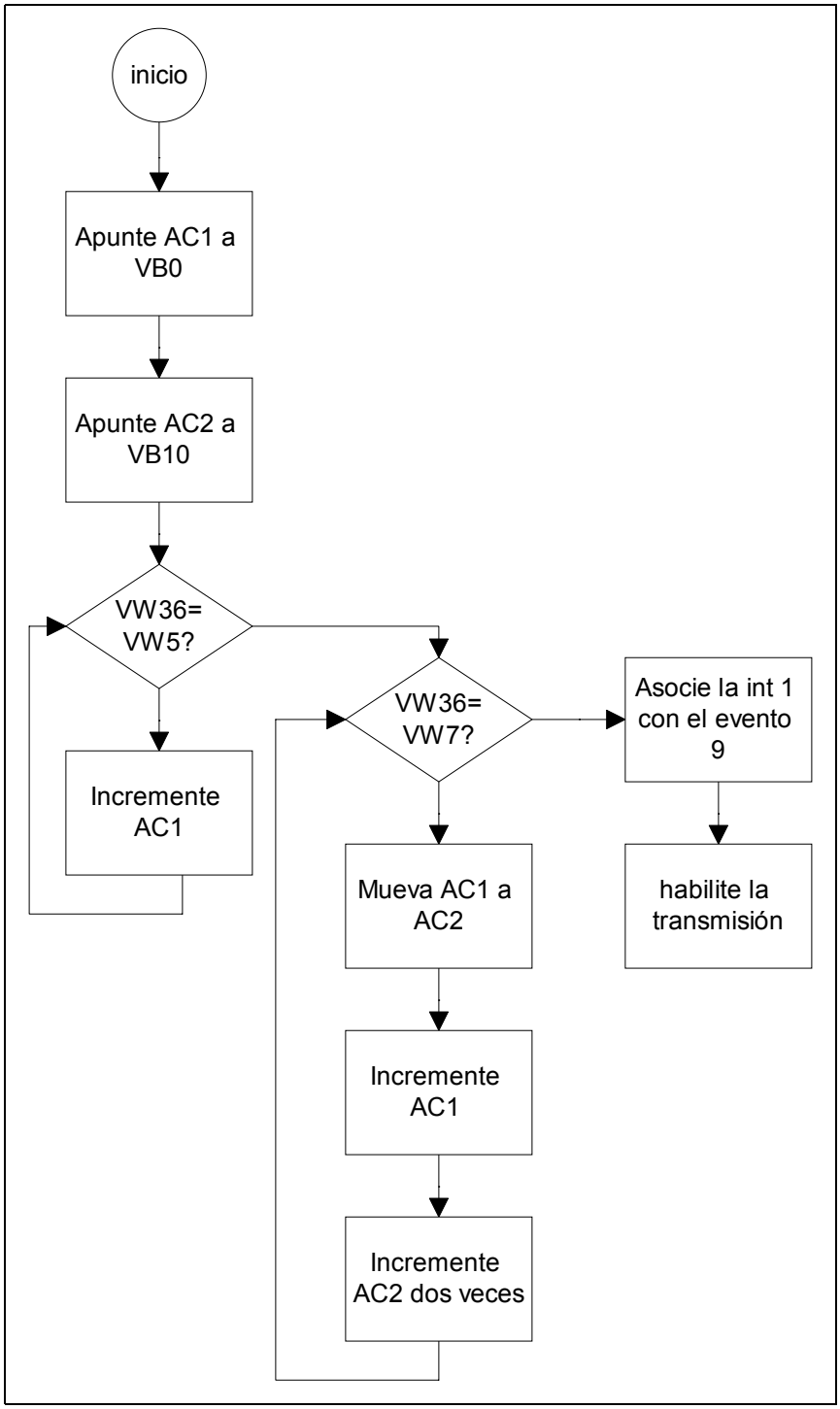

Figura 6.17 Diagrama de flujo para la subrutina 6 (código de función 16)

6.1.2.9. Interrupción 1 (Transmisión finalizada)

Esta interrupción se activa una vez que se termina de transmitir la respuesta del esclavo hacia el maestro. Consiste en el desasocie de la

interrupción con el evento, evitando interrupciones falsas, y finalmente, la activación de la recepción con el respectivo asocie de la interrupción 0 con el evento 23 (mensaje recibido). La figura 6.18 muestra el diagrama de flujo para esta interrupción.

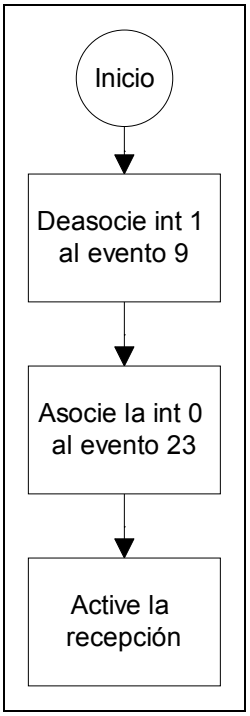

Figura 6.18 Diagrama de flujo para la interrupción 1 (Transmisión finalizada)

6.1.2.10. Subrutina 7 (Lectura de entradas)

Para darle un aplicación al sistema, se realiza una permanente lectura del byte de entradas IB0 y se transfirieren a la localidad VB400. La figura 6.19 muestra el diagrama de flujo para esta rutina.

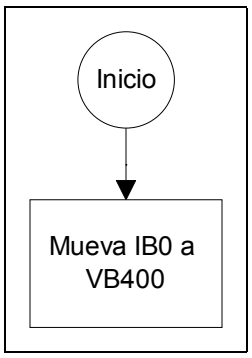

Figura 6.19 Diagrama de flujo para la subrutina 7 (Lectura de entradas)

6.1.2.11. Subrutina 3 (Conversión de decimal a ASCII)

Esta rutina es la que se encarga de tomar un valor decimal y convertirlo a ASCII. Como los valores a convertir son unidades y decenas (cada uno en una localidad diferente) se tiene AC1 apuntando a uno y AC2 apuntando al otro. El procedimiento toma AC1 y lo compara para pasar a ASCII, de igual forma se hace con AC2. La figura 6.20 muestra el diagrama de flujo para este procedimiento.

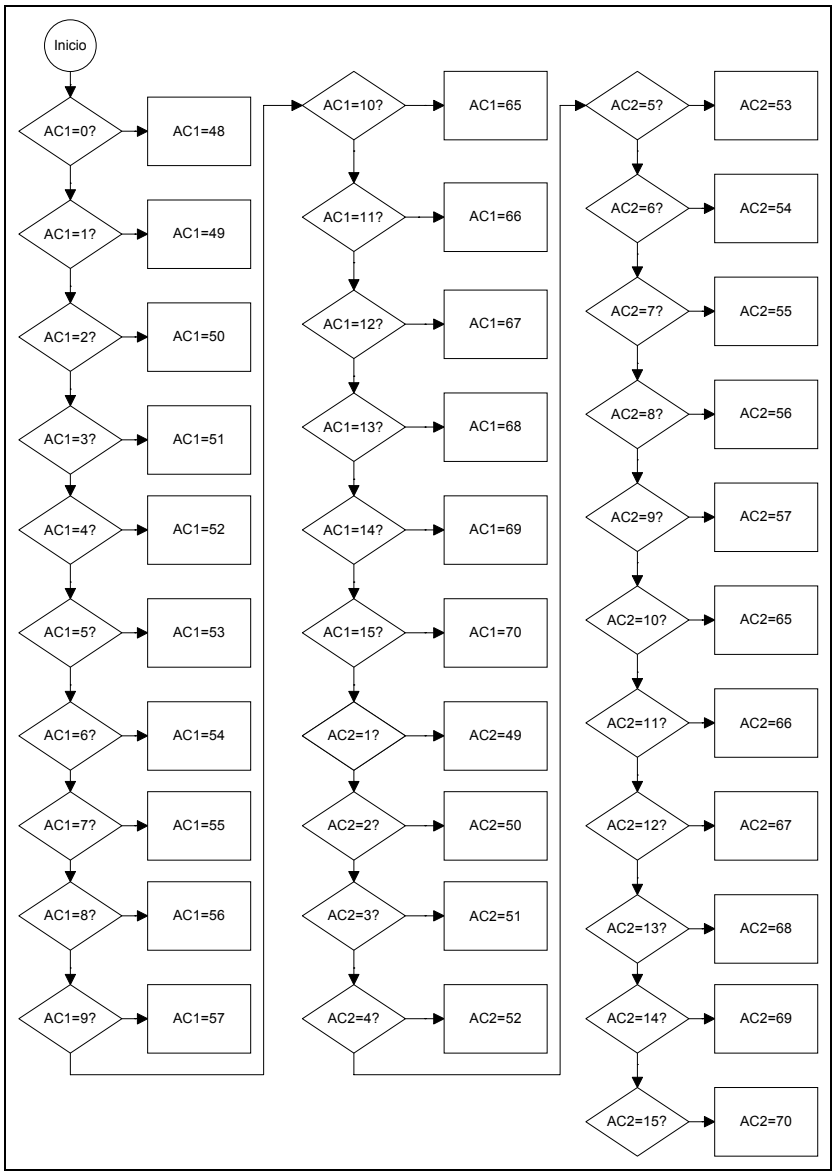

Figura 6.20 Diagrama de flujo para la subrutina 3 (Conversión de decimal a ASCII)

6.2. Alcances y limitaciones

6.2.1. Alcances

El proyecto permite una adecuada comunicación entre un maestro y una cantidad dada de esclavos. Para la aplicación desarrollada se manejan dos esclavos, sin embargo para aplicaciones desarrollados por Siemens, utilizando la aplicación se llegarán a utilizar sistemas de hasta cinco esclavos (Proyecto para el A y A en La Cruz, Guanacaste).

En caso de que el maestro envíe una solicitud y el esclavo no la reciba (situación muy probable), se resuelve adecuadamente enviando otra solicitud sin afectar los demás esclavos.

El proyecto es de fácil modificación, lo cual es muy importante ya que tiene que servir para multiples aplicaciones.

Lo que a comunicación se refiere, las variaciones en el esclavo son, prácticamente, el cambio en la dirección de este, debido a que todos los PLC´s pueden manejar cualquier solicitud que les envíe el maestro (dentro de las funciones desarrolladas).

6.2.2. Limitaciones

El protocolo Modbus no fue planeado para utilizar en PLC´s sino con microcontroladores de Modicon, por lo tanto no todas las funciones se pueden desarrollar en un PLC.

Para el proyecto se desarrollaron únicamente cuatro funciones:

- Leer byte de entradas
- Leer byte de saldias
- Leer byte en memoria
- Escribir byte en memoria

Sin embargo, estas funciones son suficientes para satisfacer cualquier futura aplicación.

Otra limitante radica en las funciones 1, 2 y 3 (leer conjunto de bobinas). Para poder saber cuales son las localidades en memoria que se van a leer, se recibe el mensaje. Dentro del mensaje se especifica la localidad inicial del bit a leer. Para llegar a este bit sería necesario direccionarlo con un puntero. El problema radica en que los bits no se pueden direccionar, sólo bytes, palabras y palabras dobles.

Se tuvo que modificar la interpretación de estas funciones y lo que se envía es el byte inicial y la cantidad de bytes a leer. La respuesta lleva los valores contenidos en dichos bytes.

Esto conlleva a que si se quiere modificar un único bit, se debe trabajar sobre el byte de manera que se modifique el bit de interés y reescribir todo el byte.

Capítulo 7. CONCLUSIONES

- El protocolo Modbus ASCII no fue diseñado para trabajar con plc´s sin embargo, es factible implementarlo para que funcione adecuadamente.
- El sistema permite una adecuada comunicación entre un maestro con dos esclavos.
- El sistema de es de fácil expansión a una cantidad determinada de esclavos
- Con las cuatro funciones desarrolladas se pueden realizar las aplicaciones necesarias requeridas por la empresa.
- El PLC Simatic no permite direccionamiento indirecto de bits, por lo tanto las funciones correspondientes a lecturas de bits se tuvieron que modificar a lecturas de bytes
- Existe una alta probabilidad de que el esclavo no reciba una solicitud, por lo tanto es indispensable la repetición de solicitudes.
- Se necesita de una rutina que deshabilite el puerto para recepción antes de realizar una transmisión (“rutina tonta”), para el caso del maestro.
- La “rutina tonta” no es necesaria para los esclavos.
- A la hora de ampliar el sistema, el maestro es el que lleva la mayor cantidad de modificaciones en su programa
- A la hora de ampliar el sistema el esclavo esta en condiciones de atender cualquier de las solicitudes diseñadas, por lo tanto sólo se debe programar lo correspondiente a la lógica de operación del sistema.

BIBLIOGRAFIA

Siemens Simatic S7-200. Applications. “Tips and Tricks”. 1995

Siemens Simatic. Sistemas de automatización S7-200. 1999

Modicon. Modbus protocol. http:www.modbus.org, 2002

APENDICES

Apéndice A.1 Glosario de términos y Lista de Abreviaturas

PLC:

Programador lógico programable. Dispositivo programable con una serie de entradas y salida configurable según los requerimiento del usuario.

PC:

Computadora personal

Polling:

Sistema utilizado para manejar un sistema maestro esclavo en el cual el maestro atiende a un esclavo; cuando termina atiene al siguiente y asi sucesivamente hasta reiniciar con el primero.

Sistema maestro esclavo:

Forma de conectar una serie de dispositivos de manera que uno (maestro) gobierna el funcionamiento de los demás (esclavos).

Modbus ASCII:

Protocolo de comunicación abierto utilizado para la comunicación entre sistemas de tipo maestro esclavo.

Protocolo:

Convención utilizada para que diferentes sistema cumplan con los mismo requerimientos y así exista entendimiento entre estos.

Protocolo abierto:

Protocolo cuyo formato y características pueden ser utilizadas por cualquier persona sin necesidad de permiso de la empresa o grupo creador del mismo.

Actuadores:

Elementos que reaccionan según instrucciones para realizar una función específica.

Paquete de información:

Grupo de datos que contienen los elementos necesarios para el envío de información según un protocolo específico.

Apéndice A.2 Generalidades del protocolo Modbus ASCII

INTRODUCCIÓN

Generalidades

Modbus es el protocolo utilizado por lo controladores desarrollados por Modicon. Este describe el proceso que el controlador utiliza para poder comunicarse con otro dispositivo, cómo va a responder a estos y la forma en que se van a detectar y reportar los errores en la transmisión.

Durante una comunicación entre dispositivos, el protocolo determina la forma en que el dispositivo va a identificar su dirección, el mensaje enviado a este, la clase de acción a ejecutar y da cualquier información complementaria que se necesite. Si se necesita una respuesta, el controlador puede construirla utilizando el protocolo Modbus. La forma en que se comunican los dispositivos utilizando el protocolo es por medio de un sistema maestro esclavo, en la cual solo el maestro puede iniciar una transacción. Los otros dispositivos responden suministrando la información solicitada por el maestro o realizando las acciones dadas en la solicitud.

El maestro puede iniciar una comunicación con un esclavo específico o puede utilizar una transacción difundida (*broadcast*) en la cual, la orden va a dirigida a todos los esclavos. Los esclavos devuelven a la solicitud cuando se realiza una comunicación directa es decir, para la transmisión difundida no se genera respuesta.

El protocolo Modbus establece el formato para la solicitud del maestro colocando la dirección del esclavo, un código de función que define la acción a realizar, cualquier información que se necesite enviar y un campo para revisar errores. La respuesta también es construida usando el protocolo Modbus, esta contiene campos confirmando la acción ejecutada, cualquier información que se necesite enviar y un espacio para revisar errores. Si ocurre un error o si el esclavo no puede ejecutar la orden, este construye un mensaje de error y envía lo envía como su respuesta.

La solicitud: Una solicitud se compone básicamente de cuatro partes

- Dirección: especifica la dirección del esclavo al que va dirigida la solicitud.
- Código de función: Este le indica a dispositivo específico que tipo de acción debe ejecutar.
- Bytes de información: Contiene cualquier dato adicional que necesite el esclavo para ejecutar la orden.
- Campo para errores: Proveen un método para que el esclavo valide la integridad del contenido del mensaje.

La respuesta: Una respuesta se compone básicamente de cuatro partes

- Dirección: es un eco de la dirección del esclavo de manera que el maestro sepa de donde proviene la solicitud.
- Código de función: También es un eco de la orden suministrada por el maestro.
- Bytes de información: Contienen la información recolectada por el esclavo tales como valores en registros o de estatus.
- Campo para errores: Permite al maestro validar el mensaje enviado por el esclavo.

Si ocurre un error, la código de función es modificado para indicar que la respuesta es una respuesta de error y los bytes de información describen el tipo de error.

7.1. Tipos de transmisión

Modo ASCII:

Cuando un controlador está configurado para comunicarse utilizando el protocolo Modbus ASCII, cada byte (ocho bits) es enviado como dos caracteres ASCII. La principal ventaja de este modo es que permite intervalos de tiempo de hasta un segundo entre envío de caracteres, sin causar error.

Bits utilizados:

1 bit de inicio (start bit)

7 bits de información, el menos significativo va primero.

1 bit de paridad (par o impar)

1 bit de paro (stop bits), dos si no se utiliza paridad.

Enmarcado del mensaje:

En comunicación ASCII se envía primero un signo dos puntos *(:)* lo cual representa el carácter 3Ah según este código, este indica el inicio de la comunicación y le permite al receptor tomar el mensaje desde el principio. Termina con un par de saltos de línea, caracteres 0Dh y 0Ah, respectivamente.

Los caracteres permitido para el resto de la información están comprendidos entre 0-9 y A-F.

Cuando se recibe la dirección cada dispositivo decodifica esta para determinar el esclavo que debe atender la orden. Se pueden dar intervalos de hasta un segundo entre caracteres, pero si de un tiempo mayor, el dispositivo receptor asume que se dio un error.

Modo RTU:

Cuando un dispositivo esta configurado para trabajar con el protocolo Modbus RTU, cada byte (ocho bits) en el mensaje contiene dos números de cuatro bits hexadecimales. La principal ventaja de este modo es que la gran densidad de caracteres permite un mejor flujo de información que lo que permite el ASCII utilizando la misma velocidad (baudios por segundo), eso si, cada mensaje debe ser transmitido continuamente.

Bits utilizados:

1 bit de inicio (start bit)

8 bits de información, el menos significativo va primero.

1 bit de paridad (par o impar)

1 bit de paro (stop bits), dos si no se utiliza paridad.

Enmarcado del mensaje:

En modo RTU el mensaje empieza con un intervalo de silencio de por lo menos 3.5 veces el tiempo que hay entre caracteres. Después de esto se puede enviar la primera parte del mensaje que contiene la dirección. Los caracteres permitidos son los valores hexadecimales 0-9 y A-F. Cuando la dirección es recibida, cada dispositivo la decodifica para determinar si este es el esclavo direccionado. Terminada la comunicación, se genera un silencio de 3.5 veces el tiempo entre caracteres. Después de esto un nuevo mensaje puede iniciar.

La transmisión debe ser continua, si se da un silencio de 1.5 veces el tiempo entre caracteres antes de terminar la transmisión el receptor desecha el mensaje y asume que la próxima información que va a recibir es una dirección de un nuevo mensaje. Igualmente, si un mensaje inicia antes de que pasen los 3.5 veces de tiempo, el dispositivo lo considera como parte del mensaje anterior. Esto provocará un error.

Manejo de la dirección

El espacio para la dirección contiene dos caracteres (ASCII) u ocho bits (RTU). Direcciones válidas para esclavos están en el tango de 0 a 247. Los diferentes esclavos se direccionan a partir de 1 hasta 247. Un maestro direcciona un esclavo poniendo la dirección de este en el espacio para dicho fin. Cuando el esclavo envía la respuesta, este coloca su propia dirección para permitirle al maestro saber quién está respondiendo. La dirección 0 se utiliza para comunicación difundida.

Manejo de los códigos de funciones

El campo para la función contiene dos caracteres (ASCII) u ocho bits (RTU). Códigos válidos están en el rango de 1 a 255 decimal. Cuando un mensaje es

enviado por el maestro, la función le indica al esclavo la operación que debe realizar. Cuando este responde, utiliza el mismo código para indicar una operación normal o para indicar un error.

En una respuesta normal el esclavo hace un eco del código de función. Para un error, el esclavo devuelve un código que es equivalente al recibido sólo que el bit más significativo se coloca en uno lógico. Además, el esclavo coloca en el espacio de datos un código que indica el tipo de error que se generó.

Manejo del espacio de datos

El espacio de datos está construido por dos dígitos hexadecimales en el rango de 00 a FF. Estos pueden ser un par de caracteres ASCII o del tipo RTU, de acuerdo al modo de transmisión.

Los datos enviados desde un maestro hacia un esclavo contiene información adicional que el esclavo debe utilizar para tomar acciones definidas por el código de función. Si no se dan errores, la respuesta de un esclavo contiene la información solicitada. Si se da error, los datos enviados contiene el código referente al tipo de error dado. Finalmente, el espacio de datos puede ser inexistente en algunos mensajes donde no se requiere de información adicional.

Manejo del espacio para revisión de errores

Cuando se utiliza modo ASCII, el espacio para errores contiene dos caracteres ASCII. Estos son el resultado de un cálculo del tipo revisión redundante longitudinal (LRC por sus siglas en inglés, *Longitudinal Redundancy Check),* que es realizado en el contenido del mensaje, excluyendo el carácter de inicio y lo dos finales. Los caracteres LRC son adicionados al final del mensaje.

Ambos, la revisión de caracteres (paridad) como la revisión del mensaje (LRC) son generados en el maestro y aplicado al mensaje antes de la transmisión. El esclavo revisa cada carácter y el mensaje entero durante la recepción para determinar posibles errores.

Método LRC:

El LRC revisa el contenido del mensaje excluyendo el carácter de inicio (3Ah) y los caracteres de final (0Dh y 0Ah). También es independiente del método de paridad utilizado o si no se utilizó del todo. El LRC es un byte que contiene ocho bits, es calculado por el dispositivo transmisor el cual anexa el LRC al final del mensaje, el receptor calcula un carácter LRC durante la recepción del mensaje y lo compara con el suministrado por el transmisor. Si no son iguales, un error se dio.

El LRC es calculado sumando sucesivamente todos los bytes del mensaje y descartando cualquier acarreo, luego el resultado se complementa a dos.

DATOS Y FUNCIONES DE CONTROL

Todas las direcciones en los mensajes Modbus están referenciados a cero. La tabla A2.1 muestra un ejemplo de una solicitud típica tanto para modo ASCII como RTU.

Tabla A2.1 Ejemplo de un solicitud en formato RTU y ASCII

Nombre del espacio	Ejemplo	ASCII	RTU
Encabezado		:	nada
dirección del esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Dirección inicial (HI)	00	0 0	0000 0000
Dirección final (LO)	6B	6 B	0110 1011
Número de registros (HI)	00	0 0	0000 0000
Número de registros (LI)	03	0 3	0000 0011
Revisión de Error		LRC (2 chr)	CRC (2 chr)
Final del mensaje		CR LF	nada
bytes totales		**17**	**8**

Para este ejemplo, el esclavo generará una respuesta para según el modo que se utilice, RTU o ASCII. La tabla A2.2 muestra lo que corresponde a una respuesta de un esclavo ante la solicitud hecha por el maestro acorde a la tabla 1.

Tabla A2.2 Ejemplo de una respuesta en formato RTU y ASCII

Nombre del espacio	Ejemplo	ASCII	RTU
Encabezado		:	nada
dirección del esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Contador de bytes	06	0 6	0000 0110
Datos (HI)	02	0 2	0000 0010
Datos (LO)	2B	2 B	0010 1011
Datos (HI)	00	0 0	0000 0000
Datos (LO)	00	0 0	0000 0000
Datos (HI)	00	0 0	0000 0000
Datos (LO)	63	6 3	0110 0011
Revisión de Error		LRC (2 chr)	CRC (2 chr)
Final del mensaje		CR LF	nada
bytes totales		**23**	**8**

FUNCIONES

A continucación se listan las diferentes funciones que se pueden utilizar para indicarle al esclavo la acción que se debe ejecutar. Se lista el código en hexadecimal de la correspondiente función, junto con su nombre y luego la descripción.

01 Leer estatus de bobina.

Lee el estado encendido o apagado de una salida discreta en un esclavo. La opción difundida no es válida.

Solicitud: El mensaje específica la bobina de inicio y la cantidad que van a ser leídas.

Respuesta: El esclavo devuelve el estado de cada bobina en registros de ocho bits, donde cada bit representa una bobina. Se inicia con la primera bobina en el bit menos significativo y se van utilizando hasta el mas significativo. Si son más de ocho bobinas, se utilizan los byte que sea necesario. Si al terminar se tiene un byte que le sobran bits, estos son dejados en cero lógico.

02 Leer estatus de entrada

Lee el estado encendido o apagado de una entrada discreta en un esclavo. La opción difundida no es válida.

Solicitud: El mensaje especifica la entrada inicial y la cantidad de entradas que van a ser leídas.

Respuesta: El esclavo devuelve el estado de cada entrada en registros de ocho bits, donde cada bit representa una entrada. Se inicia con la primera entrada en el bit menos significativo y se van utilizando hasta el mas significativo. Si son más de ocho entradas, se utilizan los byte que sea necesario. Si al terminar se tiene un byte que le sobran bits, estos son dejados en cero lógico.

03 Leer registros de retención

Lee el contenido binario de un registro de retención en el esclavo. La opción de difusión no está permitida.

Solicitud: El mensaje especifica el registro de inicio y el número de registros que van a ser leídos.

Respuesta: Los datos es un registros son empaquetados como dos bytes por registro. Para cada registro, el primer byte contiene la parte alta de los datos y el segundo byte la parte baja. El esclavo envía el contador de bytes el cual informa al maestro la cantidad de bytes que componen el área de datos. También envía los bytes con la información de todos los registros leídos.

04 Leer registros de entrada

Lee el contenido binario de un registro de entrada en un esclavo. La difusión no está permitida.

Solicitud: El mensaje especifica el registro de inicio y la cantidad de registro que van a ser leídos.

Respuesta: Los datos es un registros son empaquetados como dos bytes por registro. Para cada registro, el primer byte contiene la parte alta de los datos y el segundo byte la parte baja. El esclavo envía el contador de bytes el cual informa

al maestro la cantidad de bytes que componen el área de datos. También envía los bytes con la información de todos los registros leídos.

05 Forzar una bobina

Fuerza el valor de una bobina a un valor encendido o apagado. Para la difusión, la función indica que se deben forzar la misma bobina en todos los esclavos.

Solicitud: El mensaje especifica la dirección de la bobina a ser forzada. También se un valor de FF 00 hex si se desea forzar a uno. Un valor de 00 00 indicará que se debe forzar a cero. Cualquier otro valor es ilegal y no afectará a la bobina.

Respuesta: Una respuesta normal es un eco de la solicitud que se envía cuando el esclavo a ejecutado la acción.

06 Hacer un Preset en un registro

Le asigna un valor a un registro de retención. Cuado se utiliza la difusión, se le da el mismo valor al mismo registro en todos los esclavos.

Solicitud: El mensaje especifica la dirección del registro a modificar. El valor a modificar en el registro está dado como dos bytes uno para la parte alta y otro para la parte baja de los datos.

Respuesta: Una respuesta normal es un eco enviado una vez realizada la acción.

07 Lee el Status de Excepción (exception estatus)

Algunos controladores tienen un registro de estatus el cual en algunos casos están predefinidos y en otros se pueden programar. Esta opción realiza una solicitud de lectura de esos registros. La difusión no esta permitida.

Solicitud: El mensaje contiene la función correspondiente y la dirección del esclavo. No se necesita información adicional asi que el espacio de datos no se utiliza.

Respuesta: La respuesta contiene los valores binarios de ocho bobinas de status, por lo tanto, la respuesta es un byte donde cada bit es una bobina.

0B Contador de eventos Fetch Comm (Fetch Comm Event Counter)

Esta función devuelve una palabra de estatus y un contador desde el contador de eventos de la comunicación. Al leer el contador antes y después de una serie de mensajes, el maestro puede determina si los mensajes fueron manejados normalmente por le esclavo. La difusión no esta permitida. El contador de eventos es incrementado una vez después de que se completa exitosamente un mensaje.

Solicitud: La solicitud incluye únicamente la dirección y la función ya que no se requiere de ningún dato adicional para que el esclavo ejecute la acción.

Respuesta: Una respuesta normal contiene dos bytes para el status y dos byte para el contador de eventos. Los bytes de status son FF FF hex si una función todavía está siendo procesada por el esclavo, en caso contrario los bytes van a ser ceros.

0C Fetch Comm Event Log

Devuelve una palabra de estatus, un contador de mensaje y un grupo de bytes de eventos del esclavo. Los dos primeros son los mismos enviados por la función explicada anteriormente. El contador de mensajes contiene la cantidad de mensajes procesador por el esclavo desde el último reinicio, limpiado del contador o encendido del sistema. El grupo de bytes de eventos contienen de 0 a 64 bytes, donde cada uno corresponde al status de un Modbus enviado o recibido por el esclavo.

Este grupo trabaja como una pila donde 0 es el evento mas reciente. Al llenarse, el evento mas antiguo es desechado.

Solicitud: La solicitud incluye únicamente la dirección y la función ya que no se requiere de ningún dato adicional para que el esclavo ejecute la acción.

Respuesta: El esclavo envía un contador de bytes indicando la cantidad de bytes que conforman el mensaje. Dos bytes para el status, dos bytes para

contador de eventos, dos bytes para la cantidad de mensajes procesados y un byte para cada uno de los bytes de eventos.

0F Forzar un grupo de bobinas

Fuerza una serie consecutiva de bobinas para ponerlas en encendido o apagado. Cuando se utiliza difusión, se fuerza el mismo grupo de bobinas en cada esclavo.

Solicitud: El mensaje especifica la dirección de la primera bobina a ser forzada, la cantidad de bobinas manipular. Luego se envía un contador de bytes para indicar cuantos componen el grupo que indica los valores de las diferentes bobinas. Finalmente se colocan los bytes necesarios para especificar los valores que deben asumir cada bobina. Cada bit en el byte corresponde al estado al que se esta forzando dicha bobina.

Respuesta: Una respuesta normal devuelve la dirección del esclavo, la dirección de inicio y la cantidad de bobinas forzadas.

10 Preset de varios registros

Le asigna un valor a un grupo de registros en forma consecutiva. Cuando se utiliza difusión se le da el mismo valor al mismo de grupo de registros en todos los esclavos.

Solicitud: El mensaje especifica la dirección del primer registro a manejar, el número de registros a modificar, un contador de bytes y dos bytes por registro, que contiene el valor que va a asumir este.

Respuesta: Una respuesta normal devuelve la dirección del esclavo, el código de función, la dirección de inicio, y la cantidad de registros manipulados.

11 Reporte de la identificación del esclavo

El protocolo Modbus está diseñado para ser manejado por tarjetas de Modicon, está opción le permite saber al maestro información referente a las

características del controlador de un esclavo específico. La difusión no está permitida.

Solicitud: La solicitud incluye únicamente la dirección y la función ya que no se requiere de ningún dato adicional para que el esclavo ejecute la acción.

Respueta: El contenido de la respuesta depende de cada tipo de controlador, sin embargo, típicamente contiene la dirección del esclavo, la función, un contador de bytes, el número de identificación del esclavo, un status que indica si está trabajando o no, y finalmente, datos adicionales inherentes al tipo de controlador.

14 Leer referencias generales

Devuelve el contenido de los registro en la memoria extendida. La difusión no está permitida. La función puede leer varios grupos de referencias, los grupos pueden ser no continuos, pero las referencias dentro de un grupo deben ser continuas.

Solicitud: El mensaje contiene la dirección del esclavo, el código de función, la revisión de error y el grupo o grupos de referencias a leer.

Cada grupo debe contener:

- Un byte con el tipo de referencia.
- Dos Bytes con el número de archivo dentro de la memoria extendida.
- Dos Bytes con el registro de inicio.
- Dos bytes con la cantidad de registros a leer.

Respuesta: Una respuesta normal contiene la información de cada uno de los grupos que fueron leídos. Devuelve un contador de bytes con la cantidad total de información en el mensaje. Luego envía lo que corresponde a cada grupo, el cual contiene un contador de bytes con la cantidad de datos en ese grupo y dos bytes para cada uno de los registros leídos.

15 Escribir referencia general

Escribe en el contenido de los registros en la memoria extendida. La difusión no está permitida. Esta función puede escribir varios grupos de referencias. Los grupos pueden ser no consecutivos pero los registros dentro de un mismo grupo deben ser continuos.

Solicitud: El mensaje contiene la dirección del esclavo, el código de función, la revisión de error y el grupo o grupos de referencias a manipular.

Cada grupo debe contener:

- Un byte con el tipo de referencia.
- Dos Bytes con el número de archivo dentro de la memoria extendida.
- Dos Bytes con el registro de inicio.
- Dos bytes con la cantidad de registros a modificar.
- Dos bytes con la información que se debe escribir en el registro.

Respuesta: Una respuesta normal es un eco de la solicitud.

16 Enmascarar un registro de escritura

Modifica el contenido de un registro específico usando una combinación de mascaras AND y mascaras OR y el contenido actual del registro. Esta función puede ser utilizada para activar o desactivar bits individuales dentro del registro. La difusión no esta permitida.

Solicitud: El mensaje especifica la dirección del registro a ser manipulado, los datos a ser usados como la mascara AND y los datos a ser usados como la mascara OR. El algoritmo que se utiliza es le siguiente:

Resultado=(contenido actual & mascara AND) # (mascara OR & mascara AND)

Donde

& es la función AND

es la función OR

Respuesta: Una respuesta normal es un eco de la solicitud.

17 Leer / escribir registros

Realiza una combinación de una lectura y una escritura en una sola transacción. Se pueden leer un grupo de registros y luego escribir otro grupo. La difusión no está permitida.

Solicitud: La solicitud especifica la dirección inicial, la cantidad de registros del grupo que se va a leer. Luego da la dirección inicial, el numero de registros a escribir y la información dentro del grupo de escritura. El contador de bytes indica la cantidad de bytes que se relacionan con el proceso de escritura.

Respuesta: Una respuesta normal contiene la información correspondiente a los registros que fueron leídos. El contador de bytes especifica la cantidad de bytes que componen la información leída.

24 Leer la línea FIFO

Lee el contenido de los registros FIFO *(First in First Out)*. La función devuelve un conteo de los registros en la fila, seguido por los datos. La difusión no esta permitida.

Solicitud: El mensaje especifica la dirección del primer registro a leer.

Respuesta: Una respuesta normal se tiene un contador de bytes que indica la cantidad de bytes que siguen y los bytes que componen los registros leídos.

RESPUESTA DE EXCEPCIÓN

Excepto a la comunicación difundida, cuando un maestro envía una solicitud a un esclavo este espera una respuesta del segundo. Puede ocurrir alguno de los siguientes cuatro eventos:

- El esclavo recibe la solicitud sin error de comunicación y puede manejar la solicitud normalmente, este devuelve una respuesta normal.

- El esclavo no recibe la solicitud debido a un error de comunicación, no se envía respuesta. El maestro debe, entonces, procesar una condición de tiempo de espera excedido para dicha solicitud.
- El esclavo recibe la solicitud y detecta errores (paridad, LRC,CRC), no se envía respuesta. El maestro debe, entonces, procesar una condición de tiempo de espera excedido para dicha solicitud.
- El esclavo recibe la solicitud sin error de comunicación pero no puede manejarla, entonces, este retorna una respuesta de excepción informando el tipo de error suscitado.

Campo del código de función

Bajo una respuesta normal, el esclavo realiza un eco del código de función de la solicitud original. Todos los códigos tienen el bit más significativo en cero lógico. En una respuesta de excepción, el esclavo pone el MSB en uno lógico. De manera que le maestro pueda identificar la existencia de un error.

Campos de datos

En una respuesta normal, el esclavo retorna datos o estadísticas en el espacio para datos. En una respuesta de excepción, el esclavo devuelve un código en el espacio de datos. Este define la condición que ha causado un error. La tabla A2.3 muestra los diferentes código de excepción que se generan cuando se da un error.

Tabla A2.3 Código de excepción y su explicación

Código	Nombre	Descripción
01	Función ilegal	El código de función recibido no es un código válido para el esclavo.
02	Dirección de datos ilegal	La dirección recibida en la solicitud no es válida para el esclavo.
03	Valores de datos ilegales	Un valor en el campo de datos dentro de la solicitud no es válido para el esclavo.
04	Falla en el dispositivo esclavo	Un error irrecuperable sucedió mientras el esclavo trataba de realizar la acción en la solicitud
05	Trabajando	Una solicitud fue recibida y esta siendo procesada, pero se requiere de un período largo de tiempo para finalizarla, por lo tanto se envía la advertencia para prevenir un error de tiempo de espera finalizado.
06	Esclavo ocupado	El esclavo está procesando un comando de larga duración. El maestro debería retransmitir el mensaje luego cuando el esclavo este libre.
08	Error de paridad en memoria	El esclavo está tratando de leer la memoria extendida pero ha detectado un error de paridad en la memoria.

GENERACIÓN DE LRC

El chequeo redundante longitudinal es un byte que contiene un valor de ocho bits. El valor del LRC es calculado por el dispositivo transmisor el cual lo anexa al mensaje. El dispositivo receptor recalcula un valor LRC durante la recepción del mensaje y lo comprara con el recibido. Si son diferentes un error ha ocurrido.

El LRC es calculado sumando sucesivamente ocho bits en el mensaje, eliminando cualquier acarreo, para luego complementar a dos.

Generación del LRC

Sume todos los bits en el mensaje, excluyendo el byte inicial y los dos últimos.

Apéndice A.3 Nota de aplicación: Telemetría La Cruz de Guanacaste

A3.1 Introducción:

La siguiente aplicación se realizó utilizando el programa base desarrollado en este trabajo. Se trata de una telemetría contratada por la empresa A y A y consiste en el manejo de 5 estaciones a saber:

Tanque 1: Este tanque se encarga de almacenar el agua que va a llegar al pueblo. Cuenta con un sensor de caudal que indica el nivel del tanque. La figura A3.1 muestra el gabinete de control donde se encuentra el plc, todas las estaciones cuentan con un sistema parecido.

Figura A3.1 Foto del gabinete de control del tanque 1

Pozo 3 y 4: Se considera una sola estación porque ambos son controlados por un único plc. Se encargan de llenar el Tanque 1 cuando este llega a un nivel bajo.

Pozo 1: Es un pozo de respaldo en caso de que el pozo 3 o el 4 falle, en ese caso este pozo se activa y ayuda a llenar el tanque 1 cuando existe un nivel bajo.

Rebombeo: Esta estación cuenta con dos bombas, la principal y una de respaldo en caso de falla. Se encarga de tomar el agua del Tanque 1 y enviarla al tanque elevado.

Tanque elevado: Es el destino final, a partir de aquí se distribuye el agua a todo el pueblo. Tiene un sensor de caudal el cual determina el nivel del tanque y en caso de estar bajo, envía una orden de encendido al rebombeo. Como ilustración, la figura A3.1 muestra una foto del tanque elevado en la cual se muestra el tanque asi como la estación de control (al pie del tanque)

Figura A3.2 Foto del tanque elevado en el proyecto la Cruz de Guanacaste

En esta aplicación se va a explicar lo que se hizo para tomar el programa ya creado y modificarlo, la idea es notar lo sencillo que es ampliar el mismo para una cantidad dada de esclavos. Lo que corresponde a la comunicación se va a omitir en gran parte pues ya fue explicado en el desarrollo de este trabajo.

Como ya se comento, existen cinco estaciones lo que equivale a un maestro y cuatro esclavos. Por la localización del tanque elevado, este se seleccionó como el maestro y el resto de las estaciones son esclavos. Cada una de las estaciones se explican a continuación.

A3.2 Tanque elevado:

Como ya se mencionó este es el maestro del sistema. En cuanto a modificaciones del programa desarrollado, el maestro es el que se debe trabajar más pues se le debe indicar cuántos esclavos va a manejar y se le deben agregar las respectivas solicitudes a cada uno.

Se cuentan con las rutinas desarrolladas y se muestran las variaciones a una rutina o las rutinas que fueron agregadas. Para el caso de las rutinas modificadas se muestra sólo la variación realizada por las razones ya explicadas.

A3.2.1 Programa principal:

En este caso se incluye el llamado a las rutinas que se encargan de atender las solicitudes de cada uno de los esclavos. El valor de *esclavo* determina cual solicitud se esta haciendo, para el control del sistema se utilizan 5 solicitudes. La tabla A3.1 muestra el valor del *esclavo* con la explicación de la solicitud que se realiza.

Tabla A3.1 Valor de la variable *esclavo* y la solicitud que se realiza

Valor de *esclavo*	Solicitud que se realiza
1	Escribe el nivel (alto o bajo) del tanque elevado en el rebombeo
2	Solicita el nivel analógico al tanque 1
3	Escribe el nivel (alto o bajo) del tanque 1 en el pozo 3 y 4
4	Solicita el estado de las fallas al pozo 3 y 4
5	Escribe el nivel (alto o bajo) del tanque 1 asi como las fallas en el pozo 3 y 4 en el pozo 1

Una solicitud es repetida una vez más en caso de no obtenerse respuesta, para esto se utiliza la lógica que se muestra en la figura 9. La figura A3.3 muestra la lógica para la rutina principal.

A3.2.2 Subrutinas 1, 4, 6, 7 y 8: Generación de las solicitudes

Estas subrutinas son iguales a las desarrolladas en la subrutina 1 del programa base (figura 6.3), lo que varía es el contenido del paquete. La tabla A3.2 muestra cada uno de los paquetes para las diferentes solicitudes.

Tabla A3.2 Valor de la variable esclavo y el paquete de solicitud enviado

Valor de esclavo	Dirección de esclavo	Código de función	Información adicional
1	7C	10	Escribir a partir de la direc 190 un byte
2	C8	3	Leer a partir de la direc 190 dos bytes
3	E0	10	Escribir a partir de la direc 190 un byte
4	E0	3	Leer a partir de la direc 190 un byte
5	64	10	Escribir a partir de la direc 190 un byte

Para la rutina 6, en la cual se envía el nivel del tanque 1 al pozo 3 y 4, se debe hacer la comparación del valor analógico y convertirlo a un valor digital. Para esto se utiliza la variable VB404, en la cual si el nivel esta en alto se activa el bit V404.1 y si esta en bajo se activa el bit V404.0. Luego se hace la conversión a ASCII para hacer el respectivo envío del paquete. La figura A3.4 muestra el diagrama de flujo para esta subrutina.

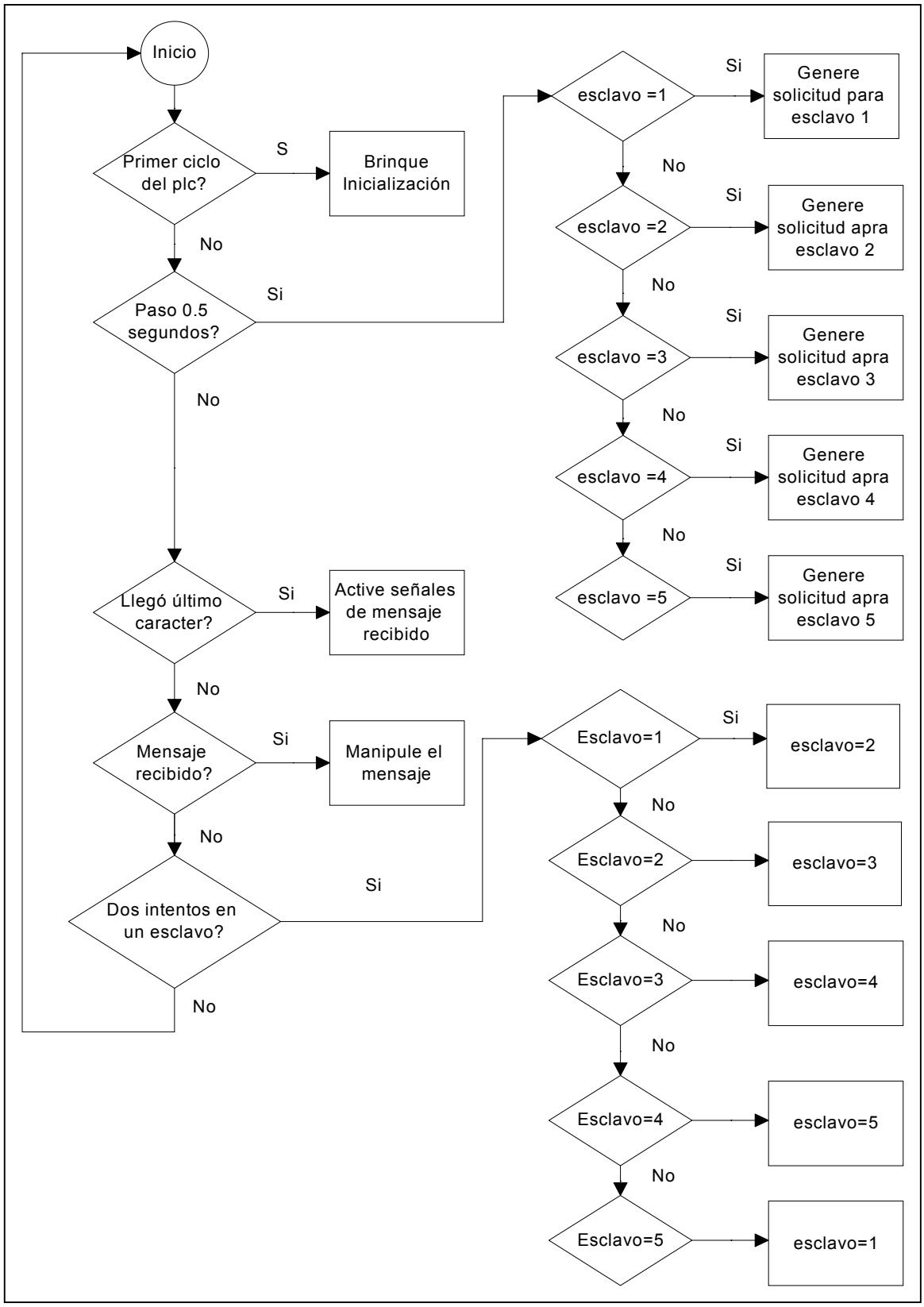

Figura A3.3 Diagrama de flujo para el programa principal

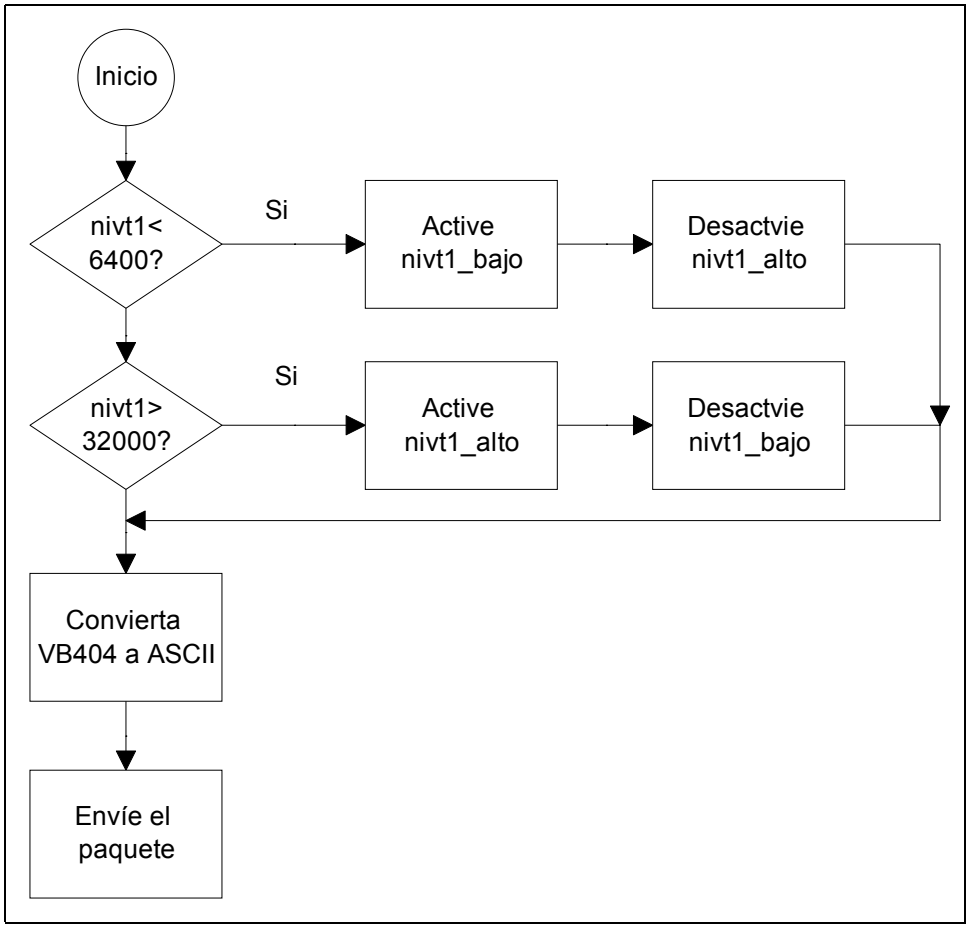

Figura A3.4 Diagrama de flujo para la subrutina 6: Escribir valor en pozo 3 y 4

De igual forma se hace con la subrutina 8, la cual escribe el valor del tanque 1 en el pozo 1. Sin embargo, también se le debe enviar el estado de las fallas en los pozos 3 y 4. Lo que se envía es un nivel digital, de manera que si existe alarma se activa un bit, si no hay falla se desactiva dicho bit. Estos valores se guardan en la variable VB56 que es la que se va a enviar en el paquete.

Se toma la variable VB405, que contiene las fallas recibidas de los pozos 3 y 4, junto con la variable VB404 (nivel del tanque 1) y se compara. El valor VB56 va a tener tres bits funcionales. El menos significativo indica nivel de tanque bajo, el siguiente el estado alto del tanque 1 y el tercero si existe alguna falla en alguno de lo pozos. Estos valores se convierten a ASCII y se envía el paquete.

A3.2.3 Subrutinas 5: Leer nivel del tanque:

Esta estación además de ser el maestro, tiene que controlar el tanque elevado y enviar el estado del nivel, alto o bajo, al rebombeo. Esta subrutina es llamada en cada ciclo del plc y se encarga de mover el valor analógico de la palabra AIW0 a la dirección VW200. Luego compara esta localidad para determinar el estado del tanque.

Finalmente la convierte a ASCII para que sea enviada cuando el valor de *esclavo* sea 1. La figura A3.5 muestra el diagrama de flujo para esta subrutina.

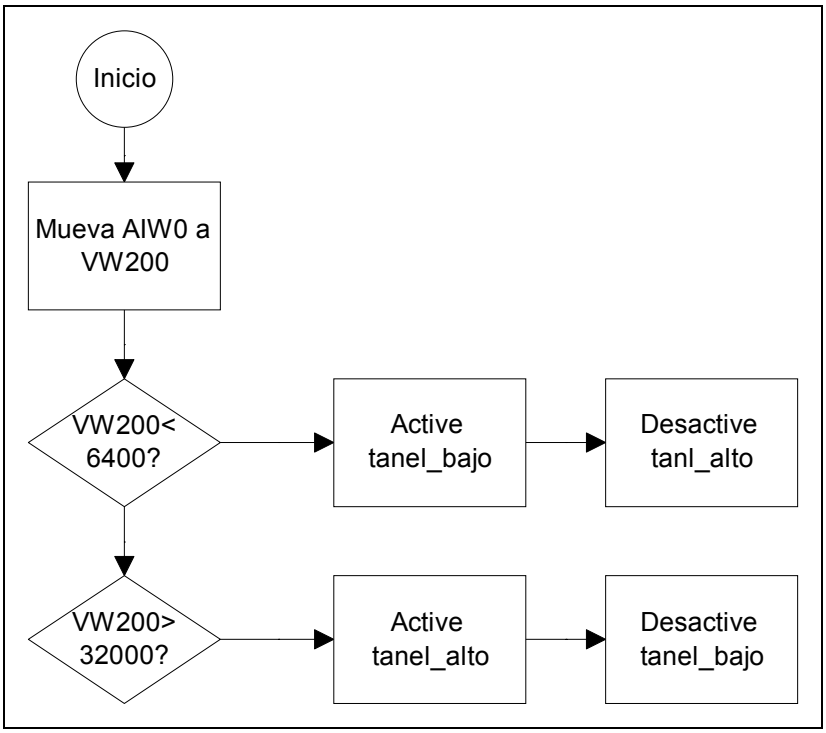

Figura A3.5 Diagrama de flujo para la subrutina 5: Leer valor del tanque elevado

A3.3 Tanque 1:

Como se mencionó, el maestro es el que sufre más modificaciones pues un esclavo cuenta con las subrutinas suficientes para interpretar cualquier solicitud. Entonces, lo que se debe modificar en cada esclavo es lo que corresponde a la lógica de operación de cada sistema específico. En este caso, el tanque toma el valor del sensor de nivel y lo guarda en la localidad VW400, ya que esta es la

dirección que el maestro le va a solicitar. El maestro solicita dos bytes a partir de la dirección VB400 lo cual forma la palabra que consiste en el valor analógico.

La figura A3.6 muestra la lógica de la subrutina 7 de este programa, el cual consiste en la lógica de operación. Esta es una operación sencilla, la cual consiste en transferir la palabra AIW0 (entrada analógica a la cual esta conectado el sensor) a la dirección VW404, esta subrutina es consultada en cada ciclo del plc para tener permanentemente actualizados.

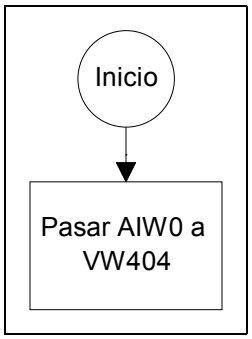

Figura A3.6 Diagrama de flujo para la subrutina 7: Leer nivel del sensor

A3.4 Rebombeo:

Esta estación llena el tanque elevado cuando este le envía un nivel de bajo y se detiene cuando recibe un nivel de alto. Tiene dos bombas, una es de respaldo de manera que cuando se da una falla o si se encuentra en manual, la segunda bomba entra a funcionar siempre y cuando no haya falla o que no esté en manual. También tiene unos sensores de nivel que cuidan que la presión de succión y descarga no sea muy baja (tuberías sin agua) o muy alta (obstrucción en alguna tubería), si esto se da, el sistema detiene las bombas espera un tiempo intenta un arranque y si el problema persiste, detiene todo y da una falla.

La subrutina 7 se encarga de manejar la lógica de operación de la estación. La subrutina es llamada en cada ciclo del plc.

Lo primero que hace es verificar los niveles de los sensores de presión, si están bien, elimina cualquier error en presión que exista. Luego verifica las

entradas I0.7 e I1.0 si están activas, pone en uno las variables *Auto_B1* y *Auto_B2* respectivamente. Esto indica si las bombas están en manual o en automático.

Si es la primera vez que se recorre al rutina, borra cualquier falla. Existe la variable *intpres* que indica cuantos intentos de arrancar la bomba se han hecho después de determinar un error en la presión. Esta variable se pone en cero al iniciar la rutina.

Para borrar un estado de falla se debe colocar el estado de la bomba en manual. Al hacerse esto se borra la falla de la respectiva bomba y se pone *intpres* en cero.

Una vez que se recibe un nivel bajo se verifican las siguientes condiciones, si todas existen se enciende la bomba 1:

- Llego nivel bajo
- La bomba 2 esta apagada
- No hay error en presión (*epresion*=0)
- La bomba 1 esta en automático (*Auto_B1*=1)
- No hay falla en la bomba 1 (*FallaB1=0)*
- El nivel del pozo no esta bajo (*pozo_bajo*=0)

Estas condiciones encienden la bomba 1, para enclavar el sistema se utilizan las condiciones de que este la bomba 1 activa y no hay nivel alto, de manera que cuando se llene el tanque se desactiva la bomba 1.

La bomba 2 es de respaldo, por lo tanto se debe encender si se dan las condiciones anteriores y si la bomba 1 esta apagada. Si se da una falla, la bomba 1 no va a arrancar y se va a encender la bomba 2. Si se restaura la bomba 1, en el próximo ciclo de llenado se van a dar las condiciones para encendido de la bomba 1 y esta va a arrancar anulando el encendido de la bomba 2.

Cuando una bomba se enciende se debe verificar que no hay ninguna falla, para esto se utiliza el contactor principal del sistema. Si este no esta activo se considera que no hay confirmación de encendido y que existe una falla. En este caso, la bomba (1 o 2) enciende y se da un tiempo (10 segundos), después de este se revisa el estado de la confirmación, la cual esta conectada a una entrada del plc, si no hay confirmación se activa la señal de *FallaB1* y *FallaB2* para la bomba 1 y la bomba 2, respectivamente.

Si se detecta una falla se activa la salida Q0.2, la cual se pone a parpadear indicando un problema.

Cuando se enciende una bomba, la presión en la tubería se vuelve inestable por un tiempo, por lo tanto se debe dar un tiempo para revisar la presión ya que si se hace al inicio se va a dar un error en presión. Este tiempo es de 3 minutos, una vez finalizado se revisa el valor de los sensores y si se detecta un error de presión se activa la variable *epresion* y se incrementa la variable *intpres*. Esto detiene la bomba en trabajo, luego se da un tiempo esperando si el sistema se estabiliza (3 minutos), al darse el tiempo se vuelve a revisar el estado de la tuberías, si hay error se incrementa *intpres*.

Cuando *intpres* tiene un valor de 2 se activa la falla 1 en la bomba que se esta tratando de arrancar. La figura A3.7 muestra la lógica de operación para esta subrutina. Se muestra una bomba como A, esto quiere decir que las mismas instrucciones existen para ambas bombas. Cuando se escoge la bomba 1 como la bomba A, entonces la bomba 2 es la B y viceversa, esto se hace para simplificar el diagrama de flujo.

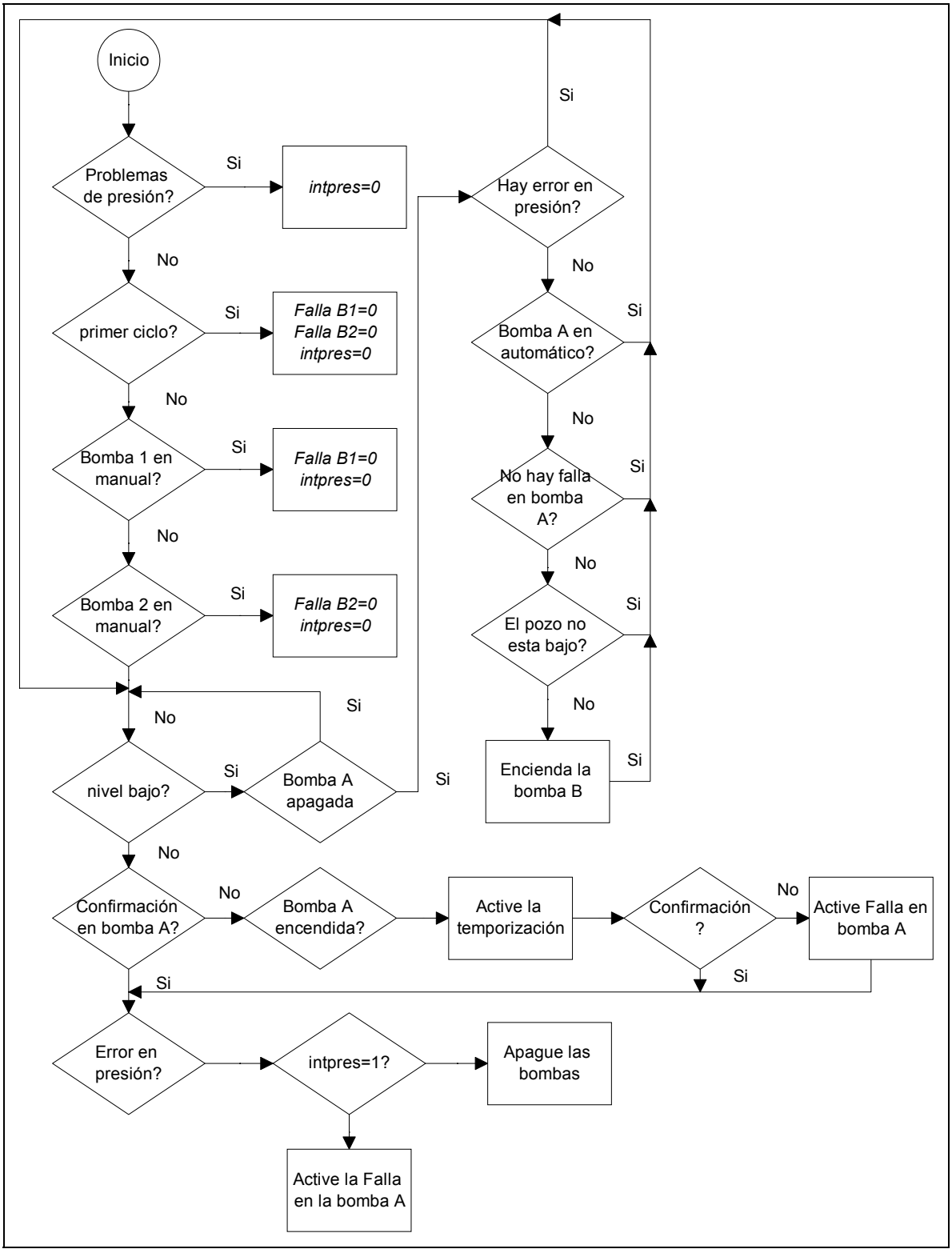

Figura A3.7 Diagrama de flujo para la lógica de operación del rebombeo

A3.5 Pozo 3 y 4:

Estos pozos son una misma estación y deben trabajar ambas bombas. Primero arranca el pozo 3 y luego el pozo 4, a menos que exista falla o alguno este en manual. Al igual que para el rebombeo, se incluye la subrutina en el programa base para manejar lo concerniente a la lógica de operación.

La figura A3.8 muestra la lógica de operación para esta rutina. Primero se revisan la entradas I0.7 e I0.4 para activar *Auto_B1* y *Auto_B2*, respectivamente. Si es el primer ciclo se borran las fallas.

Cuando se recibe un nivel bajo, se revisan las siguientes variables para encender la bomba 1:

- La bomba 1 esta en automático
- No hay Falla en la bomba 1
- El nivel del pozo no esta bajo

Una vez encendida la bomba se realiza un enclave con las siguiente dos condiciones:

- Bomba 1 activa
- No hay nivel alto

Al encenderse la bomba 1 se activa una temporización para encender la bomba 2, una vez pasada esta temporización se revisan las siguientes condiciones:

- Bomba 2 en automático
- No hay falla en bomba 2
- No hay nivel bajo en el pozo

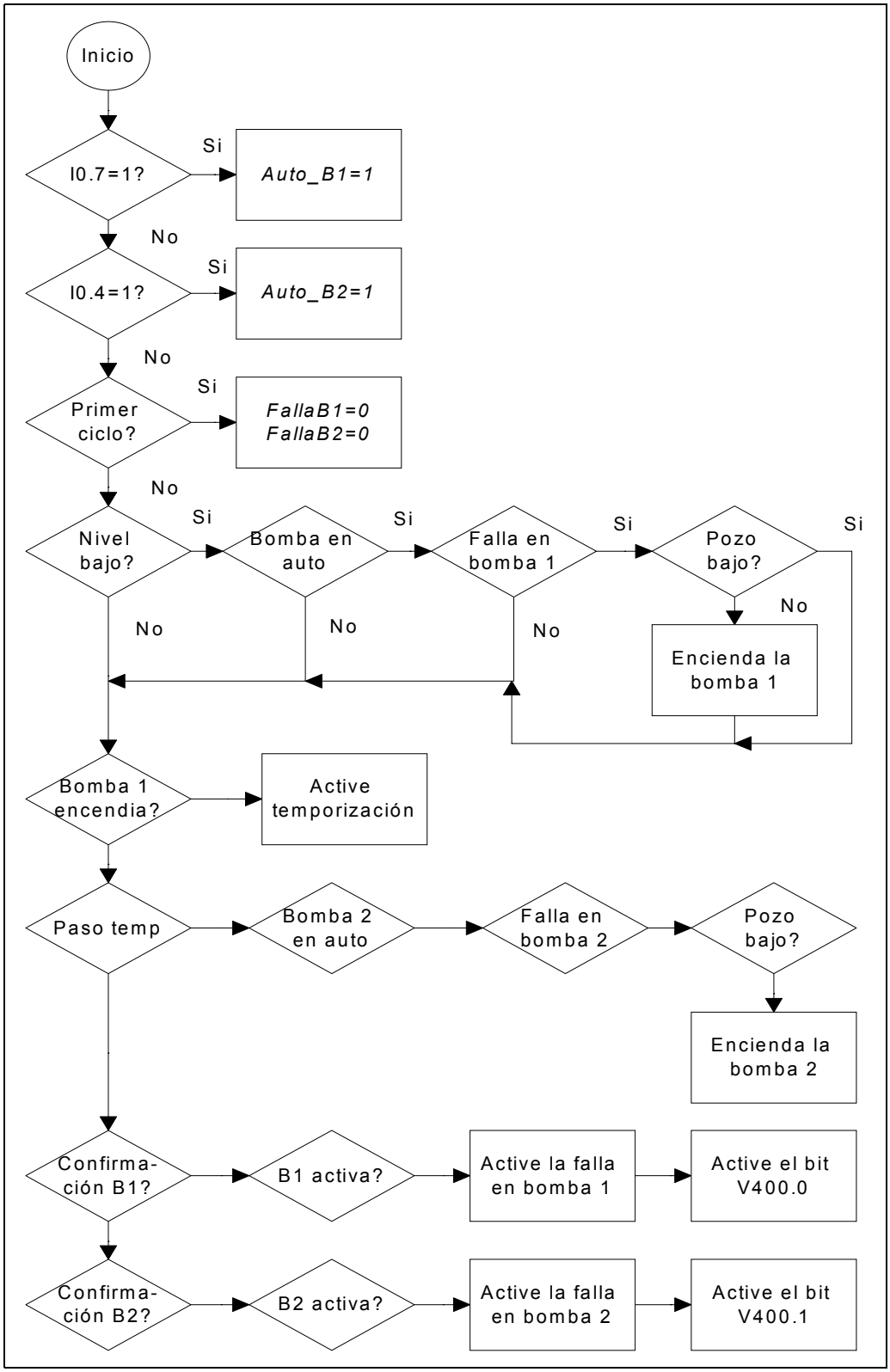

Figura A3.8 Diagrama de flujo para la lógica de operación de los pozos 3 y 4

Al igual que la bomba 1, se realiza un enclave el cual se da mientras se den las condiciones:

- Bomba 2 activa
- No hay nivel alto

Si no se recibe confirmación de alguna de las bombas y están activas, se da un temporización. Una vez finalizado este tiempo y si todavía no hay confirmación se activa la señal de alarma indicando falla en la respectiva bomba. Además si la falla es en la bomba 1 se activa el bit V400.0 y si la fallas es en la bomba 2, se activa el bit V400.1; VB400 es el byte que va a ser leído por el maestro para tener el estado de fallas y su posterior envío al pozo 1.

A3.5 Pozo 1:

La última estación es el pozo 1. Este es un sistema de respaldo el cual se enciende cuando existe una falla en el pozo 3 o el 4. Además, cuando trabaja, se activa según el nivel del tanque 1. Por lo tanto todas estas condiciones son suministradas por el maestro en una petición de escritura en la dirección VB200.

Primero se revisa el estado de la entrada I0.4 para determinar si la bomba esta en automático o manual. Si es el primer ciclo se quita cualquier falla existente.

Si el bit V200.0 (nivel del tanque 1 bajo) se activa una temporización. Una vez finalizado este tiempo se revisan las siguientes condiciones para el arranque de la bomba:

- V200.2 activo (existe una falla en pozo 3 o pozo 4)
- La bomba esta en automático
- No hay falla en la bomba

- El nivel del pozo no esta bajo

La bomba se enclava con las siguientes condiciones:

- Bomba 1 activa
- No hay nivel alto en el tanque

Después se revisa la confirmación de la bomba, si no hay y si la bomba esta encendida se activa una temporización. Si transcurre el tiempo y todavía no hay confirmación, se activa la falla en la bomba. La figura A3.9 muestra el diagrama de flujo para esta rutina.

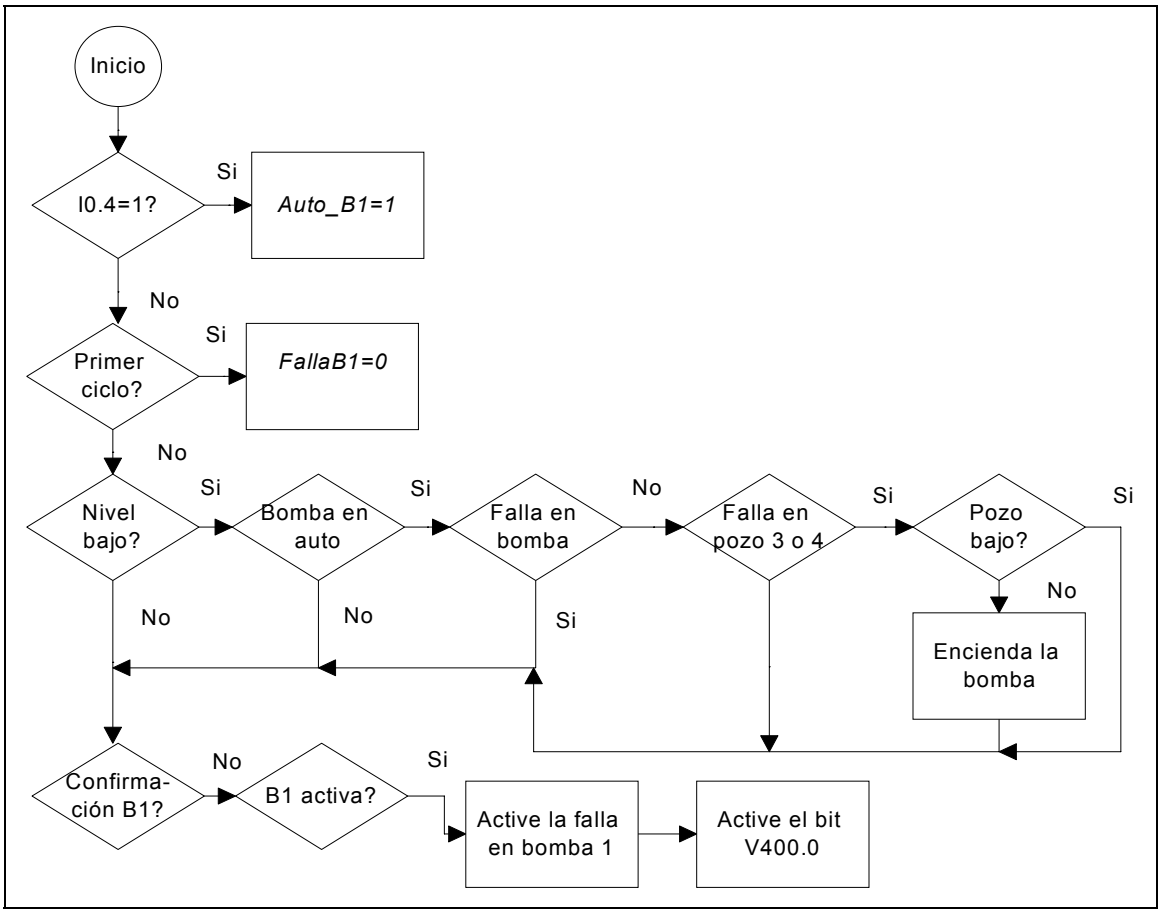

Figura A3.9 Diagrama de flujo para la lógica de operación del pozo 1