PROCESADOR DE COMUNICACIÓN MODBUS. IMPLEMENTACIÓN CON MICROCONTROLADOR

ING. JORGE ELIÉCER DUQUE PARDO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
MAESTRÍA EN INGENIERÍA
AREA DE ELECTRÓNICA
2006

PROCESADOR DE COMUNICACIÓN MODBUS. IMPLEMENTACIÓN CON MICROCONTROLADOR

Trabajo de investigación presentado como requisito
para optar al título de Magíster en Ingeniería Electrónica

ING. JORGE ELIÉCER DUQUE PARDO

DIRECTOR:
MSC JULIO AUGUSTO GELVEZ FIGUEREDO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
MAESTRÍA EN INGENIERÍA
AREA DE ELECTRÓNICA
2006

AGRADECIMIENTOS

Muchas personas contribuyeron al desarrollo de este trabajo de investigación. Primero quiero agradecer a los excelentes profesores y amigos, de quienes tuve el honor de recibir y compartir valiosas enseñanzas y consejos: Msc. Julio Augusto Gélvez Figueredo, director de la tesis, Msc. Jaime Barrero Pérez, profesor del Seminario de Arquitectura de Procesadores y Msc. Jorge Meneses, profesor del Seminario de Redes Industriales.

Por otra parte, agradezco la contribución de los ingenieros Pedro Arcila y Sinle Marcela Carreño quienes implementaron el Maestro Modbus que permitió comprobar el funcionamiento del Procesador de comunicación Modbus. Así mismo agradezco al estudiante de ingeniería Electrónica Roger Torres por su valioso aporte en la implementación de algunas rutinas Modbus RTU.

Por último, agradezco el apoyo económico brindado por la Universidad Tecnológica de Bolívar durante los dos años de realización de la Maestría. También agradezco, con todo mi corazón el amor, la paciencia y el apoyo de mi querida esposa, Dercy, y de mi hijo Christian, así como de mi querida madre, Noria, quien me soportó y consintió durante estos dos años.

Jorge Eliécer Duque Pardo

Vale más arriesgarse que nunca intentar

RESUMEN

TÍTULO: PROCESADOR DE COMUNICACIÓN MODBUS. IMPLEMENTACIÓN CON MICROCONTROLADOR*

AUTOR: DUQUE PARDO, JORGE ELIECER**

PALABRAS CLAVES: Modbus, Redes industriales, Microcontroladores.

DESCRIPCIÓN:

Este trabajo de investigación comprende el diseño y construcción de un procesador de comunicaciones que utiliza el protocolo Modbus para comunicación con diferentes dispositivos de control de procesos o instrumentación tales como PLCs, medidores de energía, transmisores y actuadores. Se utilizó un microcontrolador HCS12 de Motorola para la implementación del protocolo Modbus y para la realización de las operaciones de adquisición, conversión analógica –digital y comunicación serial.

Las características funcionales establecidas para el Procesador de Comunicación fueron:

- Capacidad de manejo de las interfaces RS-232 y RS-485
- Posibilidad de trabajar en cualquiera de los modos de transmisión Modbus ASCII o RTU.
- Implementación de 15 funciones Modbus para el acceso y manipulación de datos, tanto a nivel de bits como a nivel de registros de 16 bits, y funciones de diagnóstico y configuración.
- Capacidad para manejar hasta 2024 bits y 250 registros de 16 bits.
- Posibilidad de implementar aplicaciones en el sistema embebido, aparte de las funciones específicas de comunicación.
- Portabilidad del firmware para la implementación en otros sistemas embebidos.
- Posibilidad de configuración por hardware o software de los parámetros de comunicación: rata de baudios, paridad y tipo de interfaz.

La arquitectura abierta del protocolo Modbus permitió su implementación en diferentes plataformas de software y de hardware. Esto se comprobó al implementar el procesador Modbus esclavo en un microcontrolador de 16 bits y el maestro en un PC utilizando el software LabView. También se pudo comprobar la interoperabilidad con diferentes fabricantes de software y hardware Modbus, al conectar el procesador Modbus a una red industrial RS-485.

---

* Trabajo de Grado.

** Facultad de Ingenierías Físico-mecánicas. Maestría en Ingeniería Área de Electrónica. Julio Augusto Gélvez Figueredo.

ABSTRACT

TITLE: MODBUS COMMUNICATION PROCESSOR. IMPLEMENTATION WITH MICROCONTROLLER *

AUTHORS: DUQUE PARDO, JORGE ELIECER**

KEY WORDS: Modbus, Industrial networks, Microcontrollers.

DESCRIPTION:

This investigation work incorporates the design and construction of a communications processor that it uses the protocol Modbus for communication with different devices of control of processes or instrumentation such as PLCs, energy meters, transmitters and actuators. A microcontroller HCS12 of Motorola was used for the implementation of the protocol Modbus and for the realization of the operations of acquisition, analogical-digital conversion and serial communication.

The established functional characteristics for the Processor of Communication were:

- Capacity of handling of the interfaces RS-232 and RS-485
- Possibility to work in anyone in the transmission ways Modbus ASCII or RTU.
- Implementation of 15 functions Modbus for the access and manipulation of data, so much at bits level as at level of 16 bits registers, and diagnostic and configuration functions.
- Capacity to manage until 2024 bits and 250 registers of 16 bits.
- Possibility to implement applications in the embedded system, apart from the specific functions of communication.
- Portability of the firmware for the implementation in other embedded systems.
- Configuration possibility for hardware or software of the communication parameters: rat of bauds, parity and interface type.

The open architecture of the protocol Modbus allowed its implementation in different software platforms and of hardware. This was proven when implementing the processor Modbus slave in a microcontroller of 16 bits and the master in a PC using the software LabView. It could also be proven the interoperability with different software and hardware Modbus makers, when connecting the processor Modbus to an industrial RS-485 network.

---

* Work of Grade.

** Faculty of Engineering Physical-mechanics. Master in Electronic Engineering. Julio Augusto Gélvez Figueredo.

CONTENIDO

Capítulo 1 1
INTRODUCCION 1
Capítulo 2 6
EL PROTOCOLO MODBUS 6
2.1 INTRODUCCIÓN 6
2.2 MEDIO FÍSICO 8
2.3 ACCESO AL MEDIO 8
2.4 MODOS DE TRANSMISIÓN 9
2.4.1 MODO ASCII 10
2.4.2 MODO RTU 10
2.5 FORMATOS DE LA TRAMA MODBUS 11
2.5.1 MODO ASCII 11
2.5.2 MODO RTU 12
2.6 FUNCIONES Y CÓDIGOS 14
CÓDIGOS DE FUNCIÓN PÚBLICOS 14
CÓDIGOS DE FUNCIÓN DEFINIDOS POR EL USUARIO 15
CÓDIGOS DE FUNCIÓN RESERVADOS 15
2.7 TRANSACCIONES EN MODBUS 16
Capítulo 3 18
FUNCIONES MODBUS 18
3.1 INTRODUCCION 18
3.2 IMPLEMENTACION 22
3.3 EJEMPLO DE IMPLEMENTACION: FUNCIÓN 01 (READ COILS) 24
Capítulo 4 29
EL MICROCONTROLADOR HCS12 29
4.1 MAPA DE MEMORIA 30
4.2 INTERFAZ SERIAL ASÍNCRONA (SCI) 32
4.3 TEMPORIZADOR (TIM) 33
4.4 MANEJO DE INTERRUPCIONES 35

4.5 CONVERTIDOR ADC 39
4.6 CONVERTIDOR DAC 40
4.7 PUERTOS DE PROPÓSITO GENERAL (GPIO) 41
4.8 GRABACION DE DATOS EN MEMORIA FLASH 45
4.8.1 SECUENCIAS DE COMANDOS PARA LA FLASH 46
4.8.2 FUNCIONES PARA BORRAR Y GRABAR EN FLASH 47
Capítulo 5 51
PROCESADOR MODBUS 51
5.1 FUNCIONES DE CONFIGURACIÓN 53
5.2 COMUNICACIÓN SERIAL ASÍNCRONA 55
5.2.1 GENERALIDADES 55
5.2.2 INTERRUPCIONES EN LA SCI 56
5.2.3 TRANSMISIÓN Y RECEPCION DE DATOS 57
5.2.4 FUNCIONES DE TRANSMISIÓN Y RECEPCIÓN 60
5.3 FUNCIONES PARA EL CONTROL DE ERRORES 62
5.3.1 CHEQUEO DE REDUNDANCIA CÍCLICA (CRC) 62
5.3.2 CHEQUEO DE REDUNDANCIA LONGITUDINAL (LRC) 66
5.4 FUNCIONES DE PROCESAMIENTO DE LAS TRAMAS MODBUS 67
5.4.1 MODBUS RTU 67
5.4.2 MODBUS ASCII 71
Capítulo 6 74
APLICACION 74
Capítulo 7 83
CONCLUSIONES 83
7.1 CONCEPCIÓN DEL SOFTWARE Y DEL HARDWARE 84
7.2 INTERPRETACIÓN DEL DOCUMENTO DE REFERENCIA 88
7.3 PRINCIPALES CARACTERÍSTICAS DEL PROCESADOR 89
7. 4 FUNCIONES MODBUS DE USUARIO 91
7.5 PRODUCTOS DE LA INVESTIGACIÓN 91
7.7 FUTUROS DESARROLLOS 92
BIBLIOGRAFÍA 93

ANEXO I 97
IMPLEMENTACION DE FUNCIONES MODBUS 97
1.1. FUNCIÓN 02 (0x02): READ DISCRETE INPUTS 98
1.2. FUNCIÓN 03 : READ HOLDING REGISTERS 101
1.3. FUNCION 04 (0x04): READ INPUT REGISTERS 104
1.4. FUNCIÓN 05 (0x05): WRITE SINGLE COIL 107
1.5. FUNCIÓN 06 (0x06): WRITE SINGLE REGISTER 110
1.6. FUNCIÓN 07 (0x07): READ EXCEPTIONS STATUS 113
1.7. FUNCIÓN 08 (0x08): DIAGNOSTICS 115
1.8. FUNCIÓN 11 (0x0B): GET COMM EVENT COUNTER 123
1.9. FUNCIÓN 12 (0x0C): GET COMM EVENT LOG 126
1.10. FUNCIÓN 15 (0x0F): WRITE MULTIPLE COILS 132
1.11. FUNCIÓN 17 (0x11): REPORT SLAVE ID 135
1.12. FUNCION 22 (0x16): MASK WRITE REGISTER 138
1.13. FUNCION 23 (0X17): READ/WRITE MULTIPLE REGISTERS 141
1.14. FUNCION 100 (0X64): USER FUNCTION 144
ANEXO II 147
PROGRAMACION DE LOS REGISTROS DEL HCS1 147
2.1 REGISTROS DE LA SCI 148
2.1.1 REGISTROS DE TASA DE BAUDIOS (SCIBDH, SCIBDL) 148
2.1.2 REGISTRO DE CONTROL 1 (SCICR1) 149
2.1.3 REGISTRO DE CONTROL 2 (SCICR2) 150
2.1.4 REGISTRO DE ESTADOS 1 (SCISR1) 151
2.1.5 REGISTRO DE DATOS (SCIDRH, SCIDRL) 153
2.2 REGISTROS DEL TEMPORIZADOR (TIM) 155
2.2.1 SELECCIÓN DE ENTRADA/ SALIDA (TIOS) 155
2.2.2 REGISTRO DE CONTROL DEL TEMPORIZADOR 1 (TSCR1) 155
2.2.3 REGISTRO DE CONTROL DEL TEMPORIZADOR 2 (TSCR2) 156
2.2.4 REGISTRO DE HABILITACION DE INTERRUPCIONES (TIE) 158
2.2.5 REGISTRO PRINCIPAL DE BANDERAS (TFLG1) 158
2.2.6 REGISTRO DE CONTEO DEL TEMPORIZADOR (TCNT) 159

2.3 REGISTROS DEL ADC 159
2.3.1 REGISTRO DE CONTROL 2(ATDCTL2) 159
2.3.2 REGISTRO DE CONTROL DEL ATD 3 (ATDCTL3) 160
2.3.3 REGISTRO DE CONTROL DEL ATD 4 (ATDCTL4) 161
2.4 REGISTROS DEL DAC 163
2.4.1 REGISTRO DE CONTROL DEL DAC 0 (DACC0) 163
2.5 REGISTROS DEL BLOQUE DE MEMORIA 164
2.5.1 REGISTRO DIVISOR DEL RELOJ DE LA FLASH (FCLKDIV). 164
2.5.2 REGISTRO DE ESTADOS DE LA FLASH (FSTAT) 165
2.5.3 REGISTRO DE COMANDOS DE LA FLASH (FCMD) 167

LISTA DE FIGURAS

Figura 1. Comparación entre diversos buses de campo....2
Figura 2. Capas del protocolo Modbus....7
Figura 3. Intercambio punto a punto....8
Figura 4. Mensaje difundido....9
Figura 5. Trama del protocolo Modbus ASCII....12
Figura 6. Transmisión de tramas en Modbus RTU....13
Figura 7. Error en la transmisión de tramas en Modbus RTU....13
Figura 8. Trama del protocolo Modbus RTU....14
Figura 9. Categorías de códigos de función Modbus....15
Figura 10. Transacción MODBUS libre de errores....16
Figura 11. Ejemplo de consulta Modbus para la función 0x01....24
Figura 12. Ejemplo de respuesta Modbus para la función 0x01....25
Figura 13. Prueba de la función 0x01 con el programa WindMill ComDebug....26
Figura 14. Diagrama de flujo de la Función 01....28
Figura 15. Archivo de parámetros Modbus.prm....30
Figura 16. Mapa de memoria del MC9S12E utilizado en el Procesador Modbus....31
Figura 17. Diagrama de bloques del temporizador TIM0....34
Figura 18. Archivo *Vectors.c* utilizado en el Procesador Modbus....38
Figura 19. Diagrama de bloques del convertidor ADC....39
Figura 20. Diagrama de bloques del convertidor DAC0....40
Figura 21. Diagrama de bloques de los puertos E/S del MCS912E....42
Figura 22. Diagrama de flujo de la función Flash_Erase_Sector ( )....48
Figura 23. Diagrama de flujo de la función Flash_Write_Word ( )....49
Figura 24. Diagrama de flujo general del software....52
Figura 25. Esquema de interrupciones de la SCI en modo RTU....57
Figura 26. Diagrama de bloques de la función de transmisión de la SCI....58

Figura 27. Diagrama de bloques de la función de recepción de la SCI 59
Figura 28. Funciones en lenguaje c para la transmisión y recepción de datos 61
Figura 29. Algoritmo para el cálculo del CRC 63
Figura 30. Función para el cálculo del CRC 64
Figura 31. Vector de valores del CRC 65
Figura 32. Función en lenguaje c para el cálculo del LRC 67
Figura 33. Diagrama de estados del modo de transmisión RTU 68
Figura 34. Rutina de atención a la interrupción de la SCI: Modo RTU 69
Figura 35. Rutina de atención a las interrupciones del Timer: Canales 4 y 5 70
Figura 36. Diagrama de estados Modo de transmisión ASCII 71
Figura 37. Rutina de atención a la interrupción de la SCI: Modo ASCII (1) 72
Figura 38. Rutina de atención a la interrupción de la SCI: Modo ASCII (2) 73
Figura 39. Diagrama esquemático del módulo de interfaz para la aplicación 76
Figura 40. Panel de control del programa de aplicación en Labview 78
Figura 41. Proceso implementado para la validación del Procesador 79
Figura 42. Procesador en una red MODBUS estándar 80
Figura 43. Panel de control del maestro Modbus 80
Figura 44. Panel de control del *Sniffer* Modbus 81
Figura 45. Trama Modbus capturada con un osciloscopio 82
Figura 46. Sistema de desarrollo Adapt9s12E128 87
Figura 47. Ejemplo de consulta Modbus para la función 0x02 98
Figura 48. Ejemplo de respuesta Modbus para la función 0x02 99
Figura 49. Diagrama de flujo de la Función 02 100
Figura 50. Ejemplo de consulta Modbus para la función 0x03 101
Figura 51. Ejemplo de respuesta Modbus para la función 0x03 102
Figura 52. Diagrama de flujo de la Función 03 103
Figura 53. Ejemplo de consulta Modbus para la función 0x04 104
Figura 54. Ejemplo de respuesta Modbus para la función 0x04 105
Figura 55. Diagrama de flujo de la Función 04 106
Figura 56. Ejemplo de consulta Modbus para la función 0x05 107
Figura 57. Ejemplo de respuesta Modbus para la función 0x05 108

Figura 58. Diagrama de flujo de la Función 05 109
Figura 59. Ejemplo de consulta Modbus para la función 06 110
Figura 60. Ejemplo de respuesta Modbus para la función 06 111
Figura 61. Diagrama de flujo de la función 06 112
Figura 62. Ejemplo de consulta Modbus para la función 07 113
Figura 63. Ejemplo de respuesta Modbus para la función 07 114
Figura 64. Diagrama de flujo para la función 07 114
Figura 65. Diagrama de flujo de la función 08 122
Figura 66. Solicitud del Contador de Eventos de Comunicaciones - Consulta 124
Figura 67. Solicitud del Contador de Eventos de Comunicaciones - Respuesta 124
Figura 68. Diagrama de flujo de la función 11 125
Figura 69. Solicitud del Diario de Eventos de Comunicaciones -Consulta 127
Figura 70. Solicitud del Diario de Eventos de Comunicaciones -Respuesta 127
Figura 71. Contenido del byte de evento: Recepción de un esclavo Modbus 128
Figura 72. Contenido del byte de evento: Recepción de un esclavo Modbus 129
Figura 73. Contenido del byte de evento: Entra en Modo de Solo Escucha 129
Figura 74. Contenido del byte de evento: Restablecimiento de comunicaciones 130
Figura 75. Diagrama de flujo de la función 12 131
Figura 76. Ejemplo de consulta Modbus para la función 15 133
Figura 77. Ejemplo de respuesta Modbus para la función 15 133
Figura 78. Diagrama de flujo de la función 15 134
Figura 79. Ejemplo de consulta Modbus para la función 17 135
Figura 80. Respuesta Modbus para la función 17 136
Figura 81. Diagrama de flujo de la función 17 137
Figura 82. Ejemplo de consulta Modbus para la función 22 139
Figura 83. Diagrama de flujo de la función 22 140
Figura 84. Ejemplo de consulta Modbus para la función 23 141
Figura 85. Ejemplo de respuesta Modbus para la función 23 142
Figura 86. Diagrama de flujo de la función 23 143
Figura 87. Ejemplo de consulta Modbus para la función 100 144
Figura 88. Respuesta Modbus para la función 100 145

Figura 89. Diagrama de flujo de la función 100 146
Figura 90. Registros de Rata de Baudios (SCIBDH, SCIBDL) 148
Figura 91. Registro de Control (SCICR1) 150
Figura 92. Registro de Control 2 (SCICR2) 151
Figura 93. Registro de Estados 1 (SCISR1) 152
Figura 94. Registro de datos (SCIDRH, SCIDRL) 154
Figura 95. Registro de selección captura entrada/comparación de salida (TIOS) 155
Figura 96. Registro de control del sistema temporizador 1 (TSCR1) 156
Figura 97. Registro de control del sistema temporizador 2 (TSCR2) 157
Figura 98. Registro de habilitación de interrupciones del *Timer* (TIE) 158
Figura 99. Registro de habilitación de interrupciones del Timer (TIE) 158
Figura 100. Registro de conteo del Timer (TCNT) 159
Figura 101. Registro de control del ATD 2 (ATDCTL2) 159
Figura 102. Registro de control del ATD 3 (ATDCTL3) 160
Figura 104. Registro de control del DAC 0 (DACC0) 163
Figura 105. Registro divisor del reloj de la Flash (FCLKDIV) 165
Figura 106. Registro de estados de la Flash (FSTAT) 165
Figura 107. Registro de comandos de la Flash (FCDM) 167

LISTA DE TABLAS

Tabla 1. Funciones básicas y códigos de operación MODBUS 19
Tabla 2. Valores por defecto para las salidas en memoria RAM 23
Tabla 3. Valores por defecto para las entradas en memoria RAM 23
Tabla 4. Ubicación de las salidas en el ejemplo de la función 0x01 26
Tabla 5. Vectores de interrupción para el microcontrolador MC9S12E 36
Tabla 6. Resumen de configuración de pines de los puertos E/S 43
Tabla 7. Tabla de configuración de registros para los GPIO 44
Tabla 8. Comandos válidos para grabar en la memoria Flash 45
Tabla 9. Opciones de configuración por hardware para el Procesador Modbus 54
Tabla 10. Opciones de configuración por software para el Procesador Modbus 55
Tabla 11. Distribución de los pines del MCS912E128 utilizados en la aplicación 75
Tabla 12. Mapa de memoria del Porcesador Modbus 89
Tabla 13. Asignación de memoria del Procesador 90
Tabla 14. Programación de velocidades de transmisión para el Procesador 149
Tabla 15. Selección de la frecuencia del reloj del temporizador 157
Tabla 16. Codificación de la longitud de la secuencia de conversión 161
Tabla 17. Longitudes disponibles para la segunda fase de muestreo 162

Capítulo 1

INTRODUCCION

Tradicionalmente en la industria, la comunicación entre los diferentes dispositivos para el control de procesos se ha realizado a través de señales analógicas de 4 a 20 mA. Sin embargo, el desarrollo y evolución de los sistemas de control distribuido hacia los años noventa, así como el auge de las comunicaciones digitales propiciaron la aparición de la tecnología de los *buses de campo*, como una alternativa para la intercomunicación entre los diferentes elementos de control en una planta.

La tecnología de *bus de campo*, permite la comunicación serial y multipunto entre los elementos primarios (sensores y actuadores) y los elementos de automatización de alto nivel (PLCs y controladores locales).[1] Las ventajas más sobresalientes de los buses de campo y que han impulsado su constante crecimiento son:

- Reducción considerable del cableado.
- Incremento de la capacidad de procesamiento (inteligencia) en sensores y actuadores.
- Disminución de los costos de instalación y mantenimiento.

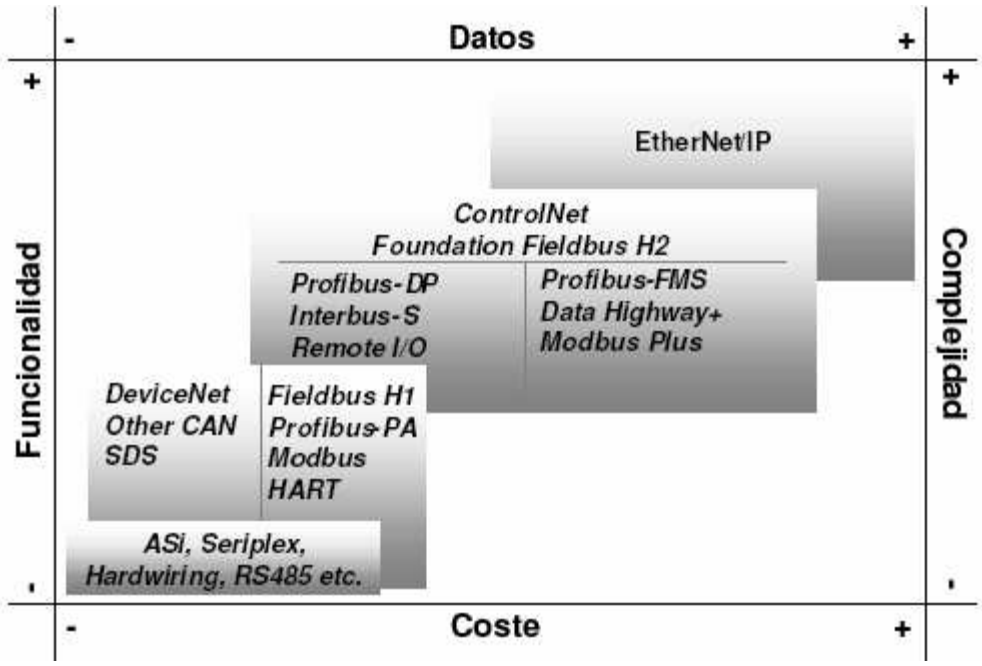
---

[1] IEC (*International Electrotechnical Commission*) e ISA (*Instruments Society of America*)

- Disposición de mayor cantidad de datos en los instrumentos (configuración, estado y mantenimiento a distancia).
- Disminución del número de barreras de seguridad en áreas clasificadas.
- Facilidad de configuración.

Actualmente, existen diferentes buses de campo, cada uno con su propio nombre y protocolo: Fieldbus-Foundation, Interbus-s, Profibus, CAN, Devicenet, Controlnet, AS-i, Modbus, y otros (Figura 1).

Figura 1. Comparación entre diversos buses de campo



Fuente: http://www.uv.es/~rosado/sid/sid.html consultada en Nov/2004

Cada protocolo está optimizado para diferentes niveles de automatización y en consecuencia responden al interés de los diferentes proveedores. Por ejemplo Fieldbus Foundation, Profibus y Modbus, están diseñados para instrumentación de control de procesos. En cambio DeviceNet y AS-i están optimizados para aplicaciones de dispositivos discretos de detectores, actuadores e interruptores, donde el tiempo de respuesta y repetibilidad son factores críticos.

Algunos de esto buses son de carácter propietario y otros son de uso tan extenso que pueden considerarse estándares de facto, como es el caso de Modbus.

La principal característica de los buses propietarios es que pertenecen a una compañía o grupo de compañías, y para utilizarlos es necesario obtener una licencia, que es concedida a la empresa que la disfruta con una serie de condiciones asociadas, y a un precio considerable.

Por el contrario, los buses abiertos tienen las siguientes características:

- Las especificaciones son públicas y disponibles a un precio razonable.
- Los componentes críticos también están disponibles.
- Los procesos de validación y verificación están bien definidos y disponibles en las mismas condiciones que los anteriores.

En Colombia, el Grupo de Investigación en Percepción y Sistemas Inteligentes de la Universidad del Valle ha trabajado en los últimos años en el desarrollo de algunos

buses de campo[2] y en la Universidad Industrial de Santander, el Grupo de Investigación en Control, Electrónica, Modelado y Simulación CEMOS ha iniciado una serie de trabajos de investigación en el área de las comunicaciones industriales específicamente en la implementación de buses de campo de arquitectura abierta.

Como un primer paso en la apropiación del conocimiento científico y tecnológico en el tema de los buses de campo el presente trabajo de investigación propone la implementación del protocolo Modbus en un microcontrolador. Las principales razones para trabajar con éste protocolo son:

- Modbus es uno de los protocolos de comunicación industrial más antiguos y difundidos en Colombia y en el mundo.
- Sus especificaciones se encuentran disponibles al público.
- Está ampliamente documentado por la organización Modbus-IDA.

En este trabajo de investigación se analizó e implementó el protocolo Modbus en un microcontrolador y se obtuvo un procesador de comunicaciones Modbus de tipo genérico con la capacidad de ser instalado en dispositivos para el control de procesos o instrumentación tales como medidores de energía, controladores digitales, transmisores y actuadores.

---

[2] Ruiz, A. "*Implementación de una red Modbus TCP/IP*". Tesis de grado.Universidad del Valle. 2001.

Este procesador de comunicaciones se implementó con las siguientes características:

- Cumple con las especificaciones del protocolo MODBUS establecidas en el documento PI–MBUS–300 Rev. J de Junio de 1996.
- Permite la interoperabilidad con sistemas SCADA.
- Permite establecer la comunicación serial entre un dispositivo de campo, (por ejemplo un transmisor o actuador) y una RTU (*Remote Terminal Unit*) o un PC.

Finalmente el aporte académico del trabajo de investigación permitió:

- Establecer una metodología para la implementación del protocolo Modbus en un microcontrolador.
- Validar la implementación del protocolo con el diseño de un procesador de comunicaciones para aplicaciones de propósito general.

Capítulo 2

EL PROTOCOLO MODBUS

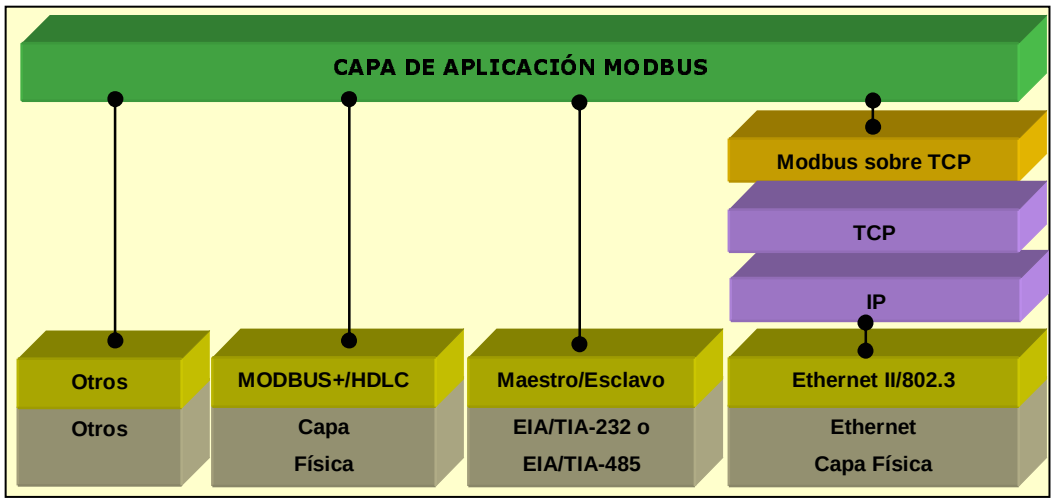
2.1 INTRODUCCIÓN

Modbus es uno de los protocolos de comunicación industrial más antiguos, apareció en 1979 como un bus para transmitir y recibir datos entre controladores y sensores a través del puerto RS-232 (comunicación punto a punto), mediante una topología de maestro/esclavo.

Modbus nació como una marca registrada de Gould Inc. y posteriormente fue adquirida por el grupo Schneider quien liberó el protocolo en el año 2000. Las especificaciones del protocolo Modbus se encuentran disponibles al público y está aceptado por la IEC como una especificación públicamente disponible (*Public Available Specification*) bajo la designación IEC PAS 6203. Actualmente es soportado por la organización independiente Modbus-IDA.

La designación Modbus no corresponde propiamente a un estándar de red que incluye todos los aspectos desde el nivel físico hasta el de aplicación, sino a un protocolo de mensajes, posicionado en la capa de aplicación o nivel 7 del modelo OSI (*Open System Interconnection*), tal como se muestra en la Figura 2

Figura 2. Capas del protocolo Modbus



Fuente: modbus.org

Modbus es un protocolo de comunicaciones tipo maestro/esclavo o cliente/servidor entre dispositivos conectados sobre diferentes tipos de redes, para el cual existen tres tipos de implementación:

- Transmisión serial asíncrona sobre diferentes medios: cable, fibra óptica y radio.
- TCP/IP sobre Ethernet
- Modbus Plus, sobre redes de alta velocidad

En este trabajo de investigación se implementó el protocolo Modbus para transmisión serial, el cual se encuentra completamente especificado en el documento PI–MBUS–300 Rev. J[3]. Las características esenciales de este protocolo se detallan a continuación:

[3] *Modicon Modbus Protocol Reference Guide* (consultado en www.modbus.org , Nov 2004)

2.2 MEDIO FÍSICO

El medio físico de conexión puede ser un bus semidúplex (half duplex) (RS-485 o fibra óptica) o dúplex (full duplex) (RS-422 o fibra óptica).

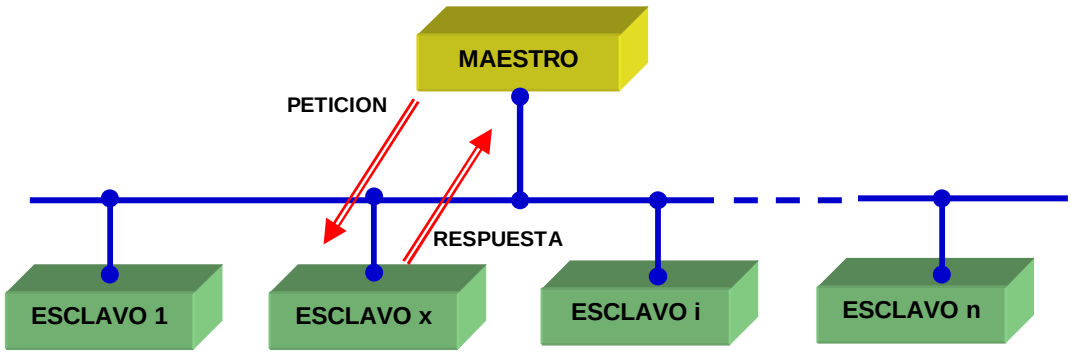
La comunicación es asíncrona y las velocidades de transmisión previstas van desde los 75 baudios a 19.200 baudios. La máxima distancia entre estaciones depende del nivel físico, pudiendo alcanzar hasta 1200 m sin repetidores.

2.3 ACCESO AL MEDIO

La estructura lógica es del tipo maestro-esclavo, con acceso al medio controlado por el maestro. El número máximo de estaciones previsto es de 247 esclavos más una estación maestra. Los intercambios de mensajes pueden ser de dos tipos:

- **Intercambios punto a punto**, que implican siempre dos mensajes: una demanda del maestro y una respuesta del esclavo que puede ser un simple reconocimiento o *"acknowledge"*. (Figura 3).

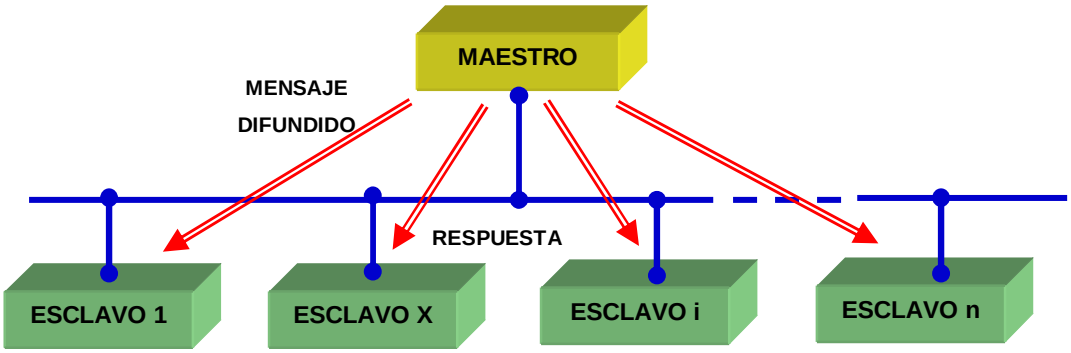
Figura 3. Intercambio punto a punto



Fuente: Balcells y Romeral

- **Mensajes difundidos**. Estos consisten en una comunicación unidireccional del maestro a todos los esclavos. Este tipo de mensajes no tiene respuesta por parte de los esclavos y se suelen emplear para mandar datos comunes de configuración, reset, y otros. (Figura 4)

Figura 4. Mensaje difundido



Fuente: Balcells y Romeral[4]

2.4 MODOS DE TRANSMISIÓN

Se pueden establecer comunicaciones en redes estándar Modbus utilizando cualquiera de estos dos modos de transmisión: ASCII o RTU. La selección de modo ASCII o RTU define los contenidos de los campos del mensaje serial transmitido por la red y determina como será empaquetada la información en los campos de código y mensaje (trama).

[4] [4] Ballcells J. y Romerall J. L. Autómatas programables. Editorial Alfaomega. Barcelona 1999

2.4.1 MODO ASCII

En el modo ASCII (Código Estándar Americano para Intercambio de Información), cada *byte* se envía como dos caracteres ASCII. La ventaja principal de este modo es que permite intervalos de tiempo de hasta un segundo entre caracteres sin causar error. El formato de cada byte en modo ASCII es:

- **Sistema de Codificación**: Hexadecimal, caracteres ASCII 0-9, A-F. Un número hexadecimal en cada carácter ASCII del mensaje.
- **Bits por Byte**: 1 bit de comienzo

  7 bits de datos, el bit menos significativo se envía primero

  1 bit de paridad par/impar; o ninguno si no hay paridad

  1 bit de fin si se usa control de paridad; ó

  2 bits de fin si no se usa control de paridad
- **Campo de Control de Error:** Control de Redundancia Longitudinal (LRC)

2.4.2 MODO RTU

En el modo RTU (Unidad Terminal Remota), cada byte, contiene dos caracteres hexadecimales de 4 bits. La ventaja principal de este modo es que su mayor densidad de caracteres permite una mejor eficiencia en el tiempo de transmisión de información que en el modo ASCII para la misma velocidad. Cada mensaje se transmite conjuntamente sin interrupción. El formato de cada byte en modo RTU es:

- **Sistema de Codificación**: 8 bits binarios, hexadecimal, 0-9, A- F

- **Bits por Byte**: 1 bit de inicio

  8 bits de datos, el bit menos significativo se envía primero

  1 bit de paridad par/impar, o ninguno si no hay paridad

  1 bit de fin si se usa control de paridad, ó

  2 bits de fin si no hay control de paridad

- **Campo de Control de Error:** Control de Redundancia Cíclica (CRC)

2.5 FORMATOS DE LA TRAMA MODBUS

2.5.1 MODO ASCII

En modo ASCII, los mensajes comienzan con dos puntos (:) Carácter (ASCII 3A hex), y terminan con el par "retorno de carro – salto de línea" (CR-LF) (ASCII 0D y 0A hex).

Los caracteres permitidos en la transmisión para todos los demás campos son 0-9, A-F. Los dispositivos conectados vigilan la red continuamente para detectar el carácter (:). Cuando se recibe, cada dispositivo decodifica el siguiente campo (Campo de dirección) para averiguar si es el dispositivo direccionado.

Se permiten intervalos de hasta un segundo entre caracteres dentro del mensaje. Si transcurre un tiempo mayor, el dispositivo receptor supone que ha ocurrido un error.

En la figura 5 se muestra el formato del mensaje.

Figura 5. Trama del protocolo Modbus ASCII

INICIO	DIRECCION	FUNCION	DATOS	LRC	FIN
1 CARÁCTER :	2 CARACTERES	2 CARACTERES	n CARACTERES	2 CARACTERES	2 CARACTERES CR LF

Fuente: modbus.org

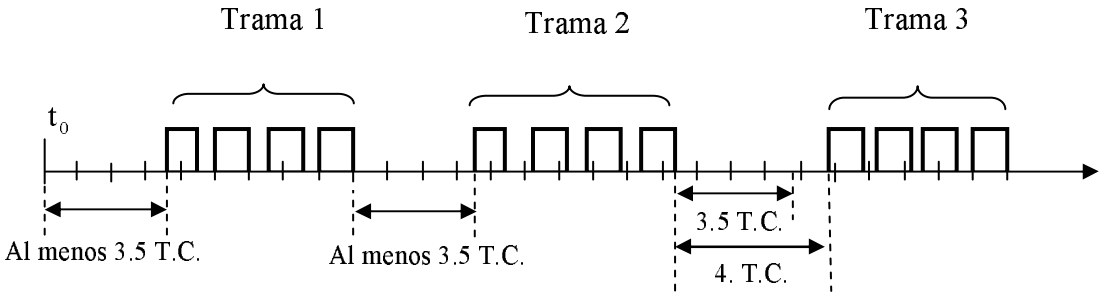
2.5.2 MODO RTU

En modo RTU, los mensajes comienzan con un intervalo de silencio de al menos 3,5 veces el tiempo de duración de un carácter. Esto se realiza esperando un tiempo múltiplo de la velocidad en baudios que se está utilizando en la red (T1-T2-T3-T4 en la Figura 6). Luego se transmite el primer campo correspondiente a la dirección del dispositivo.

Los caracteres permitidos para todos los campo son 0-9, A-F. Los dispositivos conectados vigilan el bus de red continuamente, incluso en los intervalos de silencio. Cuando se recibe el primer campo (el Campo de dirección), cada unidad lo decodifica para averiguar si es el dispositivo direccionado.

Después del último carácter transmitido se intercala un intervalo de tiempo equivalente, al menos, a 3.5 veces el tiempo de un carácter para marcar el fin del mensaje. Después de este intervalo puede comenzar un nuevo mensaje. (Fig. 6)

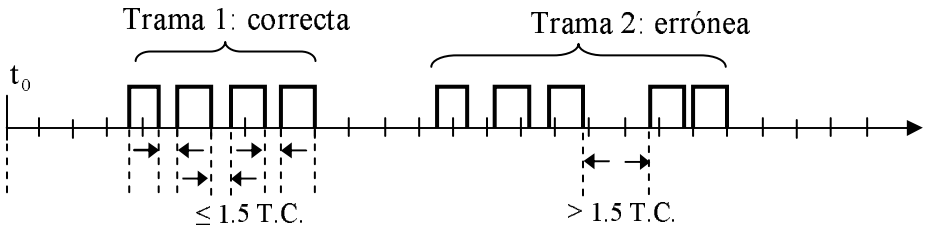
Figura 6. Transmisión de tramas en Modbus RTU



Fuente: modbus.org

El formato de mensaje completo tiene que transmitirse conjuntamente. Si se produce un intervalo de más de 1.5 veces un carácter antes de la terminación del formato el dispositivo receptor asume el mensaje como incompleto y supone que el byte *n* siguiente, será el campo de dirección de un nuevo mensaje.

Figura 7. Error en la transmisión de tramas en Modbus RTU



Fuente: modbus.org

Igualmente, si mensaje nuevo comienza antes de 3.5 veces el tiempo de un carácter, el segundo mensaje se considerará como continuación del anterior. Esto provocará un error, ya que el valor del campo CRC final no será válido por los dos mensajes combinados.

A continuación se muestra un formato de mensaje típico en el modo de transmisión RTU.

Figura 8. Trama del protocolo Modbus RTU

INICIO	DIRECCION	FUNCION	DATOS	LRC	FIN
T1-T2-T3-T4	8 BITS	8 BITS	n * 8 BITS	16 BITS	T1-T2-T3-T4

Fuente: modbus.org

2.6 FUNCIONES Y CÓDIGOS

Cada función permite transmitir datos u órdenes al esclavo. Existen dos tipos básicos de órdenes:

- Ordenes de lectura/escritura de datos en los registros o en la memoria del esclavo.
- Ordenes de control del esclavo y el propio sistema de comunicaciones

Existen tres categorías de código de funciones MODBUS:

Códigos de Función públicos:

- Son códigos de función bien definidos
- Validados y públicamente documentados por la comunidad modbus.org
- Existen pruebas de conformidad documentadas

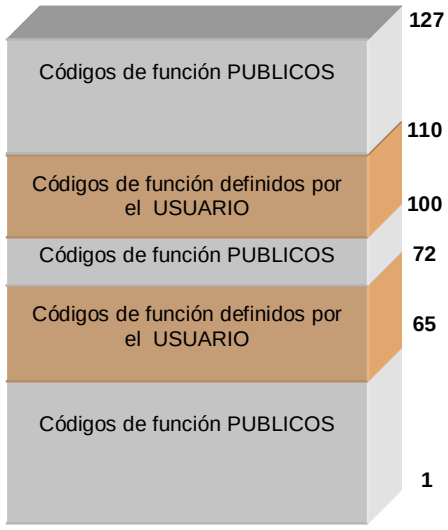
Códigos de Función definidos por el usuario:

- Son códigos de función que el usuario define para su aplicación específica. Existen dos rangos para estos códigos: 65 a 72 y de 100 a 110 (decimal).
- El usuario puede implementar estos códigos sin necesidad de su aprobación por la comunidad modbus.org

Códigos de Función reservados:

- Son códigos de función utilizados por algunas compañías para sus productos y que no son de uso público.

Figura 9. Categorías de códigos de función Modbus



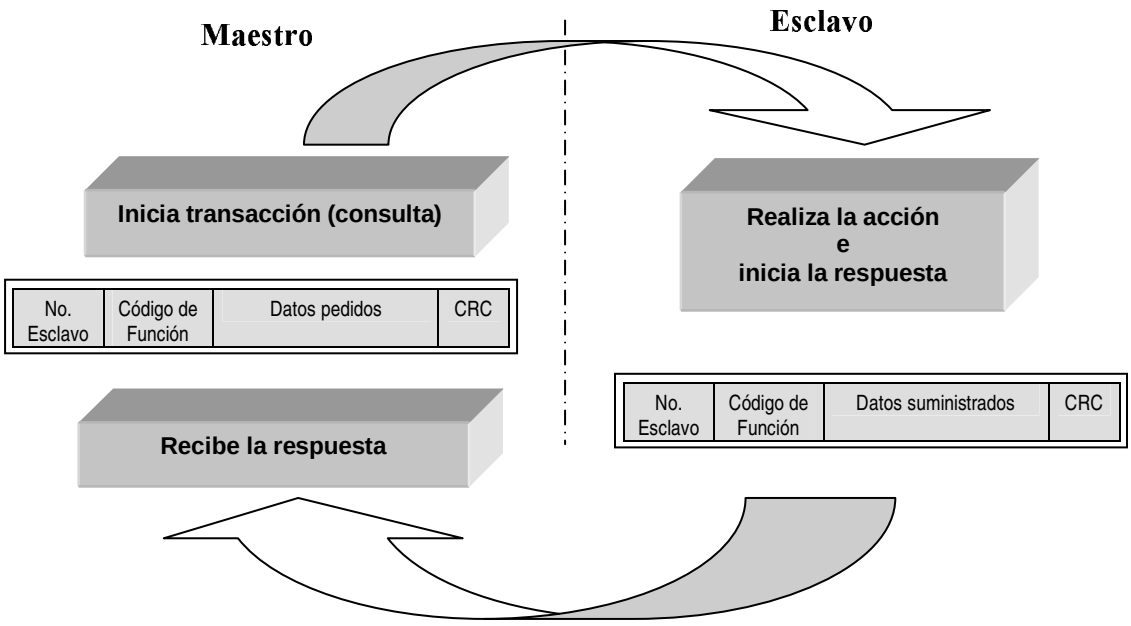
Fuente: modbus.org

2.7 TRANSACCIONES EN MODBUS

Las transacciones en MODBUS son iniciadas por el dispositivo maestro (llamadas "consultas"). Los otros dispositivos (esclavos) responden suministrando la información solicitada por el maestro, o realizando la acción solicitada en la consulta.

En la figura 10 se muestra el esquema de transacciones Modbus libre de errores.

Figura 10. Transacción MODBUS libre de errores



Fuente: modbus.org

Por ejemplo un maestro puede leer el estado (on/off) de un grupo de salidas o entradas o puede leer o escribir los datos contenidos en un grupo de registros.

Cuando el esclavo responde al maestro, este utiliza el campo de código de función para indicar bien sea una respuesta normal (libre de error) o alguna clase de error que haya ocurrido (llamado respuesta de excepción). Para una respuesta normal, el esclavo simplemente replica el código de función original

Consulta: El código de operación en la consulta dice al dispositivo esclavo el tipo de acción a realizar. Los bytes de información contienen toda la información adicional que el esclavo necesita para desempeñar la función. Por ejemplo, el código de operación 03 solicita al esclavo la lectura de registros internos y éste responde con su contenido.

El campo de información tiene que decirle al esclavo en que registro empieza la petición y cuantos registros se quieren leer. El campo de verificación de error provee un método para que el esclavo garantice la integridad del contenido del mensaje.

Respuesta: Si el esclavo envía una respuesta normal, el código de operación de la respuesta es un eco del código de operación en la consulta. Los bytes de información contienen los datos recopilados por el esclavo, como valores o estado de registros. Si se detecta un error el código de operación se modifica para indicar que la respuesta es un mensaje de error y que los bytes de información contienen el código que describe el error. La verificación del campo de error permite que el maestro pueda confirmar que los contenidos de mensaje son válidos

Capítulo 3

FUNCIONES MODBUS

3.1 INTRODUCCION

La implementación del protocolo Modbus conlleva la programación de las principales funciones descritas en los documentos disponibles al público por la organización Modbus-IDA [5,6]. Las funciones programadas para en el procesador Modbus se pueden clasificar en tres grupos:

- **Funciones de acceso de Bits:** Para la lectura de entradas discretas y la lectura o escritura de salidas discretas.
- **Funciones de acceso de palabras:** Para la lectura de registros de entrada de 16 bits y la lectura o escritura de registros de salida.
- **Funciones de diagnóstico y configuración:** Para configurar los parámetros del Procesador y obtener información acerca de los eventos y errores ocurridos en el proceso de comunicación.

Estas funciones se resumen en la Tabla 1.

---

[5] Modbus Protocol Reference Guide PI–MBUS–300 Rev. J

[6] MODBUS over Serial Line Specification & Implementation guide V1.0

Tabla 1. Funciones básicas y códigos de operación MODBUS

FUNCIONES MODBUS				CODIGOS DE FUNCION		
				CODIGO	SUB CODIGO	HEX
Acceso de Datos	Acceso de Bits	Entradas Físicas Discretas	Leer entradas discretas	02		02
		Bits Internos o Salidas físicas	Leer salidas discretas	01		01
			Escribir una sola salida	05		05
			Escribir múltiples salidas	15		0F
	Acceso de 16 Bits	Entrada física de registros	Leer registro de entrada	04		04
		Registros Internos o Salida física de registros	Leer múltiples registros	03		03
			Escribir un registro	06		06
			Escribir múltiples registros	16		10
			Leer/Escribir múltiples registros	23		17
			Escritura de registros con máscara	22		16
Diagnóstico			Leer estados de excepción	07		07
			Diagnóstico	08	00-18	
			Obtener contador de eventos	11		0B
			Obtener registro de eventos	12		0C
			Reportar identificación de esclavo	17		11

Fuente: modbus.org

La descripción detallada, ejemplos y diagrama de flujo de cada una de las funciones implementadas en el Procesador se encuentran en el anexo I.

A continuación se enumeran las funciones Modbus implementadas en el Procesador:

- **Read_Coils ( )**

  Lee hasta 2000 estados ON/OFF de salidas digitales (*coils*) en un dispositivo remoto.

- **Read_Inputs ( )**

  Lee hasta 2000 estados ON/OFF de entradas digitales en un dispositivo remoto.

- **Read_Holding_registers ( )**

  Lee hasta 125 registros internos o registros de salidas (*holding registers*) de 16 bits del Procesador de Comunicación.

- **Read_Input_registers ( )**

  Lee hasta 125 registros de entradas (*input registers*) de 16 bits.

- **Force_Single_Coil ( )**

  Fuerza el estado de una salida a uno (ON) o a cero (OFF)

- **Force_Single_Register( )**

  Escribe un valor en un único registro interno de 16 bits.

- **ReadExceptionStatus( )**

  Lee el contenido de ocho bits internos de condición de excepción (definidos por el usuario) en el Procesador.

- **Diagnostics_Function ( )**

  Proporciona una serie de pruebas para chequear el sistema de comunicación entre el dispositivo maestro y el esclavo, o para comprobar diversas condiciones internas en el Procesador Modbus.

- **Get_Comm_Event_Counter ( )**

  Devuelve una palabra de estado y un contador de eventos de comunicaciones del Procesador Modbus.

- **Get_Comm_Event_Log ( )**

  Devuelve una palabra de estado, un contador de eventos, un contador de mensajes, y un campo de bytes de evento del Procesador.

- **Write_Multiple_Coils ( )**

  Se utiliza para forzar varias salidas o bits internos consecutivos, ya sea a ON o a OFF

- **Write_Multiple_Registers ( )**

  Se utiliza para escribir en varios registros de salida o internos consecutivos.

- **Report_Slave_ID ( )**

  Lee la identificación, el estado actual, o cualquier otra información específica del Procesador Modbus

- **Mask_Write_Register ( )**

  Se utiliza para modificar el contenido de un registro interno o de salida (*holding register*) específico mediante una combinación de una máscara AND, una máscara OR y el contenido del registro

- **Read_Write_Multiple_Registers ( )**

  Realiza una operación de lectura y una operación de escritura en un registro interno o de salida en una sola transacción MODBUS.

- **User_Config ( )**

  Se utiliza para configurar por software la dirección de esclavo, interfaz RS-232/RS-485, modo ASCII/RTU y la tasa de baudios del Procesador Modbus.

3.2 IMPLEMENTACION

Uno de los aspectos claves en la implementación del protocolo MODBUS es la correcta interpretación de cada una de las funciones descritas en el documento de referencia: *Modbus Protocol Reference Guide PI–MBUS–300 Rev. J.*

Una vez entendida la mecánica de cada función se procede a su implementación en lenguaje C y a su comprobación mediante pruebas que se encuentran documentadas. Estas pruebas consisten en enviar mensajes MODBUS al Procesador y recibir las respuestas correctas de acuerdo con los ejemplos que se encuentran disponibles en el documento.

Sin embargo para poder recibir un mensaje es necesario cargar inicialmente unos datos en memoria RAM. Estos datos precargados sirven para realizar las pruebas de conformidad del Protocolo y una vez que se ha verificado el correcto funcionamiento pueden ser cambiados por el usuario o por la aplicación que se esté ejecutando en el Procesador.

Para efectos de comprobar el funcionamiento de todas las funciones del procesador Modbus, la memoria RAM se inicializa con los siguientes valores: (Tablas 2 y 3)

Tabla 2. Valores por defecto para las salidas en memoria RAM

Dirección	Datos (Hex)							
0x2000	B3	D6	96	7A	C9	AE	1B	01
0x2008	B3	D6	96	7A	C9	AE	1B	01
0x2010	B3	D6	96	7A	C9	AE	1B	01
0x2018	B3	D6	96	7A	C9	AE	1B	01
0x2020	B3	D6	96	7A	C9	AE	1B	01
0x2028	B3	D6	96	7A	C9	AE	1B	01
0x2030	B3	D6	96	7A	C9	AE	1B	01
0x2038	FF	00	FF	00	FF	00	FF	00
0x2040	B3	D6	96	7A	C9	AE	1B	01
0x2048	B3	D6	96	7A	C9	AE	1B	01
0x2050	B3	D6	96	7A	C9	AE	1B	01
0x2058	B3	D6	96	7A	C9	AE	1B	01
0x2060	B3	D6	96	7A	C9	AE	1B	01
0x2068	B3	D6	96	7A	C9	AE	1B	01
0x2070	B3	D6	96	7A	C9	AE	1B	01

Tabla 3. Valores por defecto para las entradas en memoria RAM

Dirección	Datos (Hex)							
0x2100	B3	D6	96	7A	C9	AE	1B	01
0x2108	B3	D6	96	7A	C9	AE	1B	01
0x2110	B3	D6	96	7A	C9	AE	1B	01
0x2118	B3	D6	96	7A	C9	AE	1B	01
0x2120	B3	D6	96	7A	C9	AE	1B	01
0x2128	B3	D6	96	7A	C9	AE	1B	01
0x2130	B3	D6	96	7A	C9	AE	1B	01
0x2138	FF	00	FF	00	FF	00	FF	00
0x2140	B3	D6	96	7A	C9	AE	1B	01
0x2148	B3	D6	96	7A	C9	AE	1B	01
0x2150	B3	D6	96	7A	C9	AE	1B	01
0x2158	B3	D6	96	7A	C9	AE	1B	01
0x2160	B3	D6	96	7A	C9	AE	1B	01
0x2168	B3	D6	96	7A	C9	AE	1B	01
0x2170	B3	D6	96	7A	C9	AE	1B	01

3.3 EJEMPLO DE IMPLEMENTACION: FUNCIÓN 01 (READ COILS)

Esta función se utiliza para leer hasta 2000 estados ON/OFF de salidas digitales (*coils*) en un dispositivo remoto. En el procesador Modbus se designó el área de memoria 0x2000 hasta 0x20FF para guardar los datos correspondientes a los estados de las salidas discretas del dispositivo remoto.

En el mensaje Modbus las salidas digitales (*coils*) inician en la dirección cero (0x0000) de tal forma que la salida discreta *(coil*) 1 corresponde a la dirección 0 y la 16 a la dirección 15. La consulta general no está permitida para esta función. En la figura 11 se muestra un ejemplo de consulta de lectura de los estados de las bobinas 20-38 del procesador:

Figura 11. Ejemplo de consulta Modbus para la función 0x01

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**01**
Dirección Inicial Hi	**00**
Dirección Inicial Lo	**13**
Número de salidas Hi	**00**
Número de salidas Lo	**13**
CRC Hi	**CRC**
CRC Lo	

El estado de salidas discretas (*coils*) en el mensaje de respuesta del procesador se encuentra codificado como una salida por cada bit del campo de datos. El estado se indica como: 1 = ON , 0 = OFF

Debido a que en el mensaje de respuesta se envía primero el byte menos significativo, el campo de datos será:

- Primer Byte: **0xCD** ó 1 1 0 0 1 1 0 1 (salidas 27 a 20)
- Segundo Byte: **0x6B** ó 0 1 1 0 1 0 1 1 (salidas 35 a 28)
- Tercer Byte: **0x02** ó 0 0 0 0 0 0 1 0 (salidas 38 a 36)

En el último byte de datos, los estados de las salidas 38-36 se muestran como el valor hexadecimal 0x02 o binario 0000 0010, ya que los cinco bits restantes hasta completar el byte se rellenan con ceros.

En la Figura 12 se muestra la repuesta a la consulta solicitada en el ejemplo anterior.

Figura 12. Ejemplo de respuesta Modbus para la función 0x01

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**01**
Conteo de Bytes	**03**
Estados salidas 27-20	**CD**
Estados salidas 35-28	**6B**
Estados salidas 38-36	**02**
CRC Hi	**03**
CRC Lo	**40**

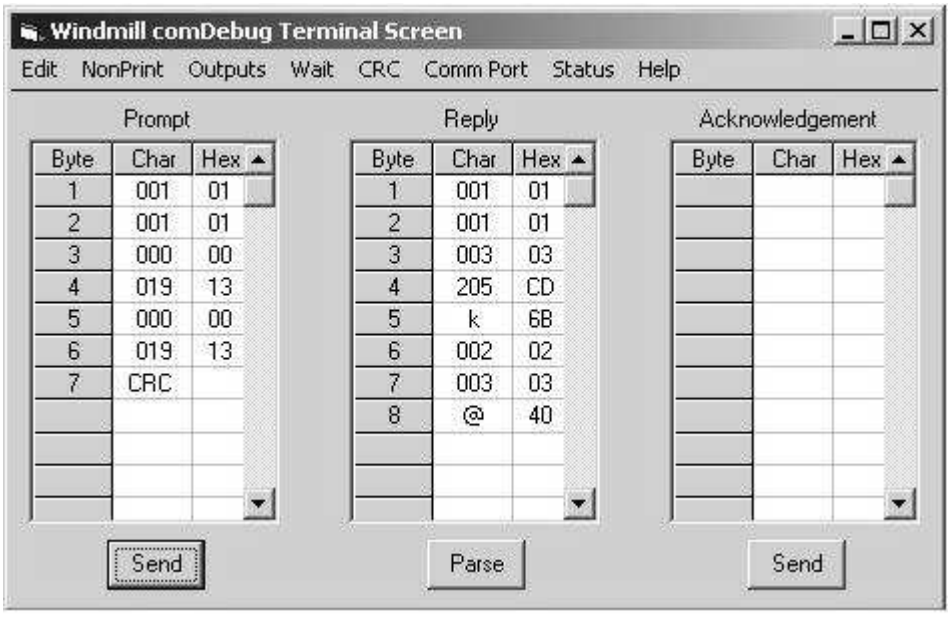
Con referencia al ejemplo mencionado anteriormente, las salidas especificadas en el mensaje de consulta se muestran en la Tabla 4.

Tabla 4. Ubicación de las salidas en el ejemplo de la función 0x01

Dirección	Área de memoria								(Hex)
0x2000	1 1	2 0	3 1	4 1	5 0	6 0	7 1	8 1	0xB3
0x2001	9 1	10 1	11 0	12 1	13 0	14 1	15 1	16 0	0xD6
0x2002	17 1	18 0	19 0	20 1	21 0	22 1	23 1	24 0	0x96
							1er Byte		
0x2003	25 0	26 1	27 1	28 1	29 1	30 0	31 1	32 0	0x7A
						2do Byte			
0x2004	33 1	34 1	35 0	36 0	37 1	38 0	39 0	40 1	0xC9
					3er Byte				

En la depuración del código implementado para la función 0x01 se utilizó el programa *shareware* de comunicación serial *Windmill ComDebug*[7] . (Figura 13):

Figura 13. Prueba de la función 0x01 con el programa WindMill ComDebug



[7] Copyright Windmill Software Ltd 2001 -2005. http://www.windmill.co.uk/serial.html

En la programación en lenguaje C de la función 0x01 del protocolo Modbus se tuvo en cuenta el manejo de memoria del microcontrolador HCS12 utilizado en el procesador de comunicaciones.

Debido a que los bits en cada una de las celdas de la memoria RAM se ordenan con el LSB a la derecha y el MSB a l izquierda, mientras que los estados de las salidas se numeran de izquierda a derecha, se hace necesario manipular los bits para enviar el mensaje de respuesta de acuerdo con dicha convención.

Esto se hace mediante las funciones de usuario: ***mirror_byte***, y ***zero_padding***, las cuales se utilizan para construir el mensaje de respuesta Modbus:

- **Función mirror_byte**

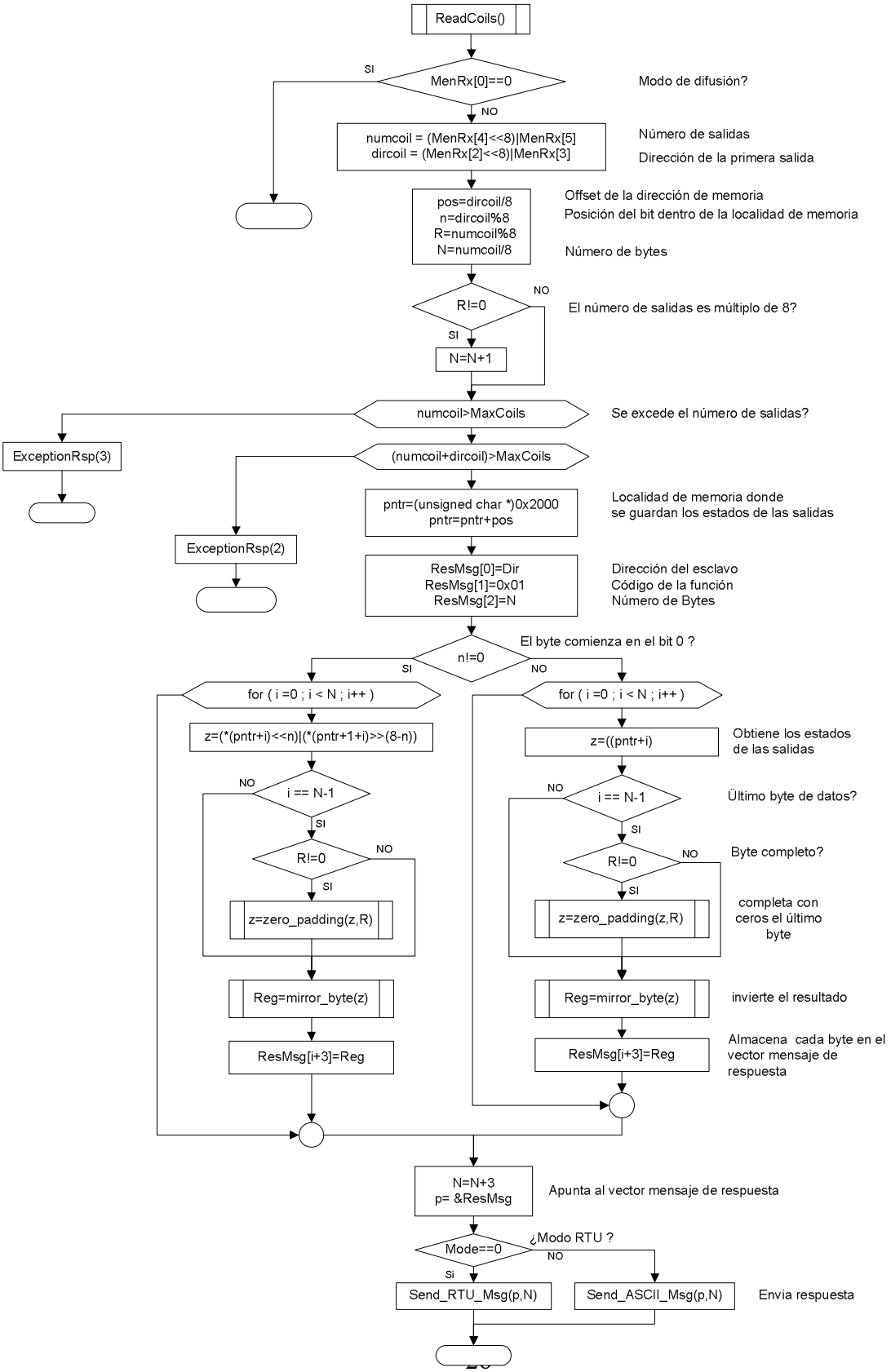
  Esta función invierte los bits leídos de las localidades de memoria asignadas a las salidas. Esto se debe a que en la memoria RAM el bit menos significativo está a la derecha mientras que en el mensaje de respuesta Modbus el estado de la salida menos significativa se encuentra a la izquierda.

- **Función zero_padding**

  Esta función completa con ceros el byte, cuando el número de bits leídos no es múltiplo de 8. Dicha función se aplica únicamente al último byte del campo de datos del mensaje.

En la Figura 14 se puede observar el diagrama de flujo utilizado para la programación de la función 01.

Figura 14. Diagrama de flujo de la Función 01



Capítulo 4

EL MICROCONTROLADOR HCS12

El microcontrolador utilizado en la implementación del Procesador Modbus es un MC9S12E de Motorola, el cual posee, entre otras, las siguientes características:

- Unidad central de procesamiento (CPU) de 16 bits
- 128 K Bytes de memoria Flash EEPROM y 8 K Bytes de memoria RAM
- Tres módulos de interfaces seriales asíncronas (SCI)
- Dos convertidores DAC de 8 bits
- Un convertidor ADC de 16 canales y 10 bits de resolución.
- Tres temporizadores (*Timers*) TIM de 4 canales cada uno.
- Frecuencia máx. de operación de 50 MHz (equivalente a 25 MHZ en el bus).
- 90 líneas de Entrada/Salida máximo.

Estas características proporcionan todas las facilidades para la implementación del Protocolo: Las SCI para la comunicación RS-232/RS-485, los *Timers* para los tiempos 1.5 TC y 3.5 TC en modo RTU, Memoria RAM para guardar los registros y bits de entrada y salida, los convertidores ADC y DAC para la aplicación y finalmente una buena cantidad de puertos de propósito general utilizados tanto en configuración del Procesador como para la aplicación (manejo de entradas y salidas digitales).

4.1 MAPA DE MEMORIA

El mapa de memoria del microcontrolador MC9S12E puede ser configurado por el usuario[8] en un archivo de parámetros ***.prm***. Para el caso del Procesador Modbus se creó el archivo ***Modbus.prm***, el cual se lista en la Figura 15. En este archivo se especifican las áreas de memoria RAM y ROM reservadas.

Figura 15. Archivo de parámetros Modbus.prm

```
NAMES

END

SECTIONS
      /* List of all sections specified on the "Build options" tab */
      RAM_2300    =  READ_WRITE           0x00002300 TO 0x00003FFF;
      ROM_C000    =  READ_ONLY            0x0000C000 TO 0x0000FEFF;
      ROM_4800    =  READ_ONLY            0x00004800 TO 0x00007FFF;
      ROM_FF10    =  READ_ONLY            0x0000FF10 TO 0x0000FF7F;
      ROM_PAGE3D  =  READ_ONLY            0x003D8000 TO 0x003DBFFF;
      ROM_PAGE3C  =  READ_ONLY            0x003C8000 TO 0x003CBFFF;
      ROM_PAGE3B  =  READ_ONLY            0x003B8000 TO 0x003BBFFF;
      ROM_PAGE3A  =  READ_ONLY            0x003A8000 TO 0x003ABFFF;
      ROM_PAGE39  =  READ_ONLY            0x00398000 TO 0x0039BFFF;
      ROM_PAGE38  =  READ_ONLY            0x00388000 TO 0x0038BFFF;
      MY_RAM1     =  READ_WRITE           0x00002000 TO 0x000020FF;
      MY_RAM2     =  READ_WRITE           0x00002100 TO 0x000021FF;
      MY_RAM3     =  READ_WRITE           0x00002200 TO 0x0000227F;
      MY_RAM4     =  READ_WRITE           0x00002280 TO 0x000022FF;
      MY_ROM1     =  READ_ONLY            0x00004000 TO 0x000043FF;
      MY_ROM2     =  READ_ONLY            0x00004400 TO 0x000047FF;
END

PLACEMENT
      DEFAULT_RAM                      INTO RAM_2300;
      DEFAULT_ROM                             INTO ROM_PAGE3D, ROM_PAGE3C, ROM_PAGE3B,
ROM_PAGE3A, ROM_PAGE39, ROM_PAGE38;
      _PRESTART, STARTUP,
      ROM_VAR, STRINGS,
      NON_BANKED, COPY                                  INTO ROM_C000, ROM_4800, ROM_FF10,
MY_ROM1,MY_ROM2;
      OUTPUTS_BUFFER                   INTO MY_RAM1;
      INPUTS_BUFFER                    INTO MY_RAM2;
```

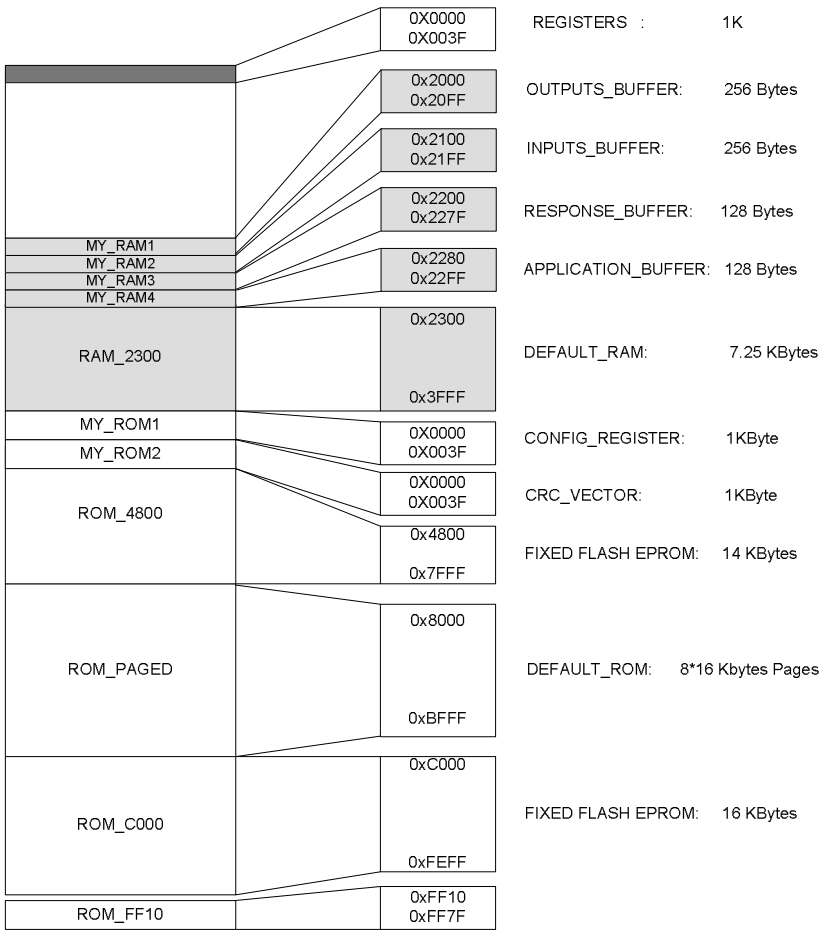
[8] Montañez, E. "*HCS12 Software Stationery*" Motorola Application Note AN2485/D. Austin, Texas 2003. pp 11-12

Las áreas de memoria que se reservaron para la implementación del Procesador Modbus fueron:

- 0x00002000 hasta 0x000020FF : Salidas y registros de salida
- 0x00002100 hasta 0x000021FF: Entradas y Registros de entrada
- 0x00002200 hasta 0x0000227F : Vector Mensaje de respuesta
- 0x00002280 hasta 0x000022FF: Área de Aplicación
- 0x00004000 hasta 0x000043FF: Vector de configuración por software
- 0x00004400 hasta 0x000047FF: Vector para el cálculo del CRC

En la figura 16 se muestra el mapa de memoria configurado para el Procesador.

Figura 16. Mapa de memoria del MC9S12E utilizado en el Procesador Modbus



4.2 INTERFAZ SERIAL ASÍNCRONA (SCI)

La SCI realiza la comunicación serial asíncrona entre el microcontrolador y otros dispositivos periféricos. En el Procesador, la SCI es el módulo que permite la implementación de la capa física del Protocolo MODBUS. En este caso se utilizaron los siguientes módulos SCI:

- SCI0: Para la comunicación en RS-232
- SCI1: Para la comunicación en RS-485

Las principales características de la interfaz serial asíncrona SCI[9] del microcontrolador son:

- Operación full-dúplex
- Formato de transmisión estándar NRZ (*Non Return to Zero*)
- Selección de la rata de baudios
- Formato de 8 o 9 bits de datos
- Habilitación separada de receptor y transmisor
- Ocho banderas de interrupción: Transmisor vacío, Transmisión completa, receptor lleno, Entrada de receptor ocupada, Error de ruido, error de trama y Error de paridad.
- Detección de error en la trama.
- Chequeo de paridad por hardware

---

[9] Motorola. "*HCS12 Serial Communications Interface Block Guide V 03.02*". Julio 03 2002

De las ocho banderas de interrupción disponibles para la SCI, se utilizó la bandera de receptor lleno (*Receiver full*).

Los registros utilizados para la programación de la SCI en la implementación del protocolo MODBUS son

- REGISTROS DE TASA DE BAUDIOS (**SCIBDH, SCIBDL**)
- REGISTRO DE CONTROL 1 (**SCICR1**)
- REGISTRO DE CONTROL 2 (**SCICR2**)
- REGISTRO DE ESTADOS 1 (**SCISR1**)
- REGISTRO DE DATOS (**SCIDRH, SCIDRL**)

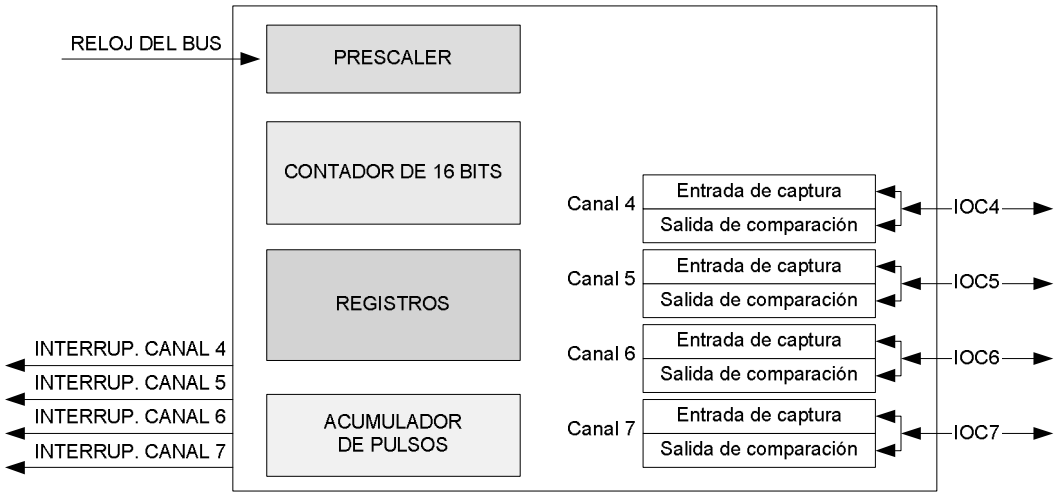
Una descripción más detallada de la programación de estos registros se encuentra en el Anexo II.

4.3 TEMPORIZADOR (TIM)

El temporizador básico consiste en un contador de 16-bits programable por software, manejado por un *prescaler* de 7 etapas. Este contador se incrementa a una tasa fija y el software puede leer su valor para conocer el tiempo actual. También puede usarse para crear pulsos, ondas cuadradas y ondas PWM.

Este temporizador contiene 4 canales de captura de entrada/salida de comparación [IOC 7:4] y un acumulador de pulsos. En la figura 17 se muestra un diagrama de bloques del temporizador.

Figura 17. Diagrama de bloques del temporizador TIM0



Fuente: Motorola[10]

En la implementación del Procesador Modbus se utilizaron los canales 4, 5 y 6 del temporizador TIM0 para generar los retardos de 1.5 T.C, 3.5 T.C. y *TimeOut* en la rutina de configuración del modo RTU.

Los registros utilizados para la programación del temporizador TIM0 en la implementación del protocolo MODBUS modo RTU fueron:

- SELECCIÓN DE CAPTURA / COMPARACION (**TIOS**)
- REGISTRO DE CONTROL DEL SISTEMA TEMPORIZADOR 2 **(TSCR2)**
- REGISTRO DE HABILITACION DE INTERRUPCIONES (**TIE**)
- REGISTRO DE BANDERAS DE INTERRUPCIONES (**TFLG1**)
- REGISTRO DE CONTEO DEL TEMPORIZADOR (**TCNT**)

Los detalles de la programación de estos registros se pueden ver en el Anexo II.

[10] Motorola Inc. *TIM_16B4C Block Guide V 1.0* Dic. 07 2001

4.4 MANEJO DE INTERRUPCIONES

En el Procesador Modbus se manejan las siguientes interrupciones:

- **Interrupción por receptor lleno en la SCI0** : utilizada para el proceso de transmisión /recepción a través de la interfaz RS-232
- **Interrupción por receptor lleno en la SCI1:** utilizada para el proceso de transmisión /recepción a través de la interfaz RS-485.
- **Interrupción por salida de comparación Canal 4 del Timer 0:** utilizada en el proceso de generación del tiempo 1.5 T.C. en el modo Modbus RTU.
- **Interrupción por salida de comparación Canal 5 del Timer 0:** utilizada en el proceso de generación del tiempo 3.5 T.C. en el modo Modbus RTU.
- **Interrupción por salida de comparación Canal 6 del Timer 0:** utilizada en el proceso de generación del TimeOut en el modo ASCII.

Cada una de las interrupciones tiene asociado un vector de 16 bits que apunta a la dirección de memoria donde se localiza la rutina que controla la interrupción. Estos vectores se guardan en los 128 bytes superiores del mapa de direcciones de memoria del Microcontrolador (0xFF80 hasta 0xFFFF).

La lista completa de posibles interrupciones del Microcontrolador MC9S12E se incluye en la tabla 5.

Tabla 5. Vectores de interrupción para el microcontrolador MC9S12E

Vector Address	Interrupt Source	CCR Mask	Local Enable
$FFFE, $FFFF	External Reset, Power On Reset or Low Voltage Reset (see CRG Flags Register to determine reset source)	None	None
$FFFC, $FFFD	Clock Monitor fail reset	None	COPCTL (CME, FCME)
$FFFA, $FFFB	COP failure reset	None	COP rate select
$FFF8, $FFF9	Unimplemented instruction trap	None	None
$FFF6, $FFF7	SWI	None	None
$FFF4, $FFF5	XIRQ	X-Bit	None
$FFF2, $FFF3	IRQ	I-Bit	INTCR (IRQEN)
$FFF0, $FFF1	Real Time Interrupt	I-Bit	CRGINT (RTIE)
$FFE8 to $FFEF	Reserved		
$FFE6, $FFE7	Standard Timer 0 channel 4	I-Bit	TIE (C4I)
$FFE4, $FFE5	Standard Timer 0 channel 5	I-Bit	TIE (C5I)
$FFE2, $FFE3	Standard Timer 0 channel 6	I-Bit	TIE (C6I)
$FFE0, $FFE1	Standard Timer 0 channel 7	I-Bit	TIE (C7I)
$FFDE, $FFDF	Standard Timer overflow	I-Bit	TSCR2 (TOI)
$FFDC, $FFDD	Pulse accumulator overflow	I-Bit	PACTL(PAOVI)
$FFDA, $FFDB	Pulse accumulator input edge	I-Bit	PACTL (PAI)
$FFD8, $FFD9	SPI	I-Bit	SPICR1 (SPIE, SPTIE)
$FFD6, $FFD7	SCI0	I-Bit	SCICR2 (TIE, TCIE, RIE, ILIE)
$FFD4, $FFD5	SCI1	I-Bit	SCICR2 (TIE, TCIE, RIE, ILIE)
$FFD2, $FFD3	SCI2	I-Bit	SCICR2 (TIE, TCIE, RIE, ILIE)
$FFD0, $FFD1	ATD	I-Bit	ATDCTL2 (ASCIE)
$FFCE, $FFCF	Port AD (KWU)	I-Bit	PTADIF (PTADIE)
$FFC8 to $FFCD	Reserved		
$FFC6, $FFC7	CRG PLL lock	I-Bit	PLLCR (LOCKIE)
$FFC4, $FFC5	CRG Self Clock Mode	I-Bit	PLLCR (SCMIE)

Fuente: Motorola[11]

[11] Motorola, Inc. *MC9S12E-Family Device User Guide V01.04.* 04 Nov 2003

Para cada una de estas interrupciones se programó una rutina de atención de interrupciones (ISR). Las rutinas de atención a las interrupciones (ISR) se manejan de forma similar como una función ordinaria.

La diferencia está en que mientras una función ordinaria se llama desde una localización específica en el software durante la secuencia del programa, la ISR ocurre cuando se produce una combinación predeterminada de eventos. Esta combinación de eventos permite interrumpir la secuencia normal del programa

Una vez que la CPU concede la petición de interrupción, se extrae la dirección del vector de interrupción. Dentro de la dirección de vector de interrupción está guardada la localización de la rutina de servicio que atenderá a la demanda.

Los vectores de interrupción sólo tienen 16 bits y por consiguiente sólo pueden apuntar a memoria no paginada. Esto significa que las rutinas de atención a las interrupciones (ISR) no deben estar localizadas en memoria no paginada[12]. Esto se logra fácilmente creando un segmento específico para las rutinas de interrupción.
Por ejemplo la rutina de atención a la interrupción provocada por la SCI0 se define en el segmento:

#pragma CODE_SEG __NEAR_SEG NON_BANKED

El uso del atributo __NEAR_SEG en la oración #pragma hace bien explícito que el segmento código tiene que ser colocado en la memoria *non_banked*.

---

[12] Stuart Robb. *MC9S12DP256 Software Development Using Metrowerk's Code warrior*. Application Note. AN2216/D Rev. 2, 4/2003

Las ISRs para el microcontrolador HCS12 se localizaron en el archivo Vectors.c, el cual se muestra en la figura 18.

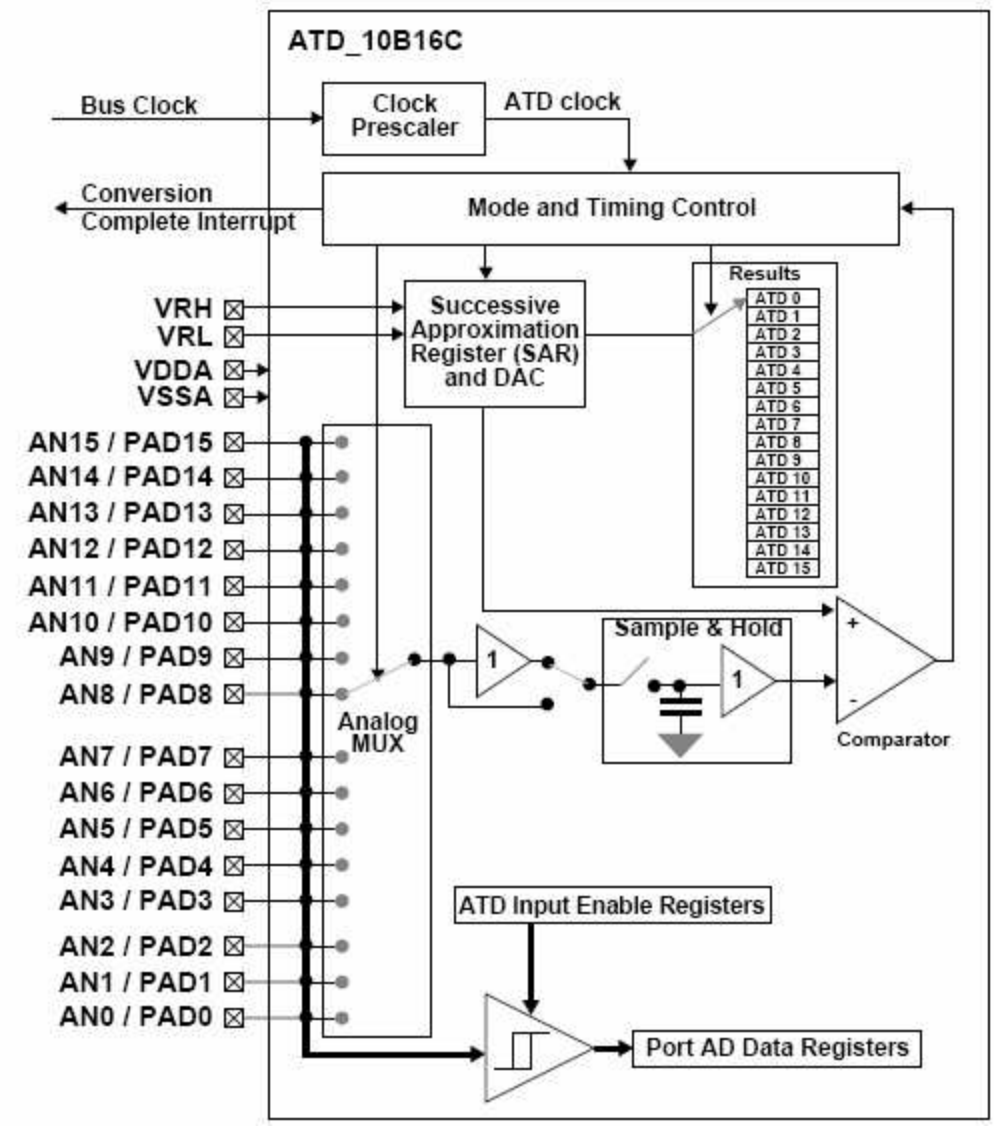
Figura 18. Archivo *Vectors.c* utilizado en el Procesador Modbus

```
#include "Cpu.h"


#pragma CODE_SEG __NEAR_SEG NON_BANKED

__interrupt void SCI0_Interrupt(void);
__interrupt void SCI1_Interrupt(void);
__interrupt void Timer15_Interrupt(void);
__interrupt void Timer35_Interrupt(void);
__interrupt void TimerOut_Interrupt(void);

#pragma CODE_SEG DEFAULT

typedef void (*near tIsrFunc)(void);
const tIsrFunc _vect[] @0xFF80 = {     /* Interrupt table */
        Cpu_Interrupt,                 /* 0 Default (unused) interrupt, address 0xFF80 */
        Cpu_Interrupt,                 /* 1 Default (unused) interrupt, address 0xFF82 */
        Cpu_Interrupt,                 /* 2 Default (unused) interrupt, address 0xFF84 */
        Cpu_Interrupt,                 /* 3 Default (unused) interrupt, address 0xFF86 */
             .                                              .
             .                                              .
             .                                              .
             .                                              .
             .                                              .
             .                                              .

        Cpu_Interrupt,                 /* 38 Default (unused) interrupt, address 0xFFCC */
        Cpu_Interrupt,                 /* 39 Default (unused) interrupt, address 0xFFCE */
        Cpu_Interrupt,                 /* 40 Default (unused) interrupt, address 0xFFD0 */
        Cpu_Interrupt,                 /* 41 Default (unused) interrupt, address 0xFFD2 */
        SCI1_Interrupt,                /* 42 Interrupt SCI1,             address 0xFFD4 */
        SCI0_Interrupt,                /* 43 Interrupt SCI0,             address 0xFFD6 */
        Cpu_Interrupt,                 /* 44 Default (unused) interrupt, address 0xFFD8 */
        Cpu_Interrupt,                 /* 45 Default (unused) interrupt, address 0xFFDA */
        Cpu_Interrupt,                 /* 46 Default (unused) interrupt, address 0xFFDC */
        Cpu_Interrupt,                 /* 47 Default (unused) interrupt, address 0xFFDE */
        Cpu_Interrupt,                 /* 48 Default (unused) interrupt, address 0xFFE0 */
        TimerOut_Interrupt,            /* 49 Interrupt Ch06,             address 0xFFE2 */
        Timer35_Interrupt,             /* 50 Interrupt Ch05,             address 0xFFE4 */
        Timer15_Interrupt,             /* 51 Interrupt Ch04,             address 0xFFE6 */
        Cpu_Interrupt,                 /* 52 Default (unused) interrupt, address 0xFFE8 */
        Cpu_Interrupt,                 /* 53 Default (unused) interrupt, address 0xFFEA */
        Cpu_Interrupt,                 /* 54 Default (unused) interrupt, address 0xFFEC */
        Cpu_Interrupt,                 /* 55 Default (unused) interrupt, address 0xFFEE */
        Cpu_Interrupt,                 /* 56 Default (unused) interrupt, address 0xFFF0 */
        Cpu_Interrupt,                 /* 57 Default (unused) interrupt, address 0xFFF2 */
        Cpu_Interrupt,                 /* 58 Default (unused) interrupt, address 0xFFF4 */
        Cpu_Interrupt,                 /* 59 Default (unused) interrupt, address 0xFFF6 */
        Cpu_Interrupt,                 /* 60 Default (unused) interrupt, address 0xFFF8 */
        Cpu_Interrupt,                 /* 61 Default (unused) interrupt, address 0xFFFA */
        Cpu_Interrupt,                 /* 62 Default (unused) interrupt, address 0xFFFC */
        _EntryPoint                    /* Reset vector, address 0xFFFE */
   };
```

4.5 CONVERTIDOR ADC

El microcontrolador MC9S12E de Motorola posee un convertidor analógico a digital (ADC) de aproximaciones sucesivas, con 16 canales de entrada multiplexados y 10 bits de resolución. En la figura 19 se muestra el diagrama de bloques del ADC.

Figura 19. Diagrama de bloques del convertidor ADC



Fuente: Motorola

Los registros utilizados para la programación del ADC en la implementación de la aplicación para el Procesador MODBUS fueron:

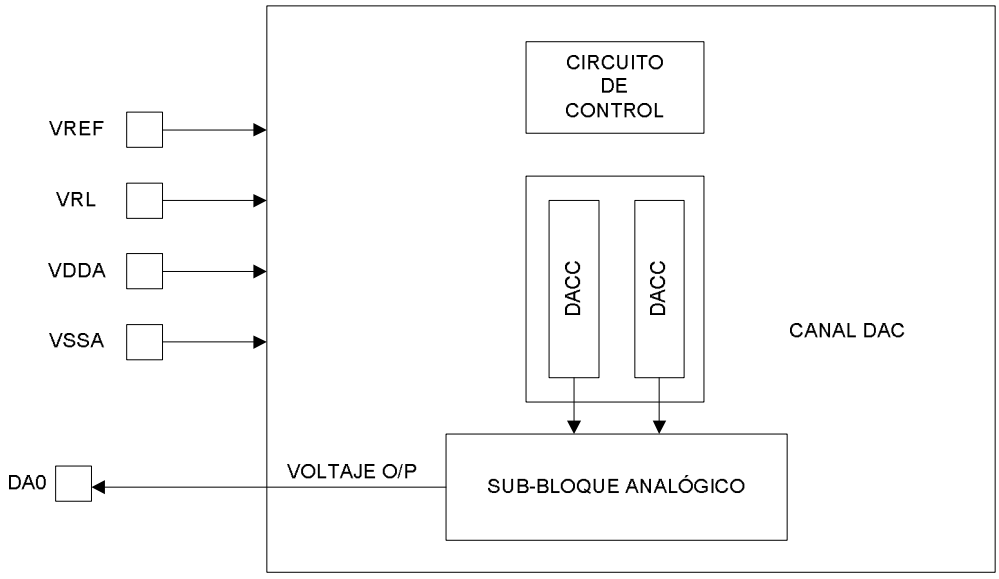
- REGISTRO DE CONTROL DEL ATD 2 (**ATDCTL2**)
- REGISTRO DE CONTROL DEL ATD 3 (**ATDCTL3**)
- REGISTRO DE CONTROL DEL ATD 4 (**ATDCTL4**)

La programación detallada de estos registros se encuentra consignada en el Anexo II.

4.6 CONVERTIDOR DAC

El microcontrolador MC9S12E de Motorola posee dos convertidores (DAC0 y DAC1) de 8 bits de resolución cada uno. En la figura 20 se muestra el diagrama de bloques del DAC0.

Figura 20. Diagrama de bloques del convertidor DAC0



Fuente: Motorola[13]

[13] Motorola, Inc. *DAC_8B1C Block Guide* V1.0 1 Jul 2003

Los registros utilizados para la programación del DAC en la implementación de la aplicación para el Procesador MODBUS fueron:

- REGISTRO DE CONTROL DEL DAC 0 **(DACC0)**

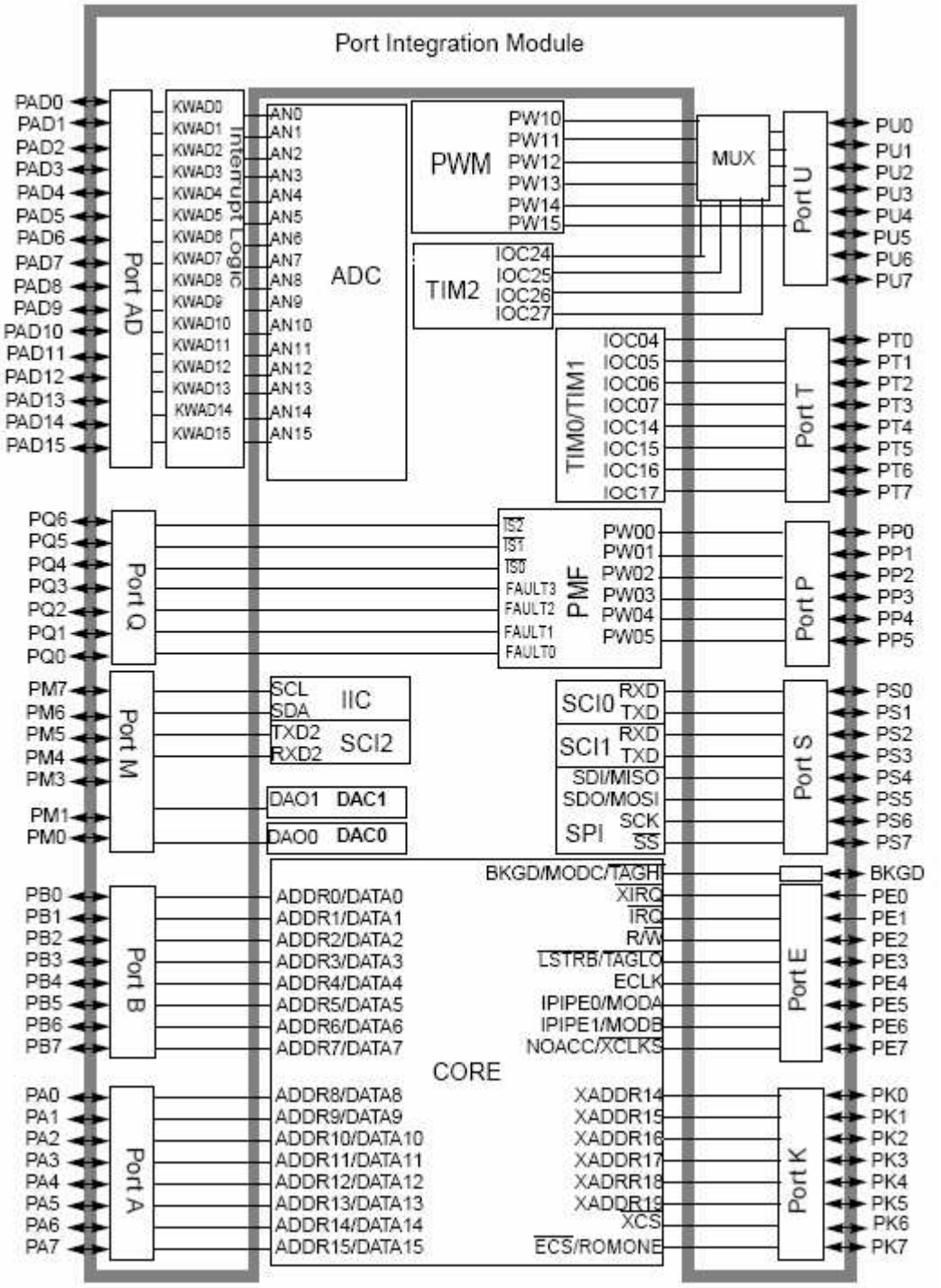
4.7 PUERTOS DE PROPÓSITO GENERAL (GPIO)

El microcontrolador MCS912E posee 10 puertos de Entrada/Salida que pueden ser programados como puertos de propósito general. Estos puertos son:

- Puertos A, B, E, y K relacionado con el núcleo y la interfaz del bus.
- Puerto T conectado a los módulos temporizadores.
- Puerto U conectado al PWM y al módulo TIM2.
- Puerto S asociado con dos módulos SCI y un módulo SPI.
- Puerto M asociado con un módulo SCI, y un módulo IIC y dos módulos DAC.
- Puertos P y Q conectados al modulo PMF.
- Puerto AD conectado al módulo ADC.

En la figura 21 se muestra el diagrama de bloques de los puertos de E/S.

Figura 21. Diagrama de bloques de los puertos E/S del MCS912E



Fuente: Motorola

La tabla 6 resume el efecto de varios bits de configuración para cada uno de los puertos: dirección de datos (DDR), nivel de salida (IO), manejo reducido del *driver* (RDR), habilitación de resistores de *pull up/down* (PER), selección de *pull up/down* (PS), y habilitación de interrupción (IE).

Tabla 6. Resumen de configuración de pines de los puertos E/S

DDR	IO	RDR	PE	PS	IE[1]	Function	Pull Device	Interrupt
0	X	X	0	X	0	Input	Disabled	Disabled
0	X	X	1	0	0	Input	Pull Up	Disabled
0	X	X	1	1	0	Input	Pull Down	Disabled
0	X	X	0	0	1	Input	Disabled	falling edge
0	X	X	0	1	1	Input	Disabled	rising edge
0	X	X	1	0	1	Input	Pull Up	falling edge
0	X	X	1	1	1	Input	Pull Down	rising edge
1	0	0	X	X	0	Output, full drive to 0	Disabled	Disabled
1	1	0	X	X	0	Output, full drive to 1	Disabled	Disabled
1	0	1	X	X	0	Output, reduced drive to 0	Disabled	Disabled
1	1	1	X	X	0	Output, reduced drive to 1	Disabled	Disabled
1	0	0	X	0	1	Output, full drive to 0	Disabled	falling edge
1	1	0	X	1	1	Output, full drive to 1	Disabled	rising edge
1	0	1	X	0	1	Output, reduced drive to 0	Disabled	falling edge
1	1	1	X	1	1	Output, reduced drive to 1	Disabled	rising edge

NOTES:
1. Applicable only on port AD.

Fuente: Motorola[14]

En la tabla 7 se detallan los puertos utilizados para la configuración del Procesador Modbus y para desarrollar la aplicación.

[14] Motorola, Inc. *PIM_9E128 Block Guide V1.00* . 04 Jun 2003

Tabla 7. Tabla de configuración de registros para los GPIO

Puerto	Función	E/S	Registros		Valor (Hex)
			Nombre	Descripción	
A[15]	Dirección de Esclavo	Entrada	**DDRA**	Registro de dirección de datos	0x0000
			PORTA	Registro de datos	---------
B[16]	Entradas Digitales	Entrada	**DDRB**	Registro de dirección de datos	0x0000
			PORTB	Registro de datos	---------
T[17]	Salidas Digitales	Salida	**DRTT**	Registro de dirección de datos	0xFFFF
			PERT	Registro de habilitación Pull-up	0x0000
			PTT	Registro de datos	--------
AD[18]	Configuración por hardware del Procesador Modbus	Entrada	**ATDDIEN1**	Registro de habilitación entradas digitales PAD0-PAD7	0xFFFF
			PERAD (Lo)	Registro de habilitación Pull-up	0x0000
			DDRAD (Lo)	Registro de dirección de datos	0x0000
A-B			**PUCR**	Registro de control Pull-up puertos A y B	0x0003

[15,13] Motorola.com/semiconductors *Multiplexed External Bus Interface (MEBI) Block User Guide* Module V3 2/2003 S12MEBIV3/D Rev. 3.00

[14] Motorola, Inc. *PIM_9E128 Block Guide V01.00* 04 Jun. 2003

[15] Motorola, Inc. *ATD_10B16C Block User Guide V02.07* 29 Aug. 2002

4.8 GRABACION DE DATOS EN MEMORIA FLASH

La función de usuario **0x64**, permite grabar la configuración del Procesador Modbus a partir de las localidades de memoria ROM: 0x4000. Para lograr este objetivo se crearon las funciones ***Flash_Write_Word*** y ***Flash_Erase_Sector***, las cuales permiten grabar datos en la memoria Flash.

La programación de la memoria Flash se controla por una máquina de estados mediante comandos[19]. La máquina de estados supervisa la secuencia de escritura de todos los comandos y verifica la validez de la secuencia de estos. Dicha máquina también es responsable de aplicar los voltajes apropiados al bloque de memoria Flash durante el tiempo requerido para grabar. Para esto requiere que la frecuencia del reloj de la FLASH se encuentre entre 150 KHz y 200 KHz. Esta frecuencia se obtiene del reloj del microcontrolador por intermedio de un *prescaler*. Los comandos válidos para grabar la memoria Flash se listan en la tabla 8.

Tabla 8. Comandos válidos para grabar en la memoria Flash

Comando	Nombre	Descripción
0x20	Programa	Programa una palabra (2 bytes)
0x40	Borra sector	Borra un sector
0x41	Borrado masivo	Borra un bloque completo

[19] Robb, S. *HCS12 NVM Guidelines*. App. Note AN2400/D Rev. 3, 07/2003. Freescale Semiconductor.

La memoria Flash se programa en unidades de palabras alineadas, o sea dos bytes al tiempo. Los datos de las palabras se escriben en direcciones pares, es decir, el bit 0 de la dirección debe ser cero (0).

La memoria Flash se borra en sectores de 1024 bytes o se borra el bloque completo. Los sectores inician en las direcciones: $xxx0, $xxx4, $xxx8 y $xxxC.

Para borrar y programar la memoria Flash se utilizan principalmente tres: el registro de estados, el registro de comandos y el registro del divisor de la señal de reloj.

- REGISTRO DIVISOR DEL RELOJ DE LA FLASH (**FCLKDIV**)
- REGISTRO DE ESTADOS DE LA FLASH (**FSTAT**)
- REGISTRO DE COMANDOS DE LA FLASH (**FCMD**)

La programación detallada de estos registros se puede consultar en el Anexo II.

4.8.1 SECUENCIAS DE COMANDOS PARA LA FLASH

Antes de grabar en la memoria Flash se debe haber borrado el sector o los sectores donde se va a escribir. Las secuencias de borrado y escritura deben estar precedidas de una secuencia de inicialización.

Secuencia de inicialización:

1. Si el bit FDIVLD está en cero (0), inicializar el registro FCKLDIV
2. Verificar que las banderas ACCERR y PVIOL en el registro FSTAT se han limpiado.

Secuencia de borrado:

1. Verificar que las banderas PVIOL y ACCERR en el registro FSTAT se han borrado.
2. Chequear que el buffer de comandos esté vacío (Bit CBEIF =1)
3. Escribir un dato cualquiera en la dirección de memoria flash a borrar.
4. Escribir en el registro de comandos, el comando de borrado (FCDM =0x30)
5. Escribir 1 en el bit CBEIF del registro FSTAT, para iniciar el comando.

Secuencia de escritura:

1. Verificar que las banderas PVIOL y ACCERR en el registro FSTAT se han borrado.
2. Chequear que el buffer de comandos esté vacío (Bit CBEIF =1)
3. Escribir la palabra en la dirección de memoria Flash especificada
4. Escribir en el registro de comandos, el comando de escritura (FCDM =0x20)
5. Colocar el bit CBEIF a 1 en el registro FSTAT, para lanzar el comando.

4.8.2 FUNCIONES PARA BORRAR Y GRABAR EN FLASH

Para llevar a cabo la escritura de los datos de configuración enviados desde la función de usuario 0x64 se programaron las siguientes funciones:

- FLASH1_Init( )
- Flash_Erase_Sector( )
- Flash_Write_Word( )
- DoOnStack( )

FLASH1_Init ( )

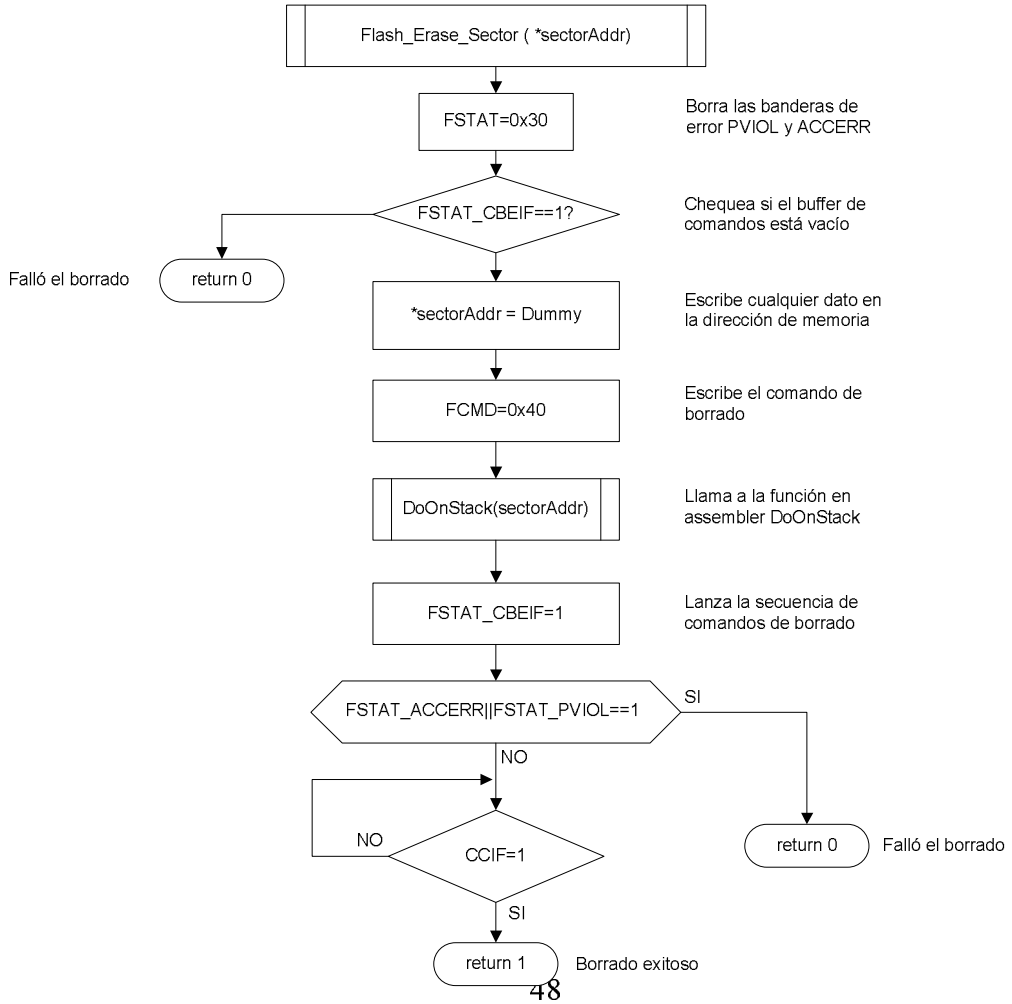
La función FLASH1_Init ( ) debe llamarse antes de utilizar las funciones de programación o de borrado de la FLASH. Esta función inicializa la velocidad del reloj de la FLASH en 200 KHz.

Flash_Erase_Sector ( )

La función Flash_Erase_Sector borra un sector entero (1024 Bytes) apuntado por la dirección especificada en el puntero **sectorAddr*.

En la figura 22 se muestra el diagrama de flujo de esta función.

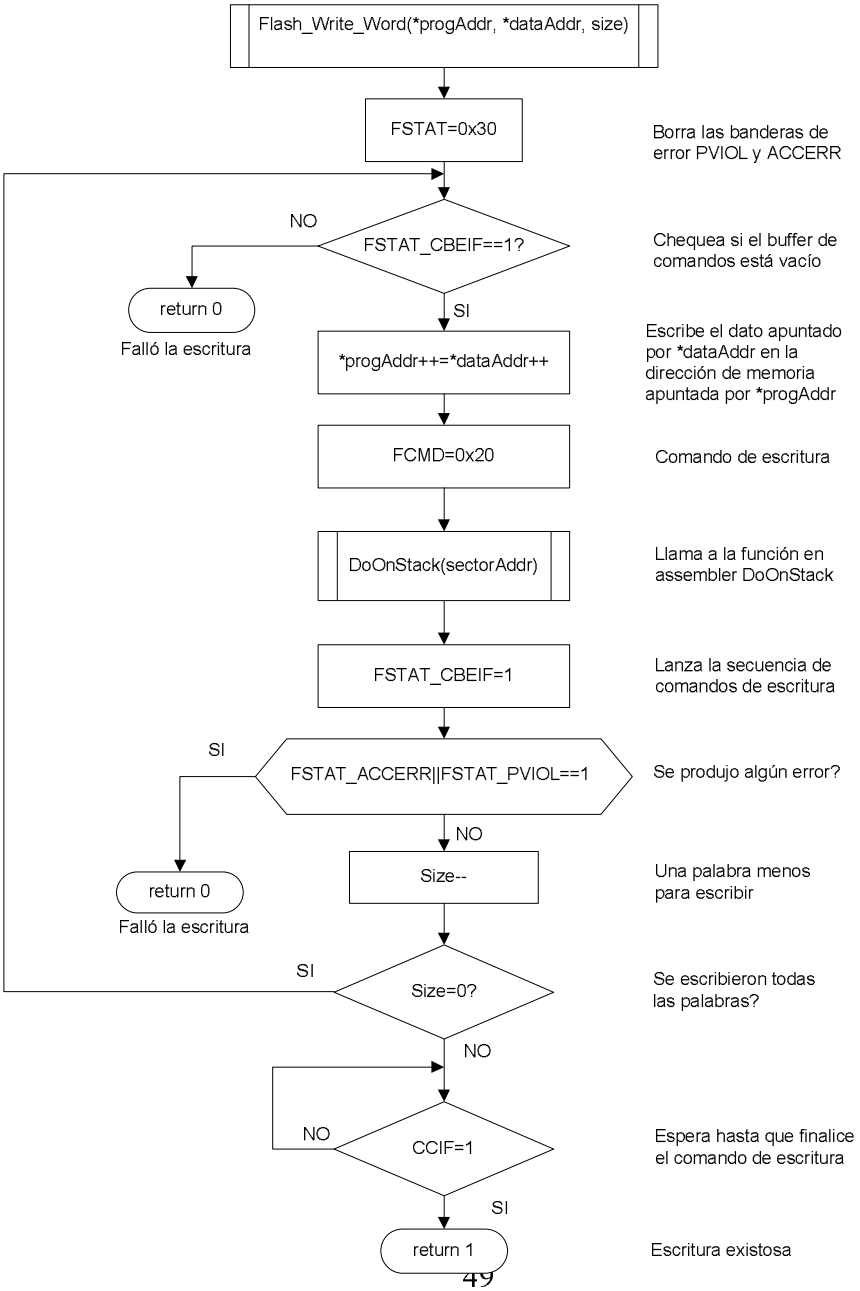
Figura 22. Diagrama de flujo de la función Flash_Erase_Sector ( )



Flash_Write_Word ( )

La función Flash_Write_Word escribe una palabra de 16 bits en la dirección de memoria FLASH apuntada por **progAddr*. En la figura 23 se muestra el diagrame de flujo de esta función.

Figura 23. Diagrama de flujo de la función Flash_Write_Word ( )



DoOnStack.asm

La idea de utilizar esta subrutina en ensamblador se encuentra en la Nota de Aplicación de Motorola AN2720/D[20]. El concepto consiste en crear una pequeña rutina en ensamblador que sea copiada en la pila de software (*Stack*) del microcontrolador, se ejecute desde allí, y luego sea removida. Esto permite que los pasos finales en los comandos de borrado/programación se ejecuten fuera de la RAM (en la pila) mientras que la FLASH está fuera del mapa de memoria. Esta rutina se puede utilizar para programar palabras en la Flash o para borrado de la misma.

[20] Jim Williams. Freescale Semiconductor, Inc. *A Utility for Programming Single FLASH Array HCS12 MCUs, with Minimum RAM Overhead.* Application Note AN2720/D Rev. 1.0 6/2004. Texas.

Capítulo 5

PROCESADOR MODBUS

Para el procesamiento de las tramas se programaron una serie de funciones con base en las especificaciones del protocolo MODBUS[21,22]. Estas funciones se programaron en Lenguaje C embebido, utilizando la herramienta de desarrollo Codewarrior[23].

Para el procesamiento de los mensajes Modbus, el programa se estructuró mediante funciones y subrutinas de atención a interrupciones en dos archivos Modbus.c y Modbus.h. En la figura 24 se muestra un diagrama de flujo general del software implementado.

Las funciones implementadas en el microcontrolador HCS12 se dividen en:

- Funciones Modbus
- Funciones de configuración
- Funciones de procesamiento de trama.
- Funciones de comunicación.

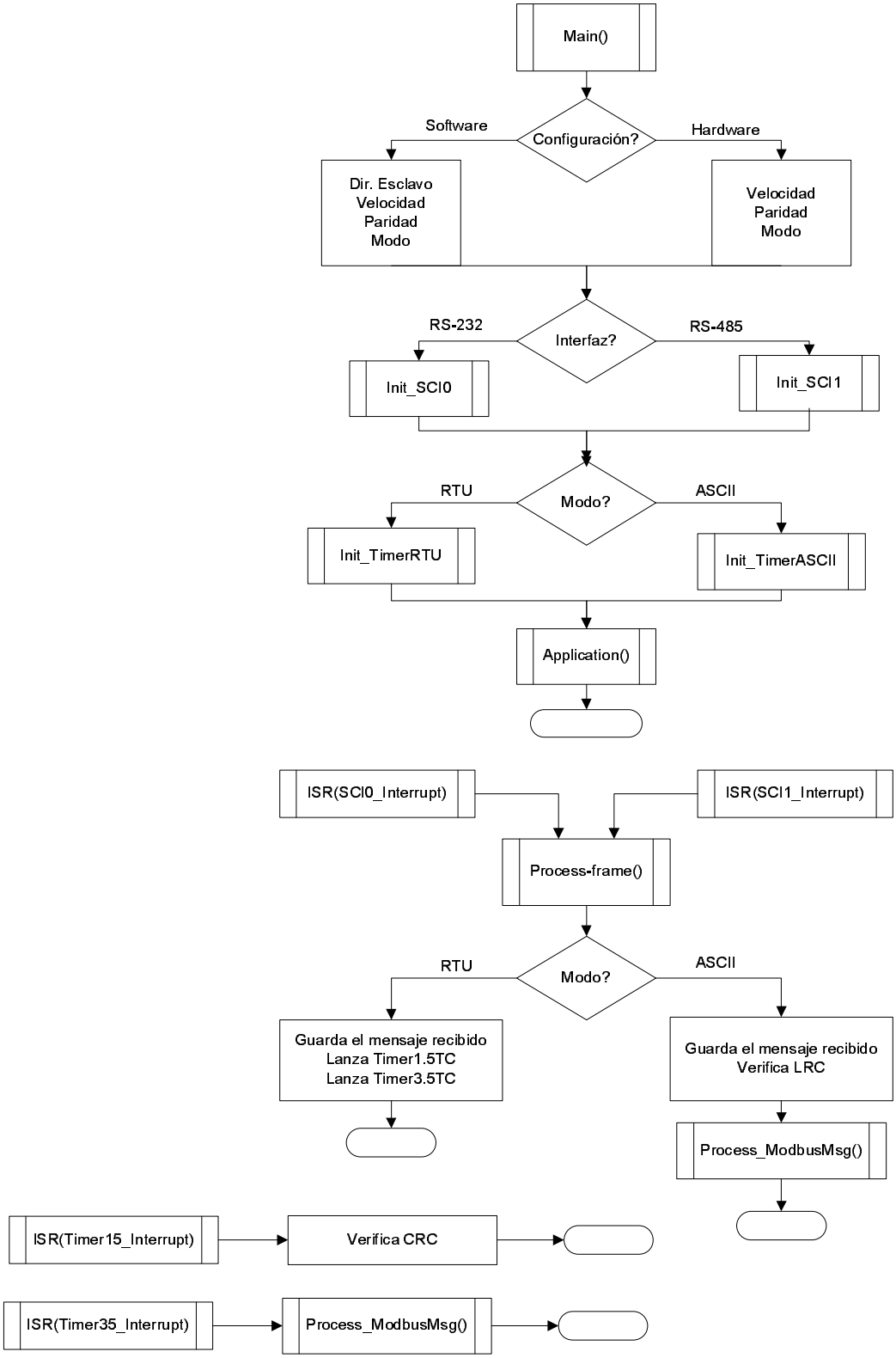
---

[21] Modbus.org. *MODBUS over Serial Line Specification & Implementation guide V 1.0* 12/02/2002

[22] Modicon, Inc. *Modicon Modbus Protocol Referencce Guide PI–MBUS–300 Rev. J* Junio 1996

[23] Metrowerks Inc.

Figura 24. Diagrama de flujo general del software.



5.1 FUNCIONES DE CONFIGURACIÓN

La configuración de los parámetros de comunicación del Procesador Modbus se puede realizar de dos maneras:

- Por hardware
- Por software

Para la configuración por hardware se implementó la función **Confg_hardware ( )**, la cual permite realizar la selección de parámetros por intermedio de de 8 micro-interruptores conectados a través de un buffer de aislamiento a la parte baja del puerto AD (PAD0 a PAD7) del Microcontrolador.

De acuerdo con la posición de cada micro-interruptor se pueden seleccionar los diferentes parámetros de configuración del Procesador Modbus:

- Interfaz: RS-232 o RS-485
- Paridad : par, impar o ninguna
- Rata de Baudios: 600, 1200, 2400, 4800, 9600 ó 19200
- Modo: ASCII o RTU
- Configuración después de reset: Por software o por hardware

En la tabla 9 se describen las posibles opciones de configuración.

Tabla 9. Opciones de configuración por hardware para el Procesador Modbus

Parámetro	Valor	Posición de los micro-interruptores							
		SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
Modo	RTU	OFF	X	X	X	X	X	X	X
	ASCII	ON	X	X	X	X	X	X	X
Interfaz	RS-232	X	OFF	X	X	X	X	X	X
	RS-485	X	ON	X	X	X	X	X	X
Rata de Baudios	19200	X	X	OFF	OFF	OFF	X	X	X
	9600	X	X	OFF	OFF	ON	X	X	X
	4800	X	X	OFF	ON	OFF	X	X	X
	2400	X	X	OFF	ON	ON	X	X	X
	1200	X	X	ON	OFF	OFF	X	X	X
	600	X	X	ON	OFF	ON	X	X	X
Paridad	Ninguna	X	X	X	X	X	OFF	OFF	X
	Par	X	X	X	X	X	ON	OFF	X
	Impar	X	X	X	X	X	OFF	ON	X
Configuración	Hardware	X	X	X	X	X	X	X	OFF
	Software	X	X	X	X	X	X	X	ON

Para la configuración por software se implementó la función **Confg_software ( )**. Esta función lee los parámetros guardados a partir de la posición de memoria Flash: 0x4000. Para la grabación de los parámetros en memoria Flash, se implementó la función de usuario 0x64, que a su vez llama a las funciones de borrado y escritura de la Flash. En la tabla 10 se muestran los parámetros de configuración por software y sus respectivas posiciones de memoria

Tabla 10. Opciones de configuración por software para el Procesador Modbus

Parámetro	Valor	Código	Posición de Memoria
Interfaz	RS-232	0x00	0x4000
	RS-485	0xFF	
Modo	RTU	0x00	0x4001
	ASCII	0xFF	
Dirección de Esclavo	1 a 247	0x01 a 0xF7	0x4002
Paridad	Ninguna	0x00	0x4003
	Par	0x10	
	Impar	0x01	
Rata de Baudios	19200	0x4B00	0x4004
	9600	0x2580	
	4800	0x12C0	
	2400	0x0960	
	1200	0x04B0	
	600	0x0258	

5.2 COMUNICACIÓN SERIAL ASÍNCRONA

5.2.1 GENERALIDADES

Para el envío y recepción de los mensajes del Procesador Modbus se utilizan las interfaces seriales asíncronas:

- RS 232: Para comunicación punto a punto
- RS 485: Para comunicación multipunto en una red Modbus estándar.

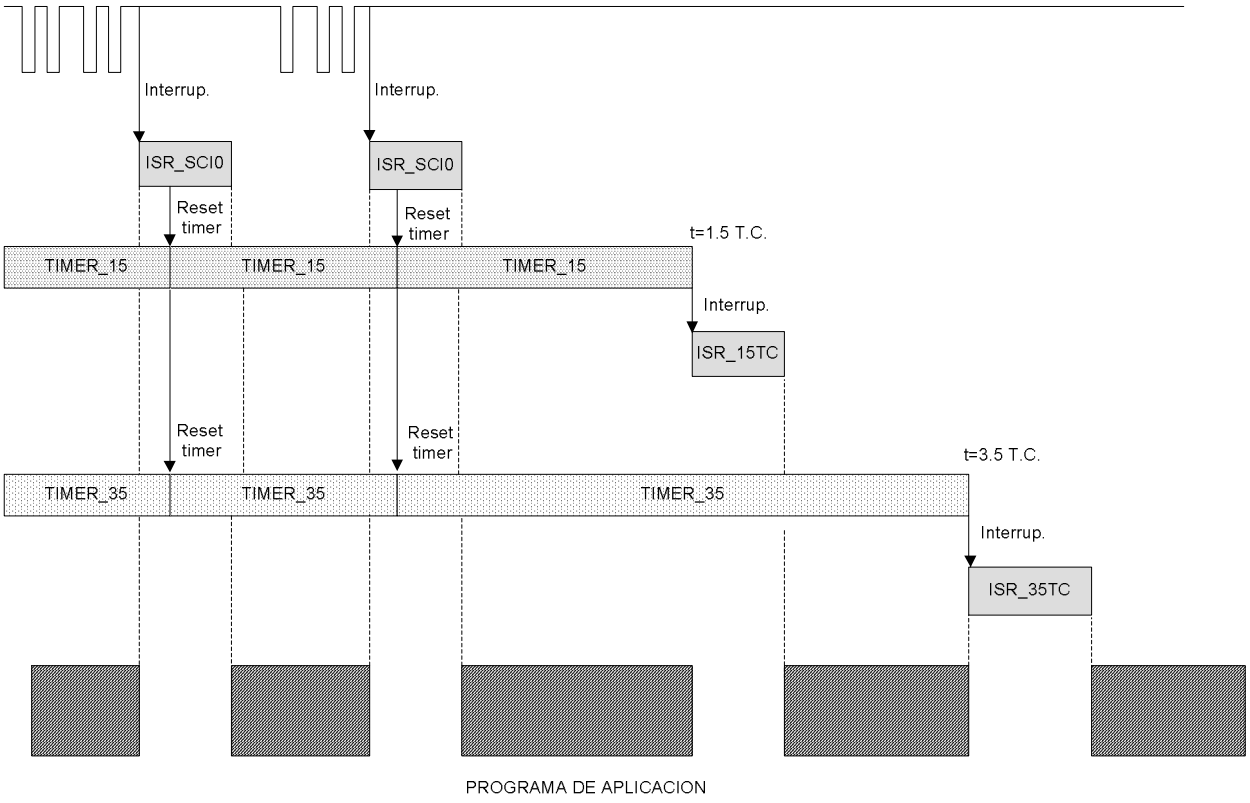
El sistema de desarrollo Adapt9S12E128 utilizado para la implementación del Procesador Modbus posee una interfaz RS-232 conectada al módulo SCI0 y una interfaz RS-485 conectada al módulo SCI21. Esto permite seleccionar cualquiera de los dos modos de transmisión dependiendo de las necesidades del usuario. Dicha selección se efectúa por software, el cual pregunta por el estado del pin de entrada PAD7 conectado al micro-interruptor SW8.

5.2.2 INTERRUPCIONES EN LA SCI

La comunicación entre el programa principal que ejecuta la aplicación y el procesamiento de las funciones Modbus, con las interfaces SCI se implementó utilizando el mecanismo de las interrupciones. De esta forma el procesador puede ejecutar diversas tareas y sólo cuando se le envía una trama Modbus, se interrumpe la ejecución normal del programa y se inicia el procesamiento del mensaje.

En la figura 25 se muestra un diagrama que ilustra el proceso de manejo de interrupciones en el procesador Modbus en el modo de transmisión RTU.

Figura 25. Esquema de interrupciones de la SCI en modo RTU



5.2.3 TRANSMISIÓN Y RECEPCIÓN DE DATOS

TRANSMISIÓN DE DATOS POR LA SCI

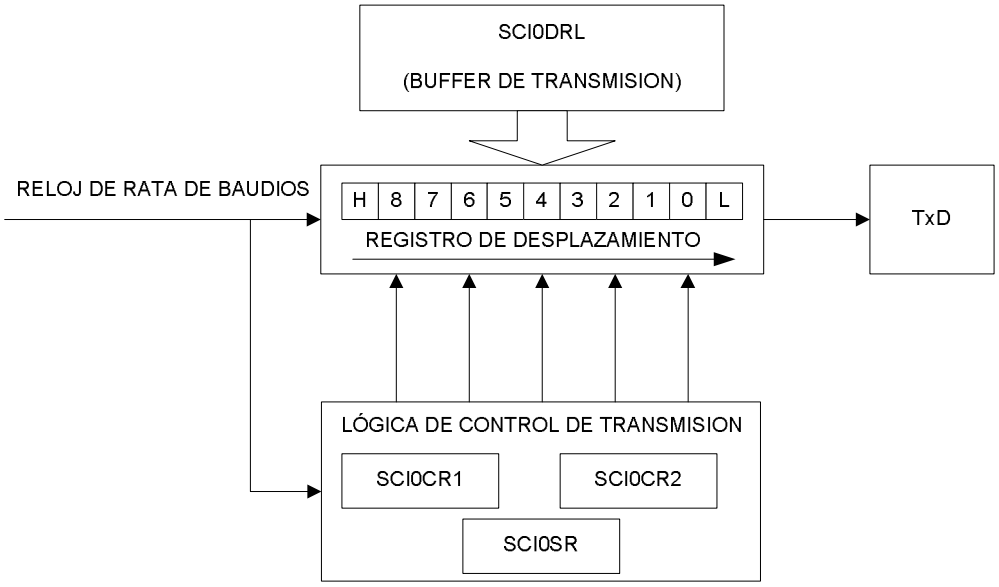
El transmisor de la SCI funciona de la misma forma que una UART (*Universal Asynchronous Receiver/Transmitter*). La función de transmisión se habilita por el bit TE en el registro SCI0CR2. Cuando se habilita el transmisor automáticamente envía un preámbulo de unos (Estado de línea ocupada).

Después de habilitado el transmisor, los bytes (caracteres) escritos en el registro de datos SCI0DRL, se moverán al registro de desplazamiento interno. El registro de

desplazamiento agrega los bits de inicio y parada a los datos y desplaza serialmente el carácter al pin de transmisión TxD.

Cada vez que un carácter se mueve del registro de datos al registro de desplazamiento para transmisión, la bandera TDRE se coloca en uno (1). Esto indica que el hardware está listo para recibir otro carácter. La bandera TC es puesta a uno cuando la transmisión se completa. La figura 26 ilustra el proceso de transmisión.

Figura 26. Diagrama de bloques de la función de transmisión de la SCI



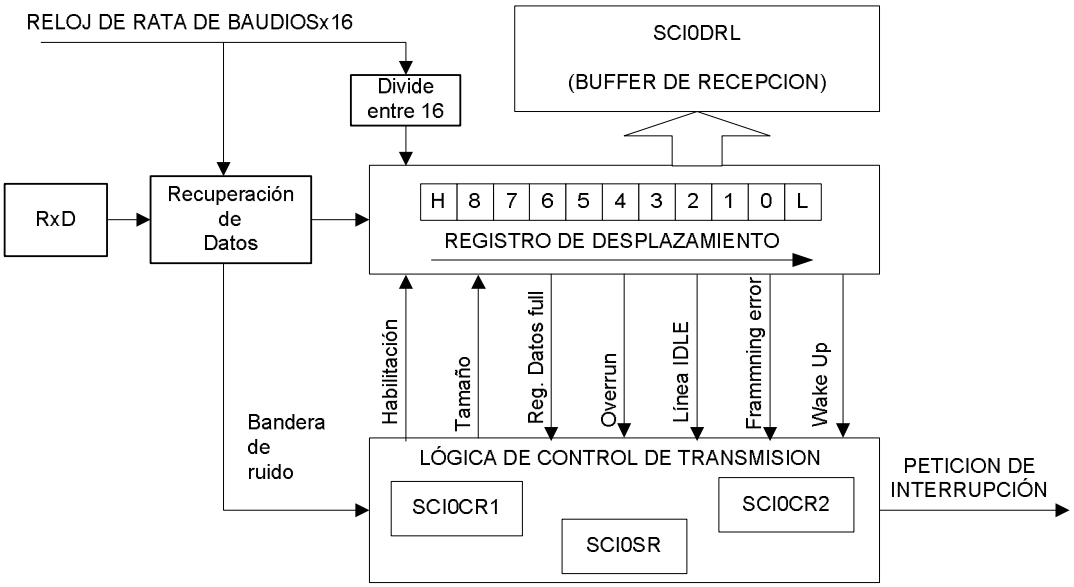
Fuente: *Technician's Guide to HC6811 Microcontroller*

El bit TE (*Transmit Enable*) del registro SCI0CR2 se utiliza para habilitar la función de transmisión de la SCI, la cual continúa con las subsecuentes escrituras en el registro SCI0DRL.

RECEPCIÓN DE DATOS POR LA SCI

El receptor de la SCI, también funciona de la misma forma que una UART (*Universal Asynchronous Receiver/Transmitter*). La figura 27 ilustra el proceso de recepción.

Figura 27. Diagrama de bloques de la función de recepción de la SCI



Fuente: *Technician's Guide to HC6811 Microcontroller*

La función de recepción se habilita por el bit RE en el registro SCI0CR2. El receptor automáticamente inicia buscando el bit de inicio. Después que el bit de inicio se detecta, el carácter se desplaza dentro del registro de desplazamiento, un bit a la vez.

Cuando el bit de parada se detecta, el carácter recibido se mueve al registro de datos SCI0DRL y las banderas se actualizan. Cada vez que un carácter se mueve del registro de desplazamiento al registro de datos SCI0DRL, la bandera RDRF se coloca

en uno (1). Esto indica al usuario que el carácter en el registro SCI0DRL necesita ser leído para evitar un error de sobre escritura.

El bit de habilitación de interrupción del receptor (RIE), por su parte, se utiliza para habilitar la interrupción de la SCI cuando el registro de datos está lleno. Cuando RIE = 1, se produce una petición de interrupción cuando el bit RDFR en el registro SCI0SR se coloca en uno (1).

5.2.4 FUNCIONES DE TRANSMISIÓN Y RECEPCIÓN DE LA SCI

En el Procesador Modbus se programaron las siguientes funciones para el envío y recepción de datos, a través de la interfaz RS-232 o RS-485:

- ***Init_SCI0 ( )*** : Inicialización de la SCI0 conectada a la interfaz RS-232
- ***Init_SCI1 ( )*** : Inicialización de la SCI1 conectada a la interfaz RS-485
- ***SCIRxByte ( )*** : Función para la transmisión de datos
- ***SCITxByte ( )*** : Función para la recepción de datos
- ***disableRx ( )*** : Función para deshabilitar la recepción cuando se transmite en modo half-dúplex (RS-485).
- ***enableRx ( )*** : Función para deshabilitar la transmisión y habilitar recepción cuando se transmite en modo half-dúplex (RS-485).

En la figura 28 se muestra un listado de las funciones implementadas en el Procesador Modbus.

Figura 28. Funciones en lenguaje c para la transmisión y recepción de datos

```
/*Interfaz RS-232 */

 void Init_SCI0(void)
{
  SCI0SR1;                  // Borra banderas de interrupción
  SCI0BD = Baud_Rate;       // Selecciona rata de Baudios BpS
  SCI0CR2 = 0x2C;           // Habilita trasmisor, Habilita receptor,
                            // y habilita interrupción por receptor lleno
  __EI();                   // Habilita interrupciones globales
}

/*Interfaz RS-485 */

 void Init_SCI1(void)
{
  SCI1SR1;                 // Borra banderas de interrupción
  SCI1BD  = Baud_Rate;     // Selecciona rata de Baudios BpS
  SCI1CR2 = 0x2C;          // Habilita trasmisor, Habilita receptor,
                           // y habilita interrupción por receptor lleno
  __EI();                  // Habilita interrupciones globales
}

void disableRx (void)

{
  Bit2_SetVal();         // PE5=1  deshabilita   recepción
  Bit3_SetVal();         // PE6=1  habilita transmisión
}

void enableRx (void)

 {
  Bit3_ClrVal();         // PE6=0  deshabilita transmisión
  Bit2_ClrVal();         // PE5=0  habilita recepción
 }

void SCIRxByte(char *dato)

{
   if (Interface==0)
         *dato=SCI0DRL;
   else
        *dato=SCI1DRL;
}

void SCITxByte (unsigned char caracter)

{
 if (Interface==0)

 {
  while (!(SCI0SR1&0x80));  // Espera que el buffer de
                            // transmisión se vacíe
  SCI0DRL = caracter;
 }

 else

 {
  while (!(SCI1SR1&0x80));   // Espera que el buffer de
                            //  transmisión se vacíe
  SCI1DRL = caracter;
 }

 }                                                              Modbus.c
```

5.3 FUNCIONES PARA EL CONTROL DE ERRORES

Para el control de errores en loa bits de las tramas Modbus recibidas y enviadas se utilizan dos métodos diferentes:

- Chequeo de redundancia cíclica (CRC) para el Modo RTU
- Cheque de redundancia Longitudinal (LRC) para el Modo ASCII

5.3.1 CHEQUEO DE REDUNDANCIA CÍCLICA (CRC)

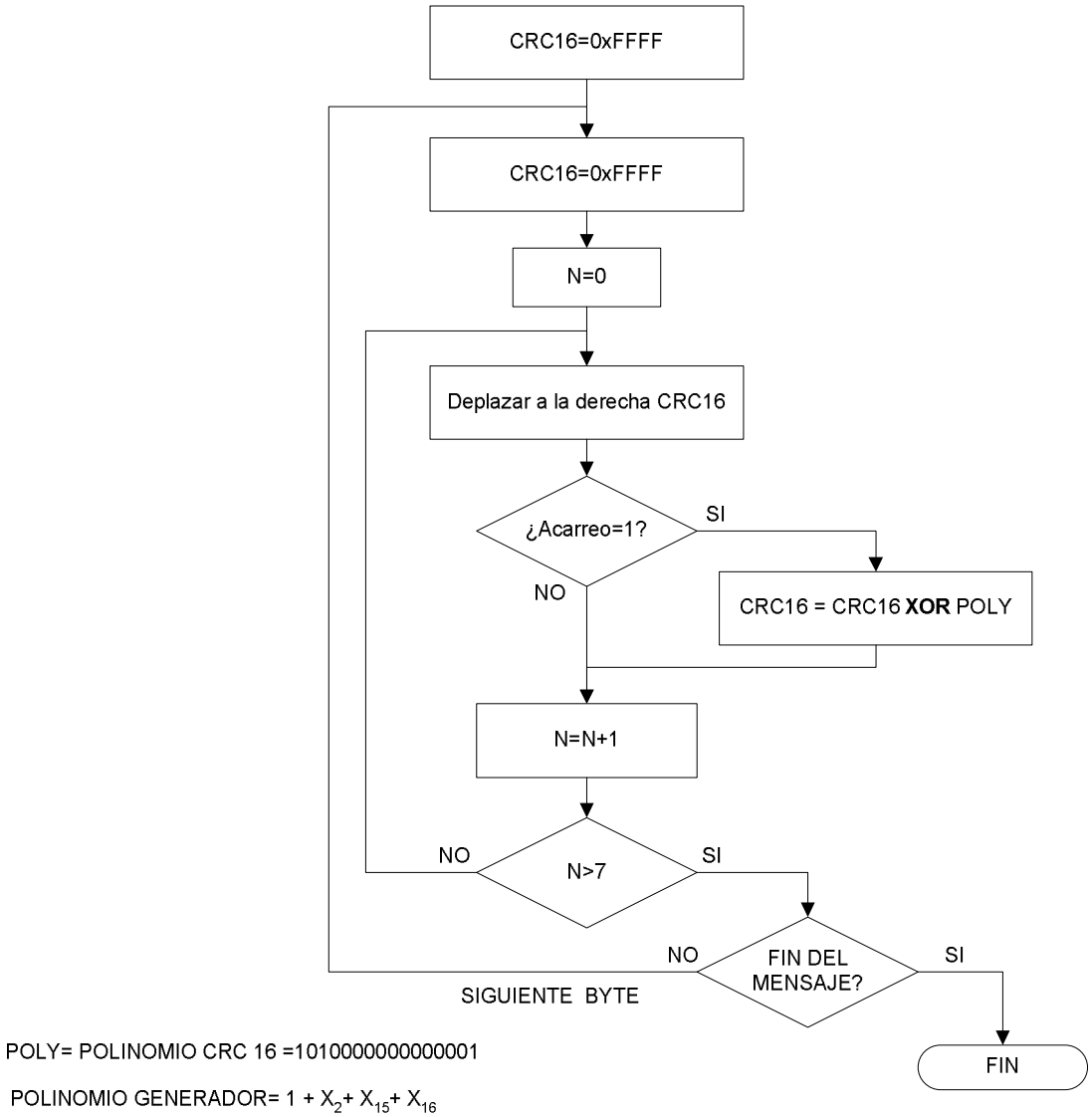
El Control de Redundancia Cíclica (CRC) es un campo de dos bytes. El CRC se calcula en el maestro Modbus y lo añade al mensaje. El procesador Modbus (Esclavo) recalcula el CRC durante la recepción del mensaje y compara el valor calculado con el campo de CRC recibido. Si no son iguales los dos valores se da un error como resultado.

El calculo de CRC se empieza cargando un registro de 16 bits todo a unos. El proceso continua aplicando una XOR sucesivamente entre los bytes de 8 bits del mensaje y el contenido actual del registro. Solo se usan los 8 bits de datos de cada carácter para la generación del CRC. No se aplican el bit de inicio, el bit de parada ni los bits de paridad.

El resultado se desplaza una vez hacia el bit menos significativo (LSB) y se coloca un cero la posición del bit mas significativo (MSB) desplazado. Luego se analiza el bit que se desborda del registro, si es 1 el registro realiza una XOR con un valor fijo denominando polinomio generador. Si es 0 no se realiza la operación.

Este proceso se repite hasta realizar ocho desplazamientos. Después del último (octavo) desplazamiento, el siguiente carácter (8 bits) realiza una XOR con el contenido del registro, y el proceso se repite ocho veces más como se ha descrito anteriormente. El valor final del registro después de haber aplicado todos los caracteres del mensaje es el valor de CRC. En la Figura 29 se muestra el algoritmo para el cálculo del CRC.

Figura 29. Algoritmo para el cálculo del CRC



Sin embargo, en lenguaje C, para la generación de CRC se precargan dos matrices con todos los valores posibles de CRC, indexadas con la función incremento del buffer de mensaje. Una matriz contiene los 256 posibles valores de CRC para el byte alto del campo de CRC de 16 bits, y la otra matriz contiene los valores del byte menos significativo.

Indexando el CRC de esta manera se consigue una mayor eficiencia que si se calculara un valor de CRC para cada nuevo carácter del mensaje. En la figura 30 se muestra la implementación del CRC utilizando los valores precargados del vector CRC (Fig. 31)

Figura 30. Función para el cálculo del CRC

```
char CRC16(unsigned char *p, char check,char dataLen)
{
 unsigned char *m = p;
 unsigned Index;
 char Len=dataLen;
 highCRC = 0xff ; // Byte alto del CRC inicializado
 lowCRC  = 0xff ; // Byte bajo del CRC inicializado

   while (Len--) // Pasa a través de todo el  buffer de mensajes
        {
           Index = (lowCRC)^(*p++); // calcula el CRC
           lowCRC = (highCRC)^(Vector_CRCHi[Index]) ;
           highCRC = Vector_CRCLo[Index] ;
        }

  if (check==1)    // Chequeo de la CRC para datos recibidos:
       if ((*(m+dataLen+1)==highCRC)&&(*(m+dataLen)==lowCRC))
       return 1;
       else
       return 0;
  else
  return 1;
}                                                                    Modbus.c
```

Figura 31. Vector de valores del CRC

```
#pragma CONST_SEG CRC_VECTOR

/* Tabla de valores CRC para el byte alto */

extern const unsigned char Vector_CRCHi[] =

{

0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40

} ;

/* Tabla de valores CRC para el byte bajo*/

extern const unsigned char Vector_CRCLo[] =

{

0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40

};

#pragma CONST_SEG DEFAULT
                                                                                Modbus.h
```

5.3.2 CHEQUEO DE REDUNDANCIA LONGITUDINAL (LRC)

El Control de Redundancia Longitudinal (LRC) es un campo de un byte, que contiene un valor binario de 8 bits. El LRC lo calcula el maestro y lo añade al mensaje. El procesador Modbus (Esclavo) recalcula el LRC durante la recepción del mensaje y compara el valor calculado con el campo de LRC recibido. Si no son iguales los dos valores se da un error como resultado.

El LRC se calcula sumando sucesivamente los bytes (8 bits) del mensaje, descartando los acarreos, y complementando a dos el resultado. El LRC es un campo de 8 bits, cada nueva suma de un carácter que de como resultado un valor mayor de 255 decimal provoca un desbordamiento hacia el noveno bit. Como no existe el noveno bit en el registro del LRC, el acarreo se descarta automáticamente.

El procedimiento para la generación de LRC es:

1. Sumar todos los bytes del mensaje, excluyendo el carácter de inicio (:) y los dos caracteres finales CR-LF. Dejar el resultado en un campo de 8 bits, y descartando todos los acarreos.
2. Restar el valor resultado a FF hex. (todo a 1's), para calcular el complemento a uno.
3. Sumar 1 al resultado para realizar el complemento a 2.

En la figura 32 se muestra la implementación del LRC utilizando el procedimiento anteriormente descrito.

Figura 32. Función en lenguaje c para el cálculo del LRC

```
unsigned char LRC(unsigned char *ptrMsg,unsigned short DataLen)

 {
  unsigned char byteLRC=0;
  while (DataLen--)
        byteLRC+=*ptrMsg++;
  return((unsigned char)(-((char)byteLRC)));
 }
```

5.4 FUNCIONES DE PROCESAMIENTO DE LAS TRAMAS MODBUS

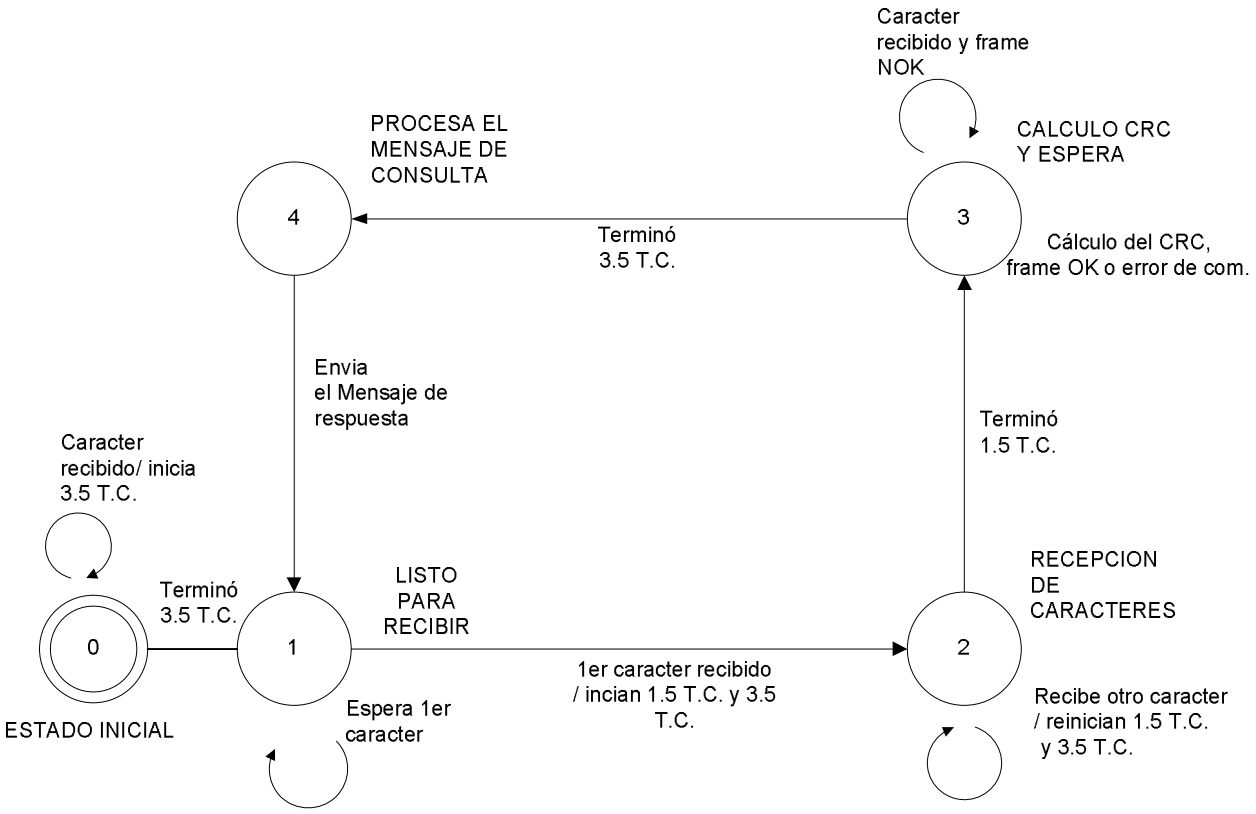
Para el procesamiento de los mensajes Modbus se tuvo en cuenta que en el Procesador se implementaron los dos modos de transmisión: Modbus RTU y Modbus ASCII. Cada uno tiene sus características propias en cuanto al procesamiento de la trama, sin embargo una vez procesados los mensajes las funciones Modbus tienen la misma estructura.

5.4.1 MODBUS RTU

La implementación de las rutinas de procesamiento de las tramas Modbus RTU implican el manejo de varias interrupciones debido a los temporizadores de 1.5 T.C y 3.5 T.C y a la recepción de los caracteres en la SCI.

En la figura 33 se describe el diagrama de estados del modo de transmisión RTU.

Figura 33. Diagrama de estados del modo de transmisión RTU



El procesamiento de las tramas en Modbus RTU comienza con un intervalo de al menos 3.5 T.C, al final del cual se espera para la llegada del primer carácter.

Luego de recibido el primer carácter, se almacena en el buffer de recepción de mensajes MenRx [] y se lanzan los temporizadores de 1.5 T.C. y 3.5 T.C.

Posteriormente se continúan recibiendo y almacenando los demás caracteres mientras no termine el tiempo de 1.5 T.C. Por último, al expirar 1.5 T.C. se realiza el chequeo CRC de la trama y si está libre de errores se espera el término de 3.5 T.C. para procesar el mensaje completo. En la figura 34 se muestra el diagrama de flujo de la ISR que procesa el mensaje RTU y en la figura 35 las ISR del Timer.

Figura 34. Rutina de atención a la interrupción de la SCI: Modo RTU

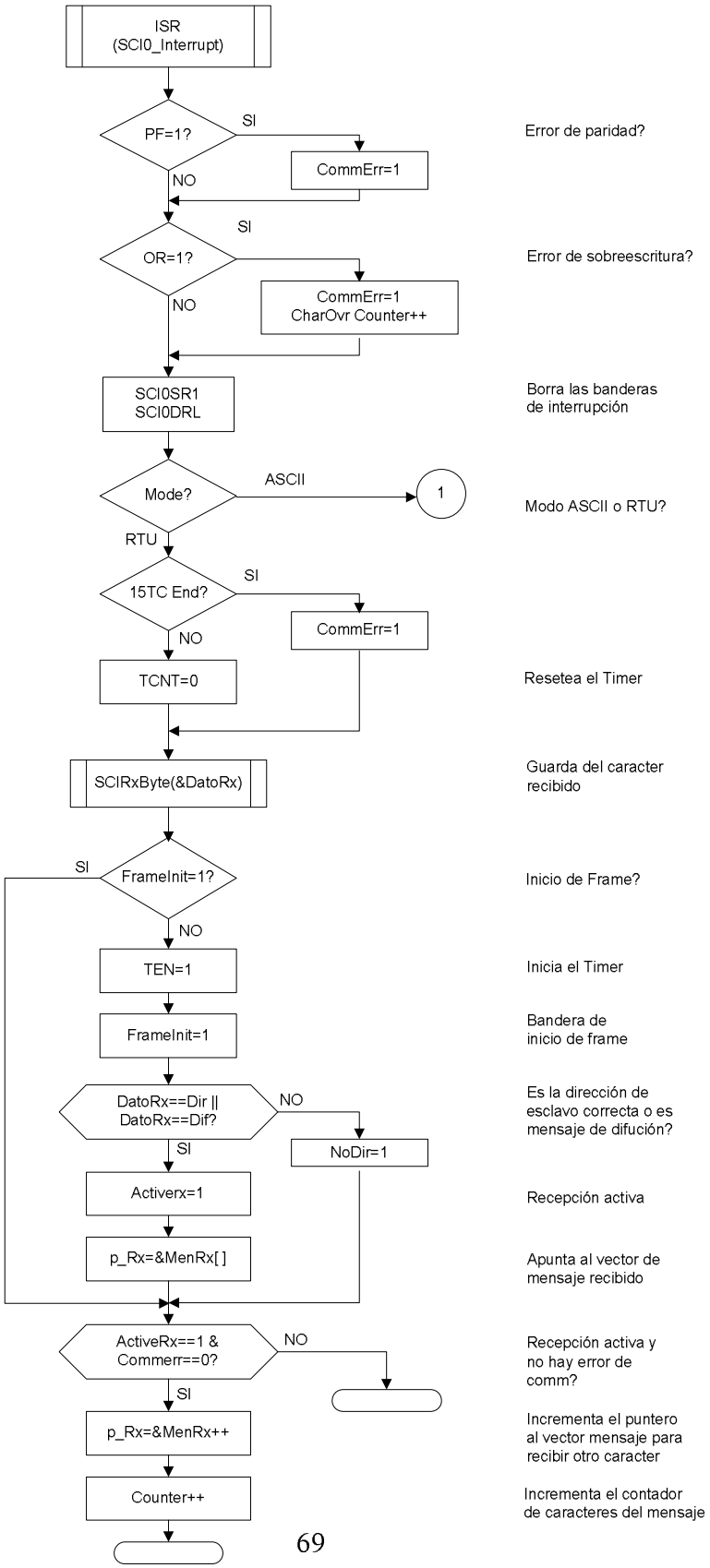
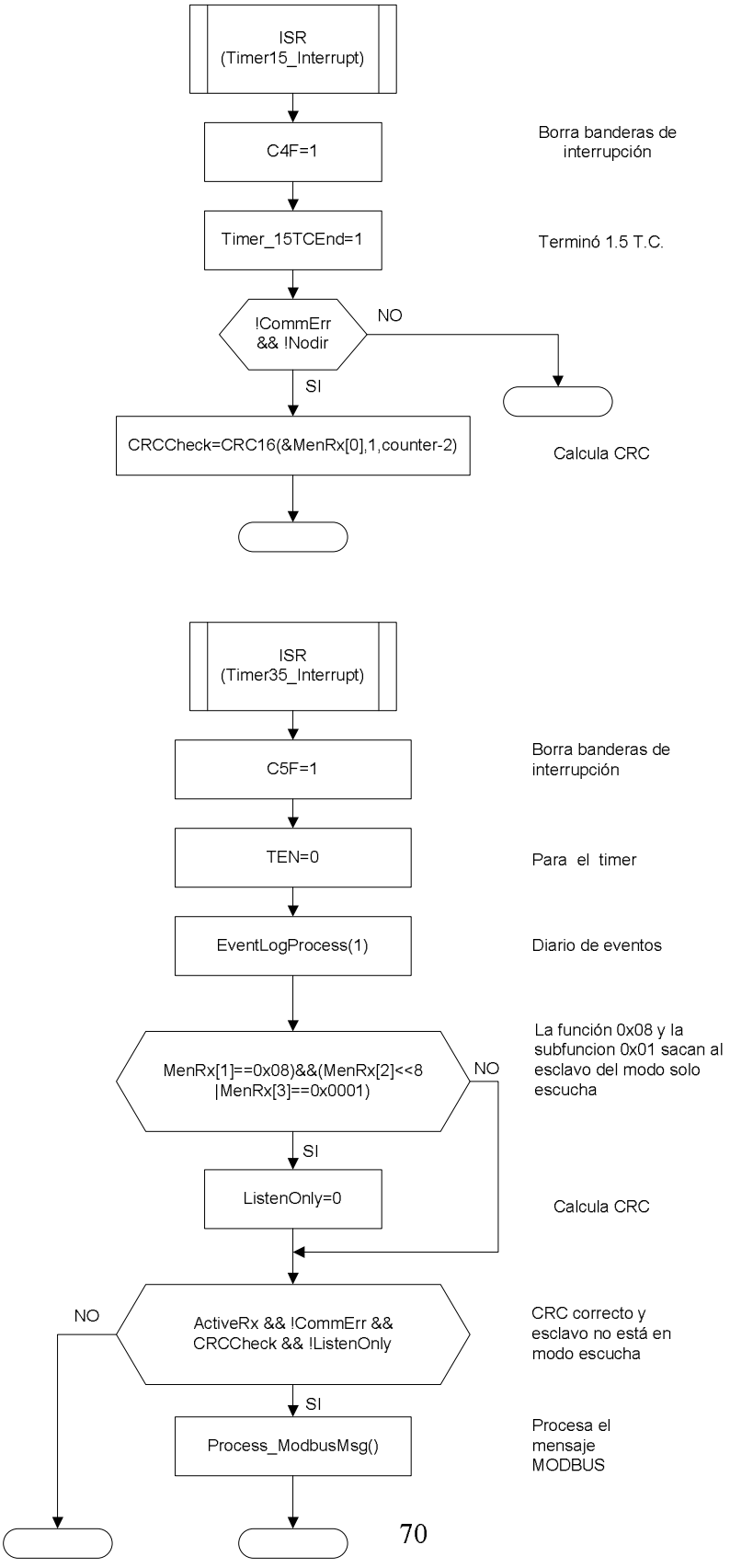


Figura 35. Rutina de atención a las interrupciones del Timer: Canales 4 y 5

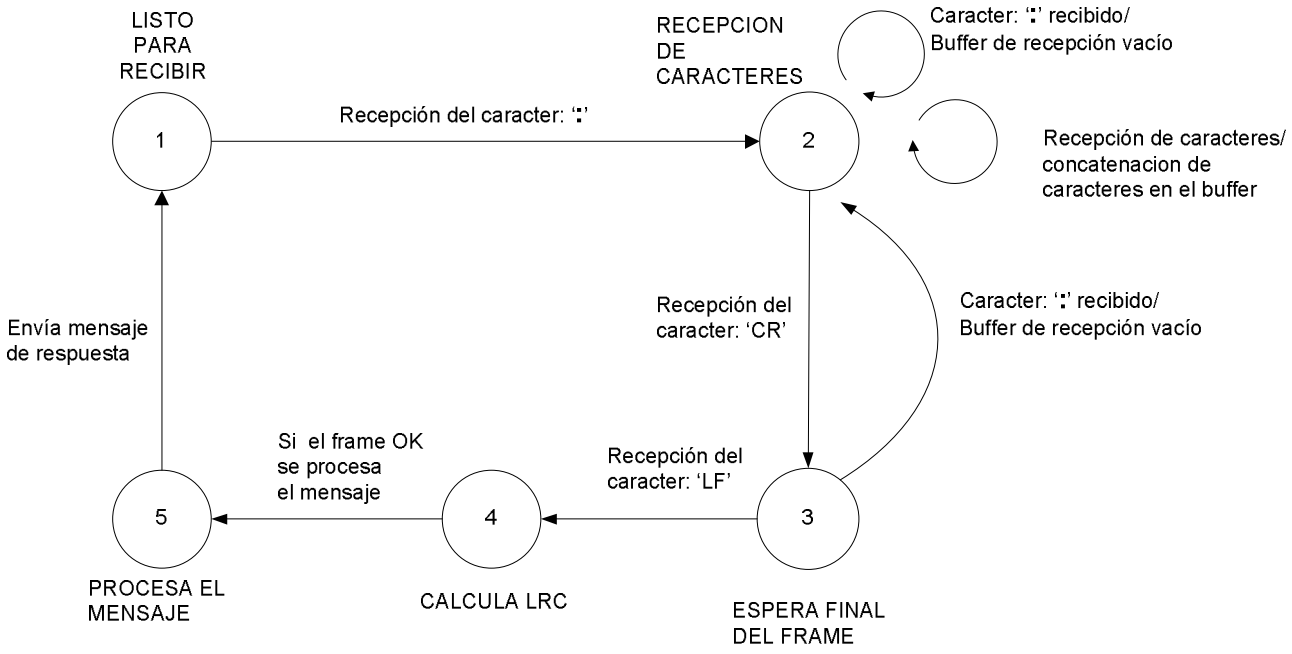


5.4.2 MODBUS ASCII

El procesamiento de las tramas Modbus ASCII implica utilizar la interrupción provocada por la SCI al recibir un carácter y la interrupción producida por un canal del *Timer* para contabilizar el *Time out* de 1 seg.

En la figura 36 se describe el diagrama de estados del modo de transmisión ASCII

Figura 36. Diagrama de estados Modo de transmisión ASCII



En las figuras 37 y 38 se describe en diagramas de flujo, la implementación de la rutina de atención a la interrupción provocada por la SCI0, la cual se encarga de procesar las tramas Modbus ASCII.

Figura 37. Rutina de atención a la interrupción de la SCI: Modo ASCII (1)

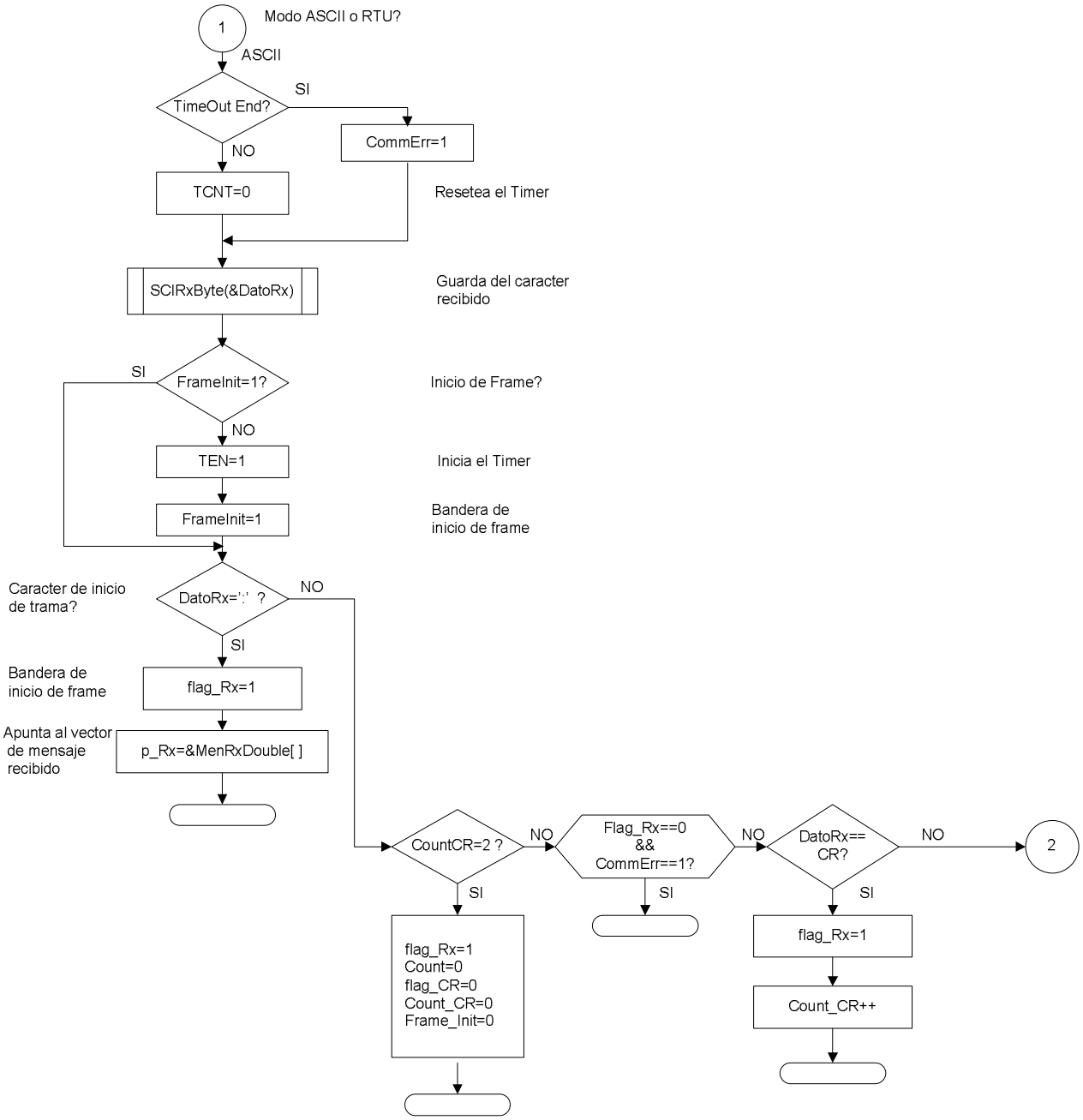
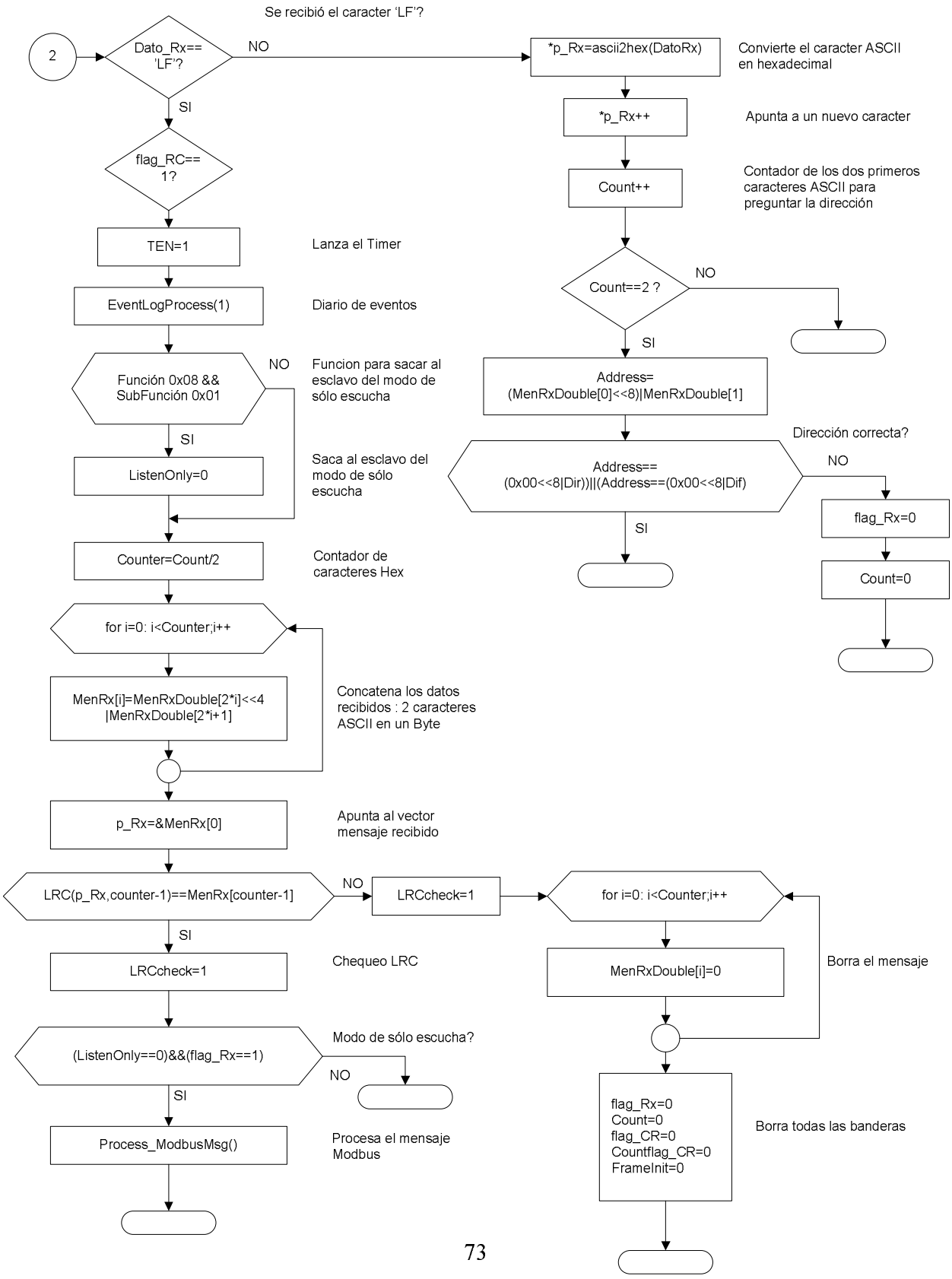


Figura 38. Rutina de atención a la interrupción de la SCI: Modo ASCII (2)



Capítulo 6

APLICACION

El propósito de la aplicación es el de comprobar la funcionalidad del Procesador Modbus en el control y/o monitoreo de procesos físicos. Para esto se creó un módulo de interfaz que permite el manejo de las siguientes señales:

- Ocho entradas digitales con niveles TTL
- Ocho salidas digitales con niveles TTL
- Dos salidas analógicas de 0-5V
- Dos entradas analógicas de 0-5V

La conexión de las entradas y salidas digitales al microcontrolador se realizó a través de los drivers 74HC74244 para lograr un manejo adecuado de las corrientes. De la misma forma, la salida del DAC se encuentra conectada en la tarjeta de desarrollo a un amplificador operacional que permite una mayor corriente de salida.

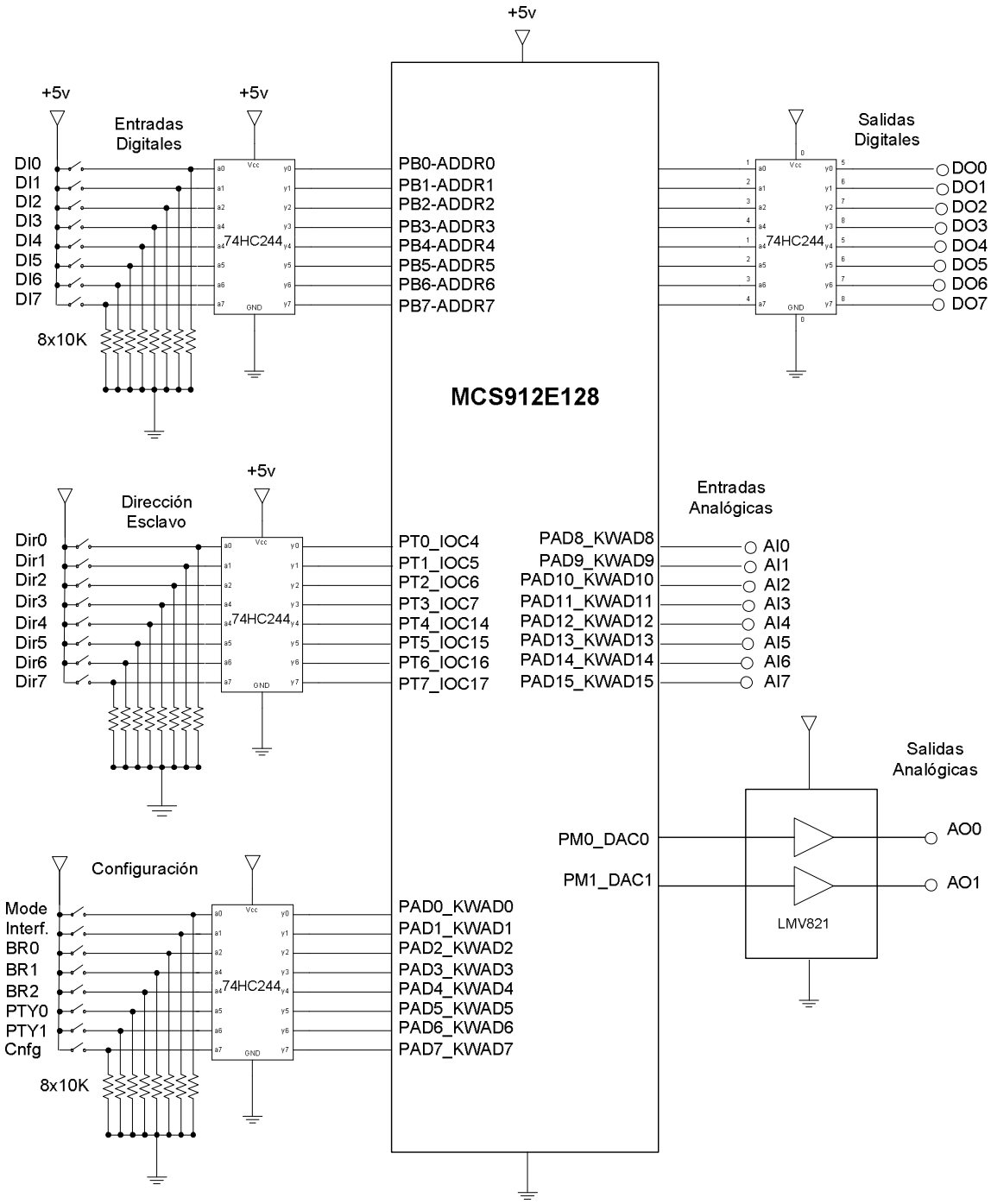
La distribución de los pines del Microcontrolador, utilizados para la implementación de la aplicación, se lista en la tabla 11.

Tabla 11. Distribución de los pines del MCS912E128 utilizados en la aplicación

Pin		Descripción	Función	Nombre	Dirección	
					Memoria	bit
H1	13	PT0_IOC4	Salida digital 0	DO0	0x2000	bit0
	12	PT1_IOC5	Salida digital 1	DO1		bit1
	11	PT2_IOC6	Salida digital 2	DO2		bit2
	10	PT3_IOC7	Salida digital 3	DO3		bit3
	9	PT4_IOC14	Salida digital 4	DO4		bit4
	8	PT5_IOC15	Salida digital 5	DO5		bit5
	7	PT6_IOC16	Salida digital 6	DO6		bit6
	6	PT7_IOC17	Salida digital 7	DO7		bit7
	22	PAD0_KWAD0	Modo: ASCII/RTU	Mode	PTADLo	
	23	PAD1_KWAD1	Interfaz: RS-232/485	Interface		
	24	PAD2_KWAD2	Rata Baudios (bit0)	BR0		
	25	PAD3_KWAD3	Rata Baudios (bit1)	BR1		
	29	PAD4_KWAD4	Rata Baudios (bit2)	BR2		
	28	PAD5_KWAD5	Paridad (bit0)	PTY0		
	27	PAD6_KWAD6	Paridad (bit1)	PTY1		
	26	PAD7_KWAD7	Configuración:HW/SW	CFG		
	14	PM1_DAC1	Salida analógica	AO0	0x2002	
	15	PM0_DAC0	Salida analógica	AO1	0x2004	
H2	16	PB0_ADDR0	Entrada digital 0	DI0	0x2100	bit0
	15	PB1_ADDR1	Entrada digital 1	DI1		bit1
	14	PB2_ADDR2	Entrada digital 2	DI2		bit2
	13	PB3_ADDR3	Entrada digital 3	DI3		bit3
	12	PB4_ADDR4	Entrada digital 4	DI4		bit4
	11	PB5_ADDR5	Entrada digital 5	DI5		bit5
	10	PB6_ADDR6	Entrada digital 6	DI6		bit6
	9	PB7_ADDR7	Entrada digital 7	DI7		bit7
	22	PAD8_KAW8	Entrada analógica 0	AI0	0x2102	
	23	PAD9_KAW9	Entrada analógica 1	AI1	0x2104	
	24	PAD10_KAW10	Entrada analógica 2	AI2	-	
	25	PAD11_KAW11	Entrada analógica 3	AI3	-	
	29	PAD12_KAW12	Entrada analógica 4	AI4	-	
	28	PAD13_KAW13	Entrada analógica 5	AI5	-	
	27	PAD14_KAW14	Entrada analógica 6	AI6	-	
	26	PAD15_KAW15	Entrada analógica 7	AI7	-	
	8	PA0_ADDR0	Dirección esclavo (bit0)	Dir0	PORTA	
	7	PA1_ADDR1	Dirección esclavo (bit1)	Dir1		
	6	PA2_ADDR2	Dirección esclavo (bit2)	Dir2		
	5	PA3_ADDR3	Dirección esclavo (bit3)	Dir3		
	4	PA4_ADDR4	Dirección esclavo (bit4)	Dir4		
	3	PA5_ADDR5	Dirección esclavo (bit5)	Dir5		
	2	PA6_ADDR6	Dirección esclavo (bit6)	Dir6		
	1	PA7_ADDR7	Dirección esclavo (bit7)	Dir7		

En la figura 39 se muestra el diagrama esquemático de conexiones del módulo de interfaz al microcontrolador

Figura 39. Diagrama esquemático del módulo de interfaz para la aplicación



Para realizar pruebas de campo se construyeron dos módulos adicionales: Un módulo de entrenamiento y un módulo de acondicionamiento de señales.

El módulo de entrenamiento permite interactuar con el Procesador Modbus para el accionamiento y visualización de las siguientes señales:

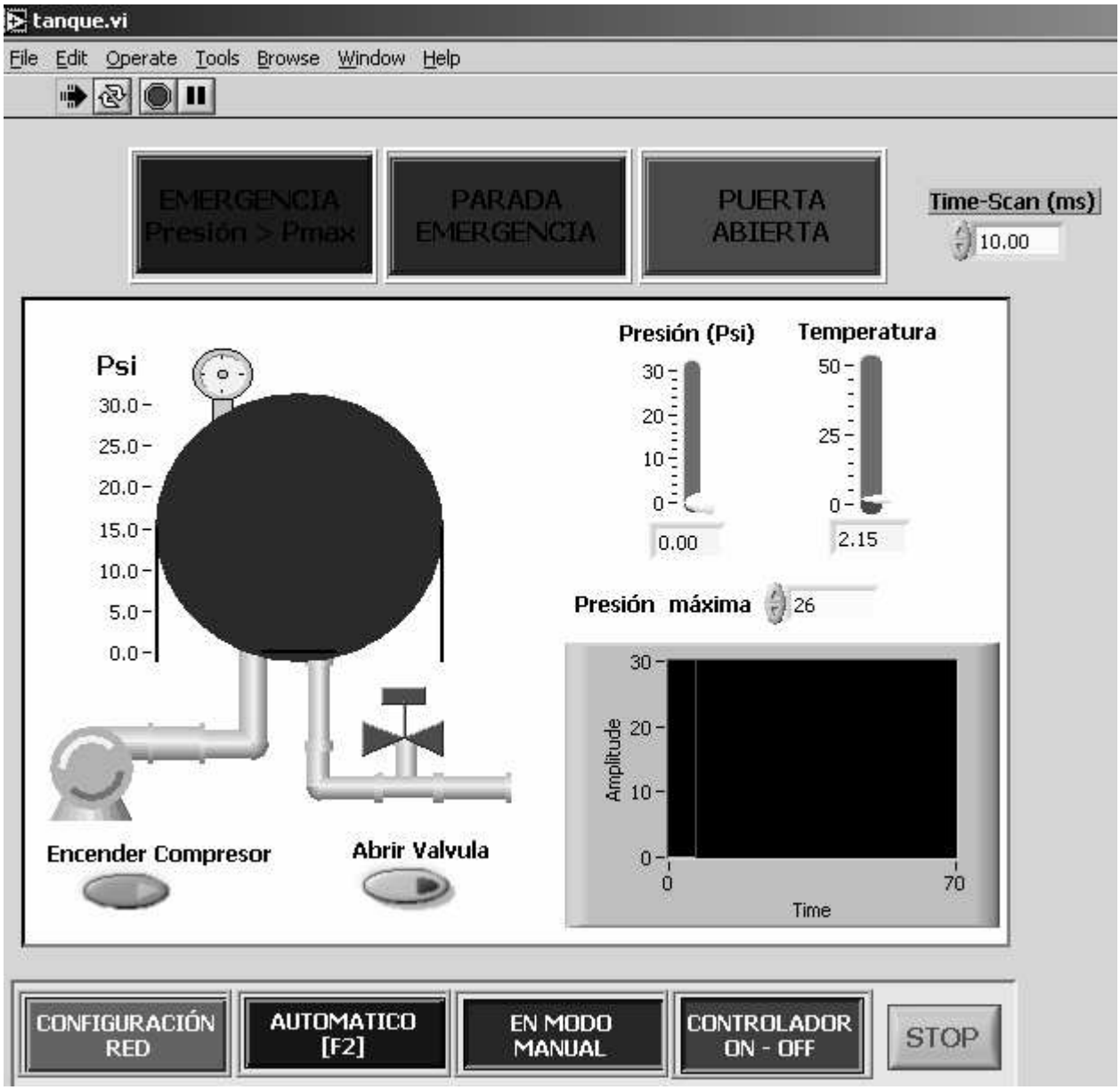
- Ocho entradas digitales accionadas por interruptores
- Ocho salidas digitales visualizadas con LEDS
- Dos entradas analógicas de 0 a 5V accionadas por potenciómetros
- Una salida analógica visualizada con un voltímetro de 0 a 5V.

Por su parte, el módulo de acondicionamiento de señales permite la interacción del Procesador Modbus con un proceso de temperatura en el rango de 0-50°c o de presión en el rango de 0 a 30 psi. Este Módulo posee las siguientes características:

- Dos salidas a contacto con capacidad de interrumpir 10A a 240 V
- Dos entradas a contacto.
- Una entrada analógica de 0 a 5V proveniente de un sensor de temperatura con una resolución de 100mV/°C, en un rango de 0-50°C
- Una entrada analógica de 0 a 5V proveniente de un sensor de presión con una resolución de 4V/30 psi, en un rango de 0-30 psi.

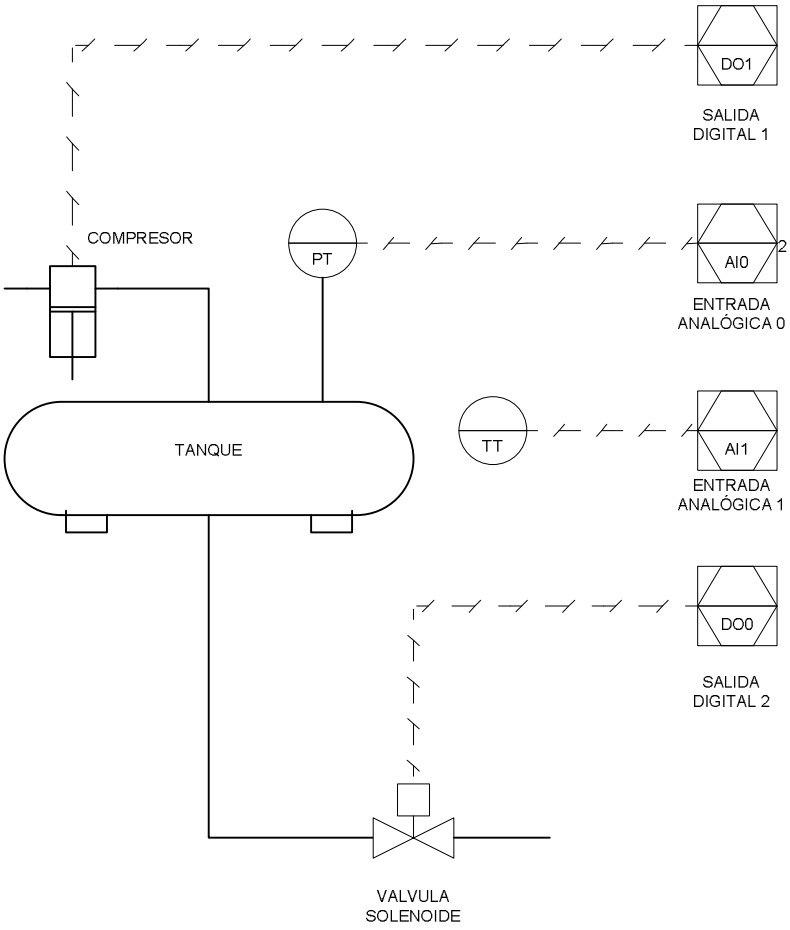
Para comprobar el funcionamiento del Procesador, se implemento un programa en Labview que permite monitorear las variables analógicas y digitales provenientes de cualquiera de los dos módulos antes mencionados. En la figura 40 se muestra el panel de control del programa en Labview.

Figura 40. Panel de control del programa de aplicación en Labview



Este programa funciona como un sistema SCADA que monitorea las variables físicas de un tanque de presión utilizando el protocolo MODBUS. En la figura 41 se muestra la aplicación implementada para la validación del Procesador

Figura 41. Proceso implementado para la validación del Procesador



Por otra parte, también se utilizó el programa Maestro Modbus implementado en Labview[24] para la comprobación del funcionamiento del Procesador en una red MODBUS estándar. En la figura 42 se muestra el esquema de conexión del Procesador a la red y en la figura 43 el panel de control del Maestro Modbus.

[24] ARDILA, Pedro, CARREÑO, Sinle M. Modbus. Monitoreo de la red empleando Labview. Tesis de grado. Universidad Industrial de Santander. 2005.

Figura 42. Procesador en una red MODBUS estándar

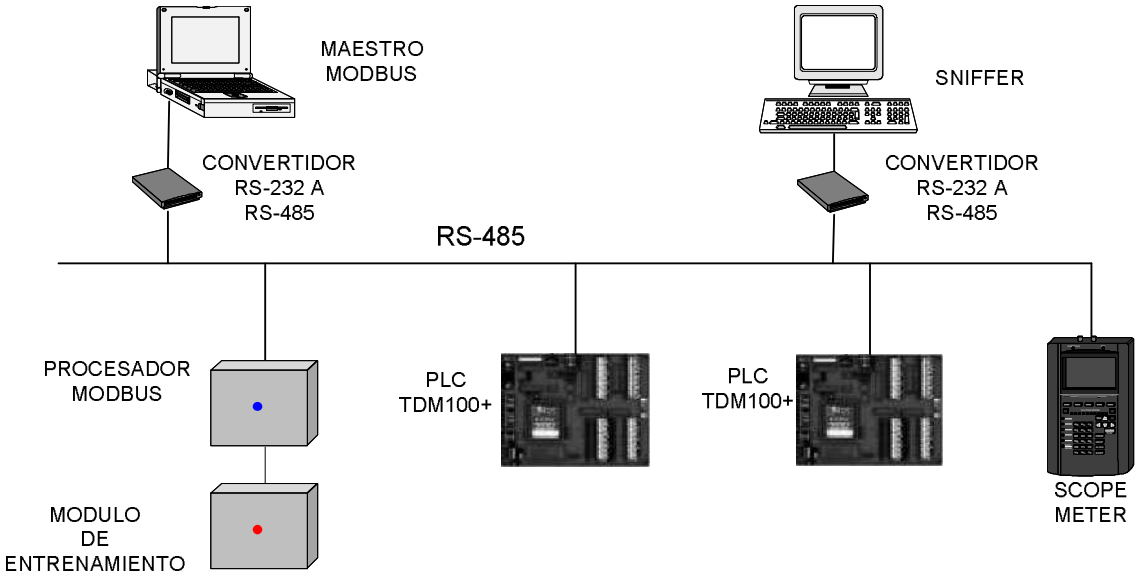
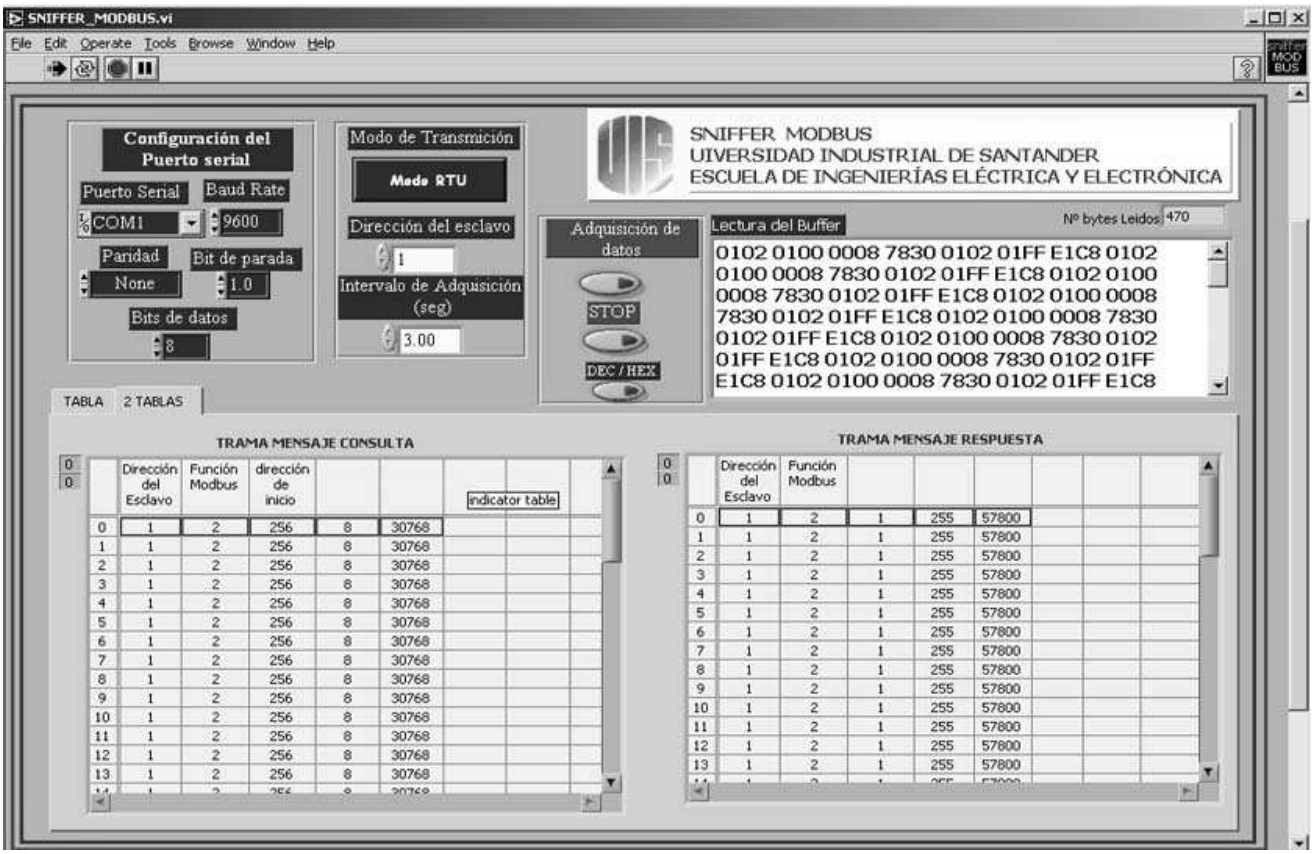


Figura 43. Panel de control del maestro Modbus

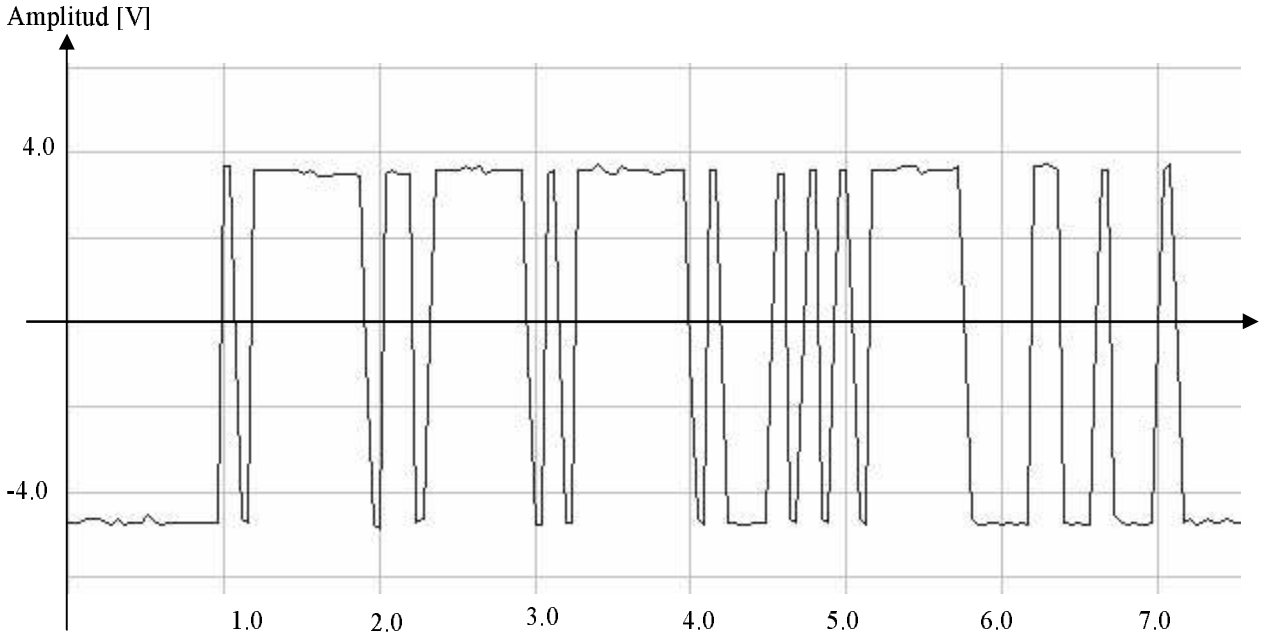
Finalmente, se utilizó la aplicación *Sniffer Modbus*[25] y un osciloscopio para monitorear el tráfico de la red, comprobando mediante los dos, que el procesador funcionaba satisfactoriamente. En la figura 44 se muestra el tráfico de la red con el Procesador conectado y en la figura 45 se muestra una trama capturada con el osciloscopio.

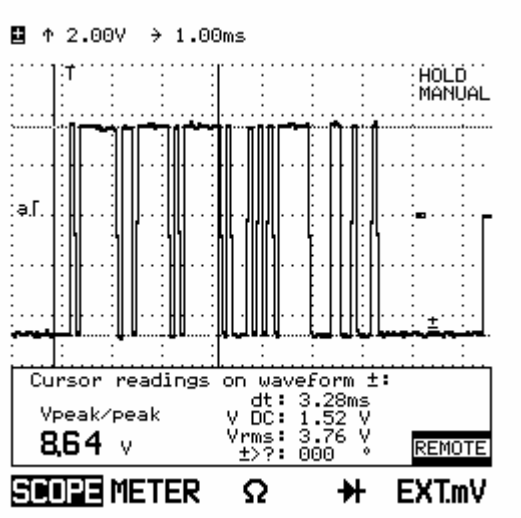
Figura 44. Panel de control del *Sniffer* Modbus



[25] ARDILA, Pedro, CARREÑO, Sinle M. Modbus. Monitoreo de la red empleando Labview. Tesis de grado. Universidad Industrial de Santander. 2005.

Figura 45. Trama Modbus capturada con un osciloscopio





Como se puede observar en el oscilograma, la señal de voltaje diferencial se encuentra dentro del rango permitido para una red Modbus estándar, que es de -7 V a + 12V de acuerdo con la norma EIA RS-485 A.

Capítulo 7

CONCLUSIONES

Este trabajo de investigación permitió que el Grupo de Investigación CEMOS y la E3T se apropiaran de la tecnología de los buses de campo, al implementar el protocolo MODBUS a nivel de Esclavo en un microcontrolador de propósito general y crear las herramientas de software para la depuración y validación del mismo.

La arquitectura abierta del protocolo Modbus permitió su implementación en diferentes plataformas de software y de hardware. Esto se comprobó al realizar el esclavo (Procesador Modbus) en un Microcontrolador de 16 bits y el maestro en LabView. Además, al estar completamente documentado sus especificaciones se conocían con exactitud y de esta manera se pudo lograr la interoperabilidad con diferentes fabricantes de software y hardware Modbus.

MODBUS es un formato de transmisión serial de datos utilizado extensamente en las comunicaciones con PLCs, pero fácilmente adaptable a cualquier tipo de instrumentación, gracias a su particular estructura de mensaje, el cual no opera con variables concretas sino con direcciones de memoria. Esto permite que, por ejemplo, las variables analógicas de un proceso puedan ser almacenadas en registros de 16 bits sin necesidad de algún procesamiento adicional. Otra ventaja que presenta el

formato MODBUS es la cantidad de variables discretas que puede manejar (hasta 1024) en un sólo esclavo, así como las diversas formas como se pueden manipular estos datos: individualmente, por grupos o por registros de 16 bits.

Algunos de los logros individuales y conclusiones más importantes de este trabajo de investigación se describen en las siguientes secciones.

7.1 CONCEPCIÓN DEL SOFTWARE Y DEL HARDWARE EMBEBIDO

La concepción del software y hardware embebido para la implementación del Protocolo Modbus se realizó con base la metodología descrita por Stuart R. Ball[26]. Esta metodología consta de las siguientes etapas:

- **Definición del producto:** En este caso se definió como producto el diseño e implementación de un Procesador de Comunicación que utilizara el protocolo MODBUS para su integración en una red estándar RS-485 y tuviese capacidad de operar con PLCs de distintos fabricantes.

- **Definición de requerimientos funcionales:** Las características funcionales establecidas para el Procesador de Comunicación fueron:
  - Capacidad de manejo de las interfaces RS-232 y RS-485.
  - Posibilidad de trabajar en cualquiera de los modos de transmisión Modbus ASCII o RTU.

[26] Ball, S.R. *Embedded Microprocessor Systems*. Newnes Press. Second edition

- Implementación de 15 funciones Modbus para el acceso y manipulación de datos, tanto a nivel de bits como a nivel de registros de 16 bits, y funciones de diagnóstico y configuración.
- Capacidad para manejar hasta 2024 bits y 250 registros de 16 bits.
- Posibilidad de implementar aplicaciones en el sistema embebido, aparte de las funciones específicas de comunicación.
- Portabilidad del firmware para la implementación en otros sistemas embebidos.
- Posibilidad de configuración de parámetros de comunicación: rata de baudios, paridad e interfaz, tanto por hardware como por software.

▪ **Selección del procesador:** Con base en las características funcionales anteriormente descritas se requería de un microcontrolador que cumpliera con los siguientes especificaciones:

- Número de pines de propósito general : mínimo 24
- Interfaces: mínimo dos (una para RS-232 y otra para RS-485)
- Requerimientos de memoria: mínimo 8 KBytes de memoria RAM y 32 KBytes de ROM. Parte de la memoria RAM se utilizaría en el manejo de bits y registros
- Interrupciones requeridas: 2 de las interfaces seriales y 3 de los temporizadores.
- *Timers*: Mínimo tres canales (dos para el modo RTU y uno para el modo ASCII)

- Convertidores ADC y DAC: para la implementación de aplicaciones que requieran procesamiento de señales analógicas.
- Ambiente de desarrollo: se requería una herramienta de desarrollo de bajo costo y con una curva de aprendizaje muy corta.

Con base en estos requerimientos se optó por el microcontrolador HCS12 de Motorola, el cual sobrepasa estas especificaciones, tal como se describió en le capítulo 4.

- **Especificaciones de software y hardware:** Las características esenciales del software es que pudiese ser fácilmente entendible por otras personas para permitir su actualización y utilización en diversas aplicaciones de ingeniería que requieran de un protocolo de comunicaciones. Por esta razón se optó por el lenguaje C como el más adecuado para la implementación del protocolo.

  En cuanto al hardware se pretendía contar con un mínimo de 8 entradas digitales, 8 salidas digitales, 8 entradas analógicas y dos salidas analógicas que permitiesen en un momento dado controlar o monitorear procesos físicos.

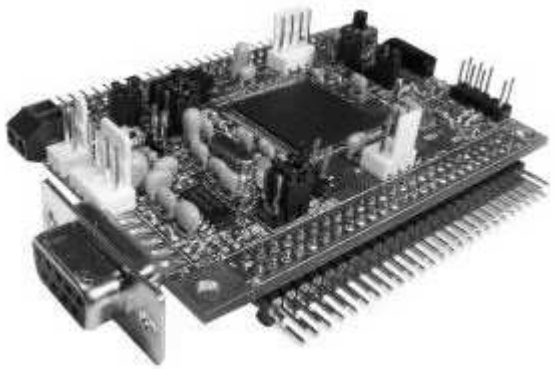
- **Diseño del firmware:** Para el diseño del firmware se seleccionó la herramienta de desarrollo Codewarrior[27]. Esta herramienta permite la compilación y depuración del software desarrollado en lenguaje C. Una de las mayores ventajas de utilizar el Codewarrior es la utilidad para depuración en tiempo real, ya que permitía acceder al contenido de la memoria y los registros del microcontrolador y de esta manera comprobar cada una de las funciones Modbus implementadas.

[27] Metrowerks Inc.

- **Diseño del hardware:** La implementación del Procesador Modbus se realizó sobre el sistema de desarrollo Adapt9s12E128 de *Technological Arts*, basado en el microcontrolador MC9S12E128 de Motorola. Este sistema facilitó el desarrollo de la capa física del protocolo debido a que ya trae integradas las interfaces RS-232 y RS-485 a través de los circuitos integrados MAX-232 y MAX-485. Otra ventaja de utilizar el sistema de desarrollo basado en el microcontrolador de 16 bits, es que permitirá utilizar el Procesador Modbus como base de otras aplicaciones que se inicien en un futuro cercano en la Escuela de Ingenierías E3T.

- **Integración:** Para la integración del Procesador Modbus con un proceso físico se diseñaron dos tarjetas de interfaz con los *buffers* adecuados para el manejo de las señales digitales de entrada y salida del microcontrolador. Las tarjetas pueden ser removidas fácilmente del sistema de desarrollo y ser reemplazadas en caso de un rediseño. Esto se debe a que el sistema de desarrollo tiene todos los pines del microcontrolador, disponibles en dos conectores laterales, tal como se muestra en la figura 46.

Figura 46. Sistema de desarrollo Adapt9s12E128



- **Verificación:** Para la verificación de cada una de las funciones implementadas se utilizaron las siguientes herramientas de software:
  - **Comdebug[28]:** Esta herramienta es un programa que permite comunicación serial con diferentes dispositivos. La característica más sobresaliente de dicha herramienta es que realiza el cálculo del CRC en forma automática y lo agrega al mensaje enviado, lo cual lo hace muy adecuado para el envío de mensajes Modbus.
  - **Maestro Modbus[29]:** Este programa desarrollado en LabView permitió la comprobación de la mayoría de las funciones Modbus implementadas. También permitió colocar el Procesador en red y comprobar su funcionamiento con otros PLCs.
  - **Aplicación:** Se realizó un programa de aplicación en LabView para el monitoreo de un sistema de control de presión en un tanque. Este programa se basó en las librerías desarrolladas para el Maestro Modbus.

7.2 INTERPRETACIÓN DEL DOCUMENTO DE REFERENCIA MODBUS

En el protocolo Modbus las direcciones de las salidas inician en cero, por lo que se hace necesario referenciar en el software las direcciones internas de la memoria del Microcontrolador a cero. De esta forma las direcciones de memoria utilizadas en el Porcesador Modbus quedan asignadas a direcciones Modbus de acuerdo con el mapa descrito en la Tabla 12.

---

[28] Windmill

[29] ARDILA, Pedro, CARREÑO, Sinle M. Modbus. Monitoreo de la red empleando Labview. Tesis de grado. Universidad Industrial de Santander. 2005.

Tabla 12. Mapa de memoria del Procesador Modbus

Datos	Dirección Modbus	Dirección del Procesador
Salidas dicretas	0x000	0x2000
Entradas discretas	0x100	0x2100
Registros internos	0x000	0x0000
Registros de entrada	0x100	0x2100

Como se puede observar, las direcciones de las salidas se comparten con las direcciones de los registros internos y las direcciones de las entradas discretas se comparten con las direcciones de los registros de entrada. Esto permite que los mismos datos se puedan manipular individualmente por bits o en grupos de 16 bits.

7.3 PRINCIPALES CARACTERÍSTICAS DEL PROCESADOR MODBUS

El aporte del trabajo de investigación acerca de la implementación del protocolo MODBUS se encuentra representado en el desarrollo de quince (15) funciones del estándar Modbus, en lenguaje C embebido para la familia de microcontroladores de 16 bits HCS12 de Motorola. Además de esto, se implementaron otras funciones de soporte y manejo del hardware embebido:

- Función de usuario 0x64: para la configuración por software del Procesador
- Manejo del convertidor ADC: para registro de variables analógicas de entrada.
- Manejo del convertidor DAC: para envío de variables analógicas de salida
- Manejo de puertos E/S: para registro y envío de datos digitales.
- Grabación de memoria Flash: utilizada en la función de configuración 0x64
- Manejo de interrupciones: para el procesamiento de las tramas Modbus
- Manejo de Temporizadores: utilizados en modo RTU y ASCII.

El Procesador Modbus posee además la asignación de memoria mostrada en la Tabla 13, que le permite el manejo de entradas y salidas tanto analógicas como digitales:

Tabla 13. Asignación de memoria del Procesador

Función	Funciones Modbus aplicables		Dirección Modbus		Dirección interna
	Descripción	Cód.	Hex.	Dec.	
8 salidas digitales	Forzar una salida Leer salidas Forzar múltiples salidas Forzar un registro Leer registros internos	**05** **01** **15** **06** **03**	0x0000	0	0x2000
8 entradas digitales	Leer entradas Leer registros de entrada	**02** **04**	0x0100	256	0x2100
Entrada analógica 1	Leer registros de entrada	**04**	0x0101	257	0x2102
Entrada analógica 2	Leer registros de entrada	**04**	0x0102	258	0x2104
Salida analógica 1	Forzar un registro	**06**	0x0000	0	0x2000

Otra característica muy importante del Procesador es que se tiene un espacio en memoria Flash de 100 Kbytes y 7,2 Kbytes de memoria RAM disponibles para implementar otras aplicaciones. Dentro del espacio de memoria disponible para el manejo de los mensajes Modbus (0x0001 a 0x00FF y 0x0103 a 0x01FF) se pueden asignar variables discretas o registros internos de 16 bits tales como:

- Temporizadores
- Contadores
- Registros del ADC
- Registros del DAC
- Estados de alarmas

- Eventos discretos

7. 4 FUNCIONES MODBUS DE USUARIO

Una de las características del protocolo Modbus, es que permite la implementación de funciones de usuario. Esta capacidad se utilizó para desarrollar la función 100 (0x64), la cual permite la configuración por software del Procesador Modbus.

7.5 PRODUCTOS DE LA INVESTIGACIÓN

Finalmente, como resultado del trabajo de investigación la E3T puede contar con las siguientes herramientas para continuar el camino en el desarrollo de aplicaciones Modbus:

- Procesador de comunicación Modbus basado en el microcontrolador MC9S128E de Motorola.
- Módulo de interfaz para implementar aplicaciones con el Procesador Modbus.
- Documentación de la implementación del Protocolo Modbus en lenguaje C embebido.
- Maestro Modbus implementado en Labview
- Espía (*Sniffer*) para visualización de las tramas en una red Modbus estándar.
- Cables de interfaz RS-232 a RS-485 para implementar una red Modbus estándar con PLCs de diferentes fabricantes

7.7 FUTUROS DESARROLLOS

Las recomendaciones para futuros trabajos que se desarrollen en el campo de las comunicaciones industriales y específicamente con buses de campo de arquitectura abierta pueden ser:

- Implementar Modbus TCP/IP, para la realización de una red de control industrial capaz de ser accedida a través de Internet ó la Intranet local, usando los protocolos TCP/IP.
- Utilizar el Procesador Modbus para el desarrollo de aplicaciones de monitoreo y control industrial, basados en el protocolo Modbus RTU o Modbus ASCII. Algunas de estas aplicaciones incluyen el monitoreo remoto y control de variables de proceso en oleoductos, acueductos y gasoductos, así como variables eléctricas en sistemas de distribución y transmisión de energía eléctrica.

BIBLIOGRAFÍA

ARDILA, Pedro, CARREÑO, Sinle M. Modbus. Monitoreo de la red empleando Labview. Tesis de grado. Universidad Industrial de Santander. 2005.

BALL, S.R. *Embedded Microprocessor Systems*. Newnes Press. Second edition. USA. 2004

BYTE CRAFT LIMITED. "*First Steps Embedded Systems*". BYTE CRAFT LIMITED. Canadá. 2002

BALCELLS, J., ROMERAL, J.L., "*Autómatas programables*", Marcombo. Serie Mundo Electrónico. Barcelona. 1997.

CARO, D., "*Automation network Selection*". ISA- The Instrumentation, Systems, and Automation Society. 2004

CUNHA, J. M. "*Protótipo de rede industrial utilizando o padrão serial RS485 e protocolo Modbus.*" I Congresso Brasileiro de Computação – CBComp 2001

KERNIGHAN, B. W. RITCHIE, D. M. "*The ANSI C programming Language*" Prentice Hall Software series. Second edition. 2003.

Klotz, D. "*Practical Embedded C Programming for the HCS12*" .Freescale Semiconductor, Inc. 2004

LIPOVSKI, G.J. "*Single- and Multi-Chip Microcontroller Interfacing for the Motorola 68HC12*" Academic Press Series in Engineering. USA. 1999

MODICON, Inc. "*Modbus Protocol Reference Guide PI–MBUS–300 Rev. J.*" Massachusetts. 1996.

MODBUS.ORG, "*MODBUS over Serial Line Specification & Implementation guide V1.0*". http://www.modbus.org/, fecha de acceso: Enero 2004

MODBUS.ORG, "*MODBUS application protocol specification V1.1*". http://www.modbus.org/, fecha de acceso: Agosto de 2004

MOTOROLA. "*HC12 CPU Reference Manual*". Motorola, Inc., 1998

MOTOROLA. "*MC9S12E-Family Device User Guide V01.04.*" Motorola, Inc. 2003

MOTOROLA. "*M68HC12 & HCS12 Microcontrollers CPU12 Reference Manual*" Motorola, Inc. 2002

MOTOROLA. "*HCS12 Microcontrollers S12CPUV2 Reference Manual*" Motorola, Inc. 2003

MOTOROLA. "*FTS128K1 Block User Guide V01.05*" Motorola, Inc.01 APR 2003

MOTOROLA. "*HCS12 Serial Communications Interface (SCI) Block Guide V03*". 8/16 Bit Division, TSPG Motorola, Inc.02 Jul 03, 2002.

MOTOROLA. "*TIM_16B4C Block Guide V1.0*" 07 8/16 Bit Division, TSPG, Austin Motorola, In. Dec 2001

MOTOROLA. "*DAC_8B1C Block Guide V1.0*". 8/16 Bit Division (TSPG) Motorola, Inc. 1 JUL 2003

MOTOROLA. "*PIM_9E128 Block Guide V01.00*". Motorola, Inc. 04 JUN 2003

MOTOROLA. "*Multiplexed External Bus Interface (MEBI) Block User Guide*". Motorola, Inc., 2003

MOTOROLA. "*ATD_10B16C Block User Guide V02.07*". Motorola Inc. 29 Aug. 2002

RAMÍREZ L., ACEVEDO C., "*Wireless System for Electrical Networks Testing Based on MODBUS Protocol*". Proceedings of the 14th International Conference on Electronics, Communications and Computers (CONIELECOMP'04) 2004.

ROBB, S. "*HCS12 NVM Guidelines*". Freescale Semiconductor, Inc. 2003

ROBB, S. "*Fast NVM Programming for the MC9S12DP256*". Freescale Semiconductor, Inc. 2001

VALVANO, J.W. "*Introducción a sistemas de microcomputadora embebidos-Simulación de Motorola 6811 y 6812*" – Editorial Thomson, 2004. México.

WILLIAMS, J. " *A Utility for Programming Single FLASH Array HCS12 MCUs, with Minimum RAM Overhead".* Freescale Semiconductor, Inc. 2004

ANEXOS

ANEXO I

IMPLEMENTACION

DE

FUNCIONES MODBUS

1.1. FUNCIÓN 02 (0x02): READ DISCRETE INPUTS

Descripción

Esta función se utiliza para leer hasta 2000 (0x07D0) estados ON/OFF de entradas digitales en un dispositivo remoto. La consulta general para esta función no está permitida. En el procesador Modbus se designó el área de memoria 0x2100 hasta 0x21FF para guardar los datos correspondientes a los estados de las entradas discretas del dispositivo remoto.

Consulta

En el mensaje Modbus las entradas digitales inician en la dirección (0x0100 ó 256 decimal) de tal forma que la entrada digital 1 corresponde a la dirección 0x0100 (256 decimal) y la 16 a la dirección 0x010F (271 decimal). En la figura 47 se muestra un ejemplo de consulta de lectura de los estados de entradas 324-343 del procesador. En este ejemplo la entrada inicial se encuentra en la dirección decimal 323 ó 0x0143 hexadecimal. El número de entradas es 20 (0x0014).

Figura 47. Ejemplo de consulta Modbus para la función 0x02

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**02**
Dirección Inicial Hi	**01**
Dirección Inicial Lo	**43**
Número de salidas Hi	**00**
Número de salidas Lo	**14**
CRC Hi	**CRC Hi**
CRC Lo	**CRC Lo**

En la Figura 48 se muestra la repuesta a la consulta solicitada en el ejemplo anterior.

Figura 48. Ejemplo de respuesta Modbus para la función 0x02

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**02**
Conteo de Bytes	**03**
Estados salidas 67-74	**79**
Estados salidas 82-75	**2D**
Estados salidas 86-83	**0D**
CRC Hi	**75**
CRC Lo	**02**

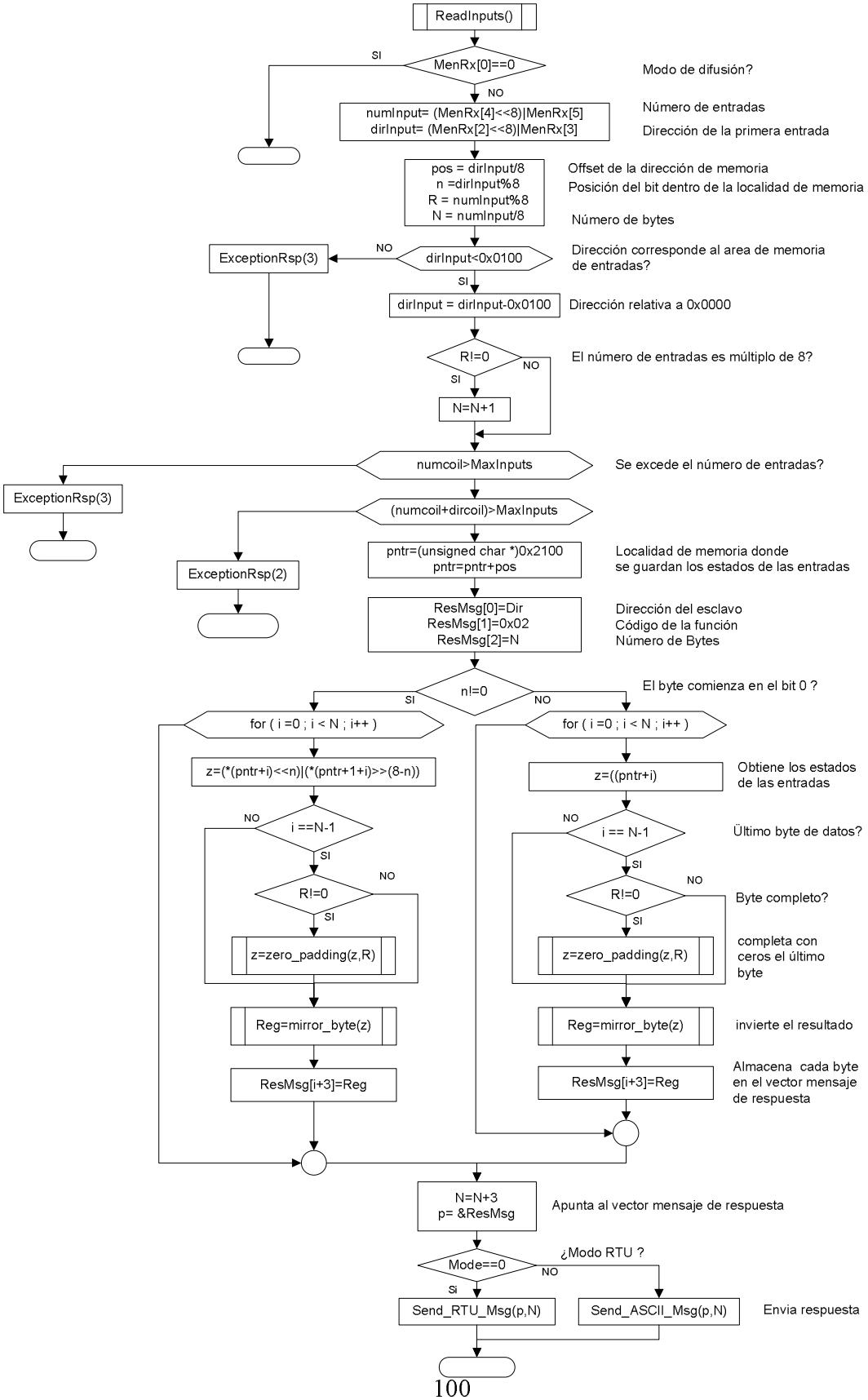
Implementación de la función 0x02

Para la implementación de la función 0x02 se utiliza la misma programación que para la función 0x01, teniendo en cuenta las siguientes consideraciones:

- El campo de código de función se cambia a 0x02
- La dirección de la entradas corresponde a 0x2100 en lugar de 0x2000

En la Figura 49 se muestra el diagrama de flujo de la función 0x02.

Figura 49. Diagrama de flujo de la Función 02



1.2. FUNCIÓN 03 : READ HOLDING REGISTERS

Descripción

Esta función se utiliza para leer hasta 125 registros internos o registros de salidas (*holding registers*) de 16 bits en un dispositivo remoto. En el procesador Modbus se designó el área de memoria 0x2000 hasta 0x20FF para guardar los datos correspondientes a los registros internos.

Consulta

En el mensaje de consulta se especifica el registro inicial y la cantidad de registros a leer. Estos registros deben ser contiguos, y el direccionamiento empieza en cero: los registros 1-16 se asocian a los bits 0-15. La consulta general no está permitida para esta función. En la figura 50 se muestra un ejemplo de consulta de lectura de los estados de los registros 4, 5 y 6 del procesador:

Figura 50. Ejemplo de consulta Modbus para la función 0x03

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**03**
Dirección Inicial Hi	**00**
Dirección Inicial Lo	**03**
Número de salidas Hi	**00**
Número de salidas Lo	**03**
CRC Hi	**CRC**
CRC Lo	

Respuesta

La información de los registros se empaqueta en el mensaje en dos bytes por registro, con el LSB a la derecha en cada byte. En cada registro, el primer byte contiene los 8 bits más significativos y el segundo contiene los 8 bits menos significativos. En la figura 51 se muestra la respuesta a la consulta solicitada en el ejemplo anterior:

Figura 51. Ejemplo de respuesta Modbus para la función 0x03

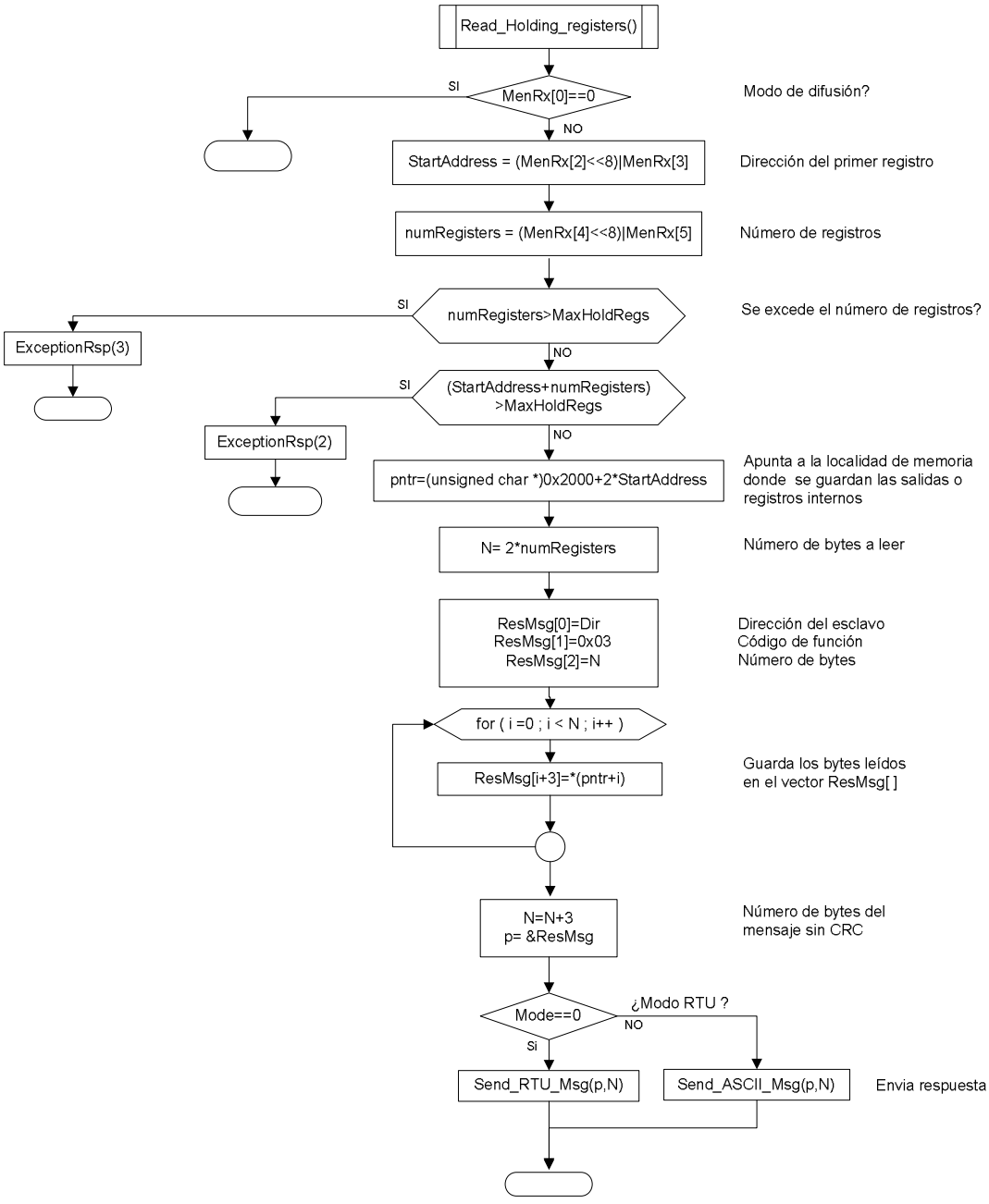
Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**03**
Conteo de Bytes	**06**
Datos Hi (Registro 4)	**1B**
Datos Lo (Registro 4)	**01**
Datos Hi (Registro 5)	**B3**
Datos Lo (Registro 5)	**D6**
Datos Hi (Registro 6)	**96**
Datos Lo (Registro 6)	**7A**
CRC Hi	**36**
CRC Lo	**61**

El contenido del registro 4 se representa como dos bytes de valor 0x1B01, el del registro 5 es 0xB3D6 y el del registro 6 es 0x967A.

Implementación

El diagrama de flujo de la figura 52 muestra la implementación de la función 03:

Figura 52. Diagrama de flujo de la Función 03



1.3. FUNCION 04 (0x04): READ INPUT REGISTERS

Descripción

Esta función se utiliza para leer hasta 125 registros de entradas (*input registers*) de 16 bits. En el procesador Modbus se designó el área de memoria 0x2100 hasta 0x20FF para guardar los datos correspondientes a los registros de entrada.

Consulta

En el mensaje de consulta se especifica el registro inicial y la cantidad de registros a leer. Estos registros deben ser contiguos, y el direccionamiento empieza en cero: los registros 1-16 se asocian a las localidades 0-15. La consulta general no está permitida para esta función. En la figura 53 se muestra un ejemplo de consulta de lectura de los estados del registro 260 (Dirección = 259 ó 0x0103 hex.) del procesador:

Figura 53. Ejemplo de consulta Modbus para la función 0x04

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**04**
Dirección Inicial Hi	**01**
Dirección Inicial Lo	**03**
Número de salidas Hi	**00**
Número de salidas Lo	**01**
CRC Hi	**CRC**
CRC Lo	

Respuesta

La información de los registros se empaqueta en el mensaje en dos bytes por registro, con el LSB a la derecha en cada byte. En cada registro, el primer byte contiene los 8 bits más significativos y el segundo contiene los 8 bits menos significativos. En la figura 54 se muestra la respuesta a la consulta solicitada en el ejemplo anterior:

Figura 54. Ejemplo de respuesta Modbus para la función 0x04

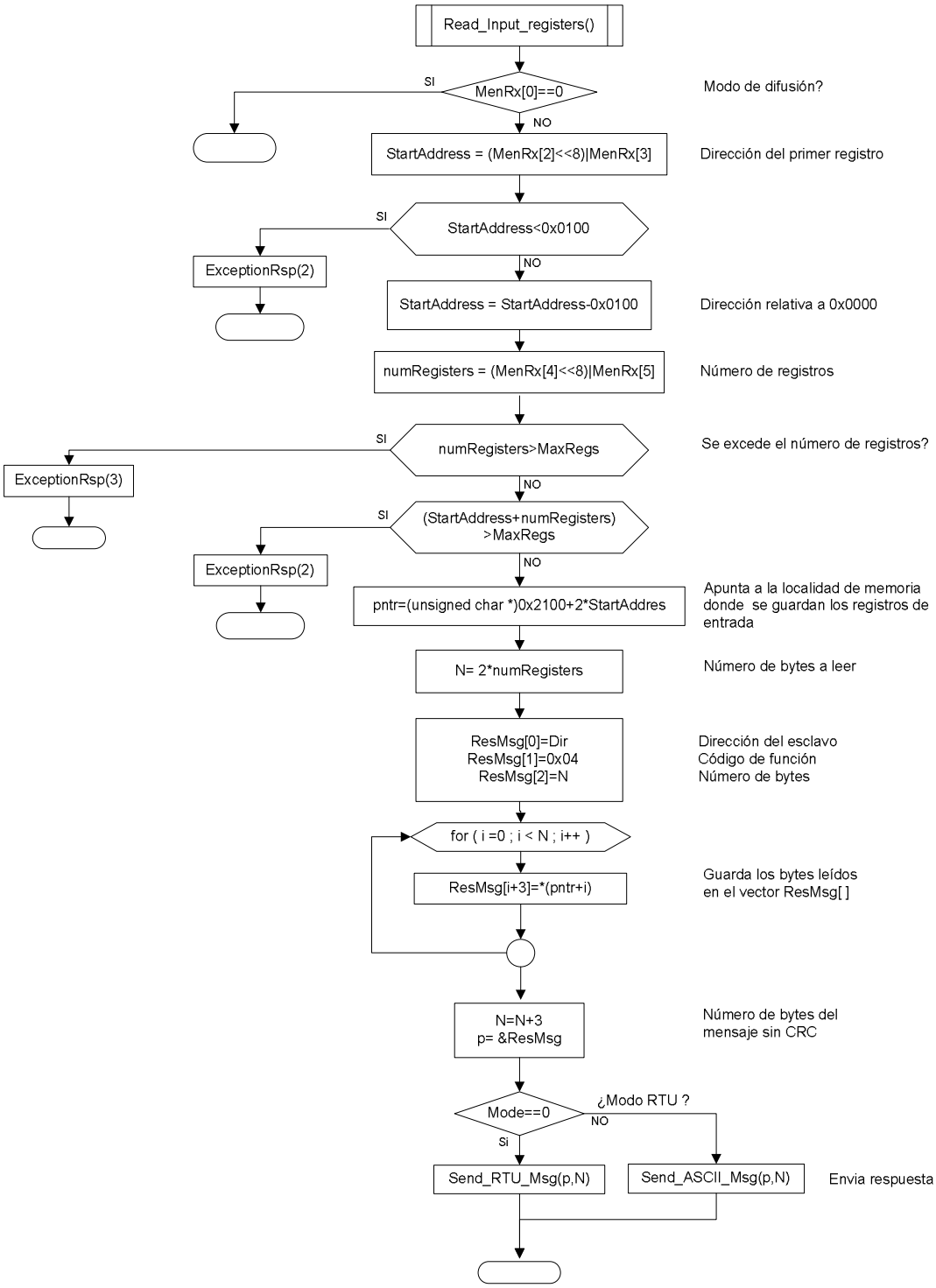
Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**04**
Conteo de Bytes	**02**
Datos Hi (Registro 260)	**1B**
Datos Lo (Registro 260)	**01**
CRC Hi	**72**
CRC Lo	**02**

El contenido del registro 260 se representa como dos bytes de valor 0x1B01.

Implementación

El diagrama de flujo de la figura 23 muestra la implementación de la función 04:

Figura 55. Diagrama de flujo de la Función 04



1.4. FUNCIÓN 05 (0x05): WRITE SINGLE COIL

Descripción

Esta función fuerza el estado de una salida a uno (ON) o a cero (OFF). En consulta general, la función fuerza la misma salida en todos los esclavos conectados a la red Modbus.

Consulta

En el mensaje de consulta se especifica la dirección de la salida a forzar. Las salidas se direccional comenzando en cero: La salida 1 se direcciona como cero (0).

La petición de estado ON/OFF se especifica con una constante en el campo de información de la consulta. El valor 0xFF00 solicita que la salida se pase a ON. El valor 0x0000 solicita que pase a OFF. Todos los demás valores son ilegales y no afectarán el estado de la bobina. En la figura 56 se muestra un ejemplo de solicitud de forzar la bobina 37 en el procesador.

Figura 56. Ejemplo de consulta Modbus para la función 0x05

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**05**
Dirección Inicial Hi	**00**
Dirección Inicial Lo	**24**
Fuerza dato Hi	**FF**
Fuerza dato Lo	**00**
CRC Hi	**CRC Hi**
CRC Lo	**CRC Lo**

Respuesta

La respuesta normal es un eco de la consulta y devuelve el estado de la salida después del forzado.

La figura 57 muestra un ejemplo de respuesta a la consulta solicitada en el ejemplo anterior

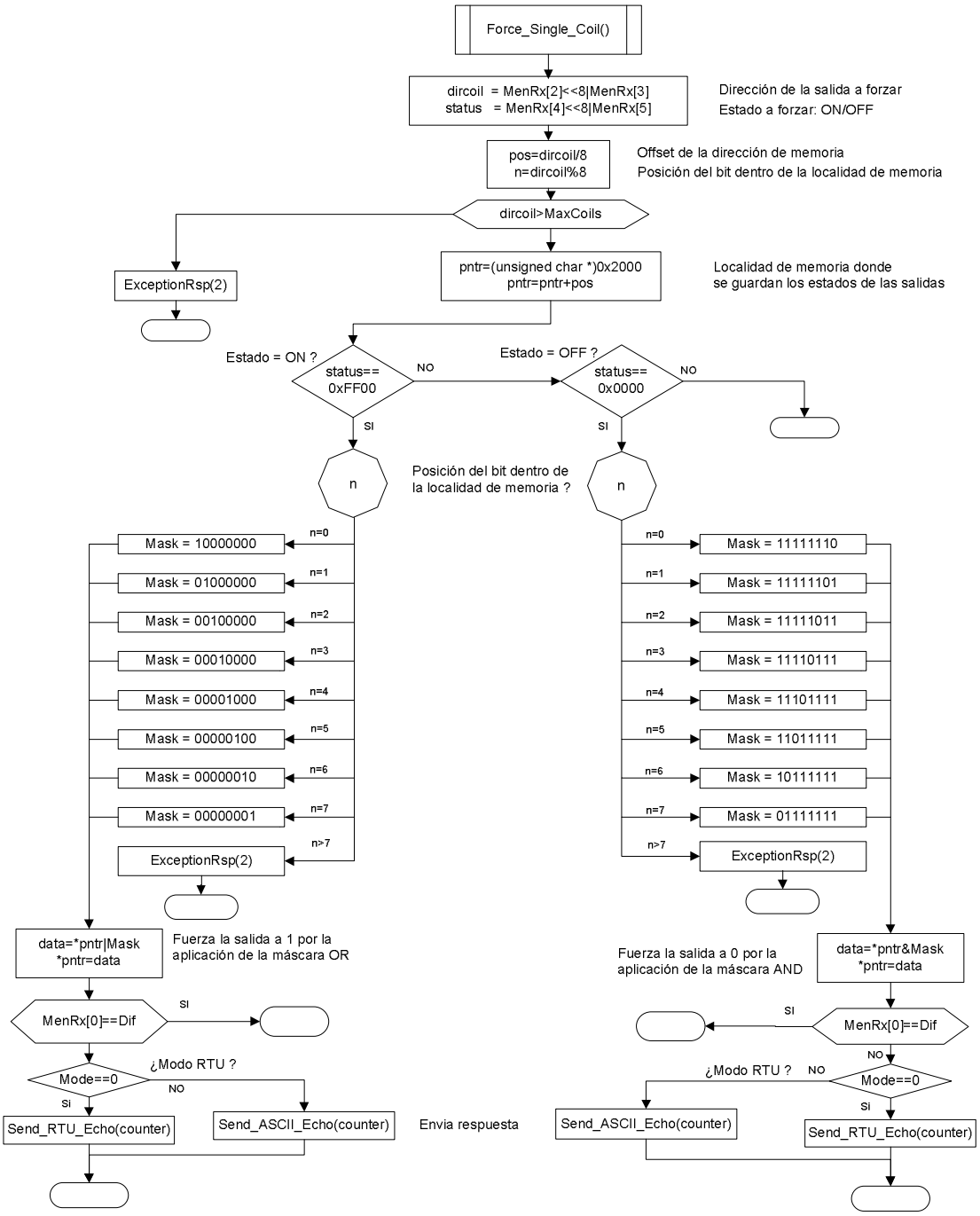
Figura 57. Ejemplo de respuesta Modbus para la función 0x05

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	01
Función	05
Dirección Inicial Hi	00
Dirección Inicial Lo	24
Fuerza dato Hi	FF
Fuerza dato Lo	00
CRC Hi	CC
CRC Lo	31

Implementación

El diagrama de flujo de la figura 58 muestra la implementación de la función 05:

Figura 58. Diagrama de flujo de la Función 05



1.5. FUNCIÓN 06 (0x06): WRITE SINGLE REGISTER

Descripción

Esta función se utiliza para escribir un valor en un registro único de 16 bits. En modo de consulta general, la función escribe en la misma dirección de registro en todos los esclavos conectados

Consulta

En el mensaje de consulta se especifica el registro inicial y el valor de 16 bits a escribir. En la figura 59 se muestra un ejemplo de solicitud para escribir el registro 10 del Procesador con el valor 0x01FF.

Figura 59. Ejemplo de consulta Modbus para la función 06

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**06**
Dirección Registro Hi	**00**
Dirección Registro Lo	**09**
Dato a escribir Hi	**01**
Dato a escribir Lo	**FF**
CRC Hi	**CRC**
CRC Lo	

Respuesta

La respuesta es un eco de la consulta, se devuelve después que el registro ha sido modificado. En la figura 60 se muestra la respuesta a la consulta solicitada en el ejemplo anterior:

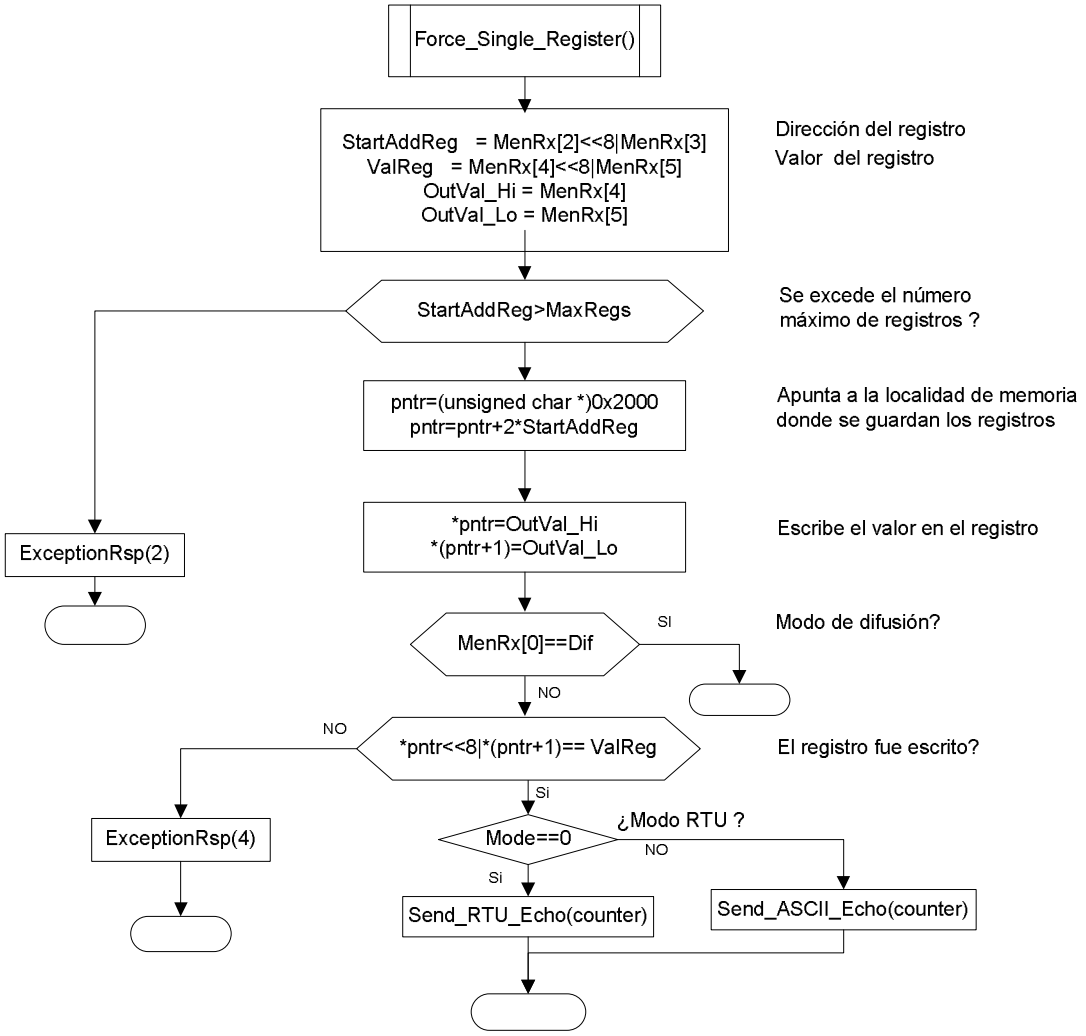
Figura 60. Ejemplo de respuesta Modbus para la función 06

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**06**
Dirección Registro Hi	**00**
Dirección Registro Lo	**09**
Dato escrito Hi	**01**
Dato escrito Lo	**FF**
CRC Hi	**18**
CRC Lo	**18**

Implementación

El diagrama de flujo de la figura 61 muestra la implementación de la función 06:

Figura 61. Diagrama de flujo de la función 06



1.6. FUNCIÓN 07 (0x07): READ EXCEPTIONS STATUS

Descripción

Esta función se utiliza para leer el contenido de ocho bits internos de condición de excepción en el procesador. Estos ocho estados de excepción son definidos por el usuario y pueden contener información acerca del estado del dispositivo remoto, por ejemplo, falla en alimentación, parada de emergencia, o cualquier otro.

Consulta

La función 07 proporciona un método simple para acceder a la información, debido a que las referencias de los estados de excepción se conocen de antemano. Debido a esto no se necesita especificar la dirección de las salidas en el mensaje de consulta. En la figura 62 se muestra un ejemplo de lectura del estado de excepción del Procesador:

Figura 62. Ejemplo de consulta Modbus para la función 07

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**07**
CRC Hi	**CRC**
CRC Lo	**CRC**

Respuesta

La respuesta normal contiene el estado de los ocho bits de estado de excepción empaquetados en un único byte.

La figura 63 muestra un ejemplo de lectura del estado de excepción del Procesador:

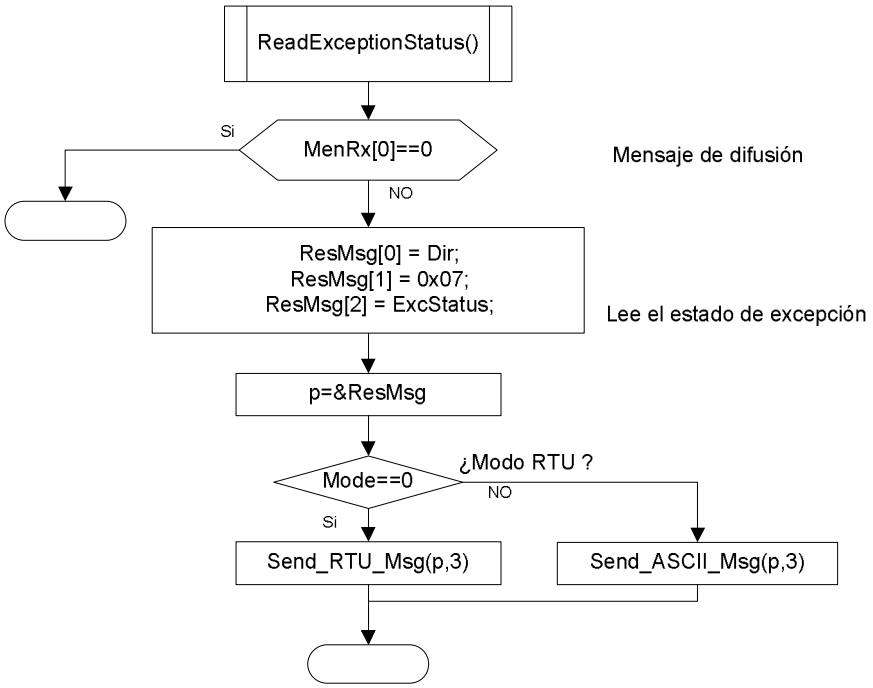
Figura 63. Ejemplo de respuesta Modbus para la función 07

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**07**
Datos de los bits de excepción	**00**
CRC Hi	**22**
CRC Lo	**30**

Implementación

El diagrama de flujo de la figura 64 muestra la implementación de la función 07:

Figura 64. Diagrama de flujo para la función 07



1.7. FUNCIÓN 08 (0x08): DIAGNOSTICS

Descripción

La función 08 proporciona una serie de pruebas para chequear el sistema de comunicación entre el dispositivo maestro y el esclavo, o para comprobar diversas condiciones internas en el procesador esclavo. La consulta general no está permitida.

Consulta

En el mensaje de consulta la función utiliza un código de subfunción de dos bytes para definir el tipo de pruebas que se van a realizar

En General, la emisión de una función de diagnóstico a un dispositivo esclavo no afecta al programa de usuario que se está ejecutando en el esclavo. La lógica de usuario, las entradas discretas y los registros, no se pueden acceder desde los diagnósticos. Ciertas funciones pueden restablecer opcionalmente contadores de error en el esclavo.

Un dispositivo esclavo puede, sin embargo, ser forzado a modo "sólo escucha", modo en el cual puede recibir mensajes por el sistema de comunicaciones pero no los responde. Generalmente, este modo de operación se utiliza para retirar un dispositivo esclavo que funciona mal en el sistema de comunicaciones.

Códigos de subfunción

00 (0x00) Retorna datos consultados

El campo de información enviado en la consulta se retorna en el mensaje de respuesta. El mensaje de respuesta entero debe ser idéntico a al mensaje de consulta:

Subfunción	Datos de consulta	Datos de Respuesta
0000	Cualquiera	Eco de la consulta

01 (0x01) Opción de restablecer comunicaciones

El dispositivo remoto se reinicializa y todos sus contadores de eventos de comunicaciones se borran. Si el esclavo esta en modo de "Sólo escucha", no se retorna respuesta. Esta función es la única forma de sacar al esclavo del modo de "Sólo escucha". Si el esclavo no está en modo de "Sólo escucha" se retorna una respuesta normal después de haberse ejecutado el reinicio del Procesador de comunicaciones.

Subfunción	Datos de consulta	Datos de Respuesta
0001	0000	Eco de la consulta
0001	FF00	Eco de la consulta

Un contenido de FF00 en el dato de consulta ocasiona que el registro de Eventos de Comunicaciones sea borrado. Un contenido de 0000 mantiene dicho registro en el valor antes del reinicio.

02 (0x02) Retorna Registro de Diagnóstico

El contenido del registro de diagnóstico de 16 bits se envía en la respuesta

Subfunción	Datos de consulta	Datos de Respuesta
0002	0000	Registro de diagnóstico

04 (0x04) Fuerza al Modo de Sólo Escucha

Fuerza al Procesador de comunicaciones al Modo de Sólo escucha para comunicaciones MODBUS. Esto aísla al Procesador de otros dispositivos en la red, permitiendo la comunicación continua sin ninguna interrupción por parte del dispositivo aislado. En este caso no se retorna repuesta alguna.

Cuando el dispositivo remoto entra en el Modo de Sólo escucha, todos los controles de las comunicaciones activas se desactivan. Mientras el dispositivo está en el Modo de Sólo escucha, todos los mensajes MODBUS dirigidos a él, se monitorean pero no se toma ninguna acción y no se envía respuesta.

La única función que se procesa después de haber entrado en el Modo de Sólo escucha es la subfunción 08-01 (Opción de restablecer comunicaciones)

Subfunción	Datos de consulta	Datos de Respuesta
0004	0000	No se retorna respuesta

10 (0x0A) Retorna Registro de Diagnóstico

Borra todos los contadores y el registro de diagnóstico. Los contadores también se borrar cuando el Procesador Modbus se apaga.

Subfunción	Datos de consulta	Datos de Respuesta
000A	0000	Eco de la consulta

11 (0x0B) Devuelve Contador de Mensaje de Bus

El campo de información de la respuesta devuelve la cantidad de mensajes que el esclavo ha detectado en el sistema de comunicaciones desde su ultimo reinicio, última operación de limpieza de contadores, o último arranque.

Subfunción	Datos de consulta	Datos de Respuesta
000B	0000	Contador de total de mensajes

12 (0x0C) Devuelve Contador de Error Bus de Comunicaciones

El campo de información de la respuesta devuelve la cantidad de errores de CRC detectados por el esclavo desde su último reinicio, última operación de limpieza de contadores, o último arranque.

Subfunción	Datos de consulta	Datos de Respuesta
000C	0000	Contador de errores CRC

13 (0x0D) Devuelve Contador Error de Excepción en el Bus

El campo de información de la respuesta devuelve la cantidad de respuestas de excepción Modbus devueltas por el esclavo desde su último reinicio, última operación de limpieza de contadores, o ultimo arranque.

Subfunción	Datos de consulta	Datos de Respuesta
000D	0000	Contador de errores de excepción

Las respuestas de excepción se describen y están listadas en la tabla 14.

Tabla 14. Códigos de excepción Modbus

Código	Nombre	Descripción
01	Función Ilegal	La función recibida en la consulta no está implementada en el esclavo
02	Dirección Ilegal	La dirección de los datos recibidos en la consulta no está permitida para el esclavo
03	Valor de datos ilegal	El valor de los datos recibidos en la consulta no está permitido en el esclavo
04	Falla en el dispositivo esclavo	Un error irrecuperable ha ocurrido mientras el esclavo está intentando procesar la respuesta
05	Reconocimiento	El esclavo ha aceptado la consulta y la está procesando, pero necesita un tiempo mayor para completar la operación. La respuesta es retornada para prevenir un error de *Time Out* en el maestro. El maestro puede realizar una nueva consulta para determinar si el procesamiento se ha completado.

14 (0x0E) Devuelve Contador de Mensajes del Esclavo

El campo de información de la respuesta devuelve la cantidad de mensajes dirigidos al esclavo o de consulta general, que el esclavo ha procesado desde su último reinicio, última operación de limpieza de contadores, o último arranque.

Subfunción	Datos de consulta	Datos de Respuesta
000E	0000	Contador de mensajes del esclavo

15 (0x0F) Devuelve Contador de No Respuesta del Esclavo

El campo de información de la respuesta devuelve la cantidad de mensajes dirigidos al esclavo y no contestados (ni en respuesta normal, ni en respuesta de excepción),

desde su último reinicio, última operación de limpieza de contadores, o último arranque.

Subfunción	Datos de consulta	Datos de Respuesta
000F	0000	Contador de No Respuesta

16 (0x10) Devuelve Contador NAK del Esclavo

El campo de información de la respuesta devuelve la cantidad de mensajes dirigidos al esclavo a los que devolvió una, respuesta de excepción, Negativa de Reconocimiento (NAK) desde su último reinicio, ultima operación de limpieza de contadores, o ultimo arranque.

Subfunción	Datos de consulta	Datos de Respuesta
0010	0000	Contador NAK

17 (0x11) Devuelve Contador de Esclavo Ocupado

El campo de información de la respuesta devuelve la cantidad de mensajes dirigidos al esclavo a los cuales el esclavo ha retornado una respuesta de excepción de Dispositivo Esclavo Ocupado desde su último reinicio, última operación de limpieza de contadores, o último arranque.

Subfunción	Datos de consulta	Datos de Respuesta
0011	0000	Contador de Esclavo Ocupado

18 (0x12) Devuelve Contador de Sobrecarga de caracteres en el Bus

El campo de información de la respuesta devuelve la cantidad de mensajes dirigidos

al esclavo y que no pudo maneja r debido a una condición de sobrecarga de caracteres (*Overrun*), desde su último reinicio, última operación de limpieza de contadores, o último arranque. Una sobrecarga de caracteres ocurre cuando los caracteres llegan al puerto mucho más rápido de lo que pueden ser almacenados, o por pérdida de caracteres debido a un mal funcionamiento del hardware.

Subfunción	**Datos de consulta**	**Datos de Respuesta**
00012	0000	Contador de Sobrecarga de caracteres

20 (0x14) Borrado de Contador de Sobrecarga y banderas

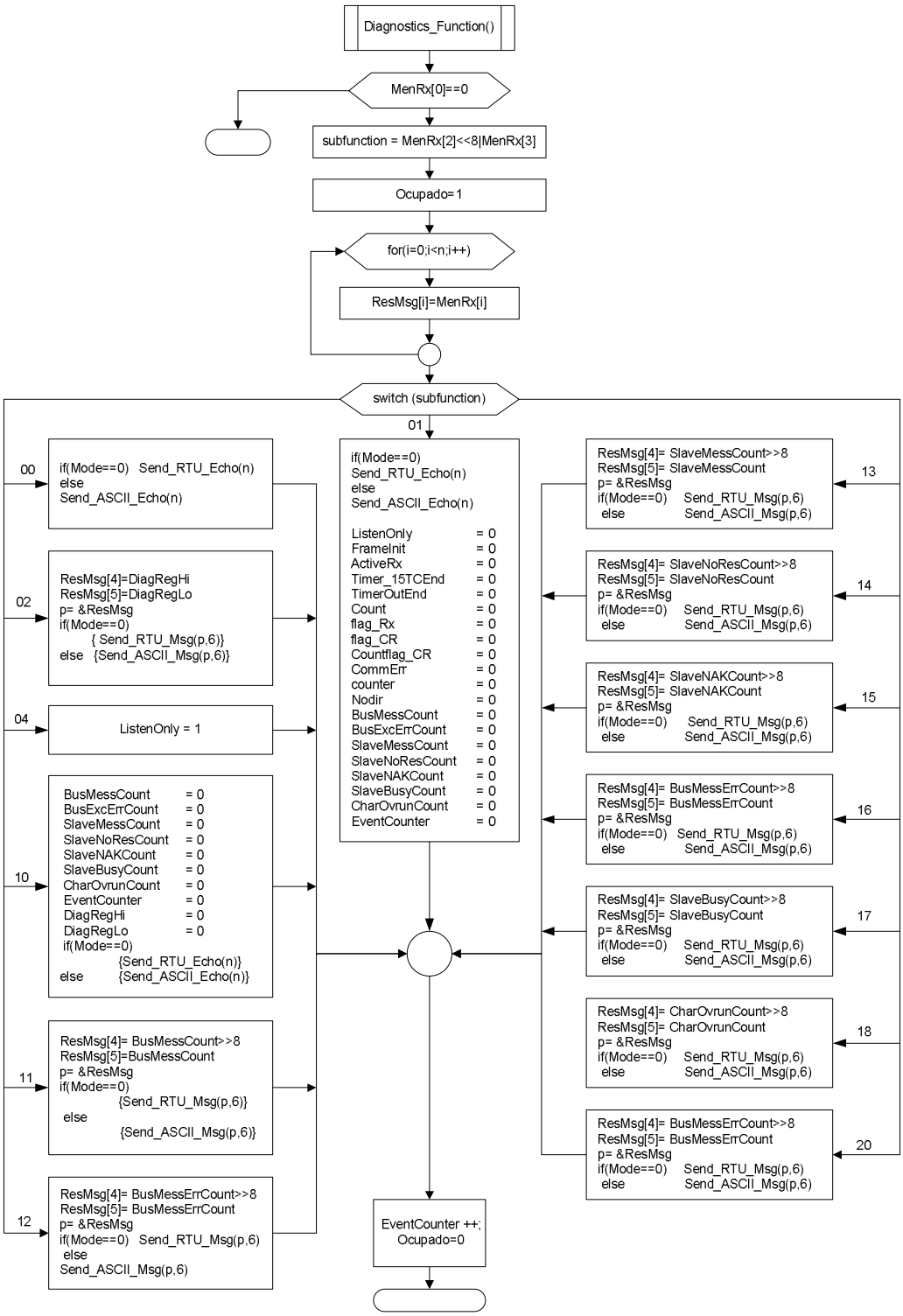
Borra el contador de errores de sobrecarga y reinicia la bandera de error.

Subfunción	**Datos de consulta**	**Datos de Respuesta**
0010	0000	Contador NAK

Implementación

El diagrama de flujo de la figura 65 muestra la implementación de la función 08:

Figura 65. Diagrama de flujo de la función 08



1.8. FUNCIÓN 11 (0x0B): GET COMM EVENT COUNTER

Descripción

Devuelve una palabra de estado y un contador de eventos de comunicaciones del esclavo. Solicitando el contador actual antes y después de una serie de mensajes, el maestro puede determinar si los mensajes se han tratado normalmente. La Consulta General no está permitida.

El contador de eventos del controlador se incrementa cada vez que se termina un mensaje correctamente. No se incrementa por respuestas de excepción, consultas generales, o comandos relacionados con el contador de eventos.

El contador de eventos se puede poner a cero por medio de la función de Diagnósticos (código 08), con la subfunción de Opción de Reinicio de Comunicaciones (código 00 01) o Limpiar Contadores y Registros de Diagnóstico (código 00 0A).

Consulta

La figura 66 muestra un ejemplo de una solicitud del Contador de Eventos de Comunicaciones en el Procesador Modbus:

Figura 66. Solicitud del Contador de Eventos de Comunicaciones - Consulta

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**0B**
CRC Hi	**CRC**
CRC Lo	**-**

Respuesta

La respuesta normal contiene una palabra de estado de dos bytes, y otros dos bytes del contador de eventos. La palabra de estado será todo unos (0xFFFF) si se está procesando un comando enviado previamente al esclavo (existirá una señal de ocupado). Si esta libre, la palabra de estado será todos ceros. (0x0000).

La figura 67 muestra un ejemplo de respuesta a la consulta realizada en el ejemplo anterior.

Figura 67. Solicitud del Contador de Eventos de Comunicaciones - Respuesta

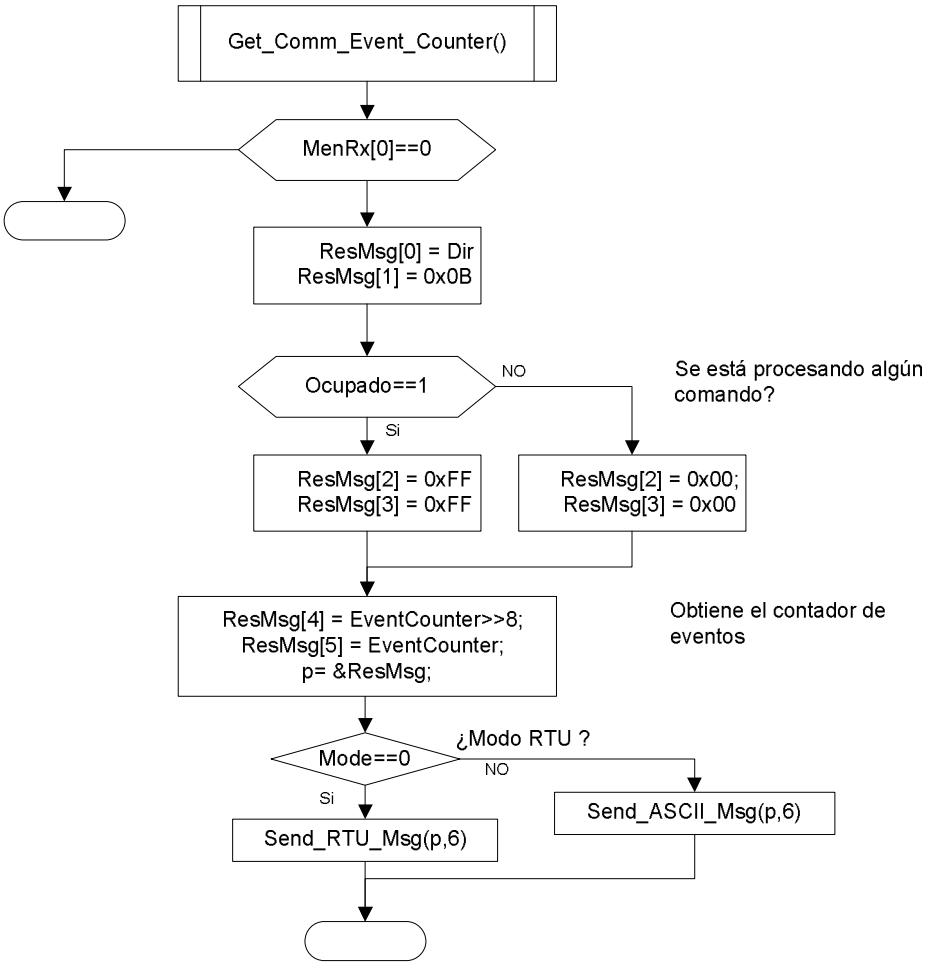
Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**0B**
Estado Hi	**FF**
Estado lo	**FF**
Contador de eventos Hi	**00**
Contador de eventos Lo	**08**
CRC Hi	**-**
CRC Lo	**-**

En este ejemplo, la palabra de estado es 0xFFFF, lo cual indica que una función del programa esta aún en proceso en el esclavo. El contador de eventos muestra que el Procesador ha registrado ocho (0x0008) eventos.

Implementación

El diagrama de flujo de la figura 68 muestra la implementación de la función 11:

Figura 68. Diagrama de flujo de la función 11



1.9. FUNCIÓN 12 (0x0C): GET COMM EVENT LOG

Descripción

Devuelve una palabra de estado, un contador de eventos, un contador de mensajes, y un campo de bytes de eventos del esclavo. La Consulta General no está permitida.
La palabra de estado y el contador de eventos es idéntico a las devueltas por la función Solicitud del Contador de Evento de Comunicaciones (11 ó 0x0B).

El contador de mensajes contiene la cantidad de mensajes procesados por el esclavo desde su último reinicio, operación de limpieza de los contadores, o arranque. Este contador es idéntico al devuelto por la función de Diagnóstico (código 0x08), subfunción Devolver Contador de Mensajes de Bus (código 0x0B).

El campo de bytes de evento contiene 0-64 bytes, cada byte corresponde al estado de una operación Modbus, envío o recepción, hacia el esclavo. Los eventos los guarda el esclavo en este campo en orden cronológico. El byte 0 es el evento más reciente. Cada nuevo byte elimina el byte más antiguo del campo.

Consulta

La figura 69 muestra un ejemplo de solicitud del diario de eventos de comunicaciones en el Procesador Modbus:

Figura 69. Solicitud del Diario de Eventos de Comunicaciones -Consulta

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**0C**
CRC Hi	**CRC**
CRC Lo	**CRC**

Respuesta

La respuesta normal contiene un campo de palabra de estado de dos bytes, un contador de eventos de dos bytes, un contador de mensajes de dos bytes, y un campo que contiene de 0-64 bytes de eventos. Un contador de un byte define la longitud total de la información en estos cuatro campos. La figura 70 muestra un ejemplo de una respuesta para la consulta realizada en el ejemplo anterior

Figura 70. Solicitud del Diario de Eventos de Comunicaciones -Respuesta

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**0C**
Conteo de bytes	**08**
Estado Hi	**00**
Estado Lo	**00**
Contador de eventos Hi	**00**
Contador de eventos Lo	**08**
Contador de mensajes Hi	**00**
Contador de mensajes Lo	**20**
Evento 0	**20**
Evento 1	**00**
CRC Hi	**-**
CRC Lo	**-**

En este ejemplo, la palabra de estado es 0x0000, indicando que el esclavo no esta procesando ninguna función de programa. El contador de eventos dice que se han contado ocho (0x0008) eventos en el esclavo. El contador de mensajes dice que se han procesado 32 (0x0020) mensajes.

El evento de comunicaciones más reciente se muestra en el byte 0 de Eventos. Su contenido (0x0020) muestra que el esclavo ha entrado últimamente en Modo Escucha. El evento anterior se muestra en el byte 1 de Eventos. Su contenido (0x0000) muestra que el esclavo recibió un Reinicio de Comunicaciones.

La disposición de los bytes de evento de respuestas se describe a continuación:

Contenido de los bytes de Evento

El byte de evento devuelto por la función de Consulta del Diario de Eventos de Comunicación puede se de cuatro tipos:

- **Evento Recepción de un Esclavo Modbus**

  Este tipo de byte de evento es almacenado por el esclavo cuando recibe un mensaje de consulta. Lo almacena después de procesar el mensaje. Este evento se define poniendo a "1" el bit 7. Los otros bits estarán en "1" si la condición correspondiente es verdadera. La relación de bits se muestra en la Figura 71.

Figura 71. Contenido del byte de evento: Recepción de un esclavo Modbus

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	Recibida Emisión general	Solo en Modo Escucha	Sobre escritura de carácter	No usado	No usado	Error de comuni-caciones	No usado

- **Evento Envío de un Esclavo Modbus**

  Este tipo de byte de evento es almacenado por el Procesador cuando finaliza un mensaje de consulta. Se almacena si el esclavo ha devuelto una respuesta normal, de excepción, o no ha respondido. Este evento se define colocando a "0" el bit 7, dejando el bit 6 a "1". Los otros bits podrán estar a "1" si la condición correspondiente es verdadera. La relación de bits se muestra en la figura 72.

Figura 72. Contenido del byte de evento: Recepción de un esclavo Modbus

Bit	Contenido
0	Enviada excepción de lectura (Códigos de excepción 1-3)
1	Enviada excepción aborto del esclavo (Código de excepción 4)
2	Enviada excepción esclavo ocupado (Código de excepción 5-6)
3	Enviada excepción Slave-Program-NAK (Código de excepción 7)
4	Ha ocurrido un time out en escritura
5	Normalmente solo en modo escucha
6	1
7	0

- **El Esclavo Entra en Modo de Solo Escucha**

  Este tipo de byte de evento lo almacena el esclavo cuando entra en Modo de Solo Escucha. El evento se define por el contenido 04 hex. La relación de bits se muestra en la figura 73.

Figura 73. Contenido del byte de evento: Entra en Modo de Solo Escucha

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	1	0	0

- **El Esclavo Inicia un Restablecimiento de Comunicaciones**

  Este tipo de byte de evento lo almacena el esclavo cuando se restablece su puerto de comunicaciones. El esclavo puede restablecerse por la función de Diagnostico (código 0x08), mediante la subfunción Opción Restablecer Comunicaciones (código 0x01).

  Esta función deja también al esclavo en el modo "Continua ante un Error" o "Para ante un Error". Si el esclavo queda en modo "Continúa ante un error", el byte de evento se añade al diario de eventos. Si el esclavo queda en modo "Para ante un Error", el byte de evento se añade al diario de eventos y el resto del diario es puesto a cero. El evento se define por un contenido de cero. La relación de bits se muestra en la figura 74.

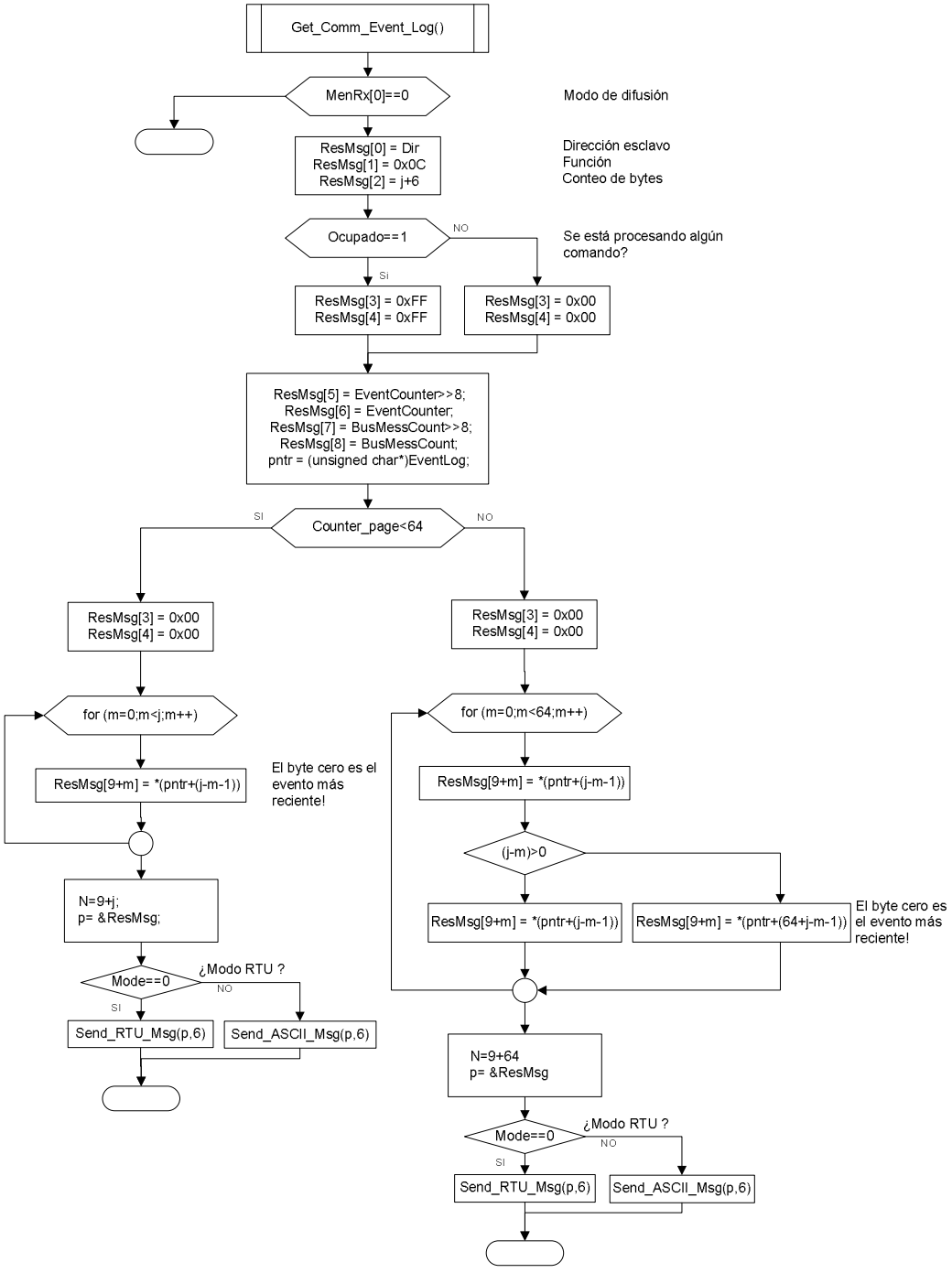
Figura 74. Contenido del byte de evento: Restablecimiento de comunicaciones

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0

Implementación

La figura 75 muestra el diagrama de flujo de la función 12

Figura 75. Diagrama de flujo de la función 12



1.10. FUNCIÓN 15 (0x0F): WRITE MULTIPLE COILS

Descripción

Esta función se utiliza para forzar varias salidas (referenciadas a 0x000) consecutivas, ya sea a ON o a OFF. Cuando es una consulta general, la función escribe las mismas referencias de salida en todos los esclavos conectados.

Consulta

En el mensaje de consulta se especifican las referencias de las salidas que se desean forzar. Las salidas se direccionan comenzando con cero: La salida 1 se direcciona como cero.

La petición de estado ON / OFF está especificada por el contenido del campo de información de la consulta. Un valor lógico "1" en una posición de bit en el campo de datos determina que la bobina correspondiente pase a ON. Un valor lógico "0" determina que la bobina pase a OFF.

En la figura 76 se muestra un ejemplo de solicitud para forzar 10 bobinas, comenzando en la bobina 12 (direccionada como 11 ó 0x0B), en el procesador Modbus. El campo de datos del mensaje de consulta posee dos bytes: 0xCD y 0x01
Los bits corresponden a las salidas de la siguiente forma:

Byte	**0xCD**								**0x01**							
Bit	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1
salida	19	18	17	16	15	14	13	12	-	-	-	-	-	-	21	20

Figura 76. Ejemplo de consulta Modbus para la función 15

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**0F**
Dirección Salida Hi	**00**
Dirección Salida Lo	**0B**
Número de salidas Hi	**00**
Número de salidas Lo	**0A**
Conteo de Bytes	**02**
Datos a forzar (salidas 19-12)	**CD**
Datos a forzar (salidas 21-20)	**01**
CRC Hi	**CRC**
CRC Lo	

Respuesta

La respuesta normal devuelve la dirección del esclavo, el código de operación, la dirección de inicio y la cantidad de salidas forzadas. La figura 77 muestra un ejemplo de respuesta a la consulta realizada en el ejemplo anterior.

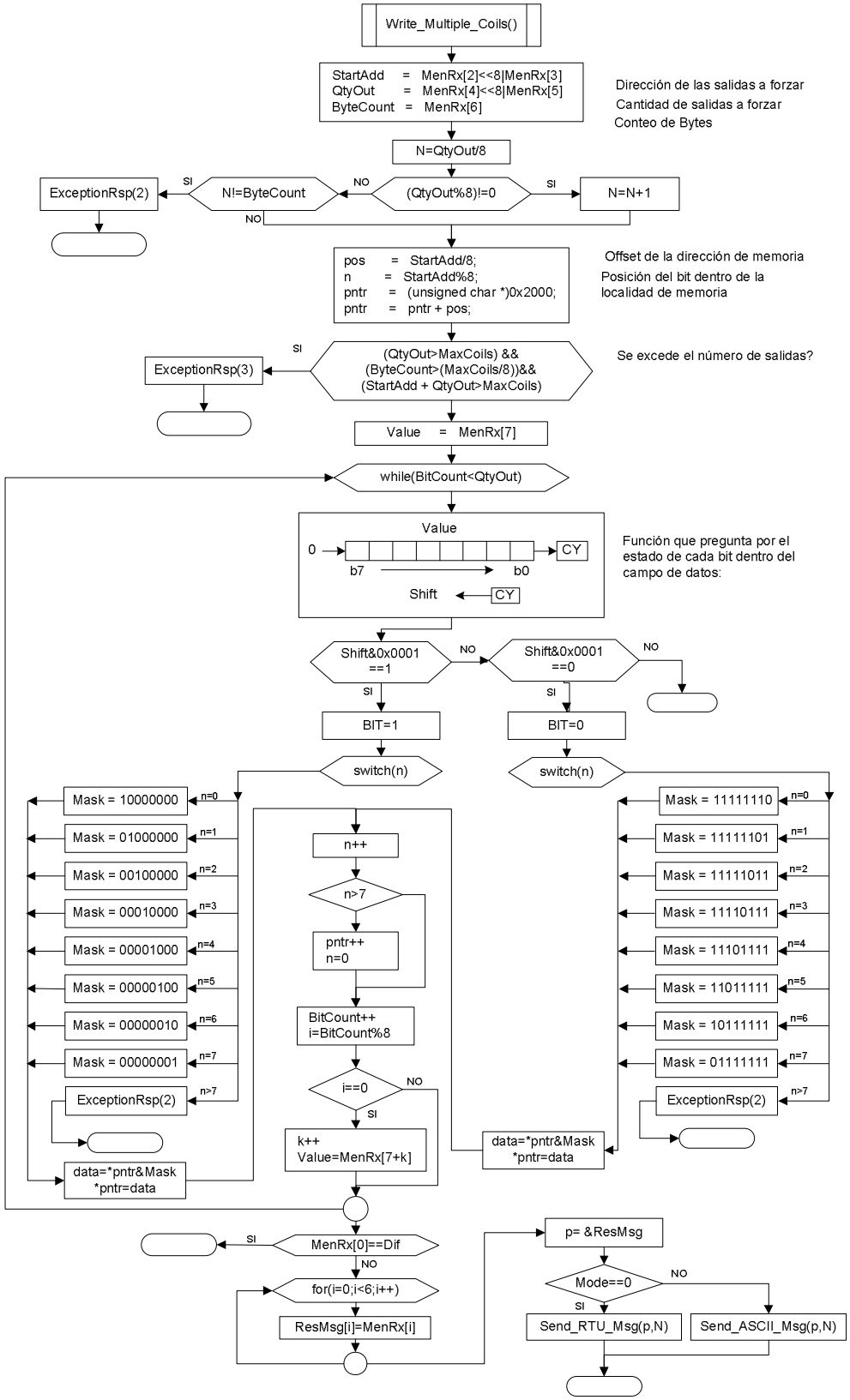
Figura 77. Ejemplo de respuesta Modbus para la función 15

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**0F**
Dirección Salida Hi	**00**
Dirección Salida Lo	**0B**
Número de salidas Hi	**00**
Número de salidas Lo	**0A**
CRC Hi	**A4**
CRC Lo	**0E**

Implementación

El diagrama de flujo de la figura 78 muestra la implementación de la función 15.

Figura 78. Diagrama de flujo de la función 15



1.11. FUNCIÓN 17 (0x11): REPORT SLAVE ID

Descripción

Esta función se utiliza para leer la descripción, el estado actual, o cualquier otra información específica del Procesador Modbus o dispositivo remoto.

Consulta

En la figura 79 se muestra un ejemplo de consulta de una solicitud de Informe de ID y estado del procesador:

Figura 79. Ejemplo de consulta Modbus para la función 17

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**11**
CRC Hi	**CRC**
CRC Lo	-

Respuesta

En el Procesador Modbus se programó la función 0x11 para que entregara información acerca de los siguientes parámetros: identificación del procesador, estado, número máximo de salidas, número máximo de entradas, número máximo de registros de entrada y de salida, y parámetros de la configuración realizada por software. En la figura 80 se muestra la respuesta del Procesador que reporta los parámetros descritos anteriormente.

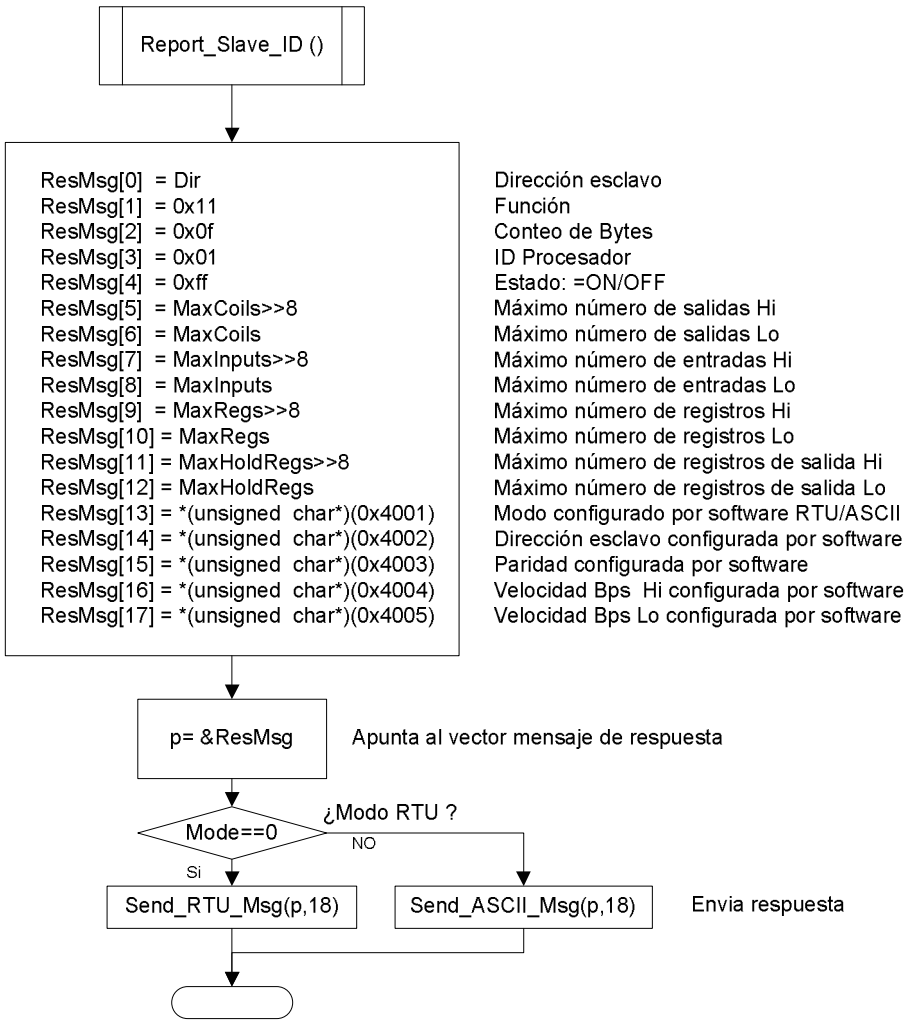
Figura 80. Respuesta Modbus para la función 17

Respuesta	
Campo	(Hex.)
Dirección esclavo	**01**
Código de función	**11**
Conteo de bytes	**0F**
ID del Procesador	**01**
Estado (0x00 = OFF, 0xFF = ON)	**FF**
Máximo número de salidas Hi	**07**
Máximo número de salidas Lo	**D0**
Máximo número de entradas Hi	**07**
Máximo número de entradas Lo	**D0**
Máximo número de registros de entrada Hi	**00**
Máximo número de registros de entrada Lo	**7D**
Máximo número de registros de salida Hi	**00**
Máximo número de registros de salida Lo	**7D**
Modo configurado por software (0x00 = RTU, 0xFF = ASCII)	**00**
Dirección esclavo configurado por software (0x01 a 0xF7)	**01**
Paridad configurada por software (0x00 = None, 0x01=Odd, 0x02=Even)	**00**
Velocidad en Baudios configurada por software Hi	**4B**
Velocidad en Baudios configurada por software Lo	**00**
CRC Hi	**CRC**
CRC Lo	**-**

Implementación

El diagrama de flujo de la figura 81 muestra la implementación de la función 17:

Figura 81. Diagrama de flujo de la función 17



1.12. FUNCION 22 (0x16): MASK WRITE REGISTER

Descripción

Esta función se utiliza para modificar el contenido de un registro de salida (*holding register*) específico mediante una combinación de una máscara AND, una máscara OR y el contenido del registro. La función puede ser utilizada para colocar a uno o a cero bits individuales. La Consulta General no está permitida.

Consulta

En el mensaje de consulta se especifica el registro de salida a ser modificado, el dato de la máscara AND, y el dato de la máscara OR. Los registros se direccionan iniciando en cero. Los registros 1-16 se direccionan como: 0-15.

El algoritmo de la función es:

$$\text{Resultado} = \text{Registro AND Máscara\_And OR (Máscara\_Or AND } \overline{\text{Máscara\_And}})$$

Ejemplo:

	Hex	**Binario**
Contenido del registro	12	0001 0010
Máscara_And	F2	1111 0010
Máscara_Or	25	0010 0101
(Máscara_And)'	0D	0000 1101
Resultado	17	0001 0111

Si el valor de la Máscara-Or es cero, el resultado es simplemente la operación AND entre la Máscara_And y el valor actual del registro.

Si el valor de la Máscara_And es cero, el resultado es igual al valor de la Máscara_Or. En la figura 82 se muestra un ejemplo de escritura con máscara en el registro 3 del procesador Modbus.

Figura 82. Ejemplo de consulta Modbus para la función 22

Consulta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**16**
Dirección registro Hi	**00**
Dirección registro Lo	**02**
Máscara_And Hi	**00**
Máscara_And Lo	**F2**
Máscara_Or Hi	**00**
Máscara_Or Lo	**25**
CRC Hi	**EF**
CRC Lo	**EE**

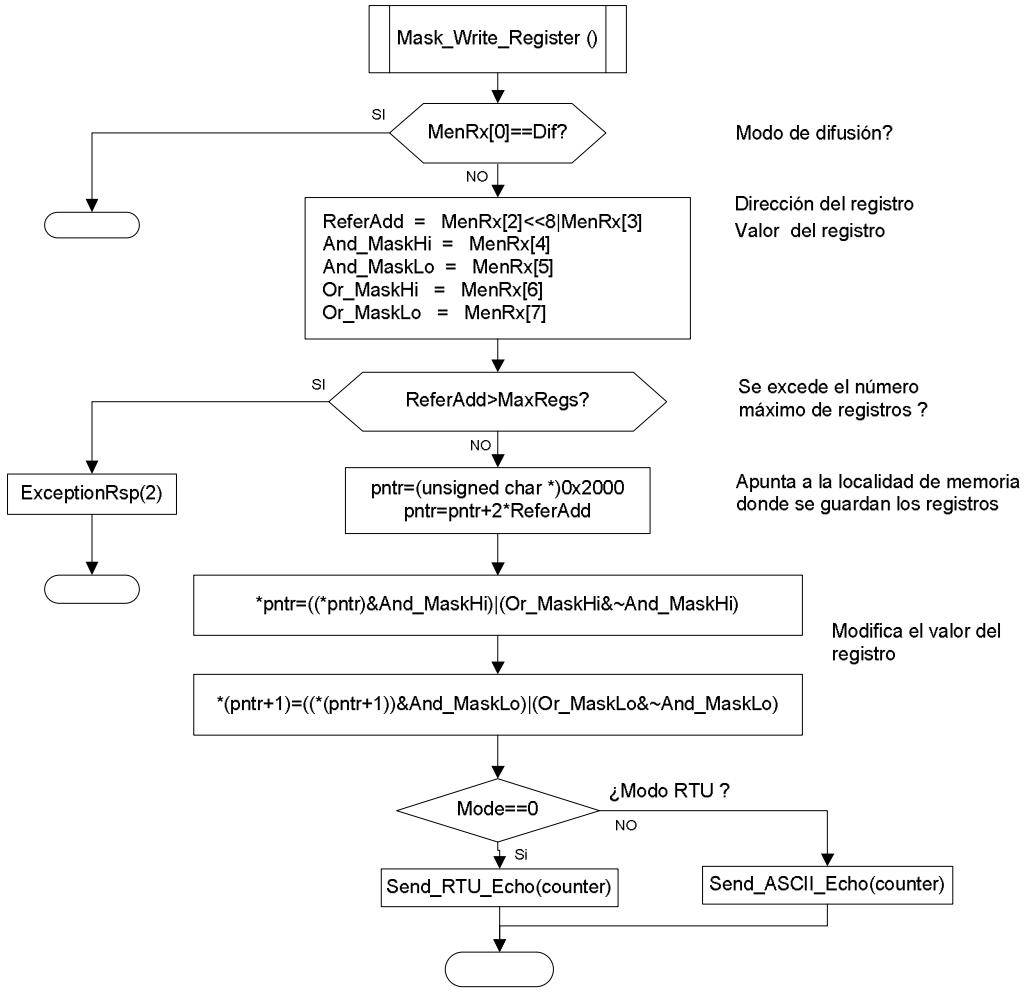
Respuesta

La respuesta normal es un eco de la consulta. La respuesta se devuelve después que se ha actualizado el valor del registro.

Implementación

El diagrama de flujo de la figura 83 muestra la implementación de la función 22 (0x16):

Figura 83. Diagrama de flujo de la función 22



1.13. FUNCION 23 (0X17): READ/WRITE MULTIPLE REGISTERS

Descripción

Esta función desempeña una combinación de una operación de lectura y una operación de escritura en una sola transacción MODBUS. La operación de escritura se desarrolla antes que la de lectura.

Consulta

En la figura 84 se muestra un ejemplo de consulta de lectura para leer 2 registros que comienzan en el registro 3, y para escribir 3 registros que comienzan en el registro 10 del procesador Modbus.

Figura 84. Ejemplo de consulta Modbus para la función 23

Consulta	
Campo	(Hex.)
Dirección Esclavo	**01**
Función	**17**
Dirección Lectura Hi	**00**
Dirección Lectura Lo	**02**
Cantidad a Leer Hi	**00**
Cantidad a Leer Hi	**02**
Dirección Escritura Hi	**00**
Dirección Escritura Lo	**09**
Cantidad a escribir Hi	**00**
Cantidad a escribir Lo	**03**
Conteo de bytes	**06**
Dato a escribir 1 Hi	**00**
Dato a escribir 1 Lo	**FF**
Dato a escribir 2 Hi	**00**
Dato a escribir 2 Lo	**FF**
Dato a escribir 3 Hi	**00**
Dato a escribir 3 Lo	**FF**
CRC Hi	**CRC**
CRC Lo	**-**

En el mensaje de consulta se especifica la dirección del registro inicial y la cantidad de registros de salida (*holding registers*) a leer, así como el número de registros de salida y los datos a ser escritos. El conteo de bytes especifica el número de bytes que siguen en el campo de datos a escribir.

Respuesta

La respuesta normal contiene los datos del grupo de registros que se han leído y el contador de bytes indica la cantidad de bytes leídos. En la figura 85 se muestra un ejemplo de respuesta para la consulta del ejemplo anterior.

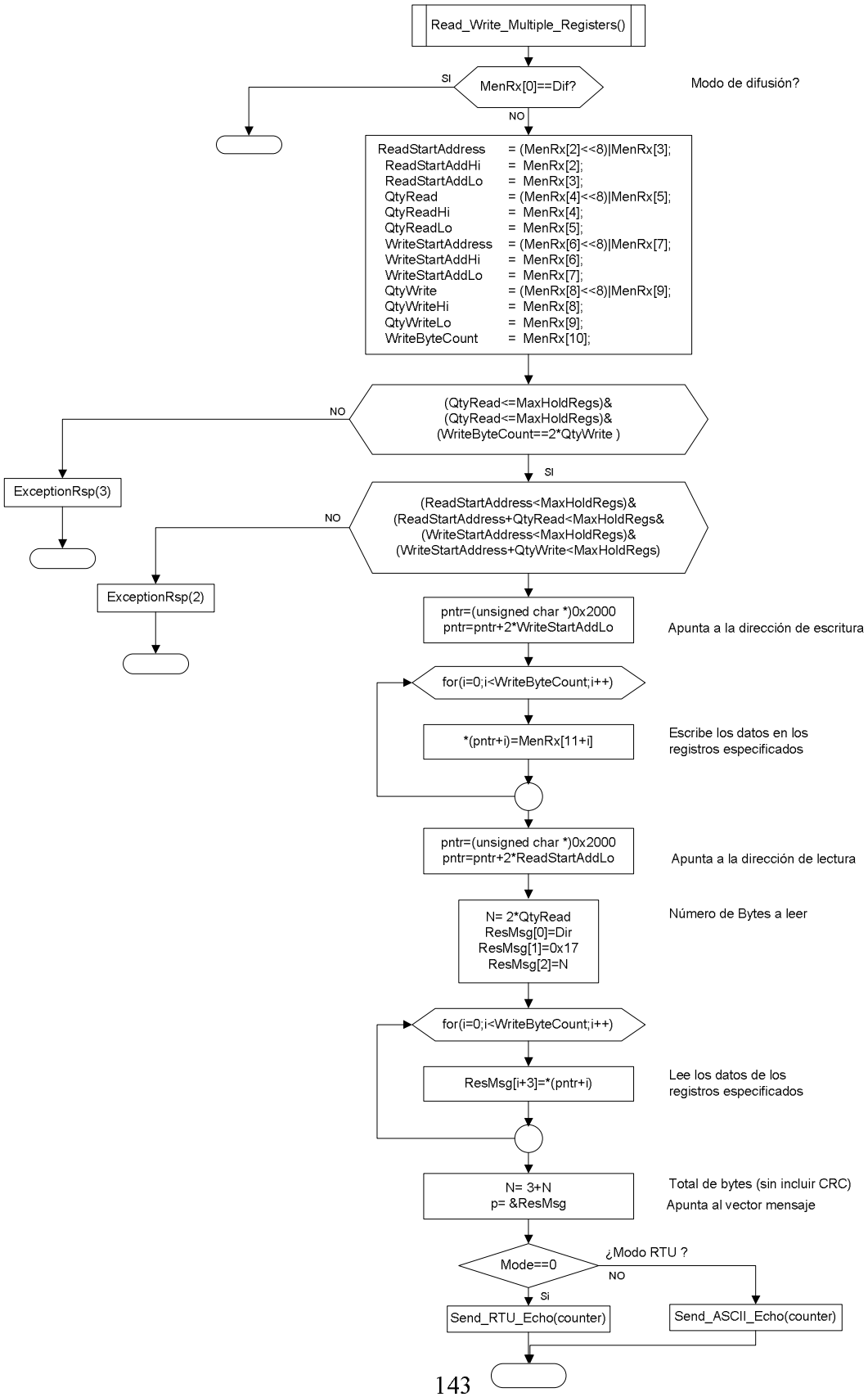
Figura 85. Ejemplo de respuesta Modbus para la función 23

Respuesta	
Campo	(Hex.)
Dirección de Esclavo	**01**
Función	**17**
Conteo de Bytes	**04**
Dato leído 1 Hi	**08**
Dato leído 1 Lo	**80**
Dato leído 2 Hi	**1B**
Dato leído 2 Lo	**01**
CRC Hi	**31**
CRC Lo	**9F**

Implementación

El diagrama de flujo de la figura 86 muestra la implementación de la función 23 (0x17):

Figura 86. Diagrama de flujo de la función 23



1.14. FUNCION 100 (0X64): USER FUNCTION

Descripción

Esta función es una función definida por el usuario. En el Procesador Modbus se utiliza para configurar por software los siguientes parámetros de comunicación:

- Dirección de esclavo
- Interfaz: RS-232 o RS-485
- Modo : RTU o ASCII
- Paridad: Ninguna, Par o Impar.
- Rata de Baudios: 150, 300, 600,1200, 2400, 4800, 9600 y 19200 Bps

Consulta

En la figura 87 se muestra un ejemplo de consulta de configuración por software del Procesador.

Figura 87. Ejemplo de consulta Modbus para la función 100

Consulta	
Campo	(Hex.)
Dirección actual del esclavo	**01**
Función	**64**
Interfaz (0x00=RS-232, 0xFF= RS-485)	**00**
Modo (0x00 = RTU, 0xFF = ASCII)	**00**
Dirección esclavo a configurar (0x01 a 0x7F)	**01**
Paridad(0x00= Ninguna, 0x01=Impar, 0x10 =Par)	**00**
Baud Rate Hi	**48**
Baud Rate Lo	**00**
CRC Hi	**CRC**
CRC Lo	**-**

Respuesta

La respuesta es un eco de la consulta. El procesador responde después de haber guardado en la memoria Flash los datos de configuración del Procesador.

En la figura 88 se muestra la respuesta del Procesador que reporta los parámetros descritos anteriormente.

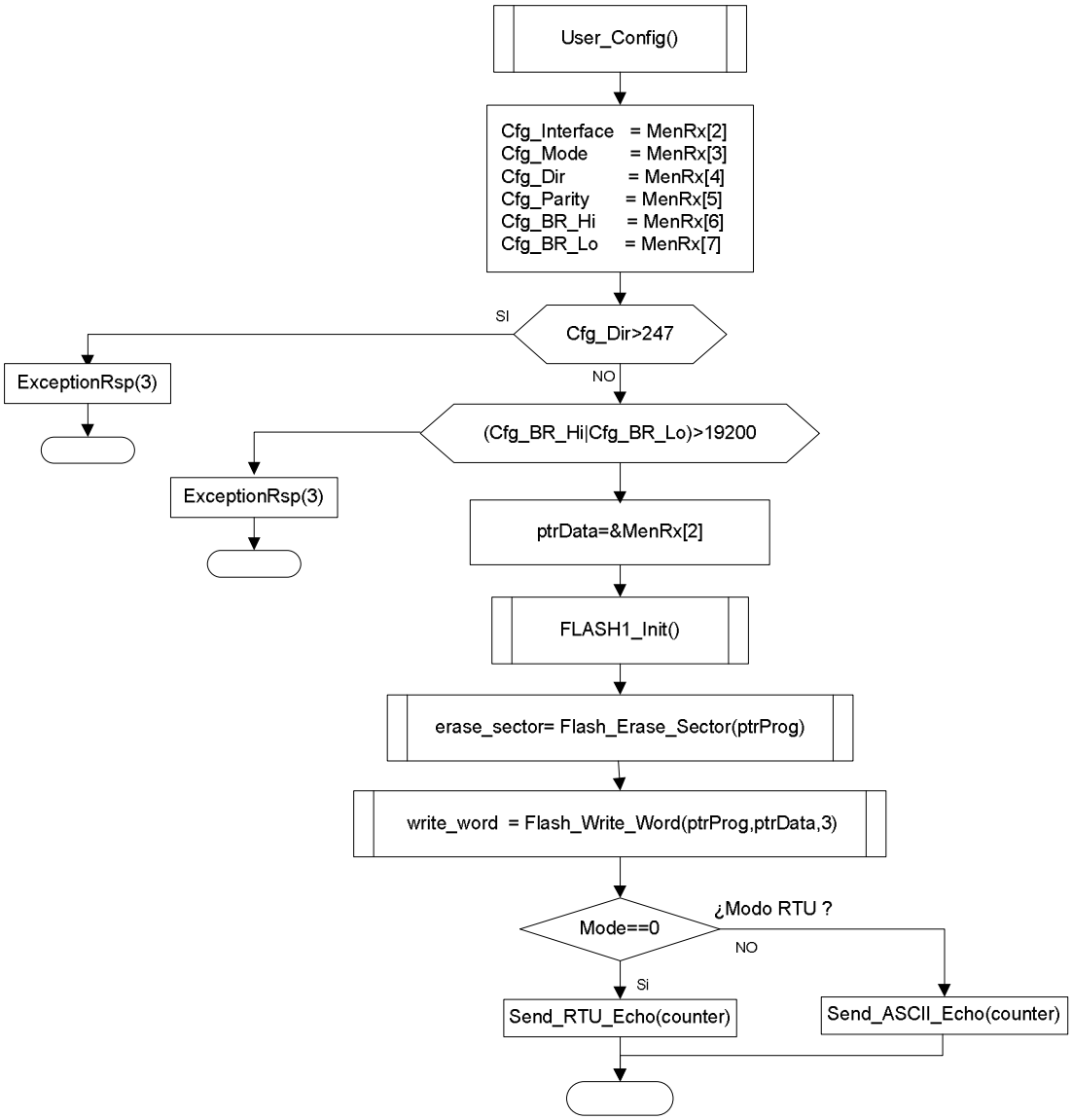
Figura 88. Respuesta Modbus para la función 100

Consulta	
Campo	(Hex.)
Dirección actual del esclavo	**01**
Función	**64**
Interfaz (0x00= RS-232, 0xFF= RS-485)	**00**
Modo (0x00 = RTU, 0xFF = ASCII)	**00**
Dirección esclavo a configurar (0x01 a 0x7F)	**01**
Paridad(0x00= Ninguna, 0x01=Impar, 0x10 =Par)	**00**
Baud Rate Hi	**48**
Baud Rate Lo	**00**
CRC Hi	**CRC**
CRC Lo	**-**

Implementación

El diagrama de flujo de la figura 89 muestra la implementación de la función 100 (0x64):

Figura 89. Diagrama de flujo de la función 100



ANEXO II

PROGRAMACION DE

LOS REGISTROS DEL HCS12

2.1 REGISTROS DE LA SCI

2.1.1 REGISTROS DE TASA DE BAUDIOS (SCIBDH, SCIBDL)

Los registros de Tasa de Baudios (SCIBDH, SCIBDL) se utilizan para determinar la velocidad en baudios de la SCI.

En la figura 90 se muestran los registros SCIBDH y SCIBDL.

Figura 90. Registros de Rata de Baudios (SCIBDH, SCIBDL)

Bit	15	14	13	12	11	10	9	8
	IREN	**TNP1**	**TNP0**	**SBR12**	**SBR11**	**SBR10**	**SBR9**	**SBR8**
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	IREN	**TNP1**	**TNP0**	**SBR12**	**SBR11**	**SBR10**	**SBR9**	**SBR8**
Reset	0	0	0	0	0	0	0	0

A continuación se detallan los bits programados:

IREN = 0 (*Infrared Enable Bit*) Deshabilita el submódulo infrarrojo

TPN1 TPNO = 00 (*Transmitter Narrow Pulse Bits*) Habilita al transmisor para enviar pulsos de 3/16 de ancho.

SBR[12:0] – (*SCI Baud Rate Bits*). La rata de baudios se calcula de la siguiente forma:

$$\text{SCI BR} = \frac{\text{SCI module clock}}{16 * \text{SBR}[12:0]} \qquad (1)$$

El reloj del módulo SCI es el mismo del Bus. En este caso la frecuencia de trabajo del Procesador Modbus es de 8 MHz. En la tabla 14 se listan las diferentes velocidades de transmisión que se programaron en el Procesador Modbus.

Tabla 14. Programación de la velocidades de transmisión para el Procesador Modbus

Bits SBR[12:0]	Velocidad calculada [Bps]	Reloj del Transmisor [Hz]	error
26	19200	19230.8	0.16 %
52	9600	9615.4	0.16%
104	4800	4807.7	0.15 %
208	2400	2403.8	0.15 %
417	1200	1199	0.08 %
833	600	600.2	0.033 %
1666	300	300.12	0.04%
3333	150	150.02	0.03 %

Debido a la división entera del módulo de reloj, la frecuencia calculada difiere de la frecuencia real del transmisor en menos del 0.2 %.

2.1.2 REGISTRO DE CONTROL 1 (SCICR1)

El registro de control SCICR1 se utiliza principalmente para determinar el tipo de paridad y el formato de la trama recibida (8 bits de datos), entre otras opciones. En la figura 91 se muestra el registro SCICR1

Figura 91. Registro de Control (SCICR1)

Bit	7	6	5	4	3	2	1	0
	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

A continuación se detallan los bits programados:

LOOPS = 0 (*Loop Select Bit*) Modo de operación normal

SCISWAI = 0 (*SCI Stop in Wait Mode Bit*) SCI habilitado en modo de espera.

RSRC = 0 (*Receiver Source Bit*) Receptor y transmisor conectados internamente.

M = 0 (*Data Format Mode Bit*) 1 bit de inicio, 8 bits de datos y 1 bit de parada

WAKE = 0 (*Wakeup Condition Bit*) Línea *Idle* es la condición que despierta la SCI

RSRC = 0 (*Receiver Source Bit*) Receptor y transmisor conectados internamente.

ILT = 0 (*Idle Line Type Bit*) El conteo del bit del carácter *Idle* comienza después del carácter de inicio.

PE = 0 (*Parity Enable Bit*) Función de paridad deshabilitada

= 1 Función de paridad habilitada

PT = 0 (*Parity Type Bit*) Paridad Par

= 1 Paridad Impar

2.1.3 REGISTRO DE CONTROL 2 (SCICR2)

El registro de control SCICR2 se utiliza principalmente para habilitar la interrupción por receptor lleno, habilitar el transmisor y habilitar el receptor, entre otras opciones. En la figura 92 se muestra la estructura del registro SCICR1

Figura 92. Registro de Control 2 (SCICR2)

Bit	7	6	5	4	3	2	1	0
	TIE	**TCIE**	**RIE**	**ILIE**	**TE**	**RE**	**RWU**	**SBK**
Reset	0	0	0	0	0	0	0	0

A continuación se detallan los bits programados:

TIE = 0 (*Transmitter Interrupt Enable Bit*) Deshabilita la interrupción por registro de transmisión de datos vacío.

TCIE = 0 (*Transmission Complete Interrupt Enable Bit*) Deshabilita la interrupción por transmisión completa.

RIE = 1 (*Receiver Full Interrupt Enable Bit*) Habilita la interrupción por registro de recepción de datos lleno. (Esta es la única interrupción de la SCI que utilizada el Procesador Modbus)

ILIE = 0 (*Idle Line Interrupt Enable Bit*) Deshabilita la interrupción por línea ocupada

TE = 1 (*Transmitter Enable Bit*) Habilita el transmisor de la SCI.

RE = 1 (*Receiver Enable Bit*) Habilita el receptor de la SCI

RWU = 0 (*Receiver Wakeup Bit*) Operación normal después del carácter de inicio.

PE = 0 (Send *Break Bit*) No envía caracteres de *Break*

2.1.4 REGISTRO DE ESTADOS 1 (SCISR1)

Los registros de estados SCISR1 y SCISR2 proporcionan entradas al Microcontrolador para la generación de interrupciones. También, estos registros pueden ser revisados por el Microcontrolador para chequear el estado de cada bit.

Los procesos de borrado de banderas requieren que el registro de estados sea leído seguido de una lectura o escritura del registro de datos de la SCI (SCIDR).

En la figura 93 se muestra la estructura del registro SCISR1

Figura 93. Registro de Estados 1 (SCISR1)

Bit	7	6	5	4	3	2	1	0
	TDRE	**TC**	**RDRF**	**IDLE**	**OR**	**NF**	**FE**	**PF**
Reset	0	0	0	0	0	0	0	0

A continuación se detallan los bits de estados o de condición de la SCI utilizados:

TDRE (*Transmit Data Register Empty Flag*) La bandera TDRE se coloca en 1 cuando el registro de desplazamiento del transmisor recibe un dato proveniente del registro de datos de la SCI. Cuando TDRE se coloca en 1, el registro de datos del transmisor (SCIDRH/L) se vacía y puede recibir un nuevo valor para transmitir. TDRE se borra por la lectura de la parte baja del registro de datos (SCIDRL).

TDRE = 1 Byte transferido al registro de desplazamiento y registro de datos vacío

= 0 No hay bytes transferidos al registro de desplazamiento.

TC (*Transmission Complete Flag*) La bandera TC se coloca en cero (0) cuando hay una transmisión en progreso. TC es puesta a uno (1) cuando la bandera TDRE está en uno (1). Cuando TC está en uno (1), la señal de salida en **Tx** permanece en el estado *Idle* (1 lógico). Para borrar TC se debe leer el registro de estados SCISR1 con TC en 1 y escribiendo entonces en la parte baja del registro de datos (SCIDRL).

TC = 0 Transmisión e progreso

= 1 No hay transmisión en progreso

RDRF (*Receive Data Register Full Flag*) RDRF es puesta a uno (1) cuando el dato en el registro de desplazamiento del receptor se transfiere al registro de datos de la SCI. RDRF se borra por la lectura del registro de estados (SCISR1) con RDRF en uno (1) y escribiendo entonces en la parte baja del registro de datos (SCIDRL).

RDRF = 1 Datos recibido está disponible en el registro de datos de la SCI

= 0 datos no disponibles en el registro de datos de la SCI

OR (*Overrun Flag*) OR ese coloca en uno (1) cuando el software falla al leer el registro de datos de la SCI antes que el registro de desplazamiento reciba la siguiente trama. El bit OR se borra por la lectura del registro de estado 1 (SCISR1) con OR en uno (1) y leyendo entonces el registro de datos de la SCI.

OR = 1 Sobre escritura (*overrun*)

= 0 No hay sobre escritura

PF (*Parity Error Flag*) PE ese coloca en uno (1) cuando se ha habilitado el chequeo de paridad y se recibe un dato que no concuerda con el bit de paridad. PF se borra por la lectura de los registros SCISR1 y SCIDRL

PF = 1 Hay Error de paridad

= 0 No hay error de paridad

2.1.5 REGISTRO DE DATOS (SCIDRH, SCIDRL)

Los registros de datos SCIDRH y SCIDRL guardan el valor de los datos recibidos o de los datos a transmitir. En la figura 94 se muestra la estructura del registro de datos de la SCI.

Figura 94. Registro de datos (SCIDRH, SCIDRL)

Bit	15	14	13	12	11	10	9	8
	R8	**T8**	**0**	**0**	**0**	**0**	**0**	**0**
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	R7	**R6**	**R5**	**R4**	**R3**	**R2**	**R1**	**R0**
	T7	**T6**	**T5**	**T4**	**T3**	**T2**	**T1**	**T0**
Reset	0	0	0	0	0	0	0	0

La función de cada uno de los bits del registro de datos se describe a continuación:

R8 (*Received Bit 8)* R8 es el noveno bit recibido cuando la SCI se configura en el formato de 9 bits de datos.

T8 (*Transmit Bit 8)* T8 es el noveno bit transmitido cuando la SCI se configura en el formato de 9 bits de datos

R7-R0 (*Received Bits)* Bits del 7 al 0 recibidos cuando la SCI se configura en el formato de 8 o 9 bits de datos.

T7-T0 (*Transmit Bits)* Bits del 7 al 0 transmitidos cuando la SCI se configura en el formato de 8 o 9 bits de datos.

2.2 REGISTROS DEL TEMPORIZADOR (TIM)

2.2.1 SELECCIÓN DE CAPTURA ENTRADA/ COMPARACION SALIDA (TIOS)

El registro **TIOS** se utiliza para configurar los canales del temporizador como entradas de captura o salidas de comparación. . En la figura 95 se muestra el registro TIOS.

Figura 95. Registro de selección captura entrada/comparación de salida (TIOS)

Bit	7	6	5	4	3	2	1	0
	IOS7	**IOS6**	**IOS5**	**IOS4**	**0**	**0**	**0**	**0**
Reset	0	0	0	0	0	0	0	0

A continuación se detallan los bits programados:

IOS [7:4] Configuración del canal como entrada de captura o salida de comparación.

IOS [7:4] = 1 El canal correspondiente actúa como salida de comparación

= 0 El canal correspondiente actúa como entrada de captura

En el Procesador Modbus se configuraron los canales 4,5 y 6 como salidas de comparación. Es decir, IOS6 = 1, IOS5 = 1 e IOS4 = 1.

2.2.2 REGISTRO DE CONTROL DEL SISTEMA TEMPORIZADOR 1 (TSCR1)

El registro **TSCR1** se utiliza para lanzar o parar el temporizador. En la figura 96 se muestra el registro TSCR1.

Figura 96. Registro de control del sistema temporizador 1 (TSCR1)

Bit	7	6	5	4	3	2	1	0
	TEN	**TSWAI**	**TSFRZ**	**TFFCA**	**0**	**0**	**0**	**0**
Reset	0	0	0	0	0	0	0	0

A continuación se detalla el propósito de cada bit:

TEN Bit de habilitación del temporizador

TEN = 1 Habilita el temporizador

= 0 Deshabilita el temporizador

En el Procesador Modbus se configuraron los canales 4,5 y 6 como salidas de comparación. Es decir, IOS6 = 1, IOS5 = 1 e IOS4 = 1.

TSWAI (*Timer Module Stops While in Wait)*

TSWAI = 0 Permite al temporizador seguir corriendo durante el modo WAIT

TSFRZ (*Timer Stops While in Freeze Mode)*

TSFRZ = 0 Permite al temporizador seguir corriendo durante el modo FREEZE

TFFCA (*Timer Fast Flag Clear All)*

TFFCA = 0 Permite borrar las banderas del temporizador.

2.2.3 REGISTRO DE CONTROL DEL SISTEMA TEMPORIZADOR 2 (TSCR2)

El registro **TSCR2** se utiliza para configurar el *prescaler* y así obtener la frecuencia del reloj del temporizador. En la figura 97 se muestra el registro TSCR2.

Figura 97. Registro de control del sistema temporizador 2 (TSCR2)

Bit	7	6	5	4	3	2	1	0
	TOI	**0**	**0**	**0**	**TCRE**	**PR2**	**PR1**	**PR0**
Reset	0	0	0	0	0	0	0	0

TOI (*Timer Overflow Interrupt Enable*) TOI = 0 Requisición de interrupción por hardware inhabilitada

TCRE (*Timer Counter Reset Enable)* TCRE = 0 Inhibe el *reset* del contador.

PR2, PR1, PR0 (*Timer Prescaler Select)* Estos tres bits seleccionan la frecuencia del temporizador a partir de la frecuencia del bus del microcontrolador como se muestra en la tabla 15.

Tabla 15. Selección de la frecuencia del reloj del temporizador

Frecuencia del bus = 8 MHz				
PR2	**PR1**	**PR0**	**Divide entre**	**Reloj timer**
0	**0**	**0**	1	125 ns
0	**0**	**1**	2	250 ns
0	**1**	**0**	4	500 ns
0	**1**	**1**	8	1µs
1	**0**	**0**	16	2 µs
1	**0**	**1**	32	4 µs
1	**1**	**0**	64	8 µs
1	**1**	**1**	128	16 µs

Para la implementación de la rutina Modbus RTU se utilizó una base de tiempo de 1 µs, lo que equivale a fijar los bits PR2 PR1 PR0 = 0 1 1 (3 decimal).

2.2.4 REGISTRO DE HABILITACION DE INTERRUPCIONES (TIE)

El registro **TIE** permite habilitar las interrupciones del temporizador. En la figura 98 se muestra la estructura del registro TIE

Figura 98. Registro de habilitación de interrupciones del *Timer* (TIE)

Bit	7	6	5	4	3	2	1	0
	C7I	**C6I**	**C5I**	**C4I**	**0**	**0**	**0**	**0**
Reset	0	0	0	0	0	0	0	0

A continuación se detalla el propósito de cada bit:

C7I-C4I (*Input Capture/Output Compare Interrupt Enable.*)

C7I-C4I = 1 Habilita la bandera de interrupción correspondiente (CnF en el registro TFLG1) para causar una interrupción del hardware.

2.2.5 REGISTRO PRINCIPAL DE BANDERAS DE INTERRUPCIONES (TFLG1)

El registro **TFLG1** contiene los estados de las banderas de interrupción de cada canal. En la figura 99 se muestra la estructura del registro TFLG1

Figura 99. Registro de habilitación de interrupciones del Timer (TIE)

Bit	7	6	5	4	3	2	1	0
	C7F	**C6F**	**C5F**	**C4F**	**0**	**0**	**0**	**0**
Reset	0	0	0	0	0	0	0	0

C7F-C4F (*Input Capture/Output Compare Channel Flag***)**

C7I-C4I = 1 Indica que ha ocurrido un evento de comparación. Estas banderas se colocan a 1 cuando ocurre un evento en el respectivo canal. Para borrarlas se debe escribir 1 en el bit CnF correspondiente.

2.2.6 REGISTRO DE CONTEO DEL TEMPORIZADOR (TCNT)

El registro **TCNT** contiene el módulo del contador ascendente de 16 bits del temporizador. En la figura 100 se muestra la estructura de este registro.

Figura 100. Registro de conteo del Timer (TCNT)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	tcnt 15	tcnt 14	tcnt 13	tcnt 12	tcnt 11	tcnt 10	tcnt 9	tcnt 8	tcnt 7	tcnt 6	tcnt 5	tcnt 4	tcnt 3	tcnt 2	tcnt 1	tcnt 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.3 REGISTROS DEL ADC

2.3.1 REGISTRO DE CONTROL DEL ATD 2 (ATDCTL2)

El registro **ATDCTL2** se utiliza para controlar el encendido del ADC, las interrupciones y el disparo externo. En la figura 101 se muestra el registro ATDCTL2.

Figura 101. Registro de control del ATD 2 (ATDCTL2)

Bit	7	6	5	4	3	2	1	0
	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
Reset	0	0	0	0	0	0	0	0

ADPU (*ATD Power Down*) Permite control on/off sobre el convertidor ADC reduciendo así el consumo de energía.

ADPU = 1 Funcionamiento normal del ATD

AFFC (*ATD Fast Flag Clear All*) Permite un borrado rápido de las banderas.

AFFC = 1 Cualquier acceso al registro de resultado borrará las banderas CCF automáticamente.

Los bits **AWAI** (*ATD Power Down in Wait Mode*), **ETRIGLE** (External Trigger Level/Edge Control), **ETRIGP** (*External Trigger Polarity*), **ETRIGE** (*External Trigger Mode Enable*) **ASCIE** (*ATD Sequence Complete Interrupt Enable*) y ASCIF (ATD *Sequence Complete Interrupt Flag*) se colocaron en sus valores de Reset.

2.3.2 REGISTRO DE CONTROL DEL ATD 3 (ATDCTL3)

El registro **ATDCTL3** controla la longitud de la secuencia de conversión de ADC, resultados FIFO y el comportamiento en modo *FREEZE*. En la figura 102 se muestra el registro ATDCTL3.

Figura 102. Registro de control del ATD 3 (ATDCTL3)

Bit	7	6	5	4	3	2	1	0
	0	**S8C**	**S4C**	**S2C**	**S1C**	**FIFO**	**FRZ1**	**FRZ0**
Reset	0	0	0	0	0	0	0	0

S8C, S4C, S2C, S1C (*Conversion Sequence Length*) Estos Bits controlan el número de conversiones por secuencia. La tabla 16 muestra todas las combinaciones.

Tabla 16. Codificación de la longitud de la secuencia de conversión

S8C	S4C	S2C	S1C	Número de conversiones por secuencia
0	0	0	0	**16**
0	0	0	1	**1**
0	0	1	0	**2**
0	0	1	1	**3**
0	1	0	0	**4**
0	1	0	1	**5**
0	1	1	0	**6**
0	1	1	1	**7**
1	0	0	0	**8**
1	0	0	1	**9**
1	0	1	0	**10**
1	0	1	1	**11**
1	1	0	0	**12**
1	1	0	1	**13**
1	1	1	0	**14**
1	1	1	1	**15**

Para el caso de la aplicación del Procesador Modbus se utilizó una conversión por secuencia y los demás bits: FIFO, FRZ1 y FRZ0 en sus valores por defecto.

2.3.3 REGISTRO DE CONTROL DEL ATD 4 (ATDCTL4)

El registro **ATDCTL4** permite seleccionar la frecuencia de conversión y la resolución del convertidor (8 ó 10 bits). En la figura 103 se muestra el registro ATDCTL4.

Figura 103. Registro de control del ATD 3 (ATDCTL3)

Bit	7	6	5	4	3	2	1	0
	SRES8	**SMP1**	**SMP0**	**PRS4**	**PRS3**	**PRS2**	**PRS1**	**PRS0**
Reset	0	0	0	0	0	1	0	1

A continuación se detallan los bits programados:

SRES8 *(A/D Resolution Select)* Este bit selecciona la resolución del convertidor

SRES8 = 1 8 bits de resolución

= 0 10 bits de resolución

SMP1, SMP0 (Sample *Time Select*) Estos dos bits seleccionan la longitud de la segunda fase del tiempo de muestreo en unidades de ciclos de reloj de conversión del ATD. El tiempo de muestreo del ADC consta de dos fases. En la primera fase se utilizan dos ciclos de reloj del ATD y en la segunda fase se añaden otros ciclos de reloj de acuerdo con los valores SMP1 y SMP0. La tabla 9 muestra los valores disponibles para la segunda fase de muestreo

Tabla 17. Longitudes disponibles para la segunda fase de muestreo

SMP1	SMP0	Longitud de la 2da fase de tiempo de muestreo
0	0	2 periodos de reloj de conversión del ADC
0	1	4 periodos de reloj de conversión del ADC
1	0	8 periodos de reloj de conversión del ADC
1	1	16 periodos de reloj de conversión del ADC

PRS4, PRS3, PRS2, PRS1, PRS0 (*ATD Clock Prescaler*) estos 5 bits representan el valor en binario del *Prescaler* (PRS). La frecuencia del reloj de conversión del ATD se calcula de la siguiente manera:

$$\text{frecuencia del ATD} = \frac{\text{frecuencia del Bus}}{[\text{PRS}+1]} * 0.5 \quad (2)$$

En la aplicación para el Procesador Modbus se seleccionó un valor de PRS = 0011 (binario), con lo cual la frecuencia del reloj del ATD es:

$$\text{frecuencia del ATD} = \frac{8\text{MHz}}{[3+1]} * 0.5 = 1\text{MHz}$$

O sea que, el período de conversión del reloj del ATD es de 1 µs. Por lo tanto el tiempo total de conversión será:

tconv = (ciclos de conv. 1era fase + ciclos de conv. 2da fase)*1µs

tconv = (2+8)*1µs = **10 µs**

2.4 REGISTROS DEL DAC

2.4.1 REGISTRO DE CONTROL DEL DAC 0 (DACC0)

El registro **DACC0** se utiliza para controlar el encendido del DAC, habilitar la salida y justificarlos datos en el registro de datos. En la figura 104 se muestra el registro DACC0.

Figura 104. Registro de control del DAC 0 (DACC0)

Bit	7	6	5	4	3	2	1	0
	DACE	**DACTE**	**0**	**0**	**DJM**	**DSGN**	**DACWAI**	**DACOE**
Reset	0	0	0	0	0	0	0	0

DACTE-DAC (*Test Enable*) Este bit está reservado para pruebas de fábrica

DACE-DAC (*Enable)* Habilita el funcionamiento del DAC.

DACE-DAC = 1 DAC está habilitado para convertir

= 0 DAC está deshabilitado y apagado.

DJM (*Data Register Data Justification*) controla la justificación de los datos en el registro de datos del DAC (DACD)

DJM = 1 Datos justificados a la derecha en el registro de datos del DAC

= 0 Datos justificados a la izquierda en el registro de datos del DAC

DSGN (*Data Register Signed*) Selecciona entre representación de datos con signo y sin signo en el registro de datos del ADC.

DSGN = 1 Representación con signo

= 0 Representación sin signo

DACWAI (*DAC Stop in WAIT mode*) Deshabilita el DAC durante el modo WAIT.

DACWAI = 1 DAC deshabilitado durante el modo WAIT

0 DAC está habilitado durante el modo WAIT.

DACOE (*DAC Output Enable*) Habilita la salida del pin DA0

DAC0E = 1 Salida en el pin DA0 está habilitada

= 0 salida en el pin DA0 no disponible para usos externo

2.5 REGISTROS DEL BLOQUE DE MEMORIA

2.5.1 REGISTRO DIVISOR DEL RELOJ DE LA FLASH (FCLKDIV)

El registro **FCLKDIV** se utiliza para controlar la temporización de los eventos en los algoritmos de programación y borrado. En la Figura 105 se muestra el registro FCLKDIV.

Figura 105. Registro divisor del reloj de la Flash (FCLKDIV)

Bit	7	6	5	4	3	2	1	0
	FDIVLD	**PRDIV8**	**FDIV5**	**FDIV4**	**FDIV3**	**FDIV2**	**FDIV1**	**FDIV0**
Reset	0	0	0	0	0	0	0	0

FDIVLD (*Clock Divider Loaded*) Indica la carga del divisor de reloj

FDIVLD = 1 El registro ha sido escrito desde el último *reset*

= 0 El registro no ha sido escrito

PRDIV8 (Enable *Prescaler by 8*)

PRDIV8 = 1 Divide entre 8 la frecuencia del reloj antes de alimentarla al divisor FCLKDIV.

PRDIV8 = 0 La frecuencia de entrada del oscilador alimenta directamente al FCLCKDIV

FDIV [5:0] (*Clock Divider Bits*) La combinación de PRDIV8 y FDIV [5:0] dividen efectivamente la entrada de reloj al módulo Flash para obtener una frecuencia de 150kHz - 200kHz.

2.5.2 REGISTRO DE ESTADOS DE LA FLASH (FSTAT)

El registro **FSTAT** define el estado de la máquina de comandos y verifica el estado de acceso, borrado y protección de la Flash. En la Figura 106 se muestra el registro FSTAT

Figura 106. Registro de estados de la Flash (FSTAT)

Bit	7	6	5	4	3	2	1	0
	CBEIF	**CCIF**	**PVIOL**	**ACCERR**	**0**	**BLANK**	**0**	**0**
Reset	0	0	0	0	0	0	0	0

CBEIF (*Command Buffer Empty Interrupt Flag*) Indica que los *buffers* de dirección, datos y comandos están vacíos y por tanto se puede iniciar una nueva secuencia.

CBEIF = 1 Los buffers están listos para aceptar un nuevo comando

= 0 Los buffers están llenos.

CCIF (*Command Complete Interrupt Flag.*) Indica que no hay más comandos pendientes.

CCIF = 1 Se completaron todos los comandos

= 0 Comandos en progreso

PVIOL (*Protection Violation.)* Indica que se está intentando programar o borrar en una dirección de memoria Flash protegida

PVIOL = 1 Ocurrió una violación de protección

= No falló.

ACCERR (*Flash Access Error*) Indica un acceso ilegal al bloque Flash causado por una violación de la secuencia de comandos o un comando ilegal.

ACCERR = 1 Ocurrió un error de acceso

= 0 No falló.

BLANK (*Array has been verified as erased*). La bandera BLANK indica que el bloque ha sido borrado

BLANK = 1 El bloque Flash verifica que ha sido borrado

0 = El bloque no ha sido borrado

2.5.3 REGISTRO DE COMANDOS DE LA FLASH (FCMD)

El registro **FCDM** define los comandos de la Flash. En la Figura 107 se muestra el registro FCDM

Figura 107. Registro de comandos de la Flash (FCDM)

Bit	7	6	5	4	3	2	1	0
	0	**CMDB**	**CMDB5**	**0**	**0**	**CMDB2**	**0**	**CDMB0**
Reset	0	0	0	0	0	0	0	0

CDBM = 0x05 Verifica borrado

= 0x20 Programa palabra

= 0x40 Borra sector

= 0x41 Borrado masivo