

Universidad Tecnológica de la Mixteca

DISEÑO Y CONSTRUCCIÓN DE UN CONTROLADOR DE CARGAS POR INTERFAZ SERIAL BAJO EL PROTOCOLO DE COMUNICACIONES MODBUS

TESIS

Para obtener el título de:
Ingeniero en Electrónica

Presenta:
Rodolfo Paz Carreño

Asesores:
M.C. Fermín Hugo Ramírez Leyva
M.C. Alejandro E. Ramírez González

Huajuapán de León, Oaxaca.

Diciembre de 2007

Dedicatoria

Dedico este trabajo a todas las personas que han formado parte de mi desarrollo como persona, especialmente a los profesores con los que tuve el gusto de compartir el aula durante mi estancia en la Universidad Tecnológica de la Mixteca.

Agradecimientos

A mis padres:

Rodolfo Paz Bravo y Carmen Gloria Carreño Arango por todo el apoyo que me han dado, y por sobre todas las cosas por darme la vida.

A mis hermanos:

Kary (la gruñona de la familia) y Julio (el desmadrazo) por ser parte importante de mi ser.

A mis asesores de tesis por tener gran paciencia para conmigo en la realización de este trabajo.

Y a todos mis amigos por dejarme ser parte de su vida.

Rodolfo.

Índice general

Índice general	vii
Índice de figuras	x
Índice de tablas	xiv
Resumen	xvi
Introducción	xviii
Objetivo general	xxi
Objetivos específicos	xxi
Contenido del documento	xxii
1. La energía eléctrica.	1
1.1. Facturación de la energía eléctrica.	1
1.2. Demanda máxima.	3
1.2.1. Controlador de demanda máxima.	5
1.3. Protocolo MODBUS.	7
1.3.1. Descripción del protocolo.	7
1.3.2. Implementación de MODBUS en una RCI de línea serial.	10
1.3.2.1. MODBUS RS-485.	10
1.3.2.2. MODBUS RS-232.	10
1.4. Interfaz USB.	12
1.4.1. Descripción.	13
1.4.2. Protocolo USB.	16
2. Diseño del Hardware.	21
2.1. Descripción general del sistema.	21
2.1.1. Módulo de comunicaciones.	22
2.1.1.1. Puerto USB.	23
2.1.1.2. Puerto RS-485.	26
2.1.2. Módulo de control.	28
2.1.3. Módulo de cargas.	31
2.2. Implementación del protocolo MODBUS en el ATmega32.	32
2.2.1. Uso de la memoria SRAM y EEPROM.	37
2.2.2. Memoria de recepción de tramas MODBUS.	39
2.2.3. Función, subfunción y direcciones de registros.	40
2.2.4. Validación de los valores de registros, códigos	

de excepción y ejecución de la función. -----	42
3. Desarrollo del software de pruebas (SPCC). -----	45
3.1. Matlab como software de desarrollo de instrumentación. ---	45
3.2. Requerimientos del programa. -----	46
3.3. Descripción general del SPCC. -----	47
3.3.1. Ejecución del programa. -----	49
3.3.2. Bloque de enviar y recibir trama. -----	51
3.4. Pruebas realizadas al SPCC. -----	54
4. Pruebas y resultados. -----	57
4.1. Pruebas del enlace de comunicaciones seriales. -----	57
4.2. Pruebas de apertura y cierre del controlador. -----	59
4.3. Pruebas del enlace en RS-485. -----	62
5. Conclusiones. -----	65
Referencias -----	69
Apéndice A. Código en lenguaje C del firmware utilizado en el CDECA. -----	A-1
Apéndice B. Manual de usuario del CDECA serial. -----	B-1
B.1. Modo de funcionamiento. -----	B-3
B.1.1. Uso de los puertos de comunicación. -----	B-3
B.1.2. Conexión de las cargas eléctricas. -----	B-5
B.1.3. Puerto de programación. -----	B-5
B.2. Diagramas del CDECA. -----	B-7
B.2.1. Diagrama esquemático. -----	B-7
B.2.2. Diagramas de montaje. -----	B-8
B.2.2.1. Lista de componentes. -----	B-9
B.2.3. Diagramas de circuito impreso. -----	B-11
Apéndice C. Manual de usuario del ComUSB. -----	C-1
C.1. Modo de funcionamiento. -----	C-2
C.1.1. Salidas en niveles TTL. -----	C-4
C.1.2. Salidas en niveles RS-232. -----	C-4
C.1.3. Señales de control para manejar niveles RS-485. -----	C-4
C.2. Diagrama esquemático. -----	C-5
C.3. Diagrama de montaje. -----	C-6
C.3.1. Lista de componentes. -----	C-6
C.4. Diagrama de circuito impreso. -----	C-8
Apéndice D. Manual de usuario del SPCC. -----	D-1
D.1. Modo de funcionamiento. -----	D-1
D.1.1. Descripción de los componentes de la interfaz gráfica. -----	D-2
D.1.2. Envío de una petición al CDECA. -----	D-3
D.2. Posibles errores. -----	D-4

Índice de figuras

Figura i.1. Diagrama a bloques del concepto de un controlador de demanda máxima centralizado. -----	xx
Figura i.2. Diagrama a bloques del controlador de demanda máxima distribuido. -----	xx
Figura 1.1. Curva típica de la demanda eléctrica en una planta. -----	4
Figura 1.2. Representación del modo dirigido de los mensajes MODBUS. -----	8
Figura 1.3. Estructura de un mensaje MODBUS. -----	8
Figura 1.4. Estructura de la interconexión de dispositivos utilizando la configuración de 2 hilos. -----	10
Figura 1.5. Configuración punto a punto MODBUS RS-232. -----	11
Figura 1.6. Conector DB-9 usado en el estándar RS-232. -----	12
Figura 1.7. Topología del bus USB. -----	14
Figura 1.8. Conectores usados en el protocolo USB. -----	16
Figura 1.9. Formato del token packet. -----	18
Figura 1.10. Trama del data packet. -----	18
Figura 1.11. Trama del Status packet. -----	19
Figura 2.1. Diagrama a bloques del CDECA. -----	22
Figura 2.2. Diagrama del módulo de comunicaciones. -----	23
Figura 2.3. Diagrama a bloques del CI FT232BM. -----	25
Figura 2.4. Diagrama esquemático del convertidor de nivel USB a TTL. -----	26
Figura 2.5. Diagrama de conexiones usado en el CI MAX489. -----	27
Figura 2.6. Encapsulado TQFP de 44 pines. -----	29
Figura 2.7. Diagrama a bloques del microcontrolador ATmega32. -----	30
Figura 2.8. Diagrama esquemático utilizado en el módulo de cargas. -----	32
Figura 2.9. Diagrama esquemático completo del CDECA. -----	33
Figura 2.10. Diagrama de flujo del programa principal del Microcontrolador. -----	35
Figura 2.11. Diagrama de flujo de la función atender_trama. -----	37

Figura 3.1.	Estructura jerárquica del SPCC. -----	48
Figura 3.2.	Entorno gráfico del SPCC. -----	49
Figura 3.3.	Diagrama de flujo general del SPCC. -----	50
Figura 3.4.	Diagrama de flujo para enviar la trama de petición MODBUS. -----	52
Figura 4.1.	Montaje para probar la tasa de errores. -----	58
Figura 4.2.	Diagrama de conexiones utilizado para probar el CDECA. -----	60
Figura 4.3.	Conexión de componentes utilizados para las pruebas del CDECA utilizando el puerto USB. -----	60
Figura 4.4.	Diagrama de flujo seguido en las pruebas del CDECA. -----	61
Figura 4.5.	Panel frontal del programa donde se activan todas las cargas. -----	62
Figura 4.6.	Panel frontal del programa al escoger un puerto serial no presente en la PC. -----	63
Figura 4.7.	Panel frontal del programa al no obtener una respuesta del dispositivo esclavo. -----	63
Figura B.1.	CDECA (vista superior). -----	B-2
Figura B.2.	CDECA (vista posterior). -----	B-2
Figura B.3.	Conexión del CDECA utilizada para hacer uso del puerto serial. -----	B-3
Figura B.4.	Conexión del CDECA utilizada para hacer uso del puerto RS-485 de dos líneas. -----	B-4
Figura B.5.	Conexión del CDECA utilizada para hacer uso del puerto RS-485 de cuatro líneas. -----	B-4
Figura B.6.	Conexión de la terminal con tornillos y la carga eléctrica a controlar. -----	B-5
Figura B.7.	Terminales móviles del CDECA. -----	B-6
Figura B.8.	Distribución de señales en el puerto de programación. -----	B-6
Figura B.9.	Diagrama esquemático del CDECA. -----	B-7
Figura B.10.	Diagrama de montaje utilizado en la placa base del CDECA. -----	B-8
Figura B.11.	Diagrama de montaje utilizado en el módulo desmontable. -----	B-8
Figura B.12.	Circuito impreso utilizado en la placa base del CDECA. -----	B-11
Figura B.13.	Circuito impreso utilizado en el módulo desmontable. -----	B-12
Figura C.1.	Vista superior del ComUSB. -----	C-2
Figura C.2.	Vista posterior del ComUSB (Lado soldadura). -----	C-2
Figura C.3.	Conexión del ComUSB a la PC. -----	C-3
Figura C.4.	Conexión utilizada para el manejo de niveles RS-485. -----	C-5
Figura C.5.	Diagrama esquemático del ComUSB. -----	C-5

Figura C.6. Diagrama de montaje utilizado en la placa del ComUSB. -----	C-6
Figura C.7. Circuito impreso utilizado en la placa del ComUSB. -----	C-8
Figura D.1. Ventana principal del SPCC. -----	D-2
Figura D.2. Pantalla obtenida al mandar ejecutar la función 0x2B. -----	D-4
Figura D.3. Pantalla resultante al no recibir la contestación del CDECA. -----	D-4

Índice de tablas

Tabla 1.1. Costo de los KWh por región para el 2007. -----	2
Tabla 1.2. Costo de la demanda máxima por región. -----	4
Tabla 1.3. Funciones más importantes de MODBUS. -----	9
Tabla 1.4. Definición de las líneas de la configuración de 2 hilos. -----	11
Tabla 1.5. Señales eléctricas del estándar RS-232. -----	11
Tabla 1.6. Asignación de los conectores del USB. -----	15
Tabla 1.7. Valores de PID por tipo de paquete. -----	17
Tabla 2.1. Descripción de las señales eléctricas del CI MAX489. -----	27
Tabla 2.2. Descripción de las variables más importantes del programa. -----	38
Tabla 2.3. Funciones implementadas en el CDECA. -----	40
Tabla 2.4. Subfunciones implementadas en el CDECA. -----	40
Tabla 2.5. Direcciones lógicas de los registros mantenidos en el CDECA. -----	41
Tabla 2.6. Direcciones lógicas para el estado de los relevadores en CDECA. -----	41
Tabla 2.7. Códigos de excepción soportados por el CDECA. -----	42
Tabla B.1. Lista de componentes utilizados en la placa base del CDECA. -----	B-9
Tabla B.2. Lista de componentes utilizados en el módulo desmontable. -----	B-10
Tabla C.1. Lista de componentes utilizados en la placa del ComUSB. -----	C-6

Resumen

El presente documento describe la forma en que se diseñó, construyó y probó un dispositivo electrónico llamado "controlador de cargas serial". Este dispositivo se encarga de activar o desactivar sus salidas a relevador, en función de los comandos MODBUS que recibe por su puerto de comunicaciones.

Las principales características del controlador de cargas, es que puede controlar hasta 8 cargas, posee un puerto RS-232, RS-485 y USB para comunicarse a otros dispositivos o a la computadora, y la interfaz de comunicaciones es independiente para que pueda ser reutilizada en otros proyectos.

El controlador de cargas fue desarrollado con base en el microcontrolador ATmega32 de la firma ATMEL. Su código fue completamente realizado en C, para facilitar su actualización o reuso. Mientras que el programa para probar al controlador se hizo en MATLAB, con el fin de aprender a utilizar el puerto serie desde este lenguaje de alto nivel.

Este sistema forma parte de un sistema para el control de la demanda máxima en forma distribuida. El presente trabajo es la continuación del proyecto de tesis presentado por el ingeniero Enmanuel Aparicio Velázquez titulado "codificador de pulsos KYZ bajo el protocolo de comunicaciones MODBUS para medidores electrónicos de energía eléctrica" [1]. Solo resta realizar el controlador de demanda con base en una computadora personal para completar el proyecto denominado *controlador de demanda máxima distribuido*.

Introducción

Actualmente la demanda de energía eléctrica es superior a la capacidad de generación en el país, esto significa que el servicio eléctrico no llega a cubrir las necesidades al 100%. Las alternativas existentes son la instalación de nuevas plantas generadoras, o establecer medidas para hacer un buen uso de la energía disponible, que a grandes rasgos consisten en la implementación de dispositivos enfocados al ahorro de energía.

El gobierno federal ha impulsado la segunda alternativa a través de programas que fomenten el consumo más eficiente y racional de la energía eléctrica. Uno de éstos es el Fideicomiso para el Ahorro de Energía (FIDE), el cual otorga estímulos fiscales a las empresas que emprendan medidas para ahorrar energía [URL1].

La forma de ahorrar energía, y así disminuir el consumo de la misma, es usando equipo más eficiente como: lámparas ahorradoras, motores de alta eficiencia, controladores automáticos de encendido y apagado del alumbrado, etc.

Los conceptos que la Comisión Federal de Electricidad (CFE) cobra a usuarios con una demanda mayor a 25kW son [URL2]:

- Cargo por Consumo de Energía.
- El factor de potencia (FP).
- Cargo por Demanda Máxima Medida.
- Derecho de alumbrado público (DAP).

El cargo por consumo de energía corresponde a la energía en Kilo-Watts hora (KWh), que el usuario consume durante el periodo de facturación.

El factor de potencia es una medida de qué tan eficientemente se consume la energía. Si el factor se mantiene entre 0.9 y 1 se bonifica, en caso contrario se hace un cargo.

Finalmente, la demanda máxima es una medida de la forma en la cual se ha realizado el consumo de energía, durante todo el período de facturación.

Una alternativa para limitar la demanda máxima, es apagando equipo eléctrico que no sea importante en un proceso de producción. De este modo se disminuye la demanda máxima y, sobre todo, se consigue un ahorro en el costo de la energía eléctrica. Hacer esto en forma manual no es práctico, por tal motivo se instalan equipos automáticos que miden la demanda máxima actual, y en base a esos valores se decide apagar o no algunas cargas.

Para medir el consumo de energía, es necesario contar con una lectura del consumo registrado por el medidor de la CFE. Para tal efecto, se pueden usar las salidas pulsadas que poseen los medidores que la CFE instala a usuarios industriales. Estas salidas se llaman *pulsos KYZ*.

Los *pulsos KYZ* son contactos secos de un relevador, que cambian de estado cada que el medidor registra un consumo de ciertos KWh (a esta razón de cambio se le llama constante de energía y se mide en KWh/pulso). Contando los pulsos y midiendo su duración, es posible conocer la energía demandada por el consumidor. Con esta información se estima la demanda máxima [1].

Actualmente existen **Controladores de Demanda Máxima**, los cuales son dispositivos que temporalmente apagan cargas eléctricas predeterminadas para mantener la demanda máxima bajo control cuando ésta rebasa un valor preestablecido. El punto prefijado debe ser cuidadosamente seleccionado para que no se afecte la producción en una industria o fábrica.

En el mercado existe una gran variedad de controladores de demanda, cada uno con diferentes grados de sofisticación, complejidad y costo. La figura i.1 muestra un diagrama a bloques para dar un ejemplo de la forma en la cual se conecta un controlador de demanda máxima típico.

En la mayoría de estos controladores, sus funciones están centralizadas en un solo equipo y es necesario cablear de éste a cada uno de los puntos a controlar, o al medidor de CFE.

Una alternativa al controlador de demanda centralizado, es un controlador realizado por módulos como el que se muestra en la figura i.2, los cuales se interconectan a través de un bus de comunicaciones industrial (PROFIBUS, CAN, MODBUS, etc.), o un enlace de comunicaciones inalámbrico. Con este arreglo es posible instalar cada módulo lo más cerca del equipo del cual se recibe o envía información.

El diagrama a bloque del controlador de demanda distribuido (figura i.2), consta de 3 módulos cuyas funciones son:

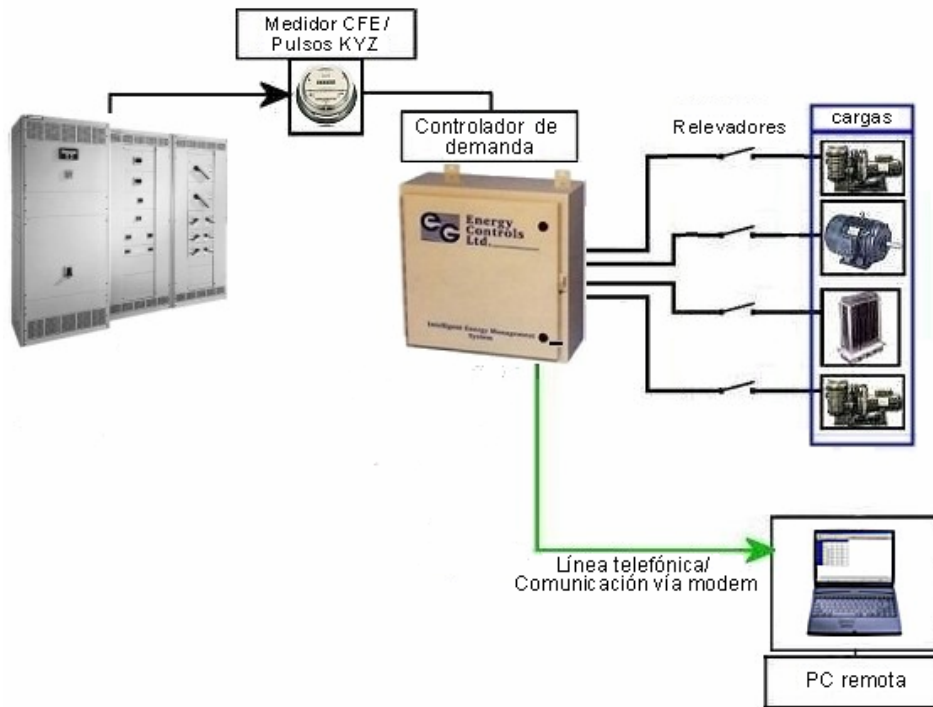


Figura i.1. Diagrama a bloques del concepto de un controlador de demanda máxima centralizado

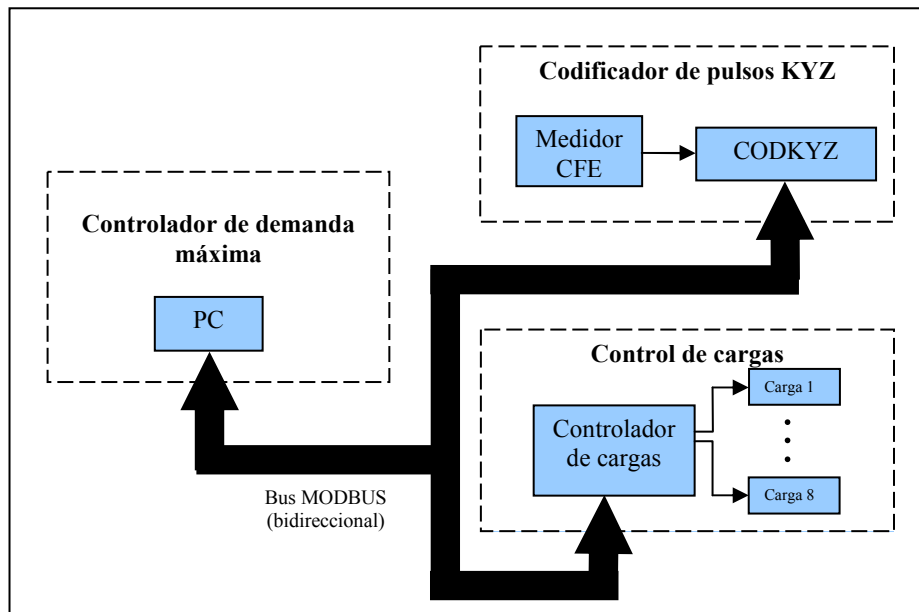


Figura i.2. Diagrama a bloques del controlador de demanda máxima distribuido.

1.- Codificador de pulsos KYZ. Se conecta al medidor de la CFE y lleva la cuenta del número de pulsos y su duración, enviando esta información al controlador de demanda para que haga las estimaciones del consumo de energía.

2.- Controlador de demanda máxima (CDM). Este bloque se basa en una computadora personal, que se conecta a los demás módulos por una interfaz de comunicaciones. Con base en la información del codificador de pulsos, estima la demanda máxima y cuando considera que se debe apagar alguna carga, le envía la orden al controlador de cargas.

3.- Control de cargas. Este módulo activa y desactiva sus salidas digitales en función de las órdenes que recibe del controlador de demanda máxima.

En un trabajo anterior, el ingeniero Aparicio Velázquez Enmanuel construyó el codificador de pulsos. Para continuar con esta línea de trabajo, en esta tesis se desarrolló el controlador de cargas. Con estos 2 módulos ya es posible realizar y probar el controlador de demanda máxima distribuido.

Objetivo general

Diseñar, construir y probar un módulo de control de cargas eléctricas bajo el protocolo de comunicaciones MODBUS, el cual será parte de un controlador de demanda máxima distribuido.

Objetivos específicos

- 8 salidas a relevador, capaces de manejar cargas de DC o AC de hasta 10 A.
- Puerto de comunicaciones RS-232 para recibir y transmitir tramas seriales de datos, con baudrate programable.
- Módulo convertidor de USB a TTL para configurar los parámetros del controlador de cargas. Este módulo será independiente, para que pueda ser usado en otros trabajos.
- Sistema con base en un microcontrolador de la firma ATMEL, programado completamente en lenguaje C para facilitar las actualizaciones del software.
- Implementación del protocolo de comunicaciones MODBUS, para la comunicación con el controlador de demanda.
- Software basado en Matlab® para mostrar la funcionalidad del módulo.
- El prototipo se dejará a nivel de circuito impreso.

Contenido del documento

Este documento de tesis consta de 5 capítulos en los cuales se hacen las descripciones que a continuación se mencionan:

En el capítulo 1 se describe la forma de cómo se factura la energía eléctrica a las industrias en México. Además, se explica la forma en la que está estructurado el protocolo de comunicaciones MODBUS y el puerto USB.

El capítulo 2 explica la forma en cómo fue diseñado y desarrollado el hardware del controlador de cargas serial. Se detallan las funciones MODBUS implementadas en el firmware del microcontrolador ATmega32, así como también se describe cómo fueron implementados los dos puertos de comunicación con los que cuenta el dispositivo.

En el capítulo 3 se explica detalladamente el desarrollo del software de pruebas, así como los requerimientos necesarios para hacerlo funcionar. También se aborda el uso del puerto serial en el lenguaje de programación MATLAB® utilizado para la realización del mismo.

El capítulo 4 muestra las pruebas realizadas al controlador de cargas. El cual se controla desde una computadora personal, con el software de control, así como los resultados finales obtenidos.

Finalmente, el capítulo 5 contiene las conclusiones a las que se llegaron en la realización de este trabajo de tesis.

Capítulo 1.

La energía eléctrica

Existen varias definiciones sobre la energía que proviene del griego "en" y "ergon" que significa acción o trabajo. Así, para un estudiante de nivel preparatoria la energía será *"cualquier cosa que pueda ser convertida en trabajo"* [2]; en tanto que los diccionarios señalan que *"la energía es la capacidad de un sistema para realizar un trabajo"* [3].

Pero en la práctica, más que la definición de la energía como un concepto, lo que interesa es saber qué beneficios proporciona. Por ejemplo, esta el calor para cocinar los alimentos o para calentar el agua para bañarse.

De todas las fuentes de energía, la más usada en el mundo moderno es la energía eléctrica. La cual se usa para iluminación, en motores eléctricos, etc. Ya que se puede transportar más fácilmente desde los centros de producción hasta el consumidor final en grandes cantidades.

1.1 Facturación de la energía eléctrica.

La energía eléctrica se mide en Watts-hora, siendo el Watt una unidad de potencia y equivale a un Joule por segundo. Pero para efectos prácticos, en la factura de consumo de energía eléctrica se cobra por la cantidad de kilowatts-hora (KWh) que se hayan consumido durante un periodo determinado. Un KWh es la energía que consume un foco de 100 watts encendido durante diez horas, ó 10 focos de 100 watts encendidos durante 1 hora.

En México, existen diferentes tarifas, ya sea por consumo de energía eléctrica de uso doméstico, servicio público, agrícola, acuícola e industrial; así mismo estas tarifas varían según la época del año en que se produzca la facturación y la región en la que se encuentre el consumidor [URL2].

Las tarifas eléctricas que tiene la Comisión Federal de Electricidad (CFE), son en función de la demanda de energía que el usuario consuma. De esta manera la tarifa de media tensión (O-M) es para una demanda menor a los 100 KW, y es la más usada por la industria de tamaño medio en el país. Los parámetros que cobra la CFE a sus usuarios son:

- La energía consumida
- Factor de potencia (FP)
- Derecho de alumbrado
- Demanda máxima

La energía consumida, es el total de energía eléctrica en KWh que la planta industrial consume durante todo el mes que se facturará.

En la tabla 1.1 se muestra el costo de facturación de la electricidad en las diferentes regiones en que se divide el país. Los precios mostrados son aplicables sólo para el mes de julio del año 2007 [URL2].

Tabla 1.1. Costo de los KWh por región para el 2007.

Región	Cargo por KWh de energía consumida (en pesos)
Baja California	0.907
Baja California Sur	1.223
Central	0.907
Noreste	0.847
Noroeste	0.840
Norte	0.847
Peninsular	0.865
Sur	0.865

El factor de potencia (FP) es una medida de la eficiencia con la que se aprovecha la energía eléctrica, ya que permite saber qué porción de la energía demandada corresponde a la potencia activa (que es la que genera trabajo) y qué porción es de potencia reactiva (que le quita capacidad al transformador). Con este parámetro se conoce qué cantidad de potencia se está ocupando realmente para producir trabajo efectivo, matemáticamente se define como se muestra en la ecuación (1) [4] [URL3].

$$FP = \cos \theta = \frac{\text{Voltaje}}{\text{Corriente}} \quad (1)$$

Para incentivar un uso eficiente de la energía eléctrica en nuestro país, la CFE bonifica al consumidor si éste tiene un FP mayor o igual a 90%, en caso contrario lo penaliza. Las ecuaciones (2) y (3) muestran como obtener el porcentaje de recargo o bonificación del recibo de consumo de energía.

$$\%_de_recargo = \frac{3}{5} * [(90 / FP) - 1] * 100 \quad (2)$$

$$\%_de_bonificación = \frac{1}{4} * [1 - (90 / FP)] * 100 \quad (3)$$

Si los valores resultantes de las operaciones llegaran a tener más de dos decimales, éstos se redondean para dejar uno solo, según sea o no menor que 5 el segundo decimal. La CFE aclara que en ningún caso se aplicarán porcentajes de recargo superiores al 120%, ni porcentajes de bonificación superiores al 2.5% [URL3].

El derecho de alumbrado público, es un porcentaje del total a pagar que se le cobra al consumidor en el mes de facturación. De esta forma la CFE solventa los gastos que se generan al suministrar energía eléctrica al alumbrado público del país.

La demanda máxima es una medida de la forma en la cual se ha realizado el consumo de energía durante todo el período de facturación. En la siguiente sección se da una explicación más detallada de este concepto.

1.2 Demanda máxima.

La demanda máxima refleja la forma en cómo una fábrica o planta distribuye su consumo de energía eléctrica durante el mes que se facturará. Para usuarios con tarifa O-M o superior, la CFE les instala medidores que mensualmente proporcionan el valor de la demanda máxima consumida por ellos. La CFE define a la demanda máxima como:

“La demanda de energía eléctrica en kilowatts durante cualquier intervalo de 15 minutos, en el cual el consumo de energía sea mayor que en cualquier otro intervalo de 15 minutos en el periodo de facturación” [URL2].

Durante todo el tiempo de facturación, se promedia la energía consumida en intervalos de 5 minutos. Este promedio entre el tiempo (los 5 minutos), proporcionan el valor promedio de la potencia (en kW) que se tuvo. Así sucesivamente se toman 3 intervalos de 5 minutos, y su promedio es la demanda máxima de ese momento. El promedio de mayor valor que se tenga, es lo que se toma como demanda máxima de todo el período de facturación.

Para ejemplificar la forma en la cual se obtiene la demanda máxima, en la figura 1.1 se aprecia un perfil de consumo de electricidad. Cada intervalo de tiempo corresponde a la demanda máxima que se tuvo. Como se puede ver, el intervalo 13 es el de mayor valor (de 250kW), por lo que si ningún otro lo superara, sería lo que se cobraría por este concepto.

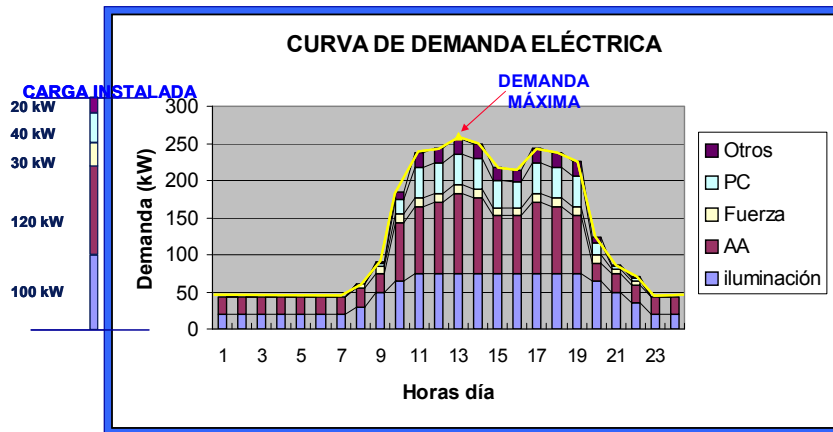


Figura 1.1. Curva típica de la demanda eléctrica en una planta.

La razón por la cual la CFE cobra este concepto, se debe a que los picos en la demanda de la planta requieren un esfuerzo extra en producción y transporte de la energía eléctrica. Lo que trae como consecuencia que aumenten los gastos en la producción de la misma.

Por lo tanto, el cobro por demanda máxima medida es necesario para solventar los gastos que representan el aumento de producción de electricidad, para quien proporciona el suministro de energía eléctrica.

En la tabla 1.2 se muestra el costo por kilowatt de demanda máxima medida para el mes de julio del año 2007, por región [URL2].

Tabla 1.2. Costo de la demanda máxima por región.

Región	Cargo por KW de demanda máxima medida (en pesos)
Baja California	107.62
Baja California Sur	119.21
Central	121.80
Noreste	111.98
Noroeste	114.31
Norte	112.42
Peninsular	125.69
Sur	121.80

La planta de la figura 1.1, si estuviera en el sur, tendría un cargo de $\$121.80 \text{ KW}^{-1} \times 250 \text{ KW} = \mathbf{\$30,450.0}$ (un cargo muy elevado). Por tal razón, en algunos casos es necesario mantener este valor limitado, para tener ahorros en la factura de la energía. El equipo que realiza esta función se conoce como controlador de demanda máxima y se describe en la siguiente sección.

1.2.1 Controlador de demanda máxima.

El controlador de demanda máxima es un aparato electrónico, que tiene la función de estar monitoreando la potencia eléctrica. Cuando la demanda de energía eléctrica se incrementa y tiende a rebasar el límite establecido, el controlador actúa automáticamente realizando la acción de desconexión de cargas que no son prioritarias en ese momento. Con esto da oportunidad a que otro tipo de cargas de mayor prioridad sigan en funcionamiento; lo cual permite al usuario mantener un menor costo de la energía eléctrica [URL4].

Para el control de la demanda, es importante asignar prioridades a las cargas. Aquellas que tienen poco o ningún impacto sobre la producción, pueden considerarse como iniciales para ponerse fuera de servicio temporalmente, las cargas que son básicas para un proceso deben de contemplarse hasta el último o inclusive no ser seleccionadas.

Existe una gran variedad de controladores de demanda disponibles, con diferentes grados de sofisticación, complejidad y costo. Algunos ejemplos de ellos son:

- El modelo MDC-01 de la compañía CARREL & CARREL LTD [URL5].



- El modelo Maximum Demand Monitor And Controller de la compañía INDUS ELECTRONICS INDIA (P) LTD [URL6].



- El modelo Maxcheck de la compañía United Machinery Corporation [URL7].



La unidad básica por lo general tiene los siguientes componentes principales:

- Transformadores de corriente, para proveer una señal de entrada desde el suministro de la CFE al controlador de demanda.
- Transductores, para convertir la señal de entrada en watts a una señal en milivolts para el panel lógico.
- Controlador de demanda, para monitorear los niveles de potencia a la entrada y actuar cuando éstos se aproximen al nivel pico de la demanda preestablecida.
- Panel relevador, para mandar señales de control a las cargas conectadas.

Los sistemas de control utilizados en los controladores de demanda se dividen en dos tipos que son:

- Controles manuales de encendido y apagado.
- Controles automáticos programables e inteligentes.

En los controles manuales de encendido y apagado, los propios operadores encienden y apagan los equipos usando sistemas de arranque/paro que posee cada equipo. El principal problema con este control, es que se tiene incertidumbre, ya que no se cuenta con un monitor para saber el valor real de la demanda máxima, y no se puede saber el momento más conveniente para apagar las cargas [URL8].

Los controles automáticos programables e inteligentes, presentan excelentes beneficios ya que en éstos se puede programar la conexión y desconexión de las cargas cuando sea necesario, sin necesidad de

preocuparse por los errores humanos. Estos sistemas se dividen en controladores programables y sistemas de control inteligente [URL9].

Los controladores programables son dispositivos de bajo y mediano costo que emplean microcontroladores. Los equipos típicos controlan 4, 8, 12, 16 ó muchas más cargas, y tienen la opción de poder expandir el número de cargas que controlan. Estos sistemas de control son fácilmente programables y son muy confiables.

Por otra parte se encuentran los sistemas de control inteligentes, los cuales usan sensores para medir el FP y la energía que se consume; esto con el fin de optimizar el consumo de la misma, al mismo tiempo que están en operación los equipos. Estos sistemas son modulares, por medio de los cuales se puede asignar prioridad a los procesos de producción y hacer que éstos sean eficientes, lo cual es posible monitoreando y controlando la demanda máxima de la planta.

1.3 Protocolo MODBUS.

Para controlar procesos industriales e integrar cada uno de los instrumentos de campo en una Red de Comunicación Industrial (RCI), es necesario tener un estándar para que puedan comunicarse entre ellos. La necesidad de establecer las reglas y procedimientos digitales mediante los cuales se pueden comunicar los elementos de una red, dio origen a los protocolos de comunicaciones tales como: PROFIBUS, MODBUS, HART, INTERBUS, etc. Ejemplo de algunos de los protocolos estandarizados para el uso en la industria.

El protocolo de comunicaciones MODBUS (Modicon Bus) fue puesto en el mercado en 1979 por la marca Modicon del grupo Schneider. Inicialmente se usó para establecer una red de comunicación multipunto entre sus controladores lógicos programables (PLC's), los cuales se utilizan para controlar procesos industriales distantes. Ahora el uso de MODBUS se ha extendido a TCP/IP sobre Ethernet, pero sobre todo se emplea en la industria como un bus de campo para monitorear equipos de medición [1].

A continuación se describen las características del protocolo MODBUS de línea serial bajo las especificaciones de transmisión en modo Unidad Terminal Remota (RTU), usada para comunicar dispositivos de medición con microcontroladores o microprocesadores.

1.3.1 Descripción del protocolo.

El protocolo MODBUS de línea serial define dos tipos de estaciones que se conectan bajo una sola configuración de red y donde el intercambio de

información es del modo maestro-esclavo. Los 2 tipos de estaciones o nodos de una red MODBUS son [1]:

- Maestro: es el encargado de controlar las transacciones de información. Solo debe existir un nodo maestro por red MODBUS y es el único que puede hacer solicitudes de información.
- Esclavo: funciona bajo las órdenes de la estación maestra. Pueden existir más de un nodo esclavo (hasta 32 en un bus RS-485), las tramas generadas por éste son llamadas respuestas.

Los nodos antes mencionados se conectan bajo una configuración de bus serial compartido, esto significa que un solo maestro y varios esclavos se conectan en una sola línea serial al mismo tiempo. La línea serial puede ser de 2 ó 4 hilos.

Existen dos modos mediante los cuales el maestro envía mensajes MODBUS a un esclavo: el modo dirigido y el modo de difusión. Para el primero, el maestro sólo se comunica con un solo esclavo, figura 1.2. En el modo de difusión el maestro les envía un mensaje a todos los esclavos y ellos ejecutan la instrucción sin responder.

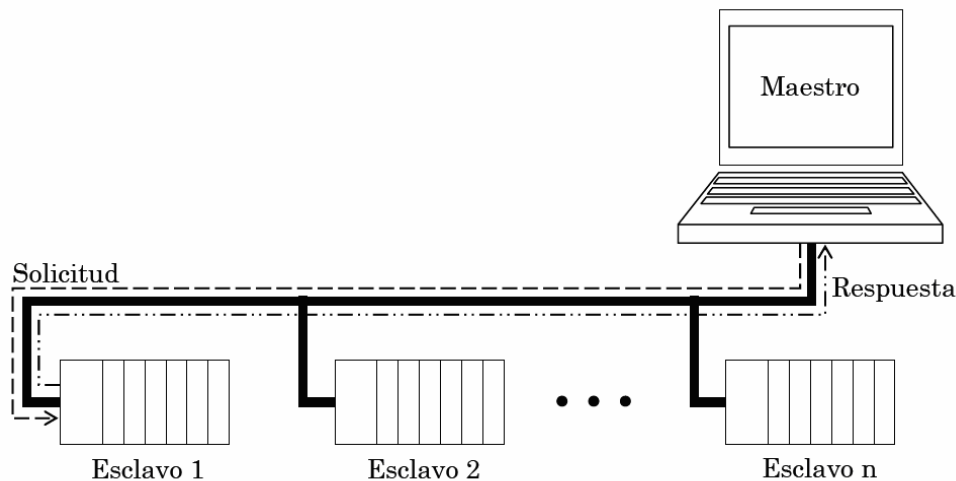


Figura 1.2. Representación del modo dirigido de los mensajes MODBUS.

En la figura 1.3 se muestra la estructura general de una trama MODBUS RTU, la cual puede ser una solicitud del Maestro en modo dirigido o de difusión, o bien, una trama de respuesta del esclavo.

Dirección del esclavo	Código de función	Subfunción o Datos	CRC-16
1 byte	1 byte	Variable	2 bytes

Figura 1.3. Estructura de un mensaje MODBUS.

Las tramas están formadas de 4 partes que son: la dirección del esclavo, la función, la subfunción o los datos y el código de error de redundancia cíclica (CRC-16). A continuación se explica cada una de ellas [5].

- Con la dirección, el maestro coloca la dirección del dispositivo esclavo con el que desea comunicarse.
- Con el código de función le indica la acción que debe de realizar, como son leer registros, entradas analógicas, digitales, etc.
- En el campo de datos se le indica al esclavo la función en particular a realizar, o éste le envía la información solicitada.
- El campo de CRC es un código de 16 bits con el cual se comprueba la integridad del mensaje recibido.

Con las funciones el maestro le indica el tipo de acción que desea que ejecute el esclavo. Existe una gran variedad de ellas, para leer o escribir variables de una palabra o de un bit. En general las más representativas para manipulación de bits se muestran en la tabla 1.3.

Tabla 1.3. Funciones más importantes de MODBUS.

Función	Descripción
0x10	Escritura de múltiples registros mantenidos. Utilizada para configuración del dispositivo, baudrate, paridad, bits de datos y número de dispositivo.
0x08	Diagnóstico del dispositivo. Sirve para llevar el registro de las funciones o comandos que ha ejecutado el dispositivo esclavo.
0x0B	Lectura del contador de eventos de comunicación. Devuelve el contador que registra cuántos comandos MODBUS se han detectado en el puerto de comunicaciones, aún cuando el comando no sea dirigido al dispositivo esclavo.
0x0F	Escribir múltiples valores de bobinas. Comando que indica al dispositivo qué número de carga(s) habrá de encender o apagar.
0x2B	Identificación del dispositivo. Función con la cual se le ordena al dispositivo que se identifique ante el dispositivo maestro.
0x01	Lectura de múltiples valores de bobinas. Devolverá el estado actual de la(s) carga(s), si se encuentra(n) encendida(s) o apagada(s).

1.3.2 Implementación de MODBUS en una RCI de línea serial.

Para que el protocolo MODBUS de línea serial pueda ser usado en una RCI, se debe cumplir con el estándar de comunicaciones EIA/TIA-485. Ese mismo estándar es conocido como RS-485 y permite conectar sistemas punto-a-punto o punto-a-multipunto [5].

Las velocidades de transición requeridas son 9600 y 19200 bits por segundo (BPS), aunque se pueden implementar otras como 1200, 2400, 4800, etc. Para la conexión de los cables se usan terminales con tornillos o conectores RJ40 o DB9.

1.3.2.1 MODBUS RS-485.

En el MODBUS RS-485, se pueden construir redes de comunicación de 2 hilos ó 4 hilos. En ambos casos, las señales se transmiten en forma diferencial, por tal razón la distancia máxima a la cual puede estar un nodo es de 1219.2 mts, según las especificaciones del protocolo. Esta distancia puede variar de acuerdo a la cantidad de dispositivos conectados, el calibre del alambre a utilizar, si existen repetidores, etc.

La principal diferencia entre el modo de 4 y 2 hilos, es que el primero puede ser configurado en modo full duplex, y el segundo solo en half duplex. En la figura 1.4 se muestra el diagrama esquemático de la forma en la cual se conecta el modo de 2 hilos, y en la tabla 1.4 el significado de cada una de las señales.

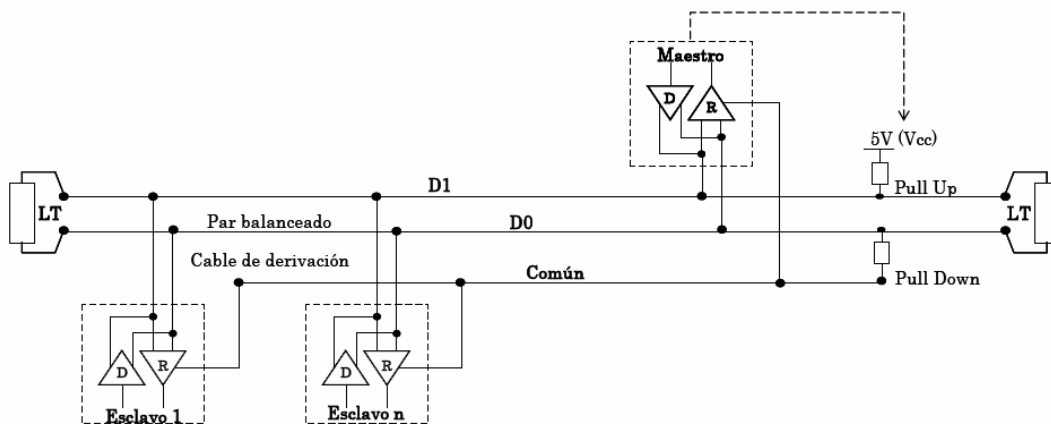


Figura 1.4. Estructura de la interconexión de dispositivos utilizando la configuración de 2 hilos.

1.3.2.2 MODBUS RS-232.

Es implementado con solo dos dispositivos, un maestro y un esclavo. Se utiliza el estándar de comunicaciones EIA/TIA-232 como interfaz eléctrica,

mejor conocido como RS-232. Una computadora normalmente es el dispositivo maestro, y el esclavo es un sistema de tarea específica con base en un microcontrolador o microprocesador. En la figura 1.5 se muestra esta configuración RS-232.

Tabla 1.4. Definición de las líneas de la configuración de 2 hilos.

Líneas requeridas	Tipo	Obligatoria	Nombre	Descripción
D1	E/S	Si	B/B'	Terminal del Voltaje V1 (transceptor), donde $(V1 - V0) > 200\text{mV}$ para un 1 binario o estado inactivo de la línea
D0	E/S	Si	A/A'	Terminal del Voltaje V0 (transceptor), donde $(V0 - V1) < 200\text{mV}$ para un 0 binario o estado activo de la línea
Común	REF	Si	C/C'	Señal común de referencia

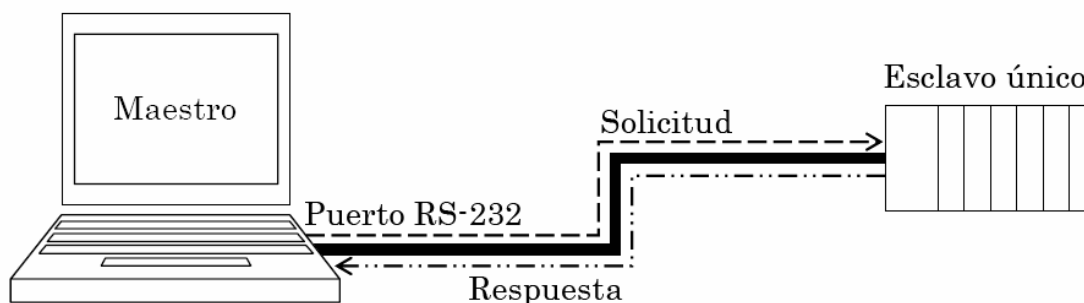


Figura 1.5. Configuración punto a punto MODBUS RS-232.

La conexión es punto a punto y la longitud del cable debe ser menor de 20m. Este estándar solo especifica las características mecánicas y eléctricas, mientras que la parte de funcionamiento se deja a cargo del usuario. El conector más usado es el DB9, y en la tabla 1.5 se muestra el tipo de señal y la función que desempeña en el estándar según la asignación de pines en el conector DB-9, figura 1.6 [6].

Tabla 1.5. Señales eléctricas del estándar RS-232.

Pin	Abreviatura	Nombre de la señal	Tipo de señal
1	CD	Carrier detect	Control
2	RD	Received data	Dato
3	TD	Transmitted data	Dato

4	DTR	Data Terminal ready	Control
5	GND	Signal ground	Referencia (tierra)
6	DSR	Data set ready	Control
7	RTS	Request to send	Control
8	CTS	Clear to send	Control
9	RI	Ring indicator	Control

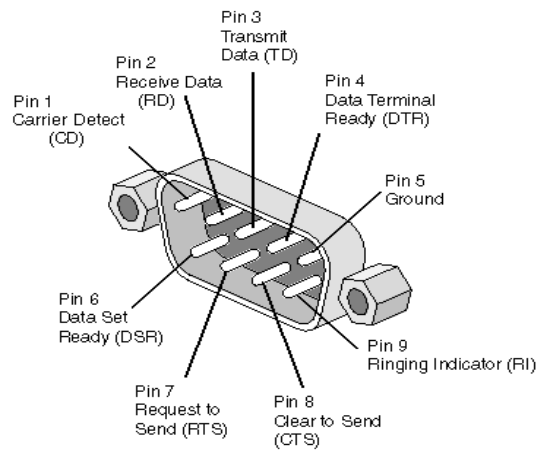


Figura 1.6. Conector DB-9 usado en el estandar RS-232.

1.4 Interfaz USB.

El protocolo USB (*Universal Serial Bus*) fue diseñado para tener conectividad con la computadora, en forma rápida, bidireccional y asíncrona. Su desarrollo fue impulsado por las compañías: Intel, Compaq Computer Corporation, Hewlett-Packard Company, Royal Philips Electronics, Lucent Technologies Inc., Microsoft corporation, NEC corporation, etc. Por lo cual es uno de los buses con más uso que se tiene en la actualidad. Las principales razones que motivaron su desarrollo fueron [7]:

- Conexión de la PC (Computadora Personal) con aparatos telefónicos, este motivo fue rápidamente desplazado por otro tipo de protocolos.
- Facilidad en su uso (Plug and Play), todo el peso del manejo de dispositivos con USB debería caer en el sistema operativo de la PC.
- Expansión de puertos, facilidad para conectar dispositivos USB que funcionen como módulos, los cuales permitan conectar más dispositivos USB.

En las siguientes secciones se dará una descripción general del bus USB, con el fin de entender la forma en la cual funciona, en caso de desear hacer

un desarrollo más extenso de este protocolo se aconseja visitar la página oficial del mismo donde se explica más detalladamente [URL10].

1.4.1 Descripción.

El protocolo USB soporta tres tipos de velocidades, que son alta velocidad (High speed) de 480 Mbits/s, velocidad completa (Full speed) de 12 Mbits/s y baja velocidad (Low speed) de hasta 1.5 Mbits/s; las cuales dependen del dispositivo que se conectará al bus USB. Estas velocidades serán constantes en cualquier intercambio de información entre el controlador (host) y el periférico. Esto con el fin de satisfacer los diferentes tipos de aplicaciones. La forma en la cual se transfiere la información entre los diferentes dispositivos y el host se puede dar de las siguientes 4 formas [7]:

- Transferencia de control, usada para mandar comandos y ver el estado de la transmisión. Es realizada por el HUB raíz.
- Transferencia de interrupción, usada por los dispositivos para requerir que el host los atienda.
- Transferencia asíncrona, usada para transmitir tramas de audio y video, ya que no se usa un reloj para sincronizar al emisor y receptor.
- Transferencia de volcado, usada para transmitir datos en cantidades no muy grandes y que no se necesiten para aplicaciones de tiempo real.

La forma en la cual se transfiere la información entre los diferentes dispositivos y el controlador es del tipo maestro-esclavo, ya que el host será el que inicie cualquier transacción de información e indique cuando está listo para enviar o recibir datos.

La arquitectura del USB permite habilitar la creación de nuevos dispositivos, con el fin de aumentar la capacidad de la PC. También es un método de conexión de bajo costo, cuando la transferencia de datos es menor a 450 Mbits/s. Da soporte para aplicaciones de tiempo real, el cual puede ser usado para la transmisión de datos de voz, audio y video.

El arreglo de conexiones en el bus USB es de tipo estrella como se muestra en la figura 1.7. Como se observa, al hub principal se le llama root hub, y en la red pueden existir un máximo de 5 hubs incluido el root hub. Los periféricos se conectan a cada hub, permitiéndose en una red hasta 127 periféricos (entre hubs y host).

Las conexiones son punto a punto entre el root hub (o host) y el hub, o entre el hub y un periférico. Para identificar a cada uno de los dispositivos, el host les asigna una dirección diferente cuando su presencia ha sido detectada en el bus USB. La distancia máxima de los cables es de 5 metros, lo que representa su mayor desventaja comparada con la longitud

utilizada para los estándares RS-232 y RS-485, aunque su velocidad de transmisión es mucho mayor que la de estos.

El host es una PC que contiene 2 componentes: el controlador de host y el root hub. Los dos, trabajando juntos, habilitan la comunicación entre el sistema operativo y el dispositivo o periférico conectado al BUS.

El controlador de host crea las tramas de información que se transmiten a los periféricos. Así mismo, traslada las tramas recibidas provenientes de un dispositivo a un formato que sea entendible para los componentes del sistema operativo.

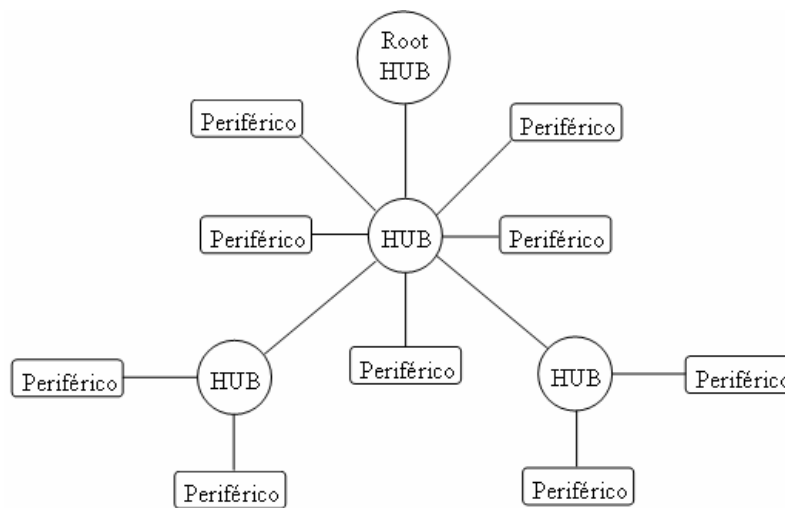


Figura 1.7. Topología del bus USB.

El root hub proporciona uno o más conectores para que puedan ser usados por los dispositivos o periféricos, además de que transporta los datos entre el controlador del host y los dispositivos en forma bidireccional. Entre el hub controlador y raíz detectan si un dispositivo se ha conectado o desconectado del bus.

Los periféricos son cualquiera de los dispositivos conectados en el bus, a excepción del root hub. Dentro de esta categoría está el hub que provee al sistema la capacidad de expansión con más puertos USB. Los demás dispositivos, son los que proveen al sistema de conectividad (memorias portátiles, joysticks, cámaras digitales, escaners, etc.).

Los dispositivos USB deberán de tener la capacidad para dar soporte al estándar USB, bajo las siguientes condiciones:

- Interpretación del protocolo USB 1.1 y/o 2.0.

- Interpretar operaciones estándar básicas, como configuración y reset.
- Tener la capacidad de proveer la información necesaria para que el host lo pueda reconocer como un dispositivo USB.

Todos los dispositivos en el BUS tienen dirección única, independiente de la interfaz que se trate, o que tenga un driver diferente de los demás dispositivos conectados en el BUS. La dirección es asignada por el root hub al momento que el dispositivo se reconoce como un periférico USB.

También existen los dispositivos compuestos, los cuales son dispositivos multifunción; es decir, un solo contenedor pero dos o más interfaces independientes. Esos dispositivos tienen solamente una dirección en el BUS, pero diferente manejador en el host.

Para que la información pueda fluir del dispositivo al host y viceversa, el protocolo define que las terminaciones de los cables transmisores deberán contar con 2 tipos de conectores:

- Conector tipo "A", este conector es siempre usado en el puerto USB de la PC.
- Conector tipo "B", este conector es el usado en el puerto USB de los dispositivos o hubs.

En la figura 1.8, se muestran los tipos de conectores usados en el protocolo USB. En ambos casos cada cable posee 4 terminales.

De las cuatro terminales que posee el cable USB, 2 son para alimentación (cuando el dispositivo la requiera) y 2 son para la transmisión de los datos. Cuando los dispositivos son alimentados por el bus, se tiene una limitación de corriente de 500 mA. En la tabla 1.6 se describe cada una de las terminales, su color y función, con respecto a los conectores de la figura 1.8 [8].

Tabla 1.6. Asignación de los conectores del USB.

Número de pin	Color del cable	Función
1	Rojo	VBUS (5 volts)
2	Negro	D-
3	Verde	D+
4	Blanco	Tierra

Los bits en el USB van en formato par diferencial, es decir, utiliza dos líneas para transmitir los datos, y la diferencia de voltaje entre las dos corresponde a un 1 ó un 0 según sea el caso. Los datos son codificados por medio del algoritmo NRZI (*Non Return to Zero, Invertid on ones*) [7].

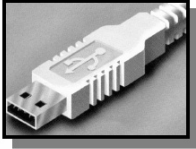
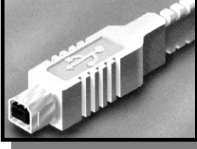
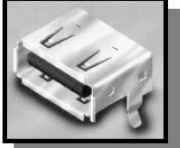
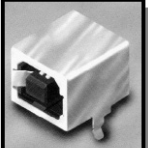
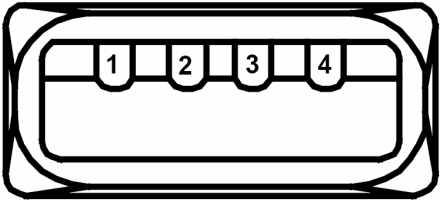
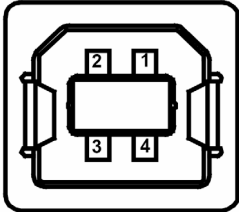
Conectores tipo "A" (orientados para ser usados por el host system)	Conectores tipo "B" (orientados para ser usados por el dispositivo USB)
 <p>Macho tipo "A". (Proveniente del USB periférico)</p>	 <p>Macho tipo "B". (Se conecta directamente al puerto del dispositivo USB)</p>
 <p>Hembra tipo "A". (Se conecta directamente al Bus USB de la PC)</p>	 <p>Hembra tipo "B". (Se conecta directamente al Bus USB del dispositivo)</p>
 <p>Numeración de los pines en el conector tipo "A"</p>	 <p>Numeración de los pines en el conector tipo "B"</p>

Figura 1.8. Conectores usados en el protocolo USB.

Para fines prácticos, el receptor tomará como un "1" lógico cuando D+ sea de un valor de 200 mV más grande que D-, y un "0" lógico cuando el valor de D+ sea 200 mV más chico que D-.

1.4.2 Protocolo USB.

El protocolo USB, a diferencia del estándar RS-232 y de otras interfaces seriales, especifica la forma en la cual se transfiere la información, haciendo uso de paquetes para transportarla. Los 3 tipos de paquetes que maneja son:

- Token packet, cabecera que indica el tipo de acción que se ha de realizar.
- Data packet (opcional), contiene los datos que serán leídos o transmitidos.
- Status packet, sirve para saber el estado de la transmisión que se ha realizado.

El host será el único que pueda iniciar cualquier transacción, esto lo realiza cuando un dispositivo nuevo se ha conectado al bus USB.

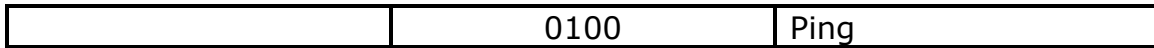
Los bits de cada byte que se transmite, se envían del menos significativo al más significativo. Para entender la forma en la cual se envía la información, ésta es codificada en tramas, las cuales están formadas por los siguientes campos:

- **Sync**, este campo corresponde a una secuencia de bits que son usados para sincronizar el reloj del transmisor con el del receptor. Su longitud es de 8 bits para *full speed* ó 32 bits para *high speed*.
- **PID**, este campo también conocido como "Packet ID", es usado para identificar qué tipo de paquete será enviado o recibido. En la tabla 1.7 se muestran los valores que puede tener.
- **ADDR**, con este campo se especifica la dirección del dispositivo al que va dirigido el paquete de información.
- **ENDP**, campo en el cual se especifica el número de endpoint (memoria física del dispositivo USB donde se guardan los datos a enviar o recibir) a usar.
- **CRC**, es el algoritmo para verificar los errores de transmisión, el CRC del token packet (paquete que informa si el host está listo para enviar a recibir datos) es de 5 bits y el del data packet (paquete de datos que han sido enviados o recibidos) de 16 bits.
- **EOP**, indica el final de la trama.

En la tabla 1.7 se especifican los 4 tipos de paquetes que son enviados por el bus.

Tabla 1.7. Valores de PID por tipo de paquete.

Paquete	Valor de PID	Identificador de paquete
Token	0001	OUT token
	1001	IN token
	0101	SOF token
	1101	SETUP token
Data	0011	DATA 0
	1011	DATA 1
	0111	DATA 2
	1111	MDATA
Handshake	0010	ACK handshake
	1010	NAK handshake
	1110	STALL handshake
	0110	NYET handshake
Special	1100	PREamble
	1100	ERR
	1000	Split



El token packet, es usado para informar al dispositivo USB o periférico que el host enviará datos, o que está listo para recibirlos. Este tipo de paquete puede ser de tres tipos:

- **In**, donde se informa al dispositivo USB que el host está en espera de recibir datos en el endpoint.
- **Out**, informa al dispositivo USB que el host enviará información a un endpoint.
- **Setup**, usado para transferencias de control.

La figura 1.9 muestra el formato de los 6 campos que conforma al token packet.



Figura 1.9. Formato del token packet.

Los paquetes de datos se llaman data packet. En la figura 1.10 se muestra el formato de cómo se codifica la trama. Hay dos tipos de data packet (Data0 y Data1) y el tipo de data packet a ser usado se selecciona en el campo PID. Son capaces de transmitir como máximo 1024 bytes de datos cada uno, donde la longitud del campo de datos depende de la velocidad a la que están conectados el host y el dispositivo, el dato se envía byte por byte.



Figura 1.10. Trama del data packet.

Los paquetes de estatus llamados status packet o handshake packet, son usados por el controlador para obtener información de las transacciones que han ocurrido. En la figura 1.11 se muestra el formato de la trama, que sólo está compuesta por tres campos.

Los tipos de operaciones que se pueden solicitar con el status packet son tres, las cuales están definidas en el campo PID. Estas son:

- **ACK**, indica que el paquete ha sido recibido satisfactoriamente.
- **NAK**, reporta que el dispositivo no puede enviar o recibir datos. También es usado para informar al host que no hay dato que mandar.
- **STALL**, sirve para solicitar la intervención del host.



Figura 1.11. Trama del Status packet.

De forma general en este capítulo se dio un panorama de lo necesario para desarrollar el controlador de cargas (CDECA), incluido el puerto USB. Si se requiere mayor información, se recomienda consultar las referencias.

Capítulo 2.

Diseño del Hardware

Para cumplir con los objetivos de la tesis, es necesario diseñar un sistema que reciba una trama MODBUS, la decodifique y con base en ella haga la apertura o cierre de los relevadores que controlan el encendido o apagado de las cargas eléctricas. Para realizar todo esto se necesita usar un microcontrolador, el cual debe ser programado para que cumpla con todos estos requerimientos (firmware).

En el presente capítulo se muestra una descripción del hardware del sistema, explicando primero la forma en que fue diseñado y fueron seleccionados los circuitos que componen al hardware. Posteriormente, el diseño del firmware. Es decir, primero se exponen las características de los circuitos y componentes electrónicos empleados para el desarrollo del controlador de cargas de corriente alterna (CDECA).

En el diseño del firmware, se exponen las funciones más importantes del programa que se empotrará en el microcontrolador ATmega32, el cual se programó con el lenguaje C. Para ello se usó el compilador WinAVR para microcontroladores de 8 bits de la firma ATMEL; por ser de libre distribución.

2.1 Descripción general del sistema.

El CDECA se basa en la trama MODBUS que recibe, activa o desactiva sus salidas a relevador. En la figura 2.1 se muestra su diagrama a bloques, observándose que se divide en 3 módulos principales:

- Módulo de comunicaciones.
- Módulo de control.
- Módulo de carga.

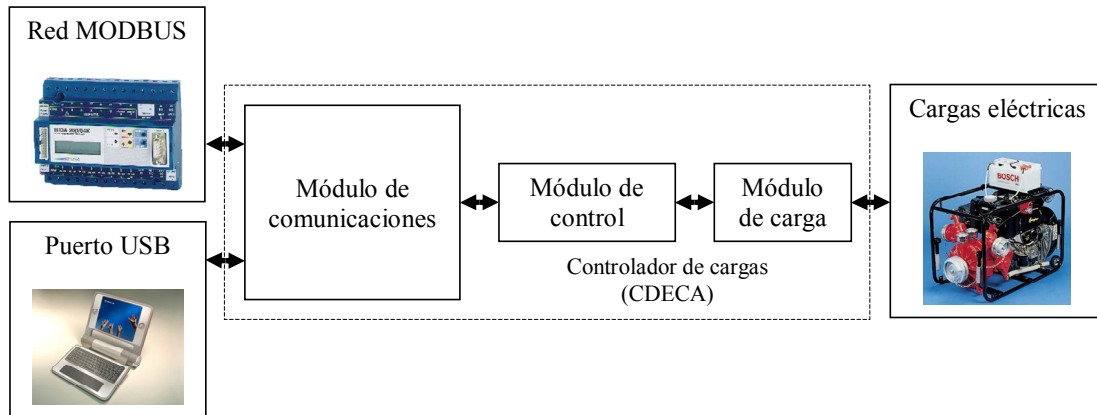


Figura 2.1. Diagrama a bloques del CDECA.

El módulo de comunicaciones está formado por los puertos de comunicaciones seriales del sistema, los cuales envían y reciben las tramas de datos en niveles de voltaje TTL, desde la USART (*Universal Synchronous/Asynchronous Receiver Transmitter*) del microcontrolador. A su vez, convierte estos niveles a los especificados para el uso de los protocolos EIA/TIA-485 (RS-485) y USB, usados para comunicaciones de línea serial.

El módulo de control se encarga de la codificación, decodificación y transmisión de los datos, acción que se realiza siguiendo las reglas especificadas por el protocolo MODBUS. Así como el envío de la orden de disparo hacia el módulo de cargas. Este módulo es básicamente el microcontrolador ATmega32, el cual posee un programa que lo configura para que efectúe las tareas que se desean. Para este propósito existe un compilador tanto de ensamblador como de C para programarlo; se decidió usar este último por ser de más fácil uso que el ensamblador.

El módulo de cargas está constituido por los relevadores con los cuales se encienden y apagan las cargas eléctricas. Las salidas del microcontrolador no pueden conmutar a los relevadores ya que estos consumen más de 20 miliampers por pin de salida, por lo cual se tiene una etapa de amplificación de corriente con base en transistores BJT. En las siguientes secciones se explicará la forma en que fue diseñado cada uno de los 3 módulos.

2.1.1 Módulo de comunicaciones.

Como se mencionó anteriormente, esta etapa se encarga de acondicionar las señales eléctricas para que se pueda establecer el intercambio de datos entre el CDECA, como dispositivo esclavo de la red, y un dispositivo maestro que se encarga de consultarlo. La comunicación alamburada se lleva a cabo por medio de los puertos de comunicaciones seriales USB y RS-485 de 2 ó 4 hilos.

En la figura 2.2 se muestra un diagrama a bloques con las partes que conforman al módulo de comunicaciones. Como se puede apreciar, éste se integra por 3 partes que son el puerto RS-485, el puerto USB y el selector del puerto (interruptor). Esta última parte es necesaria, ya que el microcontrolador solo tiene una salida de transmisión (TX) y recepción de datos (RX), y se tienen 2 interfaces de salida. En las siguientes secciones se explicarán cada una de estas partes.

2.1.1.1 Puerto USB.

El puerto USB transfiere información serial desde y hacia una computadora a una velocidad mucho mayor que el RS-232 o el RS-485. Sin embargo su manejo es mucho más complejo que los 2 anteriores, ya que involucra el manejo de señales de sincronización, manejo de errores (mecanismos de recuperación de datos ante algún fallo), inserción dinámica de dispositivos, soporte para la identificación de dispositivos defectuosos, etc. También su estándar especifica el tipo de conector a usar y la forma en que se transmiten las tramas.

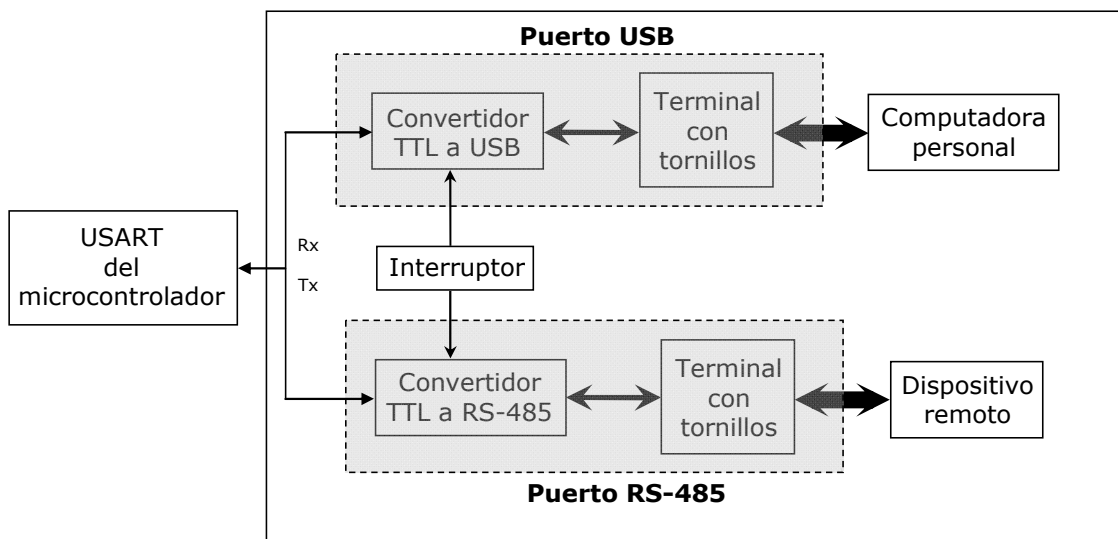


Figura 2.2. Diagrama del módulo de comunicaciones.

En la PC se requiere desarrollar un programa especial, el cual servirá de intermediario entre un dispositivo de hardware y el sistema operativo, con la finalidad de extraer al máximo las funciones del dispositivo para el cual ha sido diseñado. A este programa se le conoce como driver o controlador.

Para cada dispositivo de hardware que se conecta a la computadora, es necesario un driver en específico. Normalmente los fabricantes de los dispositivos desarrollan el controlador para sus productos, y el usuario selecciona el adecuado para el hardware que tiene. Por ejemplo, no se puede utilizar el mismo driver para controlar una impresora HP 3320 y una

HP 840C; además, por lo regular funciona con un sistema operativo en específico, es decir, para el mismo dispositivo es necesario un driver para Windows XP® y otro muy distinto para Windows Me®.

En general, desarrollar un controlador de USB es bastante complejo, ya que involucra el desarrollar software teniendo un conocimiento profundo del sistema operativo y el hardware. Sin embargo, en el mercado existen varios fabricantes de circuitos integrados que han diseñado manejadores de USB, los cuales proporcionan el driver para que sea usado su dispositivo. De tal manera que al conectar el circuito al puerto USB de la PC, a su salida se tienen los datos en forma serial o paralela con niveles TTL. Haciendo que el uso del puerto USB, por parte del usuario, sea completamente transparente; ya que la computadora reconoce al dispositivo como un puerto de comunicaciones seriales del tipo COM.

Buscando en Internet, se encontró un proveedor en México que suministra un controlador de USB modelo FT232BM de la firma FTDI (Future Technology Devices Intl.) [URL11]. Una característica importante de este dispositivo, es que se alimenta con 5 V, los cuales pueden ser proporcionados por una fuente de CD o los puede tomar directamente del bus USB.

En la figura 2.3 se muestra el diagrama a bloques del CI manejador de USB, las flechas corresponden a las señales de entrada, salida y de control. Con estas últimas se configura las diferentes opciones del CI, como: consumo de energía (pin PWRCTL), el tipo de protocolo que se usará como salida (pines TXDEN, SLEEP# y PWREN#), que pueden ser RS-232, RS-422 y RS-485. Para mayor información se recomienda leer la hoja de datos del CI [9].

Para su funcionamiento, el FT232BM requiere de un reloj a 6MHz. Por tal motivo internamente posee un oscilador a esta frecuencia, la cual se fija con un cristal que se conecta a los pines XTOUT y XTIN. El controlador, internamente requiere de señales de reloj de 12MHz y 48MHz, las cuales las consigue con un módulo multiplicador de frecuencias tomando como multiplicadores los números 2 y 8.

La salida del CI FT232BM es el módulo marcado como UART, que posee todas las terminales de un manejador de comunicaciones seriales convencional. Sin embargo, el dispositivo se conecta con la computadora a una velocidad de 12 Mbits/s (modo full speed), mientras que a la salida, la velocidad varía desde 1.2kbits/s hasta 120kbits/s. Para compensar estas diferencias, el CI tiene implementados buffers para almacenar los bytes que se reciben o envían, para de esta forma compensar la diferencia que existe en la velocidad de estos dos protocolos.

El bloque "EEPROM interface", es el encargado de controlar una memoria del tipo EEPROM serial externa (93C56 ó 93C66). En esta memoria se guarda la información de identificación del dispositivo USB, que es definida por el usuario. La memoria se graba utilizando como programador el mismo CI FT232BM y un software proporcionado por el fabricante.

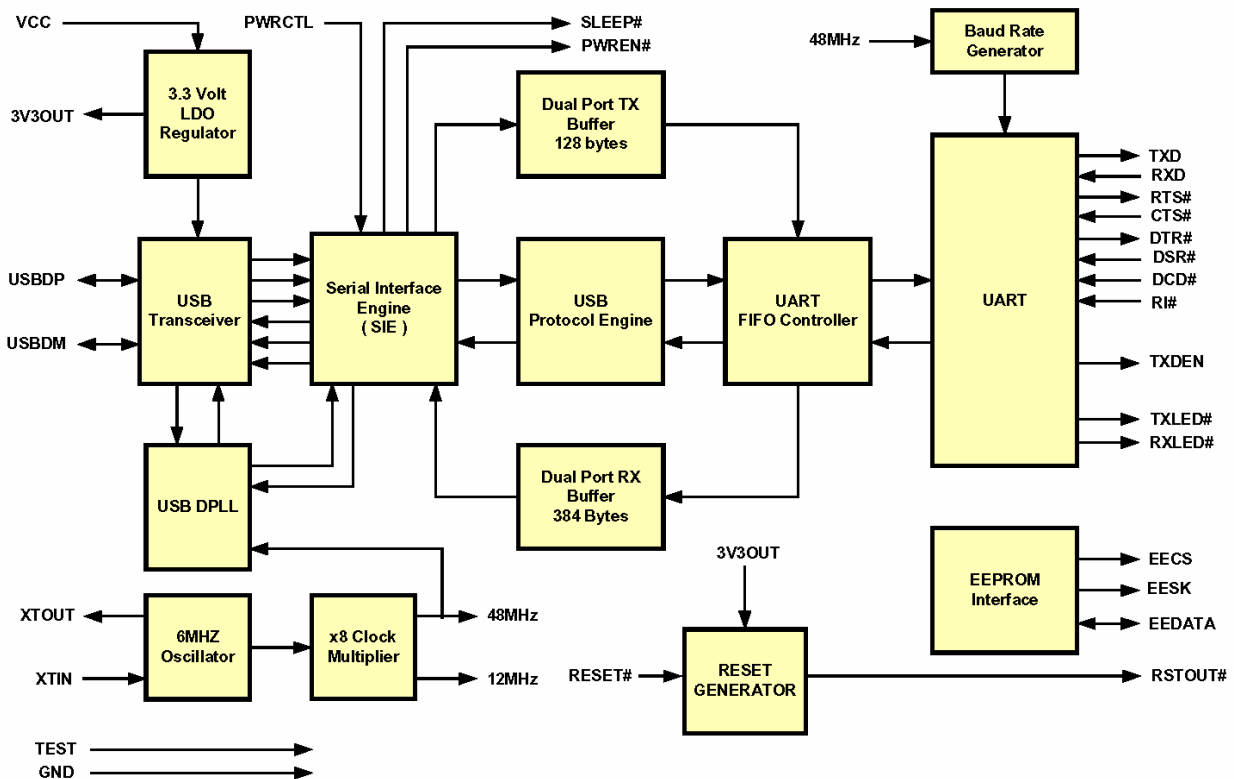


Figura 2.3. Diagrama a bloques del CI FT232BM.

Como es muy poca la cantidad de información que el CDECA intercambiará con el maestro, únicamente requiere de las señales de Recepción (RXD) y Transmisión (TXD) de datos. Además, el modo de comunicación del CDECA es semiduplex, es decir, puede transmitir y recibir datos, pero sólo una acción a la vez.

En la figura 2.4 se muestra el diagrama esquemático de toda la circuitería necesaria para que funcione adecuadamente el FT232BM como convertidor USB a niveles TTL. Los elementos que requiere son capacitores, resistores, el cristal y la memoria serial 93C66. La alimentación es proporcionada por una fuente de voltaje de 5 volts, aunque puede funcionar con la alimentación que proporciona el puerto USB de la PC. Las señales del USB se conectan al conector J1 y las salidas seriales al conector P1.

2.1.1.2 Puerto RS-485.

Para adaptar los niveles TTL del microcontrolador y generar las señales del puerto RS-485 de 2 ó 4 hilos siguiendo la norma EIA/TIA-485, se usa el CI MAX489 de la firma MAXIM. La figura 2.5 muestra el diagrama de conexiones usado por este circuito, y en la tabla 2.1 se muestra la descripción de las señales eléctricas.

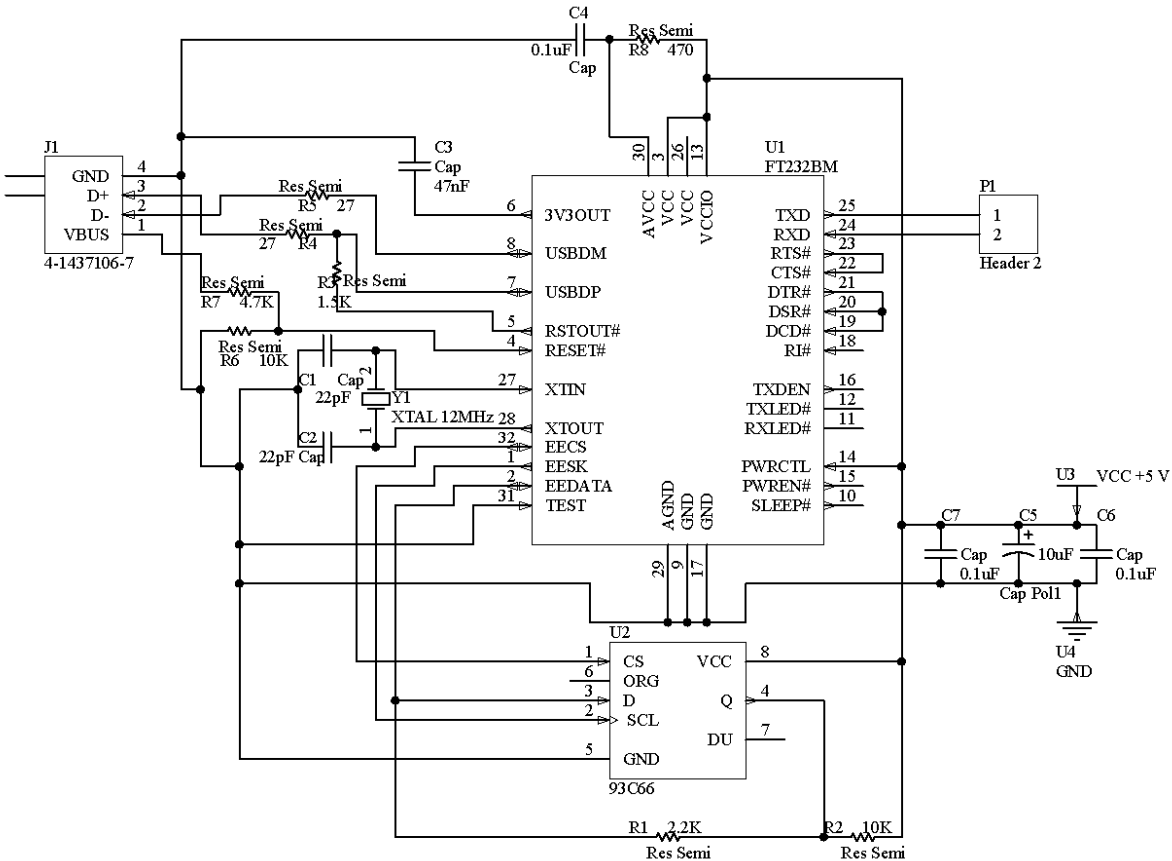


Figura 2.4. Diagrama esquemático del convertidor de nivel USB a TTL.

Este CI proporciona una comunicación de tipo full-duplex, en la cual se pueden transmitir y recibir datos al mismo tiempo a una velocidad de hasta 250kbts/s, permite conectar hasta 32 tranceptores de su mismo tipo en el BUS. Por la configuración en modo diferencial de sus salidas, es menos susceptible a ser afectado por el ruido de tipo EMI (*ElectroMagnetic Interference*) causado principalmente por campos eléctricos externos.

En configuración de 4 hilos, el MAX489 usa las señales A-B para recepción y Y-Z para transmisión. Las señales de habilitación RE y DE hacen que las salidas se comporten en tercer estado, lo cual es muy útil cuando se

trabaja en modo half-duplex, que es la que se utiliza en comunicaciones tipo maestro-esclavo de MODBUS. En este modo, estas señales se interconectan para ser manipuladas por una sola señal de control, esto para habilitar la recepción o transmisión, solo una a la vez por ser excluyentes; es decir, mientras una se activa en alto, la otra se activa en bajo.

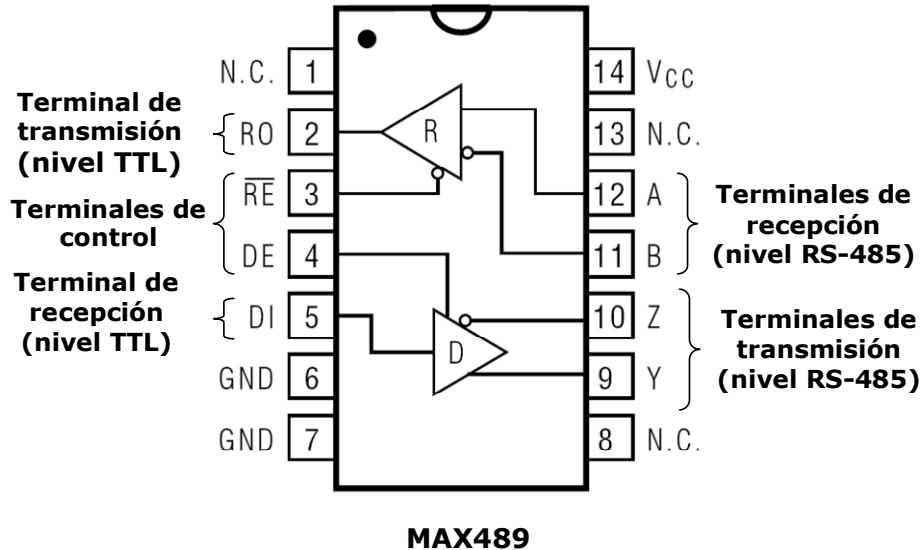


Figura 2.5. Diagrama de conexiones usado en el CI MAX489.

Tabla 2.1. Descripción de las señales eléctricas del CI MAX489.

MAX489			
Número de pin	Nombre del pin	Tipo	Función
1	N.C.	Ninguno	Ninguna
2	RO	Salida	Salida del receptor: si $A > B$ por 200mV, RO se mantiene en alto; si $A < B$ por 200mV, RO se pone en bajo.
3	\overline{RE}	Entrada	Habilitador del pin RO. RO es habilitado cuando \overline{RE} se encuentra en bajo; RO se pone en alta impedancia cuando \overline{RE} se encuentra en alto.
4	DE	Entrada	Habilitador de los pines Y y Z, estos se habilitan poniendo en alto el pin DE, cuando este se encuentre en bajo, los pines Y y Z estarán en alta impedancia.
5	DI	Entrada	Entrada del manejador: un bajo en DI, fuerza a la salida Y a ponerse en bajo y a la salida Z en alto; un alto en DI, fuerza a la salida Z a ponerse en bajo y a la salida Y en alto.

6	GND	Entrada	Tierra
7			
8	N.C.	Ninguno	Ninguna
9	Y	Salida	Salida no invertida
10	Z	Salida	Salida invertida
11	B	Entrada	Entrada no invertida
12	A	Entrada	Entrada invertida
13	N.C.	Ninguno	Ninguna
14	V _{CC}	Entrada	Alimentación

La conexión con 2 hilos se realiza al aprovechar la alta impedancia que posee el CI en las salidas del transmisor y receptor. Esta característica se obtiene interconectando las señales A-Y y B-Z para construir el puerto RS-485 de 2 hilos.

Por lo tanto, cuando el CDECA espera datos, habilita el receptor del CI MAX489 activando la señal RE (en bajo) y desactivando DE. Cuando se transmite, se activa la señal DE (en alto) y desactivando la señal RE.

2.1.2 Módulo de control.

Este módulo se encarga de recibir las tramas seriales MODBUS que vienen del módulo de comunicaciones, decodificarlas y si son válidas, ejecutar la tarea solicitada. También envía las señales de disparo que activa a los 8 relevadores del módulo de cargas. Dentro del manejo del protocolo MODBUS, se especifica que se mantenga la información, que no se borre cuando el dispositivo no se encuentre con alimentación.

Para cumplir con los requerimientos planteados, el circuito integrado que puede hacer este trabajo es un microcontrolador. De todos los microcontroladores que hay en el mercado, se decidió usar uno de la firma ATMEL, por contar con experiencia en el desarrollo de aplicaciones con los dispositivos de esta firma.

Para determinar el mejor dispositivo a usar en el presente proyecto, se tomaron en cuenta las siguientes características:

- Memoria de programa (memoria flash) grande, de al menos 16 kbytes, ya que se va a desarrollar todo su firmware en C, y ocupa una mayor cantidad de memoria que si se programará en ensamblador.
- Memoria EEPROM para guardar información que no se pierda en caso de que se le retire la alimentación.
- Puerto de comunicaciones seriales (USART).
- Al menos 10 pines de entradas salida.
- Tamaño pequeño.

La mayoría de microcontroladores de tamaño pequeño con encapsulado DIP tienen una memoria de programa de hasta 2 kbytes, lo cual no cumple con uno de los requerimientos. El microcontrolador ATmega32 tiene 32kBytes de memoria de programa, pero tiene 40 terminales, el cual es muy grande en encapsulado DIP. Por tal motivo se decidió usar uno de tipo TQFP de 44 pines como el que se muestra en la figura 2.6, el cual es de montaje superficial.



Figura 2.6. Encapsulado TQFP de 44 pines.

El microcontrolador seleccionado cuenta con las siguientes características utilizadas en este proyecto de tesis:

- 131 instrucciones en lenguaje ensamblador.
- 32 registros de propósito general, cada uno de 8 bits.
- La gran mayoría de instrucciones se realiza en solo un ciclo de reloj.
- 32 kbytes de memoria flash con una duración de 10000 ciclos de borrado/escritura.
- 1024 bytes de memoria EEPROM con una duración de 100000 ciclos de borrado/escritura.
- 2 kbytes de memoria SRAM.
- 2 timer/counters de 8 bits con preescalador separado.
- Watchdog timer programable.
- 4 puertos, cada uno con 8 líneas de entrada/salida.
- Voltaje de alimentación de 2.7 a 5.5 volts.

En la figura 2.7 se presenta el diagrama a bloques del microcontrolador ATmega32, en la cual se aprecia la complejidad del ensamble interno del mismo. En forma general está conformado por el núcleo (AVR CPU) utilizado en todos los microcontroladores de 8 bits de la firma ATMEL.

Para su funcionamiento es necesario proporcionarle una señal de reloj en los pines XTAL1 y XTAL2, aunque también posee la capacidad de omitir esa señal y funcionar por medio de un oscilador interno; el cual deberá ser configurado al momento de cargar el firmware en la memoria flash.

Generalmente la señal de reloj es proporcionada por un resonador de cristal, cuya frecuencia depende del proyecto que se va a desarrollar. Específicamente, para el CDECA se utilizó un cristal de 7.3728 MHz, con el cual se pueden alcanzar velocidades de transmisión/recepción de hasta 230.4 kbits/s con un 0% de error; lo cual es idóneo, ya que no se pasa de la máxima velocidad de transmisión/recepción del CI MAX489.

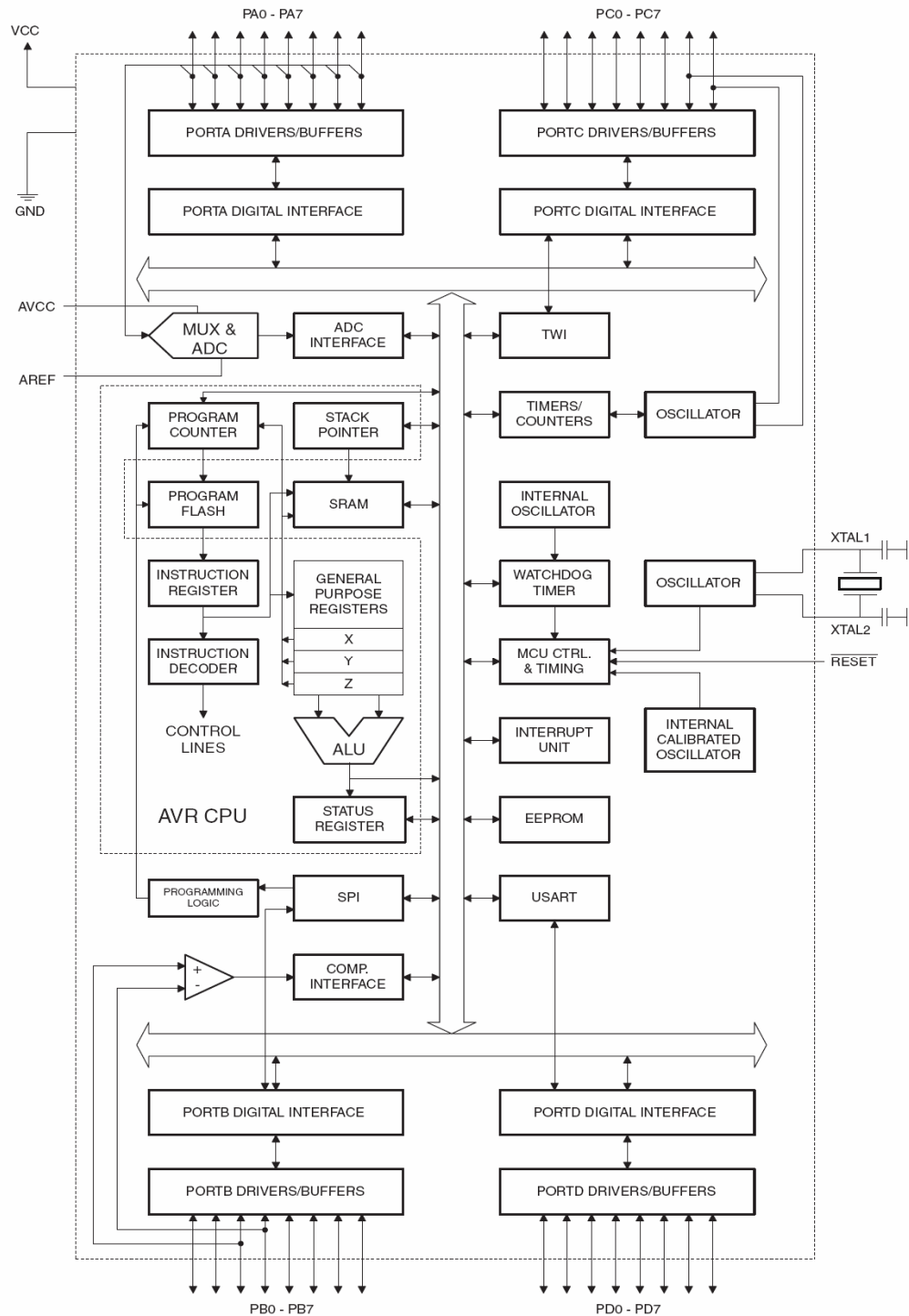


Figura 2.7. Diagrama a bloques del microcontrolador ATmega32.

Las 32 líneas de entrada/salida se agrupan en cuatro puertos, los cuales pueden funcionar como entrada o salida según sean programados. También, su uso puede cambiar en función de los recursos del microcontrolador, como son: convertidores analógicos digitales, puerto serial (USART), puerto SPI para programación de memoria flash y EEPROM, señales de interrupción y puerto para simulación "on chip" (JTAG). En esta aplicación solo se usa el USART y los pines de salida para controlar al módulo de cargas.

La característica más importante por la que se optó por este microcontrolador es la velocidad con que realiza las operaciones, además que todo el software de desarrollo es de libre distribución y fácil de usar.

2.1.3 Módulo de cargas.

Este módulo se encarga de controlar la activación y desactivación de las cargas, es decir, encender y apagar los relevadores de salida de acuerdo a las órdenes recibidas por el dispositivo maestro. Los relevadores pueden conmutar cargas de DC de hasta 24 volts o AC de hasta 120 volts.

Las salidas digitales del microcontrolador proporcionan voltajes con niveles TTL, con una corriente de salida máxima de 20 mA. Ya que los relevadores que se utilizarán se alimentan con un voltaje de 5 V y tienen un consumo de 72 mA, obliga a usar una etapa de potencia entre el microcontrolador y el disparo de los relevadores conformada por un transistor tipo NPN como se muestra en la figura 2.8.

Cuando la salida del microcontrolador esta en alto, el transistor esta en saturación y dispara al relevador. Cuando la salida esta en bajo, el transistor se pone en corte y se desactiva el relevador.

La resistencia de base sirve para polarizar positivamente el transistor, ya que la corriente que conducirá el colector será la máxima permitida por la resistencia interna de la bobina presente en el relevador de 69 Ω . Por tal razón se optó por usar el transistor BC547 tipo NPN que permite conducir 100 mA de corriente máxima en el colector, ya que la bobina del relevador tiene un consumo de 72 mA.

En la figura 2.9 se muestra el diagrama esquemático completo del CDECA serial, en el cual se aprecian todos los componentes como son los manejadores del puerto USB (CI FT232BM), el puerto RS-485 (CI MAX489), el módulo de cargas y el microcontrolador (ATmega32).

La selección del puerto de comunicaciones se hace con los puentes P1 y P2. Con los cuales se conectan los pines RXD (recepción) y TXD (transmisión)

del microcontrolador a las terminales RXD y TXD de los circuitos integrados que manejan el USB o el RS-485.

El conector P4, es un puerto por el cual se permitirá actualizar el firmware del CDECA. El puerto esta diseñado con la misma distribución de pines que usan los programadores comerciales de la firma ATMEL. Debido a que algunos de estos necesitan ser alimentados con 5 volts, se les suministra este voltaje en las terminales P5; si el programador a usar necesita ser alimentado, se colocar un jumper entre las terminales del conector P5.

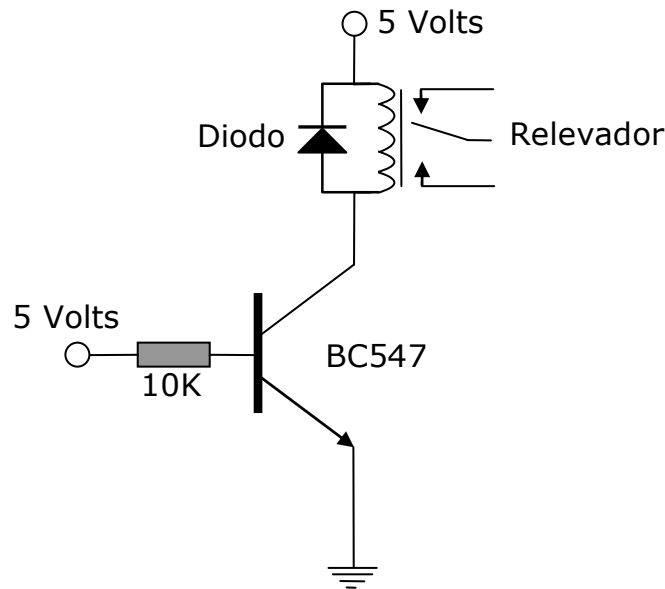


Figura 2.8. Etapa de potencia utilizada en el módulo de cargas.

Se recomienda leer las hojas de especificaciones del programador que se utilizará para evitar posibles daños al programador o al microcontrolador, ya que la configuración de pines puede variar a la que se implementó en el CDECA.

Para hacer la conexión que alimentará las cargas, se optó por el uso de terminales con tornillos (terminales de la P6 a la P13). Solamente es necesario conectar una de las dos líneas que vayan a servir de alimentación a la carga.

2.2 Implementación del protocolo MODBUS en el ATmega32.

El programa que controla al microcontrolador ATmega32, es el que se encarga de coordinar todas las tareas que éste ejecuta. Siendo las más importantes, la implementación del protocolo MODBUS y el control del módulo de control. El conjunto de tareas que desarrolla son:

- Recepción de la trama MODBUS y decodificación del mensaje.
- Actualización de las salidas en función de la petición realizada.
- Manejo de los contadores de eventos.
- Ejecución de la función solicitada.
- Construir la respuesta MODBUS en función de la petición realizada por el maestro.
- Construir la respuesta MODBUS de excepción en caso de que el dispositivo no soporte la función solicitada u ocurriera un error de lectura/escritura de un registro.

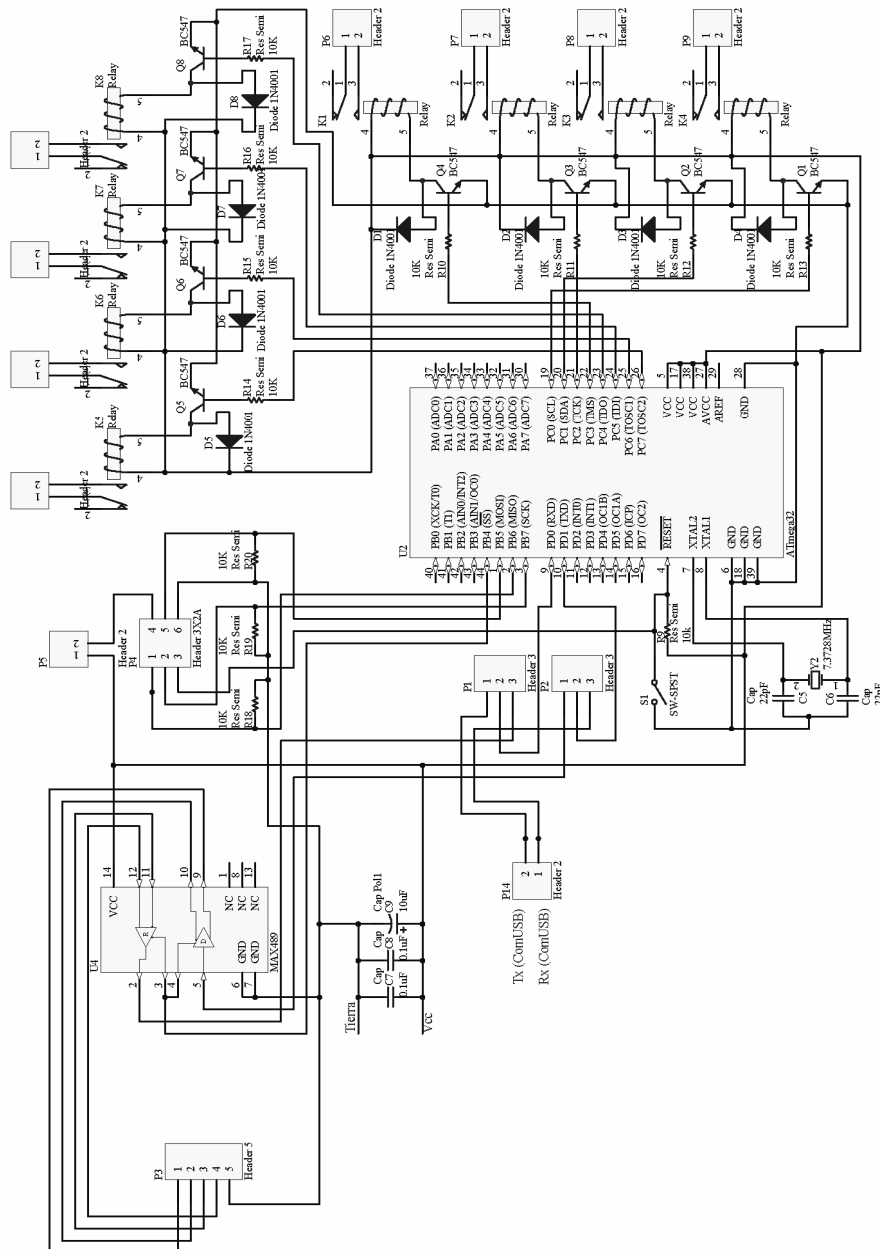


Figura 2.9. Diagrama esquemático completo del CDECA.

Para que las actualizaciones del firmware sean más fáciles de llevar a cabo, así como de entender el código del programa, se decidió usar el lenguaje de alto nivel C. El compilador que se usó es el WinAVR, en su versión 20070122 [URL12].

Para explicar la forma en que está estructurado el programa, se hará uso de diagramas de flujo y se explicarán las funciones más importantes. En la figura 2.10 se muestra el diagrama de flujo de la forma en que se ejecuta el programa.

La función `configurar_puertos`, se encarga de designar los puertos que servirán para controlar el encendido o apagado de las cargas, así como para controlar la señal que activa y desactiva el envío y recepción de datos en el puerto RS-485.

La función `configurar_USART` configura los registros que manipula la USART del microcontrolador, como la velocidad de recepción/transmisión y los bits de paro. Otras opciones que configura el programa, son la dirección del dispositivo en la red MODBUS. Todos estos datos están almacenados en la memoria EEPROM, por lo cual tiene incluida una rutina que se encarga de la lectura de esta memoria.

La función `configurar_temporizador`, es la que se encargará de configurar los parámetros necesarios para que el temporizador (Timer Counter 2 del microcontrolador) sirva de base para medir el tiempo de silencio necesario para que se pueda tomar como finalizada la recepción de una trama MODBUS, proveniente de un dispositivo maestro.

Dado que el tiempo de silencio es 3.5 veces el tiempo de transmisión de datos (incluyendo los bits de inicio de trama, paridad y paro), la función `configurar_temporizador` configura al temporizador 2 para que se ajuste a ese tiempo, en función de la velocidad de transmisión/recepción que tenga configurada el CDECA. Las velocidades que soporta son: 2.4, 4.8, 9.6, 19.2, 38.4, 57.6, 115.2, 230.4 en kbits/s.

El temporizador 2 se activa cada que se ha terminado de leer el dato presente en la USART, y se desactiva cada que se detecta un dato en el puerto serial o USART del microcontrolador; esto con el propósito de saber cuándo se cumplirá con la condición de tiempo de silencio. Una vez que ha transcurrido este tiempo, se procede a checar que la trama MODBUS recibida no se encuentre corrupta.

La función CRC16 revisará que la trama enviada por el Maestro y recibida por el esclavo no haya sido corrompida. Para ello calcula el CRC16 con los datos recibidos y la compara con la que viene en la trama. Si son iguales

toma el mensaje como válido, en caso contrario lo desecha e incrementa los contadores de eventos y genera el reporte de error. El algoritmo de cómo se calcula el CRC16 puede ser consultado en [1].

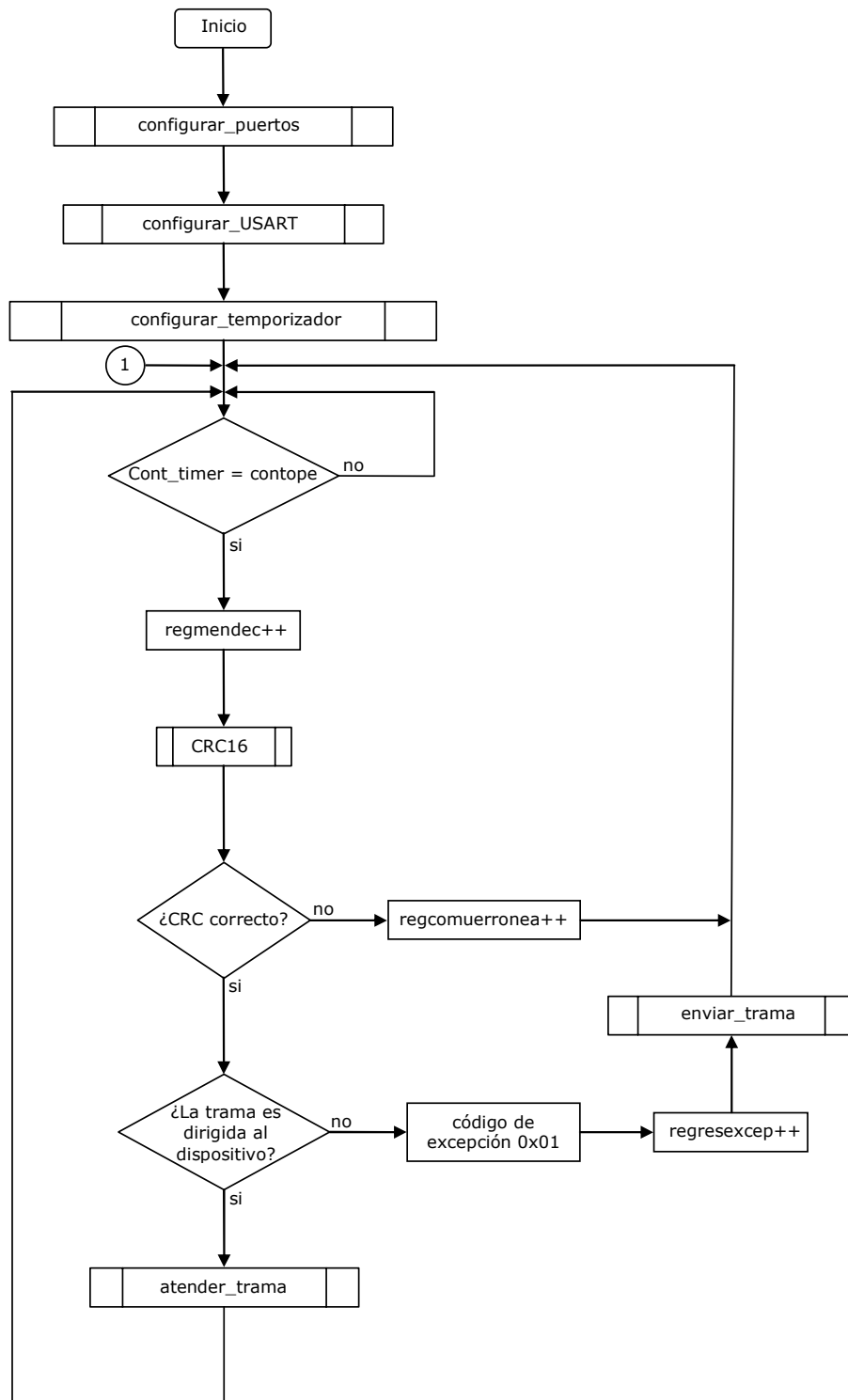


Figura 2.10. Diagrama de flujo del programa principal del microcontrolador

Cuando el CRC16 es válido, verifica que la dirección corresponda con la del dispositivo. En caso de que la dirección sea diferente, genera el código de excepción 0x01 e incrementa el contador de respuestas de excepción (la variable `regresexcep`), inmediatamente después se envía la trama de respuesta.

Si no hay ningún error, se ejecuta la función `atender_trama`. Esta función decodifica el resto de la trama MODBUS y ejecuta la petición que hace el maestro al esclavo, posteriormente genera la trama de respuesta, para indicar que la función se ha ejecutado correctamente. En caso de que se le pida una opción que no este implementada en el dispositivo, se envía una respuesta de excepción con el código de error correspondiente. Como es una de las funciones más importantes se explica con mayor detalle.

En la figura 2.11 se muestra el diagrama de flujo de la función `atender_trama`, las diferentes opciones que tiene son:

- Verifica si es de tipo difusión, si es así, incrementa el contador "contmendif", en caso contrario continúa con la ejecución de la función.
- La siguiente opción es verificar que la función pedida se encuentre implementada en el dispositivo, si es así se procede a la siguiente opción (verificar el número de registro). En caso contrario genera la respuesta de excepción 0x01, y aumenta el contador "regresexcep" y desecha la petición del maestro.
- Verifica que el número de registros a leer/escribir sea correcto, con ello se comprueba que el registro a utilizar por la función a ejecutar sea soportado por ésta. En caso de que no sea así, se genera una respuesta de excepción 0x02, aumenta el contador "regresexcep" y desecha la petición del maestro.
- Si todas las condiciones son cumplidas, se incrementa el contador "regcomexec" y la función solicitada por el dispositivo maestro es atendida por la función `ejecutar_función`. `Ejecutar_función` atiende la petición que le solicitan al usuario y crea la trama de respuesta, que se le envía al dispositivo maestro.

Una vez que se mostró de forma general cómo está estructurado el programa, en las siguientes secciones se explicarán los registros que se implementaron en el CDECA, las variables que se crearon en la memoria RAM, EEPROM y los códigos de excepción.

2.2.1 Uso de la memoria SRAM y EEPROM.

Los datos más importantes del esclavo son: la dirección del dispositivo y los datos de configuración de la USART (velocidad de transmisión, bits de datos, paridad y bits de paro). Estos datos se almacenan de manera permanente en la memoria EEPROM, con el fin de que no se pierdan cuando se desenergiza al dispositivo.

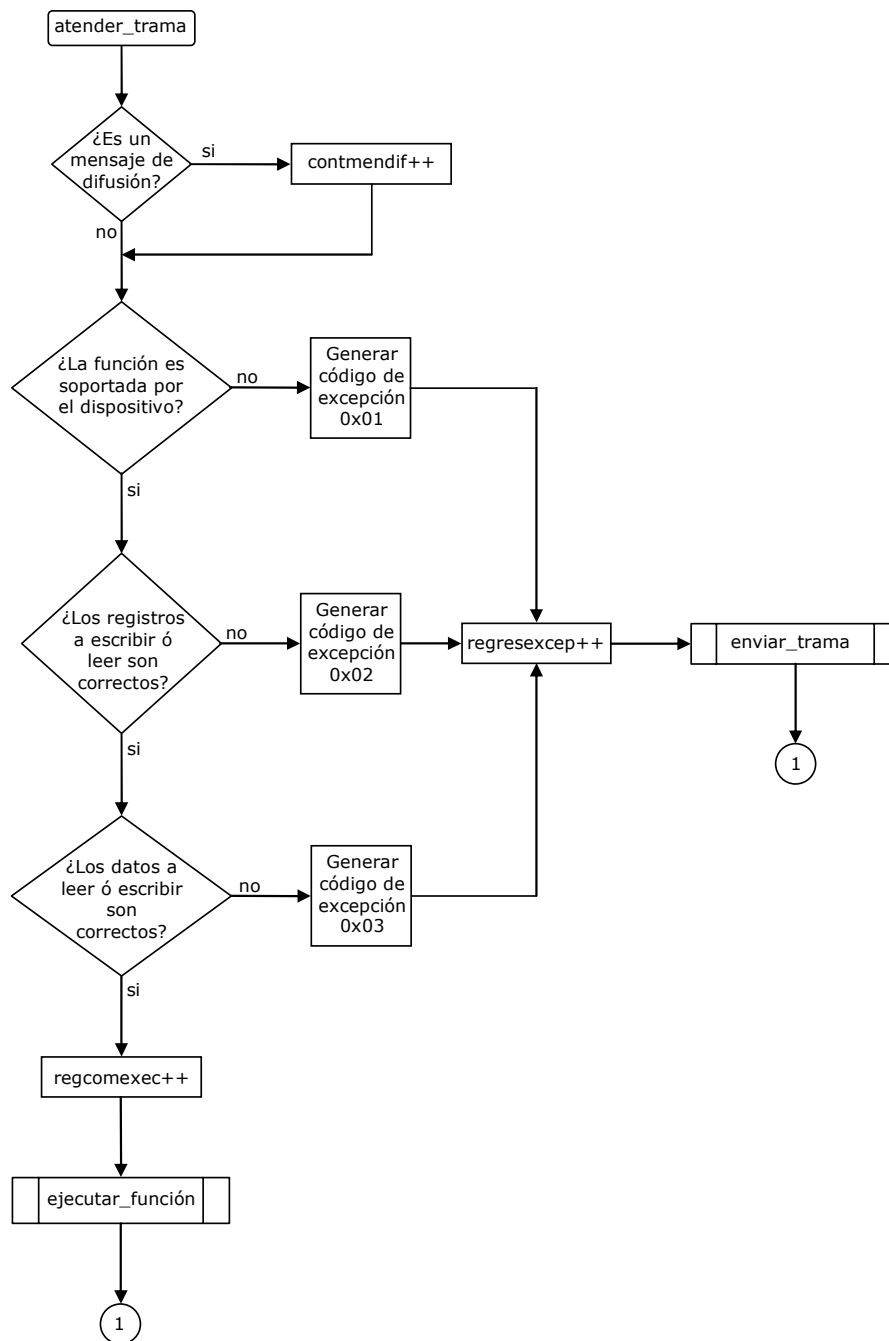


Figura 2.11. Diagrama de flujo de la función atender_trama.

La información del estado de las cargas se almacena en registros del tipo mantenidos. Éstos son guardados también en la memoria EEPROM. Cada registro mantenido es de 2 bytes, aunque el valor que almacena no lo requiera, debido a que el protocolo MODBUS especifica que todos los registros de datos (mantenidos o de entrada) deben tener esta longitud (16 bits).

Cada vez que el CDECA se inicializa, realiza una rutina de autoconfiguración, cargando los registros mantenidos almacenados en la EEPROM.

El microcontrolador ATmega32 posee 2 Kbytes de memoria SRAM, la cual es usada para el almacenamiento de información de tipo volátil o variable. Como en este trabajo se va a desarrollar el programa en C, este espacio de memoria es completamente controlado por el compilador. De tal manera que el uso de la memoria SRAM es asignada totalmente por el compilador, de tal manera que no es posible saber en qué parte de la memoria se almacena cada variable.

Se van a manejar 2 tipos de datos (variables) de 8 y 16 bits. Los primeros son para el almacenamiento de los datos provenientes del dispositivo maestro, es decir la trama de solicitud. Los segundos se destinan a guardar los eventos de comunicación MODBUS detectados por el CDECA. Las variables más importantes se enumeran en la tabla 2.2.

Los valores de las variables que se muestran en la tabla 2.2, serán los valores de los registros utilizados para crear la trama de respuesta MODBUS. El valor de la variable que se utilizará para la trama de respuesta, depende de la función que se haya pedido ejecutar por parte del dispositivo maestro.

Tabla 2.2. Descripción de las variables más importantes del programa.

Variable	Tipo	Función que realiza
trama03	uint8_t de tipo arreglo	En ella se guarda la velocidad y paridad a la que funciona el CDECA.
CCE	uint16_t	Registro en el cual se guarda la cantidad de mensajes dirigidos al dispositivo, exceptuando las peticiones de eco.
regcomuerronea	uint16_t	Sirve para llevar el conteo de los mensajes recibidos con error de redundancia cíclica.
regresexcep	uint16_t	Contiene el número de respuestas de excepción que ha generado el dispositivo.
regcomexec	uint16_t	Lleva la cantidad de funciones que han sido

		ejecutadas a petición del dispositivo maestro.
regmendec	uint16_t	En ella se guardan los mensajes que han sido detectados en el bus, aunque no sean dirigidos al dispositivo.
contmendif	uint16_t	Registro en el cual se guarda la cantidad de mensajes de difusión ejecutados por este dispositivo.
cadpeti	uint8_t de tipo arreglo	Variable tipo arreglo en la cual se guarda la trama de petición enviada por el dispositivo maestro.

La variable de memoria utilizada para recibir los mensajes de solicitud (cadpeti), se va llenando conforme se reciben los datos (bytes) que conforman la trama de solicitud del dispositivo maestro. El almacenamiento de los datos en la variable "cadpeti", termina una vez que ha transcurrido el tiempo de silencio; tres veces y medio un tiempo de un byte o carácter sin recibir ningún dato por parte del dispositivo maestro.

2.2.2 Memoria de recepción de tramas MODBUS.

Una trama MODBUS no debe exceder los 45 bytes de longitud, si esto ocurriera, la trama se descarta automáticamente, pues ninguno de los mensajes de solicitud destinados al CDECA debe de ser mayor. Esta longitud de trama es la longitud máxima para variables tipo arreglo que puede manejar el compilador WinAVR sin que introduzca datos basura al contenido de cualquier variable tipo arreglo.

Antes de atender el mensaje, el paquete se somete a la verificación del algoritmo CRC16 (verifica que no se encuentre corrupto) y se asegura que vaya dirigido al CDECA. Si se detecta un error se actualizan las variables "regcomuerronea", la cual lleva el conteo de las tramas corruptas recibidas por el dispositivo y "regmendec", que lleva el conteo de los mensajes detectados en el bus de comunicaciones. Mientras se verifican estas opciones, la trama MODBUS permanece en la variable de tipo arreglo "cadpeti".

Al presentarse un determinado evento de comunicación, se incrementa el valor de la variable correspondiente en 1. Estos registros o variables pueden ser consultados únicamente enviando un mensaje de solicitud con el código de función 0x08 y la subfunción relacionada al contador del evento de comunicación requerido.

2.2.3 Función, subfunción y direcciones de registros.

Con la información de los campos de función y subfunción se especifica la tarea que se quiere que ejecute el CDECA (esclavo). Cuando se solicita una opción que no tenga implementado se genera un código de excepción.

De todo el conjunto de funciones que especifica el protocolo MODBUS, 6 son las funciones que se decidieron implementar y 9 subfunciones. En la tabla 2.3 se enumeran las funciones y en la tabla 2.4 las subfunciones. En estas mismas tablas se explica que hacen.

Tabla 2.3. Funciones implementadas en el CDECA.

Función	Descripción
Lectura de los estados de las bobinas [0x01].	Devuelve el estado (encendido o apagado) en el que se encuentren los relevadores.
Escribir múltiples registros mantenidos [0x10].	Modifica el valor de los registros usados para la configuración de este dispositivo.
Diagnóstico del dispositivo [0x08].	Devuelve el estado en el que se encuentran los contadores de eventos de comunicación. Esta función esta compuesta por las subfunciones descritas en la tabla 4.
Devolver el contador de eventos de comunicación [0x0B].	Devuelve el valor de la variable "CCE".
Escribir múltiples valores de bobinas [0x0F].	Modifica el estado de los relevadores.
Identificación del dispositivo [0x2B].	Devuelve los caracteres con los cuales el dispositivo se identifica ante cualquier maestro cuando se le solicite.

Tabla 2.4. Subfunciones implementadas en el CDECA.

Subfunción	Descripción
Solicitud de eco [0x0000]	Devuelve la misma trama recibida.
Reinicio de comunicaciones [0x0001]	Pone a ceros los registros (variables) que llevan los eventos de comunicación y reinicia la USART del microcontrolador.
Activa el modo de solo escuchar [0x0004]	Hace que el dispositivo solamente ejecute la función

	solicitada, no envía respuesta.
Reinicio de contadores de comunicación [0x000A]	Pone a ceros los registros (variables) que llevan los eventos de comunicación.
Devuelve el contador "regmendec" [0x000B]	Devuelve la cantidad de mensajes detectados en el BUS.
Devuelve el contador "CRC" [0x000C]	Devuelve la cantidad de errores de redundancia cíclica que se han detectado.
Devuelve el contador "regresexcep" [0x000D]	Devuelve la cantidad de respuestas de excepción a enviado el dispositivo.
Devuelve el contador "regcomexec" [0x000E]	Devuelve la cantidad de funciones que han sido ejecutadas en el dispositivo.
Devuelve el contador "contmendif" [0x000F]	Devuelve la cantidad de mensajes de difusión que se han recibido.

Desde el punto de vista del protocolo, a cada variable a leer se le especifica una dirección, a la cual se le llama dirección lógica. Internamente se procesa con una dirección diferente. Los registros mantenidos van de la dirección 40001 a la 40003, y los valores de las bobinas, van de las 30001 a la 30008. El dígito 4 ó 3, se utilizan para distinguir cada tipo de registro. En las tablas 2.5 y 2.6 se muestran las direcciones lógicas y variables en las cuales se guardan el valor de los registros.

Tabla 2.5. Direcciones lógicas de los registros mantenidos en el CDECA.

Dirección lógica	Descripción	Nombre de la variable
40001	Velocidad que manejará el dispositivo al transmitir datos.	trama03[1]
40002	Paridad que manejará el dispositivo al transmitir datos.	trama03[2]
40003	Dirección que usará el dispositivo en la red MODBUS, a la que esta conectado.	trama03[3]

Tabla 2.6. Direcciones lógicas para el estado de los relevadores en CDECA.

Dirección lógica	Descripción	Nombre de la variable
30001		

30002	El estado que tendrán los 8 relevadores (encendido "1" o apagado "0").	estadoboninas
30003		
30004		
30005		
30006		
30007		
30008		

2.2.4 Validación de los valores de registros, códigos de excepción y ejecución de la función.

Para que un registro mantenido pueda ser modificado, el mensaje de solicitud de escritura debe portar datos afines al registro que desea escribir. Un ejemplo de ello es si se quiere modificar el registro mantenido "trama[1]" (variable en lenguaje C que guarda el valor del baudrate, al que deberá transmitir y recibir este dispositivo), el campo de datos debe portar sólo valores de velocidad que el dispositivo soporta.

La validación de los valores se realiza examinando la información del campo de datos, se identifican los registros que van a ser modificados y se comprueba que los valores proporcionados por el dispositivo maestro se encuentren dentro de los rangos de valores aceptados para cada registro mantenido. Si la identificación reporta algún dato no válido, se responde con un mensaje de excepción de datos con código de error. En la tabla 2.7 se muestran los 3 códigos que puede reportar el CDECA al producirse cualquiera de los errores de validación.

Tabla 2.7. Códigos de excepción soportados por el CDECA.

Número de código	Descripción del código
0x01	Se ha solicitado al dispositivo ejecutar una función que no soporta.
0x02	Se ha intentado leer o escribir una dirección ilegal.
0x03	El valor que se desea escribir en un registro no es válido.

Por último si la trama MODBUS no fue rechazada y no se generó una respuesta de excepción, se procede a ejecutar la función que el mensaje indica y se construye la trama que contiene la información solicitada o la confirmación de que ya se ha efectuado la operación requerida.

Capítulo 3. Desarrollo del software de pruebas

Para verificar que se hayan cumplido los objetivos marcados en el inicio del desarrollo del CDECA, se creó un programa para computadora con base en el lenguaje de programación MATLAB®. Este lenguaje normalmente es usado para realizar cálculos de índole científica, ya que cuenta con funciones para obtener una solución más rápida de problemas matemáticos. Sin embargo, las últimas versiones permiten tener accesos a puertos, e incluso controlar tarjetas de adquisición de datos y cámaras de video, con lo cual se hacen sistemas de control con base en la PC.

Por tal motivo, en el presente trabajo se decidió utilizarlo para explorar su rendimiento cuando se hace uso del puerto serie, además de poseer una herramienta con la cual se pueden crear interfaces gráficas de forma muy sencilla, utilizada para el desarrollo del panel frontal de la aplicación. Se hizo una investigación de las tesis que se han realizado en la UTM y no se encontró alguna que lo usara en una aplicación similar.

En el presente capítulo se explicará el proceso seguido durante el desarrollo del Software de Pruebas del Controlador de Cargas (SPCC), además de explicar detalladamente la manera en que se puede acceder a los puertos seriales con los que cuenta una PC haciendo uso del lenguaje de programación MATLAB®.

3.1 Matlab como software de desarrollo de instrumentación.

MATLAB® es la abreviatura de Matrix Laboratory (laboratorio de matrices). Es un programa de análisis numérico creado por The MathWorks en 1984.

Está disponible para las plataformas Unix, Windows y Mac OS X [URL13, URL14].

Es un lenguaje de alto desempeño computacional para cálculos técnicos, orientado a matrices y vectores. Por tanto desde el principio hay que pensar que todo lo que se pretenda hacer con él, será mucho más rápido y efectivo si se piensa en términos de matrices y vectores.

Integra cálculos, visualización y programación en un ambiente sencillo, donde los problemas y las soluciones se expresan en una notación matemática familiar y sencilla.

Se pueden ampliar sus capacidades con accesorios especiales llamados "Toolboxes", que son paquetes para simular de forma más rápida: Sistemas con procesado digital de señales, adquisición de datos, economía, inteligencia artificial, lógica difusa, etc. También cuenta con otras herramientas como Simulink, que sirve para simular sistemas, basándose en fórmulas matemáticas [URL15].

MATLAB[®] puede utilizarse para casi cualquier caso o problema que necesite de cálculos. Tales como el desarrollo de algoritmos, modelado, simulación y desarrollo de prototipos, análisis de datos, exploración y visualización, para graficar ensayos de tipo científico o simplemente una tarea universitaria y para desarrollo de aplicaciones, las cuales incluyen el desarrollo de interfaces gráficas para el usuario (GUI, Graphical User Interface).

Usa un lenguaje de programación creado en 1970 para proporcionar un sencillo acceso al software de matrices LINPACK y EISPACK sin tener que usar Fortran [URL16, URL17].

Es un software muy usado en universidades, centros de investigación y por ingenieros. En los últimos años ha incluido muchas más capacidades, como la de programar directamente procesadores digitales de señales, crear código VHDL y otras [URL13].

3.2 Requerimientos del programa.

El programa de pruebas debe tener la capacidad de mostrar cómo funciona el CDECA. Por tal motivo, debe ser capaz de tener implementadas las funciones MODBUS más importantes, así como cambiar el estado de las cargas y mostrar los estados actuales. En forma puntual, los requerimientos del programa son:

- Comunicación a través del puerto RS-232 y USB.
- Menú de configuración de las opciones del puerto serial como Baud Rate, número de bits, dirección del esclavo.

- Menú de configuración donde se seleccionen las opciones del protocolo: lectura de múltiples valores de bobinas, diagnóstico del dispositivo, lectura del contador de eventos de comunicación, escribir múltiples valores de bobinas, escritura de múltiples registros mantenidos e identificación del dispositivo.
- Recepción de las tramas MODBUS y despliegue del estado de las cargas.
- Envío de la trama MODBUS, con la configuración del estado de las cargas.
- Ventana de errores.
- Detección de los puertos seriales disponibles en la PC.
- Deberá poder ejecutarse en máquinas que no dispongan del lenguaje MATLAB[®] previamente instalado.

3.3 Descripción general del SPCC.

Para que el SPCC fuera entendible y fácil de usar para el usuario se decidió crear una interfaz gráfica, la cual fue diseñada con la utilería "GUIDE", proporcionada en el paquete de instalación del lenguaje MATLAB[®]. Esta utilería tiene la ventaja de poder acortar el tiempo de programación que se necesita para crear las interfases gráficas. Sin embargo puede llegar a generar demasiado código, lo cual lo hace difícil de entender y modificar. Para mayor información del desarrollo de interfaces consultar [URL18].

Para cumplir con los requerimientos del programa, se dividió según la estructura jerárquica mostrada en la figura 3.1. De forma funcional, el programa se divide en tres módulos que son: Selección de opción, Procesamiento y Reporte.

El módulo de Selección de opción, es la parte en la cual se definen los parámetros de comunicaciones del esclavo con el que se va a establecer comunicación, la velocidad de transmisión y la paridad, así como también la función MODBUS a ejecutar. Este módulo se divide en 3 bloques que son: Función MODBUS a solicitar, Dirección del dispositivo y Configuración del dispositivo. Estas opciones se ejecutan hasta que se de la orden de enviar, en ese momento se llama al módulo de procesamiento.

El módulo de procesamiento elabora y envía la trama MODBUS por el puerto serial, con los datos configurados en el módulo Selección de opción. Este módulo se compone de 3 bloques:

- Crear trama. Elabora la trama MODBUS.
- Generar CRC. Con base en la trama a enviar calcula el código de error (CRC) correspondiente.
- Enviar trama. Es el encargado de enviar los datos por el puerto serie.

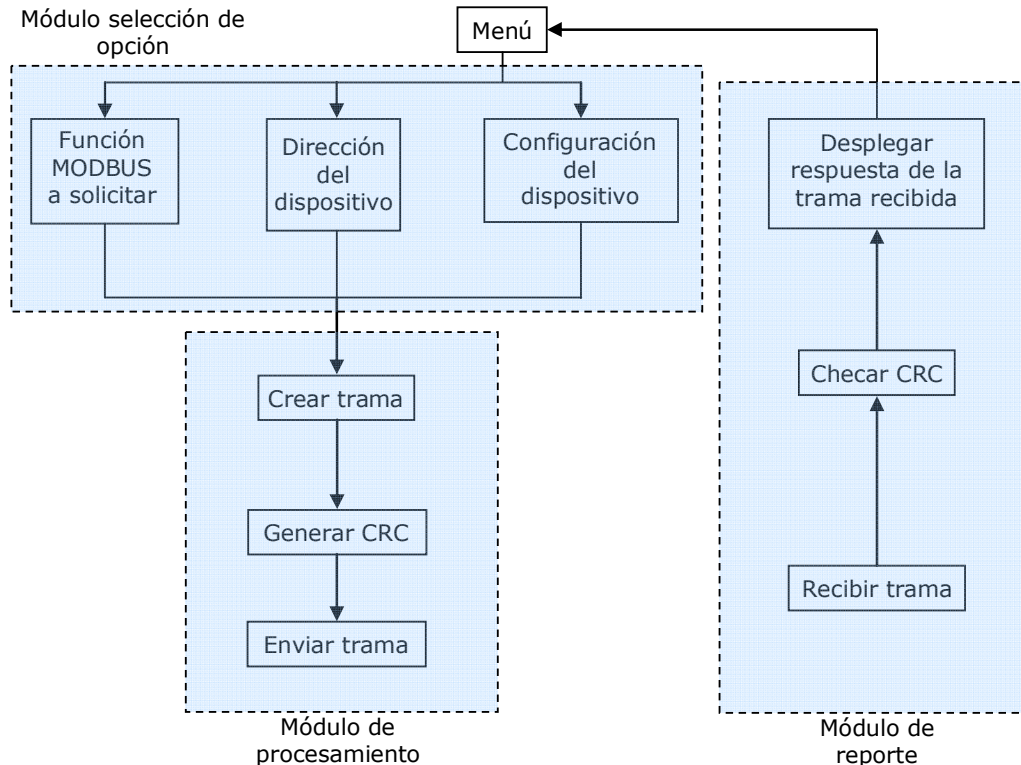


Figura 3.1. Estructura jerárquica del SPCC.

El módulo de reporte es el responsable de obtener los datos recibidos del CDECA que envía como respuesta, y muestra al usuario el resultado de la misma de forma que pueda ser entendible para él. El módulo se compone de tres bloques que son:

- Recibir trama, encargada de leer los datos del puerto serial.
- Checar CRC, calcula el CRC y compara que los datos de los campos correspondientes al CRC que ha enviado el dispositivo esclavo, sean los mismo a los que se han calculado.
- Desplegar respuesta de la trama recibida, se encarga de desplegar los resultados contenidos en la trama de respuesta del dispositivo esclavo.

Al iniciarse el programa, lo primero que se realiza es la creación del entorno gráfico, figura 3.2. Una vez que éste se ha ejecutado, el usuario es libre de configurar el dispositivo mediante las opciones del programa, como son: la dirección del dispositivo, velocidad de recepción/transmisión, tipo de paridad a usar y el puerto serial de la computadora. Estas opciones se encuentran en los apartados "**Configuración del puerto y Dirección del dispositivo**".

En la siguiente sección se explicarán con más detalle, los bloques de procesamiento y de reporte. Debido a que son los de mayor relevancia para la ejecución del programa.

3.3.1 Ejecución del programa.

En la figura 3.3 se muestra el diagrama de flujo general del programa. En él se aprecia el flujo que sigue la información para poder enviar y recibir tramas MODBUS.

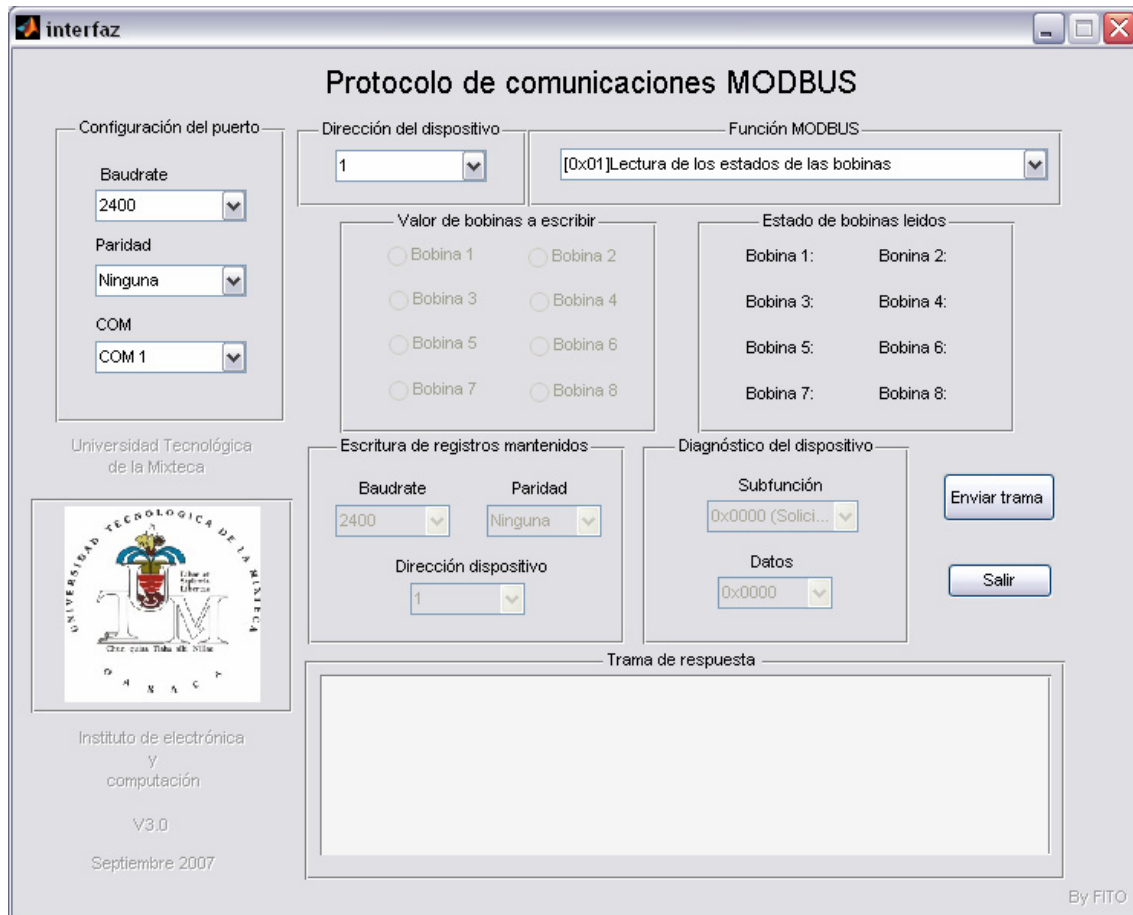


Figura 3.2. Entorno gráfico del SPCC.

Por defecto, el SPCC inicia con los valores presentes en la primer pantalla, es decir configura el "COM1" a una velocidad de 2400 BPS, sin paridad, dos bits de datos y 1 (uno) como dirección del dispositivo. Por lo tanto si el usuario decidiera dar la orden de enviar trama, presionado el botón del mismo nombre, la función que se ejecutará será la 0x01 (lectura de los estados de las bobinas).

Cuando se modifica alguna de las opciones, el cambio se guarda en la variable correspondiente. Por ejemplo, si se cambia el baudrate, el cambio se guarda en la variable BPS. Cuando se presiona la tecla de enviar trama, se ejecuta la parte que envía la trama serial y espera la respuesta del

esclavo. Cuando se presiona la tecla de salir, se termina la ejecución del programa y se cierra.

En el apartado "**Función MODBUS**" se encuentra el menú con las funciones que pueden ser ejecutadas por el CDECA, dependiendo de qué función se elija, los apartados "**Valor de bobinas a escribir**", "**Estado de bobina leídos**", "**escritura de registros mantenidos**" y "**Diagnóstico de dispositivo**" se irán desbloqueando para que pueda ser reunida la información que será enviada al CDECA.

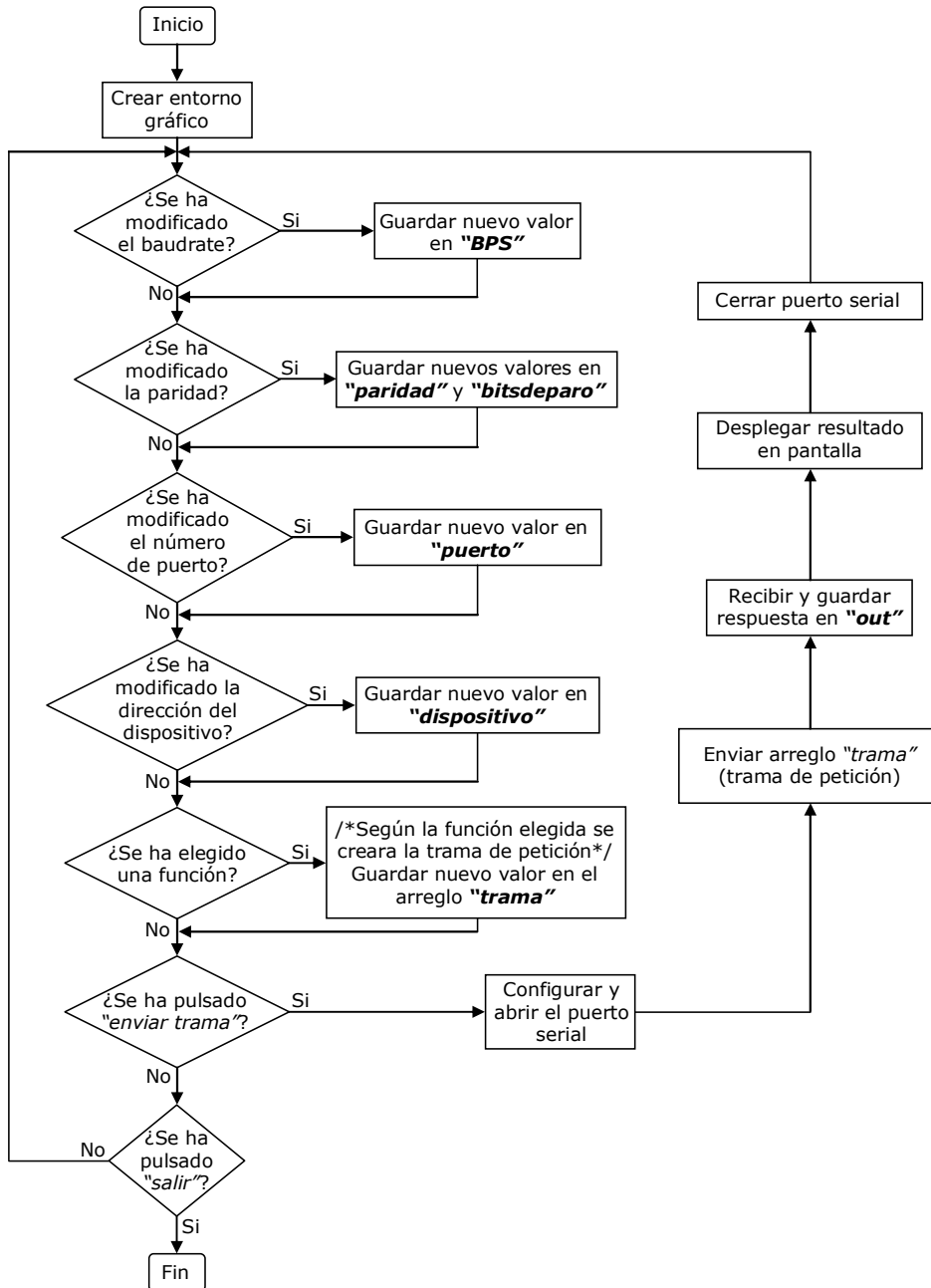


Figura 3.3. Diagrama de flujo general del SPCC.

Por ejemplo, para cambiar el estado de las cargas se debe elegir la función MODBUS número 0x0F, en ese instante se desbloqueará el apartado "**Valor de bobinas a escribir**" y se podrá decidir qué relevadores tendrán estado de apagado y encendido. Lo último que restaría es presionar el botón "Enviar trama" para enviar la solicitud de ejecución al CDECA y esperar la respuesta de éste.

En el apartado "**Trama de respuesta**", se desplegarán los mensajes que informarán al usuario si la ejecución de la función MODBUS enviada fue satisfactoria o errónea. Para salir del SPCC solo basta con presionar el botón "Salir".

3.3.2 Bloque de enviar y recibir trama.

En la figura 3.4 se muestra el diagrama de flujo del bloque Enviar trama, que pertenece al módulo de Procesamiento, así como también el de Recibir trama, que pertenece al módulo de Reporte. A continuación se explicará cada parte del diagrama de flujo, así como el código que lo realiza.

Al pulsar el botón "enviar" lo primero que se lleva a cabo es checar que la PC cuente con el puerto serial seleccionado, ya que si se usa algún puerto no definido, se produciría un error que terminaría en un fallo general del SPCC. El código que ejecuta esta opción es:

```
banderapuertotemp = 0;
puertosdisponibles = instrhwinfo('serial');
numpuertosdisponibles =
size(puertosdisponibles.AvailableSerialPorts);
for contador = 1:numpuertosdisponibles(1),
    if
        (char(puertosdisponibles.AvailableSerialPorts(contador)
) == handles.puerto)
            banderapuertotemp = 1;
        end
    end
end
```

La función `instrhwinfo` proporciona información del hardware indicado en su argumento, devolviendo un arreglo para su manipulación. Con este código, en caso de que no exista el puerto, no se manda la trama de solicitud y se despliega en pantalla el mensaje: "La máquina no cuenta con el puerto seleccionado".

El segundo paso a realizar es **configurar el puerto serial** a utilizar, lo que se hace con las siguientes líneas de código:

```
handles.s = serial(handles.puerto);
```

```

handles.s.InputBufferSize = 500;
handles.s.OutputBufferSize = 500;
set(handles.s, 'BaudRate', handles.BPS, 'Parity', handles.pari
dad, 'DataBits', 8, 'StopBits', handles.bitsdeparo);

```

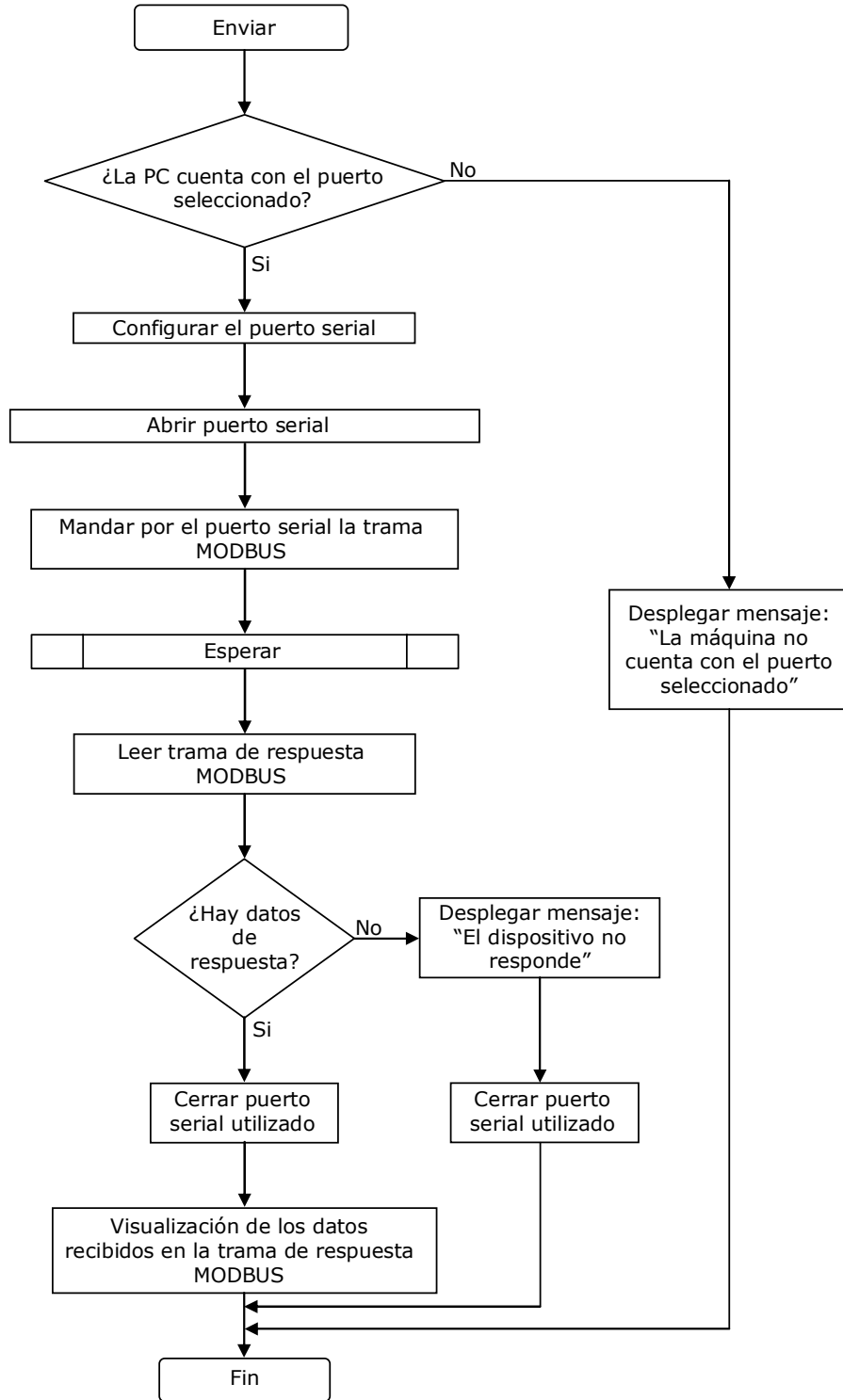


Figura 3.4. Diagrama de flujo para enviar la trama de petición MODBUS

La variable `handles.puerto` contiene el valor del puerto serial que se abrirá, por ejemplo, si se fuera a utilizar el primer puerto serial de la PC, la variable deberá tener el valor "COM1". Una vez escogido qué puerto utilizar, se crea una variable de tipo apuntador, la cual servirá para modificar y configurar los parámetros a los que se desee que funcione el puerto. Esto se hace por medio de la función "serial".

Se tiene que definir un buffer de entrada y salida donde se almacenan los bytes a enviar y recibir. En este caso ambos buffers tienen el mismo tamaño. Para obtener los datos recibidos y vaciar el buffer, sólo es necesario leer su contenido. Para enviar un dato, se escribe en el buffer y de forma automática el programa se encarga de enviar los datos por el puerto.

La función "set", sirve para establecer los valores de configuración del puerto serial a manipular. Esta función sirve para cambiar más de dos valores, los cuales pueden ser direccionados directa o indirectamente; es decir se pueden poner directamente los valores a utilizar o mediante el uso de variables.

El tercer paso, es **abrir el puerto serial** para tener comunicación con el dispositivo. Lo cual se logra utilizando las siguientes instrucciones:

```
if (banderapuertotemp == 1)
    fopen(handles.s);
end
```

Simplemente se pasa como argumento el apuntador `handles.s` creado por la función "serial" a la función `fopen`.

Es necesario aclarar que solamente se revisa que la PC cuente con el puerto serial seleccionado; cuando la trama MODBUS de petición es enviada y no se recibe una trama de respuesta por parte del dispositivo esclavo se despliega el mensaje: "El dispositivo no responde".

El cuarto paso es **mandar por el puerto serial la trama MODBUS**, lo cual se realiza con las siguientes líneas de código:

```
if (banderapuertotemp == 1)
    fwrite(handles.s,handles.trama,'uint8','async');
    while(handles.s.BytesToOutput ~= 0)
        end
end
```

La función `fwrite` es la encargada de enviar los datos, que conforman la petición, al puerto serial. El buffer de salida es un arreglo (`handles.trama`)

en el cual se almacenan los datos que serán enviados. La longitud de éste no debe sobrepasar el valor de la variable `OutputBufferSize` declarado al abrir el puerto serial.

Una vez que han sido enviados los datos, se da una pausa de 1 segundo para leer los datos de respuesta enviados por el dispositivo esclavo. De ello se encarga la subfunción `esperar`.

El quinto paso es **leer la trama de respuesta MODBUS** con la siguiente instrucción:

```
out = fread(handles.s,handles.s.BytesAvailable,'uint8');
```

La función `fread` vacía el buffer de entrada, guardando su contenido en el arreglo `out`.

El sexto paso es **cerrar el puerto serial utilizado** para evitar que la próxima trama de respuesta se contamine con datos basura. Por lo cual se implemente el siguiente código:

```
stopasync(handles.s);  
fclose(handles.s);  
clear handles.s;
```

La función `stopasync`, se encarga de detener la comunicación asíncrona que se dió entre el SPCC y el dispositivo esclavo, mientras que la función `fclose` cierra el puerto serial. Una vez que se ejecuta, si se manda un dato por el puerto, el intérprete del sistema marca error de comunicaciones.

La instrucción `clear` simplemente libera de la memoria RAM de la PC el espacio utilizado por el apuntador `handles.s`.

Por último, se tiene la **visualización de los datos recibidos en la trama de respuesta MODBUS**. Esta parte depende de la respuesta que haya enviado el esclavo; es decir, la visualización de los resultados depende del tipo de función ejecutada por el CDECA.

3.4 Pruebas realizadas al SPCC.

Las pruebas que se realizaron al SPCC para verificar su buen funcionamiento fueron de integración, por ser las más convenientes, ya que la herramienta "GUIDE" de MATLAB® genera un módulo o función por cada objeto creado en el entorno.

Se optó por este tipo de pruebas enfocadas a la interacción que tienen los módulos; es decir, que se verifica el intercambio de información entre ellos.

MATLAB® ofrece la posibilidad de ejecutar y probar rutinas independientemente de que sean invocadas o no por el programa principal. Sin embargo, las pruebas se realizaron con todos los módulos que integran el sistema.

Una vez concluida en su totalidad la programación del SPCC, se realizaron pruebas de funcionamiento utilizando directamente el CDECA diseñado como elemento del sistema. Se comprobó que todas las tareas implementadas funcionaran correctamente y proporcionaran los resultados adecuados. En el capítulo siguiente se describen con más detalle el resultado de las pruebas hechas al sistema.

Capítulo 4.

Pruebas y resultados

En este capítulo se describirán las pruebas realizadas al CDECA para verificar su buen funcionamiento. Éstas fueron realizadas utilizando el SPCC, descrito en el capítulo anterior, y el CDECA descrito en el capítulo 2.

En general una vez instalado el dispositivo, su funcionamiento es relativamente simple; se envía la trama MODBUS con la petición que se desea ejecutar, recibe la trama, la decodifica y en caso de que sea correcta la petición, la ejecuta y envía la respuesta. Por tal motivo solo se realizaron 3 tipos de pruebas que son:

- Enlace de comunicaciones y tasa de error.
- Apertura y cierre del CDECA
- Enlace RS-485.

4.1 Pruebas del enlace de comunicaciones seriales.

Para probar el buen funcionamiento del puerto de comunicaciones seriales, así como la tasa de error que se puede conseguir, se programó el microcontrolador del CDECA con un programa que retransmite los datos que recibe por el puerto USB. Las tramas fueron de 255 datos y el proceso se repitió 1000 veces. Calculando la tasa de error con la ecuación 4.1. Se hizo un programa en MATLAB® el cual envía la cadena de datos por el puerto serie, recibe la respuesta del CDECA y hace una comparación de los datos recibidos para verificar si ocurrió un error.

$$\text{Tasa de error} = \frac{\text{Tramas erróneas recibidas}}{\text{Total de tramas recibidas}} \times 100\% \quad (4.1)$$

El proceso descrito en el párrafo anterior se repitió para todas las velocidades que soporta el puerto USB, desde 2200 BPS hasta 115200 BPS.

En la figura 4.1 se muestra el montaje usado para esta prueba, en la cual se observa, la computadora, el CDECA y el convertidor USB-serial (ComUSB); este último se conectó a una PC por medio de un cable USB.



Figura 4.1. Montaje para probar la tasa de errores

Los datos de esta prueba dieron como resultado una tasa de error del 0% para todas las velocidades configuradas.

Hay que aclarar que las pruebas se realizaron en un ambiente libre de ruido electromagnético y la longitud de los cables utilizados para enviar y recibir los datos fue demasiado corta como para afectar a las transmisiones. El resultado obtenido puede variar si se incrementa la longitud de los cables.

Un problema que se presentó al realizar la prueba fue el no dejar tiempos de espera adecuados a cada velocidad de transmisión, en el programa que controla a la computadora. Se implementó un programa en MATLAB[®] en el cual se hicieron varias modificaciones de código hasta obtener el resultado adecuado.

El programa en MATLAB[®] usado para la prueba de velocidad se muestra a continuación. Creemos importante incluirlo en el documento para que sirva como referencia en trabajos futuros. La velocidad de transmisión a la que se configura el puerto de comunicaciones es de 115200 BPS.

```
clear
clc
BPS = 115200;
paridad = 'none';
bitspordato = 8;
```

```

bitsdeparo = 2;
s = serial('COM3');
s.InputBufferSize = 1014;
s.OutputBufferSize = 1012;
s.Timeout = 100;
set(s, 'BaudRate', BPS, 'Parity', paridad, 'DataBits', bitspordato, 'StopBits', bitsdeparo);
fopen(s);
trama = 1:1:255;
trama = uint8(trama);
for j=1:1000
    j
    fwrite(s, trama, 'uint8', 'async');
    while(s.BytesToOutput ~= 0)
    end
    while (s.BytesAvailable ~= 257)
    end
    if (j==1)
        out = fread(s, s.BytesAvailable, 'uint8');
    else
        out2 = fread(s, s.BytesAvailable, 'uint8');
        out = cat(2, out, out2); %1
    end
end
end

stopasync(s);
fclose(s);
delete(s);
clear s

```

4.2 Pruebas de apertura y cierre del controlador.

El funcionamiento del CDECA se probó con la configuración mostrada en la figura 4.2. El CDECA se conectó a una computadora personal por medio de los puertos de comunicación que ésta posee, en la computadora se ejecutó el programa de pruebas desarrollado en MATLAB®. Las interfaces de comunicación usadas fueron el puerto USB, RS-232 y el RS-485.

La primera prueba es verificar que el controlador funcione de acuerdo a lo programado. Para ello se usó una PC, el ComUSB, un convertidor RS-232 a RS-485 y el CDECA. Para verificar que efectivamente se activan o no las cargas se utilizó un foco, figura 4.3.

Los pasos a seguir para usar el CDECA se muestran en el diagrama de flujo de la figura 4.4. En resumen éstos son:

- Configuración de los puertos de comunicación del CDECA.
- Envío de petición MODBUS al dispositivo.

- Verificación de que ha sido ejecutada la petición.
- Verificación de la respuesta del dispositivo.

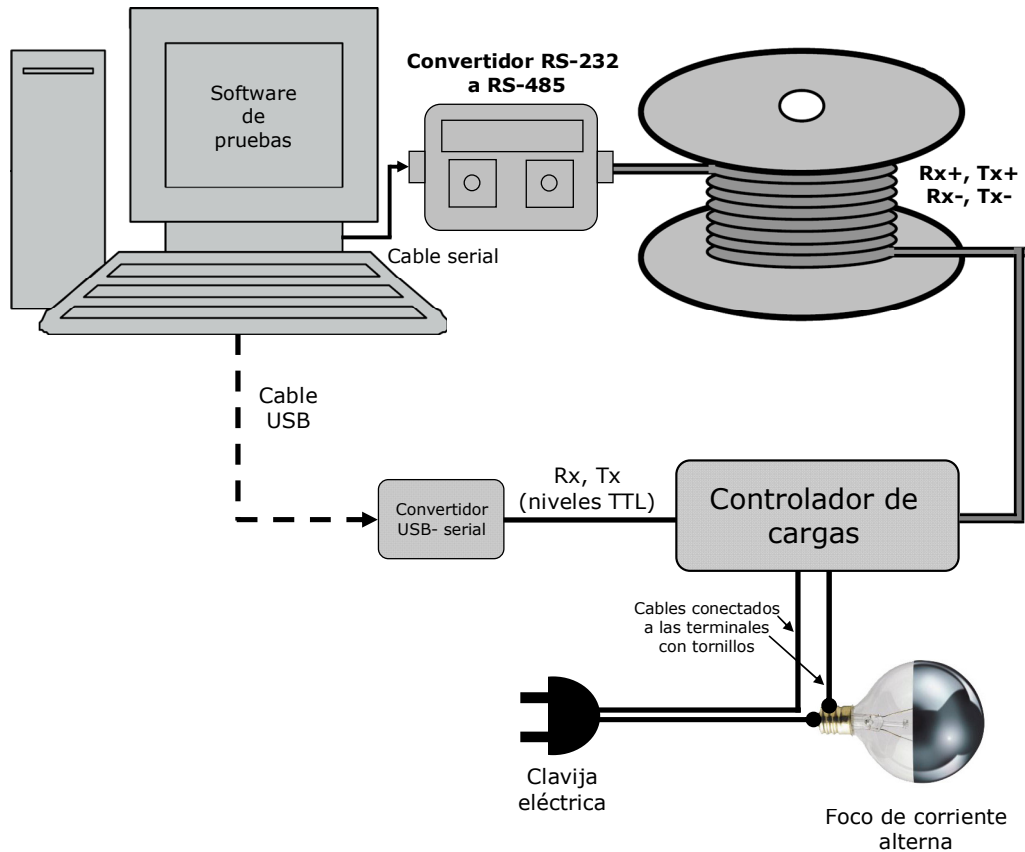


Figura 4.2. Diagrama de conexiones utilizado para probar el CDECA.

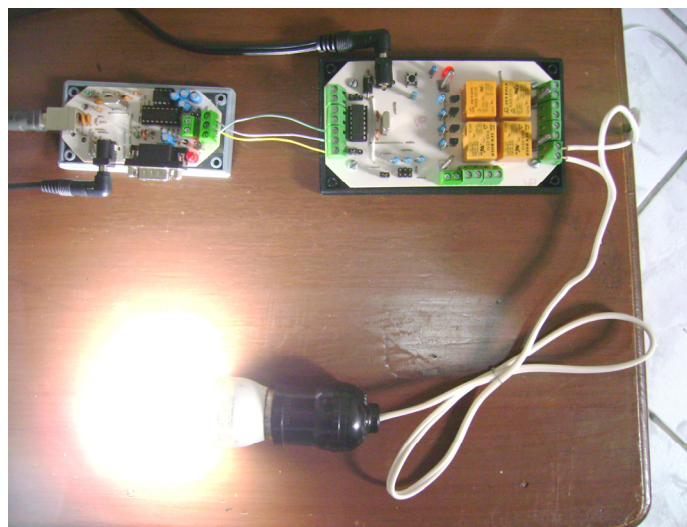


Figura 4.3. Conexión de componentes utilizados para las pruebas del CDECA utilizando el puerto USB.

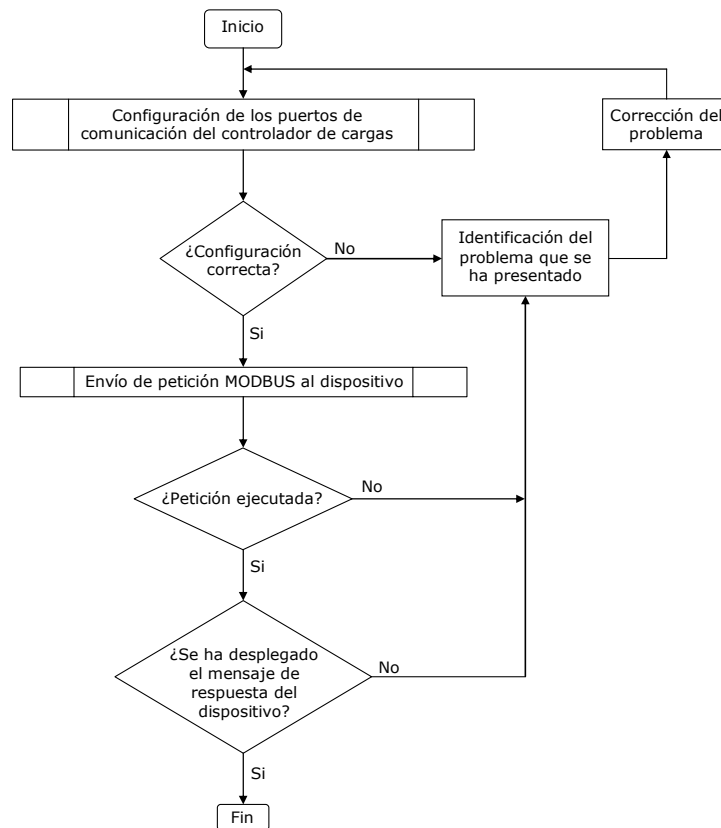


Figura 4.4. Diagrama de flujo seguido en las pruebas del CDECA.

En la figura 4.5 se muestra la pantalla del programa cuando todas las salidas del CDECA están activas. Para tener esta configuración se selecciona la función "(0x0F) Escribir múltiples valores de bobinas" que se selecciona en la sección "Valor de bobinas a escribir". Al presionar "Envía Trama", y al recibir la respuesta del CDECA, la sección "Estado de las bobinas leídas" se ponen en ON, en "Trama de Respuestas" aparece el mensaje "El valor de las bobinas ha sido modificado". Cuando esto ocurre, se activa la salida del relevador que tiene conectado el foco, y hace que se encienda (figura 4.3).

En caso de que desde el programa de control se seleccione un puerto serial no definido, la respuesta que genera el programa se muestra en la sección "Trama de Respuestas". En ella aparece el mensaje "La máquina no cuenta con el puerto seleccionado", como se muestra en la figura 4.6. Esta opción se implementó debido a que si se usa un puerto no definido, se generaba un error de ejecución y se terminaba abruptamente la ejecución del programa.

En la figura 4.7 se muestra la pantalla resultante cuando se manda a ejecutar una función MODBUS y por algún motivo el dispositivo esclavo no

devuelve respuesta alguna. Las condiciones bajo las que ocurre esto son: que el mensaje no fue dirigido al CDECA, que la velocidad del dispositivo maestro es distinta a la del esclavo, que fue un mensaje de difusión, etc.

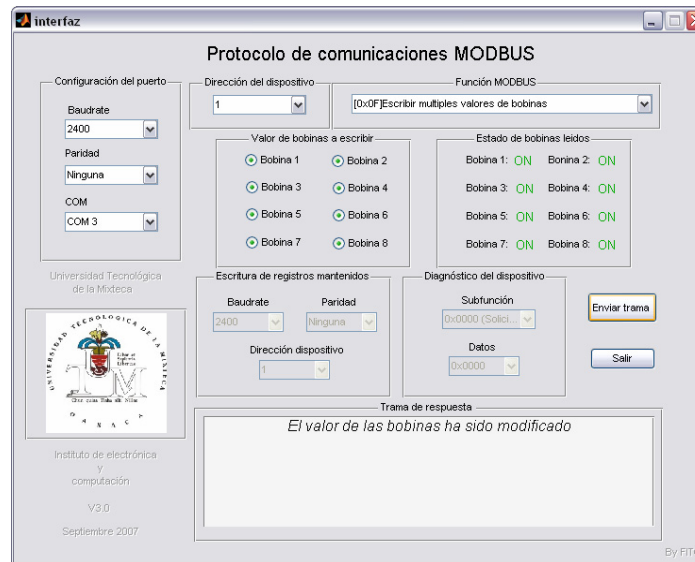


Figura 4.5. Panel frontal del programa donde se activan todas las cargas

Las pruebas aquí mostradas se llevaron a cabo varias veces, y en todas se obtuvo la respuesta esperada, por tal motivo se puede concluir que los resultados obtenidos son satisfactorios.

Algo importante que hay que mencionar acerca del desempeño del programa de control, es que en algunas ocasiones se queda congelado. Es decir, no responde a las órdenes que se le dan desde el teclado o el ratón. Para descartar que fuera resultado de una mala programación se probó cada módulo del programa en forma independiente y funcionó sin ningún problema. Por tal motivo creemos que el problema radica en la interfaz usuario que se desarrolla con la herramienta "GUIDE", ya que genera mucho código que tiene que ejecutar la computadora.

4.3 Pruebas del enlace en RS-485.

La última prueba fue verificar el funcionamiento del puerto RS-485, el cual viene directamente montado sobre el CDECA. Para conectarlo a la computadora, se utilizó un convertidor de RS-485 a RS-232 de la firma MAXIM modelo MAX489. En la computadora se ejecutó el programa de control desarrollado en MATLAB®.

Se usó un tramo de cable de 100 metros con categoría telefónica (tipo Cat 1). La computadora con el convertidor MAXIM funcionó como el maestro y

el esclavo fue el CDECA. Esta prueba se uso a esta distancia, debido a que una ventaja del puerto RS-485 es que puede transmitir señales a una distancia de hasta 1km.

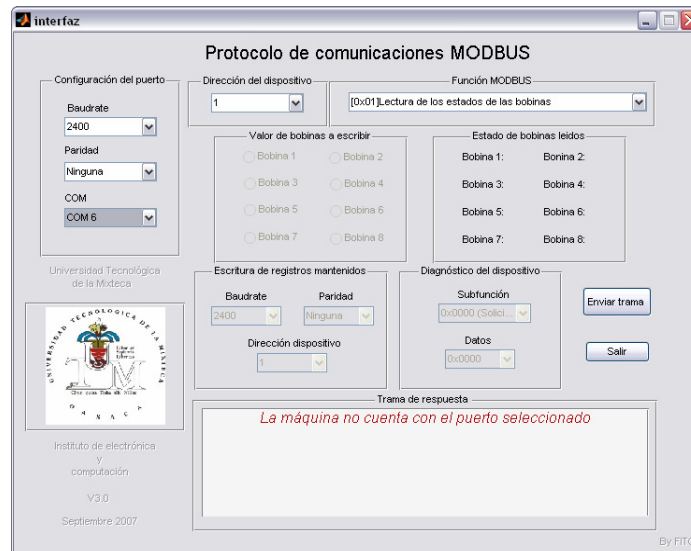


Figura 4.6. Panel frontal del programa al escoger un puerto serial no presente en la PC.

El diagrama de conexiones se muestra en la figura 4.2 con todos los componentes usados para probar este puerto. Se hicieron las mismas pruebas que en el apartado anterior, pero en este caso con la interfaz RS-485. Al igual que en el apartado anterior se obtuvieron los mismos resultados.

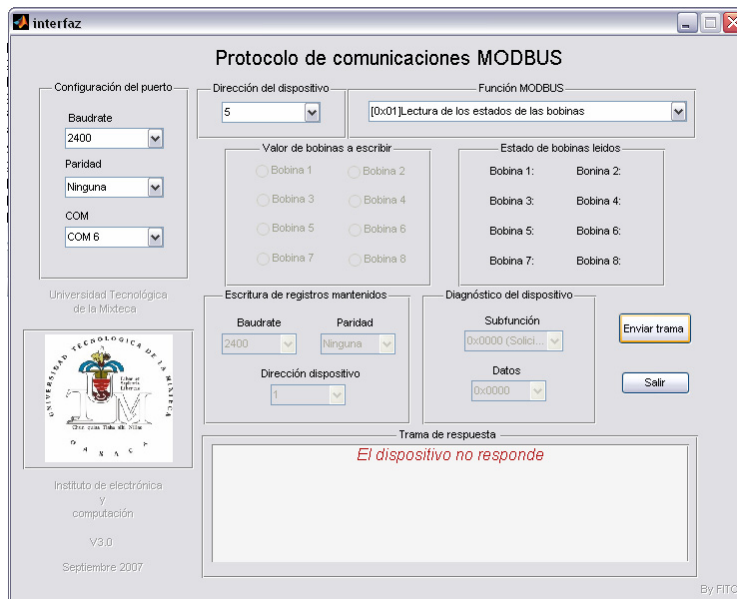


Figura 4.7. Panel frontal del programa al no obtener una respuesta del dispositivo esclavo.

Capítulo 5.

Conclusiones

En el presente documento se describió el desarrollo realizado para construir el "CDECA serial", así como también el del SPCC para la activación y desactivación de las cargas eléctricas.

No hay que olvidarse también que se desarrolló el ComUSB para tener la opción de implementar un puerto USB a algún proyecto que así lo necesite. Todo sin necesidad de saber cómo funciona realmente este puerto, siendo totalmente transparente su uso gracias al puerto serial virtual creado por los drivers del CI utilizado en el dispositivo.

La realización de este proyecto de tesis se hizo con la finalidad de contribuir al mejor uso o consumo de la energía eléctrica y por ende bajar la quema de fuentes de energía fósiles. Una vez finalizado el desarrollo del proyecto se ha llegado a las siguientes conclusiones:

- El CDECA es un dispositivo terminado a nivel de circuito impreso, protegido por un chasis para evitar algún daño físico del mismo y listo para ser instalado en una red industrial que utilice el protocolo de comunicaciones MODBUS.
- El firmware del CDECA fue realizado mediante la utilización del compilador WinAvr, con el cual se pueden compilar programas de lenguaje C adaptado para ser utilizado en los microcontroladores de 8 bits de la firma ATMEL[®]. Esto facilita y acorta el tiempo de desarrollo de proyectos con estos microcontroladores, pero tiene la desventaja de generar más código para cargar en la memoria flash del que se crearía al utilizar el lenguaje ensamblador para este tipo de CI.

- Se optó por tener un puerto serial que manejase niveles TTL para tener la opción de poder conectar radio modems que hagan uso de este mismo tipo de señales, esto para tener una opción más de comunicación sin necesidad de utilizar cables para transmitir las tramas de información.
- Las pruebas realizadas utilizando el SPCC como dispositivo maestro y al CDECA como dispositivo esclavo fueron satisfactorias, ya que cada una de las funciones MODBUS implementadas en el CDECA funcionaron correctamente.
- El SPCC se desarrollo en MATLAB[®] para probar las rutinas que hacen uso del puerto serie. Sin embargo el principal problema que se tuvo es que después de estar corriendo el programa, éste de repente dejaba de funcionar. Creemos que esto se debe a la interfaz gráfica y a la función que lee el reloj interno de la PC, ya que cuando este programa se corría sin los dos componentes mencionados no ocurría ningún problema.
- El funcionamiento del ComUSB fue mejor de lo esperado, ya que al realizar las pruebas y obtener el porcentaje de error se observó que este es del 0%. Hay que aclarar que esta cifra se obtuvo después de una serie de pruebas realizadas utilizando el lenguaje MATLAB[®] y el ComUSB.

Trabajos futuros

Los posibles trabajos futuros que se pueden realizar tomando como base éste y otros trabajos desarrollados previamente son:

- Realizar el controlador de demanda distribuido, debido a que se cuentan con el codificador de pulsos "CODKYZ" desarrollado en una tesis anterior por el ingeniero Enmanuel Aparicio Velázquez, y a que se tiene disponible el CDECA.
- Para estar a la vanguardia en lo que se refiere a la implementación del protocolo de comunicaciones MODBUS, un trabajo futuro sería el implementar el protocolo usando su versión TCP/IP en los dispositivos ya realizados. Ésta puede ser una buena opción ya que se podría aprovechar la infraestructura de red tipo LAN de la planta, en caso de que ésta contase con ella.
- Una mejora de hardware del dispositivo CDECA es la de verificar que las cargas a controlar, se encuentren realmente conectadas a las terminales de control, y así evitar mal funcionamiento en lo que se refiere al control de la demanda máxima de la planta.

- Otra opción a tomar en cuenta para la implantación de este sistema es usar el protocolo X10 para transmitir los datos por la red eléctrica interna de la planta. Esta opción solamente sería válida si se llegase a la conclusión de que "la línea de red eléctrica de una planta industrial es un buen medio para transmitir información a velocidades moderadas".

Referencias

Bibliografía

- [1] Aparicio Velásquez Emmanuel; Codificador de pulsos KYZ bajo el protocolo de comunicaciones modbus para medidores electrónicos de energía eléctrica; Tesis de licenciatura; Universidad Tecnológica de la Mixteca; 2004.
- [2] Tippens, Paul E.; Física, conceptos y aplicaciones; 2da. Edición; McGraw-Hill; México; 1991.
- [3] Océano práctico, diccionario de la lengua española; 1ra. Edición; Editorial Océano; México; 1995.
- [4] Hayt, William H.; Análisis de circuitos en ingeniería; McGraw-Hill; 1994.
- [5] MODBUS APPLICATION PROTOCOL SPECIFICATION; V1.1b; MODBUS organization; USA; 2006.
- [6] Matlab R2006b; Help system; The mathworks; USA; 2006.
- [7] Universal Serial Bus Specification; Rev 2.0; Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips; USA; 2000.
- [8] Jan Axelson; USB COMPLETE; 2da. Edición; Lakeview research; Madison, USA; 2001.
- [9] FT232BM USB UART (USB – Serial) I.C; V1.8; Future Technology Devices International Ltd.; USA; 2005.

Internet

- [URL1] http://www.fide.org.mx/el_fide/resultados.html, "Fideicomiso para el Ahorro de Energía", última visita: 28 de noviembre del 2007.
- [URL2] <http://aplicaciones.cfe.gob.mx/aplicaciones/ccfe/tarifas/tarifas/Tarifas.asp?Tarifa=OM>, "Tipos de tarifas eléctricas en México", "Costo de la energía eléctrica en México", "Definición de demanda máxima medida", última visita: 28 de noviembre del 2007.
- [URL3] <http://www.cfe.gob.mx/es/InformacionAlCliente/conocetutarifa/disposicionescomplementarias/2007/3/>, "Fórmulas de bonificación o cargo por F.P", última visita: 28 de noviembre del 2007.
- [URL4] http://www.conae.gob.mx/wb/CONAE/CONA_2408_sistemas_de

- _control_, "Controlador de demanda", última visita: 28 de noviembre del 2007.
- [URL5] <http://www.carrel.co.nz/transducer/mdc-01.htm>, "Compañía CARREL & CARREL LTD", última visita: 28 de noviembre del 2007.
- [URL6] http://www.induselect.com/maximum_demand.html, "Compañía INDUS ELECTRONICS INDIA (P) LTD, última visita: 28 de noviembre del 2007.
- [URL7] http://www.alibaba.com/catalog/11308123/Maximum_Demand_Controller.html, "Compañía United Machinery Corporation", última visita: 28 de noviembre del 2007.
- [URL8] http://www.conae.gob.mx/wb/CONAE/CONA_2409_controles_manuales_d, "Controladores de demanda máxima manuales", última visita: 28 de noviembre del 2007.
- [URL9] http://www.conae.gob.mx/wb/CONAE/CONA_2410_controles_automatizado, "Controladores de demanda máxima automáticos", última visita: 28 de noviembre del 2007.
- [URL10] <http://www.usb.org/home>, "Página principal de la organización USB", última visita: 28 de noviembre del 2007.
- [URL11] <http://www.ftdichip.com>, "Fabricante del CI FT232BM", última visita: 28 de noviembre del 2007.
- [URL12] <http://winavr.sourceforge.net>, "Compilador WinAVR", última visita: 28 de noviembre.
- [URL13] <http://www.mathworks.com/company/aboutus/index.html>, "Historia y características Matlab[®]", última visita: 28 de noviembre del 2007.
- [URL14] http://www.mathworks.com/support/sysreq/current_release, "Sistemas operativos en los cuales puede ser instalado Matlab[®]", última visita: 28 de noviembre del 2007.
- [URL15] <http://www.mathworks.com/products/distribtb>, "Descripción de las Toolboxes de Matlab[®]", última visita: 28 de noviembre del 2007.
- [URL16] http://people.scs.fsu.edu/~burkardt/m_src/linpack/linpack.html, "Descripción de la librería LINPACK", última visita: 28 de noviembre del 2007.
- [URL17] <http://www.cisl.ucar.edu/softlib/EISPACK.html>, "Descripción de la librería EISPACK", última visita: 28 de noviembre del 2007.
- [URL18] http://www.mathworks.com/access/helpdesk/help/techdoc/creating_guis/bqz79mu.html, "Guía en línea para la creación de interfaces gráficas en Matlab[®]", última visita: 28 de noviembre del 2007.
- [URL19] <http://www.maxim-ic.com>, "Fabricante de los CI's MAX489 y MAX232", última visita: 28 de noviembre del 2007.

Apéndice A. Código en lenguaje C del firmware utilizado en el CDECA

Para el mejor entendimiento de éste código es recomendable que el lector tenga conocimiento del protocolo MODBUS.

```

/*****
*
* Titulo      : CDECA (Controlador de cargas)
* Version     : 3.0
* Ultima actua. : Agosto 17 del 2007
* Tarjeta     : ATmega32L
* Archivo     : MODBUS.c
* Autor      : Rodolfo Paz Carreño
*
* Este software es de libre distribucion, fue compilado con el WinAVR-20070122.
*
* DESCRIPCION
*
* El protocolo de comunicaciones MODBUBS es muy usado por la industria
* para poder controlar los dispositivos conectados en una red serial.
*
* Se han implementado los siguientes comandos:
*
* - Función 0x01: lectura de los estados de las bobinas
* - Función 0x08: Diagnóstico del dispositivo
* - Función 0x0B: Devolver el contador de eventos de comunicación
* - Función 0x0F: Escribir multiples valores de bobinas
* - Función 0x2B: Identificación del dispositivo
*
*
* NOTA
*
* Para obtener mayor información sobre el protocolo implementado
* visitar http://www.modbus.org
*****/

#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <compat/deprecated.h>

void configurar_puertos();
void configurar_USART();
void configurar_temporizador();
void atender_trama();
uint16_t CRC16(uint8_t [], uint8_t, uint8_t);
void enviaresp(uint8_t [], uint8_t);
void funcion0F(uint8_t []);
void funcion10(uint8_t []);
void funcion0B(uint8_t []);
void funcion2B(uint8_t []);
void funcion01(uint8_t []);
```

```

void funcion08(uint8_t []);

volatile uint16_t cont_timer=0x0000; //cuenta los sobreflujos del timer para verificar fin de recepcion
volatile uint8_t contUDR0=0; //contador de los datos recibidos en la USART
volatile uint8_t cadpeti[45]={0}; //arreglo donde se guardan las peticiones "modificar en caso de que la
trama a recibir contenga mas de 45 bytes de datos"
volatile uint8_t direcescla; //se guarda la direccion del esclavo
uint8_t MSE=0; //guarda el estado de solo escuchar '0: desactivado' , '!=0 activado'
uint8_t trama03[2]= {3,3}; //se guarda la configuracion que tendra la USART
uint8_t estadobobinas = 0; //se guarda el estado de las bobinas
uint16_t contope; //tope del timer counter para verificar fin de recepcion
uint16_t CCE = 0x0000; //contador mensajes exceptuando las peticiones de eco
uint16_t regcomuerronea = 0x0000; //contadores errores de CRC16
uint16_t regresexcep=0x0000; //contador respuestas de excepcion
uint16_t regcomexec=0x0000; //contador de peticiones ejecutadas
uint16_t regmendec=0x0000; //contador de mensajes detectados en el BUS
uint16_t contmendif=0x0000; //contador de mensajes difundidos
uint8_t veeprom, valorUCSRC, valorUBRRH, valorUBRRL; //variables usadas para guardar valores
temporalmente

ISR (USART_RXC_vect) //vector de interrupción (al recibir un dato) de la USART 0
{
uint8_t c;

    TCCR2 = 0x00; //para detener el conteo del timer counter 2
    cont_timer = 0; //reset al contador de sobreflujos del counter 2
    TCNT2 = 0x00; //carga el inicio de conteo del timer counter 0
    c = UDR; //guarda el valor recibido en la USART
    cadpeti[contUDR0] = c; //para guardar los bytes recibidos de la trama MODBUS
    TCCR2 = 0x01; //para iniciar el conteo del timer counter 2
    contUDR0++; //incrementa el indice para el arreglo 'cadpeti'
}

ISR(TIMER2_OVF_vect) //vector de interrupción (al haber overflow) del TIMER/COUNTER 0
{
    cont_timer++; //contador de sobreflujos
}

int main()
{
configurar_puertos();

configurar_USART();

configurar_temporizador();

sei (); //activa las interrupciones de la USART 0 y Timer Counter 2

while(1)
{

MODBUS

    if (cont_timer == contope) //verifica el tiempo de espera que indica el fin de la trama
    {

        PORTB = 0xFF; //deshabilita la recepción de datos en modo RS-485
        UCSRB = 0x18; /*deshabilita la interrupcion cuando se recibe un dato en
        la USART 0*/
        TCCR2 = 0x00; /*para parar el timer counter 0*/
        regmendec++;
        if ((CRC16(cadpeti,contUDR0-2,1)) == 0xFFFF) /*llamar a una rutina para
        que cheque el CRC16*/
        {
            if (cadpeti[0]==0x00)
            {
                contmendif++;
            }
            if (cadpeti[3]!=0x08 && (cadpeti[5]!=0x00 || cadpeti[5]!=0x0B ||
            cadpeti[5]!=0x0C || cadpeti[5]!=0x0D || cadpeti[5]!=0x0E ||
            cadpeti[5]!=0x0F))
            {
                CCE++;
            }
            if (MSE==1 && cadpeti[1]==0x08 && cadpeti[2]==0x00 &&
            cadpeti[3]==0x01)
            {
                funcion08(cadpeti);
            }
        }
    }
}

```

```

        cadpeti[0] = direcescla + 1; //para evitar que entre en la
                                siguiente comparación
    }
    if ((cadpeti[0] == direcescla || cadpeti[0] == 0) && (MSE!=1)) //verifica
    que el mensaje sea dirigido a éste dispositivo y el modo de solo escuchar
    este desactivado
    {
        atender_trama();
    }
    contUDR0 = 0;
    cont_timer = 0;
    UCSRB = 0x98; //reinicia la interrupcion de la USART
}
else
{
    regmendec++;
    contUDR0=0;
    cont_timer = 0;
}
}
}
return 0;
}

/*****función que verifica el CRC16 de la trama de solicitud, asi como crea el CRC16 de la trama
de respuesta*****/
uint16_t CRC16(uint8_t trama[], uint8_t numelem, uint8_t creacheca)
{
    uint8_t regCRCbajo=0xFF, regCRCalto=0xFF, cont=0, contcorri=0, temp=0;
    uint16_t CRCcreado=0x0000;
    for (cont = 0; cont < numelem; cont++)
    {
        regCRCbajo = regCRCbajo ^ trama[cont];
        while (contcorri <= 7)
        {
            do
            {
                regCRCbajo = regCRCbajo >> 1;
                cbi(SREG, 0); //limpia bandera de sobreflujo
                regCRCalto = regCRCalto >> 1;
                if (bit_is_set(SREG, 0))
                {
                    regCRCbajo = regCRCbajo | 0x80; //se le agrega el bit del
                    corrimiento
                }
                contcorri++;
                temp = regCRCbajo & 0x01;
            } while (((temp != 1) && (contcorri <= 7)));
            if (temp == 1)
            {
                regCRCbajo = regCRCbajo ^ 0x01;
                regCRCalto = regCRCalto ^ 0xA0;
            }
        }
        contcorri = 0;
    }
    switch(creacheca)
    {
        case 1:
            {
                if ( ((trama[numelem] == regCRCbajo) && (trama[numelem + 1] == regCRCalto))
                || ((trama[numelem] == regCRCalto) && (trama[numelem + 1] == regCRCbajo)) )
                {
                    CRCcreado=0xFFFF;
                }
                else
                {
                    CRCcreado=0x7FFF;
                    regcomuerronea++;
                }
            }
            break;
        case 2:
            {
                CRCcreado = CRCcreado | regCRCalto;
            }
    }
}

```

```

        CRCreado = CRCreado << 8;
        CRCreado = CRCreado | regCRCbajo;
    }
    break;
}
return(CRCreado);
}
/*****
*****

/*****funcion para transmitir la trama de
respuesta*****/
void enviaresp(uint8_t tramaout[], uint8_t numelemin)
{
uint8_t cont=0, CRCoutbajo = 0, CRCoutalto = 0;
uint16_t CRCreado=0x0000;
CRCreado = CRC16(tramaout,numelemin,2);
CRCoutbajo = CRCoutbajo | CRCreado;
CRCreado = CRCreado >> 8;
CRCoutalto = CRCoutalto | CRCreado;
for (cont = 0;cont < numelemin;cont++)
{
    UDR = tramaout[cont];
    loop_until_bit_is_set(UCSRA,UDRE);
}
if (CRCoutbajo < CRCoutalto)
{
    UDR = CRCoutbajo;
    loop_until_bit_is_set(UCSRA,UDRE);
    UDR = CRCoutalto;
    loop_until_bit_is_set(UCSRA,UDRE);
}
else
{
    UDR = CRCoutalto;
    loop_until_bit_is_set(UCSRA,UDRE);
    UDR = CRCoutbajo;
    loop_until_bit_is_set(UCSRA,UDRE);
}
}
PORTB = 0x00; //Habilita la recepción de datos en modo RS-485
}

/*****
*****

/*****funcion para escribir multiples registros
mantenidos*****/
void funcion10(uint8_t tramaout10[])
{
uint8_t tramagene[9]={0},temp=1,temp2=0,temp3=0,temp4=0,temp5=0,temp6=0,temp7=0,temp8=0;
UCSRB = 0x00;//desabilita todo de la USART
if (tramaout10[2]!=0x00 || tramaout10[3] > 0x02 || tramaout10[4]!=0 || tramaout10[5]>0x03 ||
tramaout10[6]>0x06)
{
    tramagene[0]=direcescla; //*****código de excepcion 0x02: se ha intentado
                                leer un registro con direccion ilegal
    tramagene[1]=0x90;
    tramagene[2]=0x02;

    if (tramaout10[0]!=0x00)
        enviaresp(tramagene,3);
    regresexcep++;
}
else
{
    direcescla = tramaout10[12];
    eeprom_write_byte (2,tramaout10[12]);
    temp2=UBRRL;
    temp3=UCSRC;
    temp4=trama03[0];
    temp5=trama03[1];
    temp6=contope;
    switch(tramaout10[8])
    {
        case 1:
        {

```

```

        trama03[0]=1;//velocidad de 2400 BPS
        temp7 = 191;
        contope = 462;
        eeprom_write_byte (0,1);//escribe un byte en la EEPROM
    }
    break;
    case 2:
    {
        trama03[0]=2;//velocidad de 4800 BPS
        temp7 = 95;
        contope = 231;
        eeprom_write_byte (0,2);//escribe un byte en la EEPROM
    }
    break;
    case 3:
    {
        trama03[0]=3;//velocidad de 9600 BPS
        temp7 = 47;
        contope = 116;
        eeprom_write_byte (0,3);//escribe un byte en la EEPROM
    }
    break;
    case 4:
    {
        trama03[0]=4;//velocidad de 19200 BPS
        temp7 = 23;
        contope = 58;
        eeprom_write_byte (0,4);//escribe un byte en la EEPROM
    }
    break;
    case 5:
    {
        trama03[0]=5;//velocidad de 38400 BPS
        temp7 = 11;
        contope = 29;
        eeprom_write_byte (0,5);//escribe un byte en la EEPROM
    }
    break;
    case 6:
    {
        trama03[0]=6;//velocidad de 57600 BPS
        temp7 = 7;
        contope = 20;
        eeprom_write_byte (0,6);//escribe un byte en la EEPROM
    }
    break;
    case 7:
    {
        trama03[0]=7;//velocidad de 115200 BPS
        temp7 = 3;
        contope = 10;
        eeprom_write_byte (0,7);//escribe un byte en la EEPROM
    }
    break;

    default:
        tramaout10[10]=6;
    break;
}
switch(tramaout10[10])
{
    case 1://paridad par (even), un bit de paro
    {
        trama03[1]=1;
        temp8 = 0xA6;
        eeprom_write_byte (1,1);//escribe un byte en la EEPROM
    }
    break;
    case 2://paridad impar, un bit de paro
    {
        trama03[1]=2;
        temp8 = 0xB6;
        eeprom_write_byte (1,2);//escribe un byte en la EEPROM
    }
    break;
    case 3://sin paridad, dos bits de paro

```

```

        {
            trama03[1]=3;
            temp8 = 0x8E;
            eeprom_write_byte (1,3); // escribe un byte en la EEPROM
        }
        break;

        default:
            temp = 2;
            break;
    }
    if (temp == 1)
    {
        UCSRB = 0x18; //habilita todo de la USART
        tramagene[0]=direcescla;
        tramagene[1]=0x10;
        tramagene[2]=tramaout10[2];
        tramagene[3]=tramaout10[3]+1;
        tramagene[4]=tramaout10[4];
        tramagene[5]=tramaout10[5];
        if (tramaout10[0]!=0x00) //checa que la solicitud no sea un mensaje de
                                                                    difucion
            enviaresp(tramagene,6);
        regcomexec++;
        UCSRC = 0x00;
        UBRRL = 0;
        UBRRL=temp7;
        UCSRC=temp8;
    }
    else
    {
        UCSRC = 0x00;
        UBRRL = 0;
        UBRRL=temp2;
        UCSRC=temp3;
        contope = temp6;
        trama03[0]=temp4;
        trama03[1]=temp5;
        UCSRB = 0x98; //habilita todo de la USART
        tramagene[0]=direcescla; //*****código de excepcion 0x03: el valor
                                                                    que desea escribir en un registro no es valido
        tramagene[1]=0x90;
        tramagene[2]=0x03;
        if (tramaout10[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
}
}
}
/*****
*****

/*****funcion que devuelve el contador de eventos de
comunicacion*****/

void funcion0B(uint8_t tramaout0B[])
{
    uint8_t tramagene[9]={0};
    uint16_t CCEtemp;
    tramagene[0] = direcescla;
    tramagene[1] = 0x0B;
    tramagene[2] = 0x00;
    tramagene[3] = 0x00;
    CCEtemp = CCE;
    tramagene[5] = CCEtemp | tramagene[5];
    CCEtemp = CCEtemp >> 8;
    tramagene[4] = CCEtemp | tramagene[4];

    if (tramaout0B[0]!=0x00)
        enviaresp(tramagene,6);
    regcomexec++;
}
/*****
*****

/*****funcion para enviar la identificacion del
producto*****/

```

```

/*****
/*
/* Para modificar los mensajes de identificacion
/* solo es necesario cambiar el valor de los
/* arreglos:
/*     vendname,
/*     codiprod,
/*     versprod.
/*
/*****
void funcion2B(uint8_t tramaout2B[])
{
uint8_t tramagene[114]={0},cont=0;
uint8_t vendname[]={ "Rodolfo Paz Carreño"};
uint8_t codiprod[]={ "CDECA"};
uint8_t versprod[]={ "V3.0"};
    if (tramaout2B[2]!= 0x0E || tramaout2B[3]!=01 || tramaout2B[4]!= 00)
    {
        tramagene[0]=direcescla; /*******código de excepcion 0x01: se ha intentado
                                ingresar una funcion que el dispositivo no soporta
        tramagene[1]=0xAB;
        tramagene[2]=0x01;

        if (tramaout2B[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
    else
    {
        tramagene[0]= direcescla;
        tramagene[1]= 0x2B;
        tramagene[2]= 0x0E;
        tramagene[3]= 0x01;
        tramagene[4]= 0x01;
        tramagene[5]= 0x00;
        tramagene[6]= 0x00;
        tramagene[7]= 0x03;
        tramagene[8]= 0x00;
        tramagene[9]= strlen(vendname);
        for (cont=0;cont<tramagene[9];cont++)
        {
            tramagene[cont+10]= vendname[cont];
        }
        tramagene[strlen(vendname)+10]= 0x01;
        tramagene[strlen(vendname)+11]= strlen(codiprod);

        for (cont=0;cont<tramagene[strlen(vendname)+11];cont++)
        {
            tramagene[cont+(strlen(vendname)+12)]= codiprod[cont];
        }
        tramagene[strlen(codiprod) + strlen(vendname)+12]= 0x02;
        tramagene[strlen(codiprod) + strlen(vendname)+13]= strlen(versprod);

        for (cont=0;cont<tramagene[strlen(codiprod) + strlen(vendname)+13];cont++)
        {
            tramagene[cont+strlen(codiprod) + strlen(vendname)+14]=
                                versprod[cont];
        }

        if (tramaout2B[0]!=0x00)
            enviaresp(tramagene,cont+strlen(codiprod) + strlen(vendname)+14);
        regcomexec++;
    }
}
/*****
                                *****/

/*****funcion para leer el estado de las
bobinas*****/
void funcion01(uint8_t tramaout01[])
{
uint8_t tramagene[6]={0};//,tempo=0;
    if (tramaout01[2]!=0 || tramaout01[3]>7 || tramaout01[4]!=0 || tramaout01[5]>8)
    {
        tramagene[0]=direcescla; /*******código de excepcion 0x01: se ha intentado

```

```

                                ingresar una funcion que el dispositivo no soporta
tramagene[1]=0x81;
tramagene[2]=0x01;

if (tramaout01[0]!=0x00)
    enviaresp(tramagene,3);
regresexcep++;
}
else
{
    tramagene[0]=direcescla;
    tramagene[1]=0x01;
    tramagene[2]=0x01;
    //tempo = PINA;
    //tempo = tempo << (8 - (tramaout01[3] + tramaout01[5]));
    //tempo = tempo >> (8 - tramaout01[5]);
    //tramagene[3]=tempo;
    tramagene[3] = estadobobinas;
    if (tramaout01[0]!=0x00)
        enviaresp(tramagene,4);
    regcomexec++;
}
}
/*****
*****

/*****funcion para diagnostico de
dispositivo*****/
void funcion08(uint8_t tramaout08[])
{
    uint8_t tramagene[34]={0};
    uint16_t CCEtemp=0x0000;
    if (tramaout08[2]!=0)
    {
        tramagene[0]=direcescla; //*****código de excepcion 0x01: se ha intentado
                                ingresar una funcion que el dispositivo no soporta
        tramagene[1]=0x88;
        tramagene[2]=0x01;

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
    else
    {
        switch(tramaout08[3])
        {
            case 0x00:
            {
                tramagene[0]=tramaout08[0];
                tramagene[1]=tramaout08[1];
                tramagene[2]=tramaout08[2];
                tramagene[3]=tramaout08[3];
                tramagene[4]=tramaout08[4];
                tramagene[5]=tramaout08[5];

                if (tramaout08[0]!=0x00)
                    enviaresp(tramagene,6);
                regcomexec++;
            }
            break;
            case 0x01:
            {
                if (MSE==1)
                {
                    UCSRA = 0;
                    UCSRB = 152;
                    UCSRC = 0x00;
                    UBRRH = valorUBRRH;
                    UBRRL = valorUBRRL;
                    UCSRC = valorUCSRC;
                    CCE=0;
                    regcomuerronea = 0;
                    regresexcep=0;
                    regcomexec=0;
                }
            }
        }
    }
}

```



```

        regmendec=0;
        contmendif=0;
        MSE=0;
        regcomexec++;
    }
    else
    {

        UCSRA = 0;
        UCSRB = 152;
        UCSRC = 0x00;
        UBRRH = valorUBRRH;
        UBRRL = valorUBRRL;
        UCSRC = valorUCSRC;
        tramagene[0]=direcescla;
        tramagene[1]=0x08;
        tramagene[2]=0x00;
        tramagene[3]=0x01;
        tramagene[4]=tramaout08[4];
        tramagene[5]=tramaout08[5];

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,6);
        CCE=0;
        regcomuerronea = 0;
        regresexcep=0;
        regcomexec=0;
        contmendif=0;
        regcomexec++;
    }

}
break;
case 0x04:
{
    MSE=1;
    regcomexec++;
}
break;
case 0x0A:
{
    CCE=0;
    regcomuerronea = 0;
    regresexcep=0;
    regcomexec=0;
    contmendif=0;
    regcomexec++;
    tramagene[0]=direcescla;
    tramagene[1]=tramaout08[1];
    tramagene[2]=tramaout08[2];
    tramagene[3]=tramaout08[3];
    tramagene[4]=tramaout08[4];
    tramagene[5]=tramaout08[5];
    if (tramaout08[0]!=0x00)
        enviaresp(tramagene,6);
}
break;
case 0x0B:
{
    if ((tramaout08[4]==0) && (tramaout08[5]==0))
    {
        tramagene[0]=direcescla;
        tramagene[1]=0x08;
        tramagene[2]=0x00;
        tramagene[3]=0x0B;
        CCEtemp = regmendec;
        tramagene[5] = CCEtemp | tramagene[5];
        CCEtemp = CCEtemp >> 8;
        tramagene[4] = CCEtemp | tramagene[4];

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,6);
        regcomexec++;
    }
    else

```

```

    {
        tramagene[0]=direcescla; //*****código de
                                excepcion 0x01: se ha
                                intentado ingresar una funcion
                                que el dispositivo no soporta

        tramagene[1]=0x88;
        tramagene[2]=0x01;

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
}
break;
case 0x0C:
{
    if ((tramaout08[4]==0) && (tramaout08[5]==0))
    {
        tramagene[0]=direcescla;
        tramagene[1]=0x08;
        tramagene[2]=0x00;
        tramagene[3]=0x0C;
        CCEtemp = regcomuerronea;
        tramagene[5] = CCEtemp | tramagene[5];
        CCEtemp = CCEtemp >> 8;
        tramagene[4] = CCEtemp | tramagene[4];

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,6);
        regcomexec++;
    }
    else
    {
        tramagene[0]=direcescla; //*****código de
                                excepcion 0x01: se ha
                                intentado ingresar una funcion
                                que el dispositivo no soporta

        tramagene[1]=0x88;
        tramagene[2]=0x01;

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
}
break;
case 0x0D:
{
    if ((tramaout08[4]==0) && (tramaout08[5]==0))
    {
        tramagene[0]=direcescla;
        tramagene[1]=0x08;
        tramagene[2]=0x00;
        tramagene[3]=0x0D;
        CCEtemp = regresexcep;
        tramagene[5] = CCEtemp | tramagene[5];
        CCEtemp = CCEtemp >> 8;
        tramagene[4] = CCEtemp | tramagene[4];

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,6);
        regcomexec++;
    }
    else
    {
        tramagene[0]=direcescla; //*****código de
                                excepcion 0x01: se ha
                                intentado ingresar una funcion
                                que el dispositivo no soporta

        tramagene[1]=0x88;
        tramagene[2]=0x01;

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
}
}

```

```

}
break;
case 0x0E:
{
    if ((tramaout08[4]==0) && (tramaout08[5]==0))
    {
        tramagene[0]=direcescla;
        tramagene[1]=0x08;
        tramagene[2]=0x00;
        tramagene[3]=0x0E;
        CCEtemp = regcomexec;
        tramagene[5] = CCEtemp | tramagene[5];
        CCEtemp = CCEtemp >> 8;
        tramagene[4] = CCEtemp | tramagene[4];

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,6);
        regcomexec++;
    }
    else
    {
        tramagene[0]=direcescla; //*****código de
                                //excepcion 0x01: se ha
                                //intentado ingresar una funcion
                                //que el dispositivo no soporta

        tramagene[1]=0x88;
        tramagene[2]=0x01;

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
}
break;
case 0x0F:
{
    if ((tramaout08[4]==0) && (tramaout08[5]==0))
    {
        tramagene[0]=direcescla;
        tramagene[1]=0x08;
        tramagene[2]=0x00;
        tramagene[3]=0x0F;
        CCEtemp = contmendif;
        tramagene[5] = CCEtemp | tramagene[5];
        CCEtemp = CCEtemp >> 8;
        tramagene[4] = CCEtemp | tramagene[4];

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,6);
        regcomexec++;
    }
    else
    {
        tramagene[0]=direcescla; //*****código de
                                //excepcion 0x01: se ha
                                //intentado ingresar una funcion
                                //que el dispositivo no soporta

        tramagene[1]=0x88;
        tramagene[2]=0x01;

        if (tramaout08[0]!=0x00)
            enviaresp(tramagene,3);
        regresexcep++;
    }
}
break;
default:
{
    tramagene[0]=direcescla; //*****código de excepcion
                                //0x02: se ha intentado leer un registro
                                //con direccion ilegal

    tramagene[1]=0x88;
    tramagene[2]=0x01;

    if (tramaout08[0]!=0x00)
        enviaresp(tramagene,3);
}

```

```

        regresexcep++;
    }
    break;
}
}
}
/*****
*****
*****/

/*****funcion para escribir los valores de las
bobinas*****/
void funcion0F(uint8_t tramaout05[])//funcion para escribir un solo valor de coil
{
uint8_t tramagene[9]={0};
if ((tramaout05[3] > 0x07) && (tramaout05[5] > 0x08)) //verifica que la solicitud no pida mas de los 3
registros activos
{
    tramagene[0]=direcescla; //*****código de excepcion 0x02: se ha intentado
    leer un registro con direccion ilegal
    tramagene[1]=0x8F;
    tramagene[2]=0x02;

    if (tramaout05[0]!=0x00
        enviaresp(tramagene,3);
    regresexcep++;
}
else
{
    PORTC = tramaout05[8];
    estadobobinas = tramaout05[8];
    tramagene[0]=direcescla;
    tramagene[1]=0x0F;
    tramagene[2]=tramaout05[2];
    tramagene[3]=tramaout05[3];
    tramagene[4]=tramaout05[7];
    tramagene[5]=tramaout05[8];

    if (tramaout05[0]!=0x00
        enviaresp(tramagene,6);

    regcomexec++;
}
}
/*****
*****
*****/

/*****
*****
*****/

void configurar_puertos()
{
    DDRA = 0x00; //puerto A como entrada
    DDRC = 0xFF; //puerto C como salida
    DDRB = 0xFF; //puerto B, pin PB4 como salida
    PORTB = 0x00; //Habilita la recepción de datos en modo RS-485
}
/*****
*****
*****/

/*****
*****
*****/

void configurar_USART()
{
/*****configuracion de la USART 0 y direccion del
dispositivo*****/
UCSRA = 0;
UCSRB = 152;
/*****
/*Los valores con los cuales se */
/*configura la USART 0, se encuentran */
/*guardados en la EEPROM: */
/* Paridad y velocidad, */
/* asi como la dirección de dispositivo. */
/* */
/*Los cuales son los registros mantenidos del */
/*dispositivo. */
/* */

```

```

/*
/*IMPORTANTE:
/*
/* los valores que se cargan a los
/* registros UCSRC, UBRRH, UBRRL y a*/
/* la variable contope son valores
/* para un cristal de 7.3728MHz.
/*
/*****
veeprom = eeprom_read_byte(2);
direcescla = veeprom;
veeprom = eeprom_read_byte(0);
switch(veeprom)
{
    case 1:
    {
        trama03[0]=1;
        UCSRC = 0x00;
        UBRRH = 0;
        UBRRL = 191; //velocidad de 2400 BPS
        contope = 462;

    }
    break;
    case 2:
    {
        trama03[0]=2;
        UCSRC = 0x00;
        UBRRH = 0;
        UBRRL = 95; //velocidad de 4800 BPS
        contope = 231;

    }
    break;
    case 3:
    {
        trama03[0]=3;
        UCSRC = 0x00;
        UBRRH = 0;
        UBRRL = 47; //velocidad de 9600 BPS
        contope = 116;

    }
    break;
    case 4:
    {
        trama03[0]=4;
        UCSRC = 0x00;
        UBRRH = 0;
        UBRRL = 23; //velocidad de 19200 BPS
        contope = 58;

    }
    break;
    case 5:
    {
        trama03[0]=5;
        UCSRC = 0x00;
        UBRRH = 0;
        UBRRL = 11; //velocidad de 38400 BPS
        contope = 29;

    }
    break;
    case 6:
    {
        trama03[0]=6;
        UCSRC = 0x00;
        UBRRH = 0;
        UBRRL = 7; //velocidad de 57600 BPS
        contope = 20;

    }
    break;
    case 7:
    {
        trama03[0]=7;
        UCSRC = 0x00;

```

```

        UBRRH = 0;
        UBRRL = 3; //velocidad de 115200 BPS
        contope = 10;
    }
    break;
}
}
valorUBRRH = UBRRH;
valorUBRRL = UBRRL;
veeprom = eeprom_read_byte(1);
switch(veeprom)
{
    case 1://paridad par (even), un bit de paro
    {
        trama03[1]=1;
        UCSRC = 0xA6;
    }
    break;
    case 2://paridad impar, un bit de paro
    {
        trama03[1]=2;
        UCSRC = 0xB6;
    }
    break;
    case 3://sin paridad, dos bits de paro
    {
        trama03[1]=3;
        UCSRC = 0x8E;
    }
    break;
}
}
valorUCSRC = UCSRC;
}
/*****
*****
*****/
/*****
*****/
void configurar_temporizador()
{
    TIMSK = 0x40; /*habilita la interrupcion de sobreflujo del timer couter 2*/
}
/*****
*****/
/*****
*****/
void atender_trama()
{
    uint8_t tramagene[4]={0};
    switch(cadpeti[1])
    {
        case 0x0F://Escribir multiples valores de bobinas
        {
            funcion0F(cadpeti);
        }
        break;
        case 0x08://Diagnostico del dispositivo
        {
            funcion08(cadpeti);
        }
        break;
        case 0x10://Escribir multiples registros mantenidos
        {
            funcion10(cadpeti);
        }
        break;
        case 0x0B://Devolver el contador de eventos de comunicacion
        {
            funcion0B(cadpeti);
        }
        break;
        case 0x2B://Leer datos de identificacion

```

```

        {
            funcion2B(cadpeti);
        }
        break;
        case 0x01://funcion para leer el estado ON/OFF de las salidas que controlan el
            encendido-apagado
        {
            funcion01(cadpeti);
        }
        break;
        default:
        {
            tramagene[0]=direcescla; //*****código de excepcion 0x02: se ha
                intentado leer un registro con direccion ilegal
            tramagene[1]=cadpeti[1]+0x80;
            tramagene[2]=0x01;
            if (cadpeti[0]!=0x00)
                enviaresp(tramagene,3);
            regresexcep++;
        }
        break;
    }

}
/*****
*****/

```


Apéndice B.

Manual de usuario del CDECA serial

El CDECA serial es un dispositivo que permite controlar de forma remota el encendido o apagado de cargas eléctricas o bancos de capacitores. Esto con el fin de controlar o reducir el consumo de energía eléctrica, o corrección del factor de potencia, y con ello disminuir el pago del recibo de energía eléctrica. Es controlado remotamente por el controlador de demanda máxima (dispositivo maestro) ya que este dispositivo en una red MODBUS es un dispositivo esclavo.

Tiene la característica de poseer dos puestos de comunicaciones, el puerto serial con niveles TTL y el puerto RS-485. El puerto serial posee tres señales que son:

- Rx, es la señal que sirve para recibir los datos enviados por la estación maestra.
- Tx, tiene la función de enviar los datos de la respuesta creada por el CDECA.
- GND, señal de referencia.

El puerto serial TTL puede ser utilizado para interconectarse con radio-modems a nivel TTL, con este tipo de interfaz funcionan algunos de los fabricados por la firma MAXSTREAM. También se dejó a este nivel ya que se cuenta con un módulo conversor de señales USB, el cual recibe con este nivel y tiene interfaces USB y RS-232.

El puerto RS-485 usa 5 señales, 2 para transmisión, 2 para recepción y una para tierra, ya que opera en modo diferencial. Esta característica da una mejor inmunidad al ruido eléctrico, haciendo posible la transmisión de datos a una mayor distancia.

Las salidas del CDECA son 8 relevadores, los cuales son capaces de manejar hasta 8 cargas eléctricas con un consumo máximo de 10 A, ya sea de CD o CA. Por cuestiones de espacio se diseñó en 2 bloques: el primero controla 4 cargas y posee el puerto de comunicaciones (en si es el CDECA); el segundo solo posee los 4 relevadores de salida restantes y es controlado por el primero, además es desmontable.

El CDECA serial tiene como chasis de protección un gabinete color gris claro con dimensiones: 7.5 x 13.5 x 5 cm (ancho x largo x alto). En la figura B.1 se muestra físicamente el CDECA serial montado sobre la base del gabinete, en la figura B.2 se muestra la parte posterior donde se observa el microcontrolador ATmega32.



Figura B.1. CDECA (vista superior).

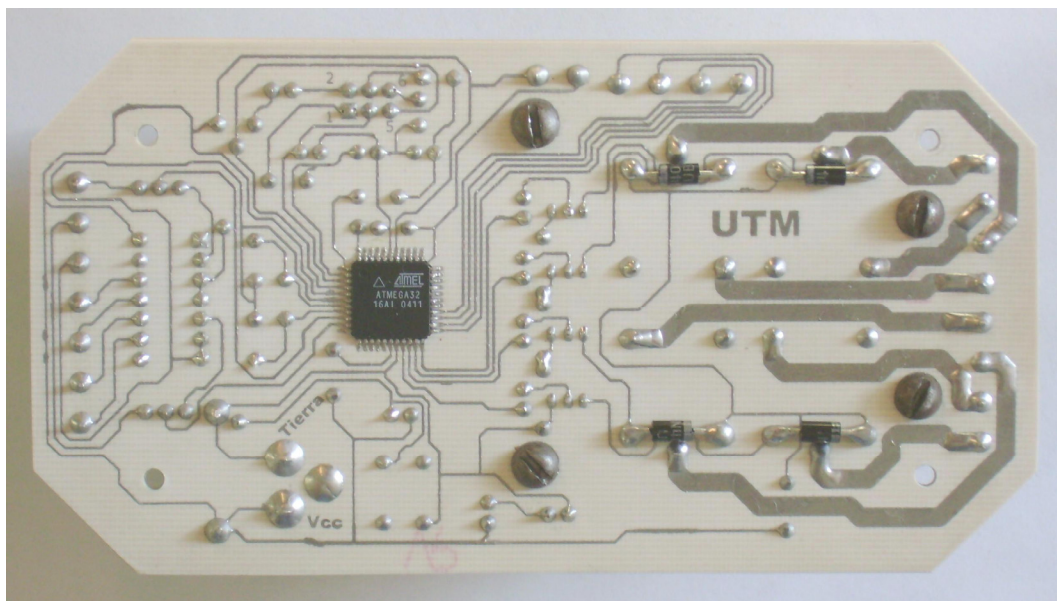


Figura B.2. CDECA (vista posterior).

B.1 Modo de funcionamiento.

El funcionamiento de este dispositivo es muy sencillo. Lo primero que hay que hacer es encender el dispositivo, para lo cual se necesita conectar una fuente que proporcione un voltaje entre 4.75V y 5.25V, con capacidad de corriente de 1A. La alimentación se conecta en el jack denominado "Alimentación" de la figura B.1. Si todo esta bien, se deberá encender el "LED indicador".

B.1.1 Uso de los puertos de comunicación.

En la figura B.1 se muestra la localización del puerto serial TTL y el RS-485, los cuales se encuentra en la parte izquierda de la figura. Para seleccionar el puerto a usar, se usan 2 puentes (*jumper*), marcados como "jumper 1 y jumper 2".

Para utilizar el puerto serial TTL, se deberán puentear las dos primeras terminales (de izquierda a derecha de la figura 1) de los jumpers 1 y 2. La configuración que tienen los puentes de la figura 1, corresponden a esta opción.

Las terminales que se usan en comunicaciones seriales TTL son Rx, Tx y GND, las cuales deberán conectarse hacia el dispositivo maestro como se muestra en la figura B.3. Estas señales, se conectan al ComUSB y de ahí a la computadora.

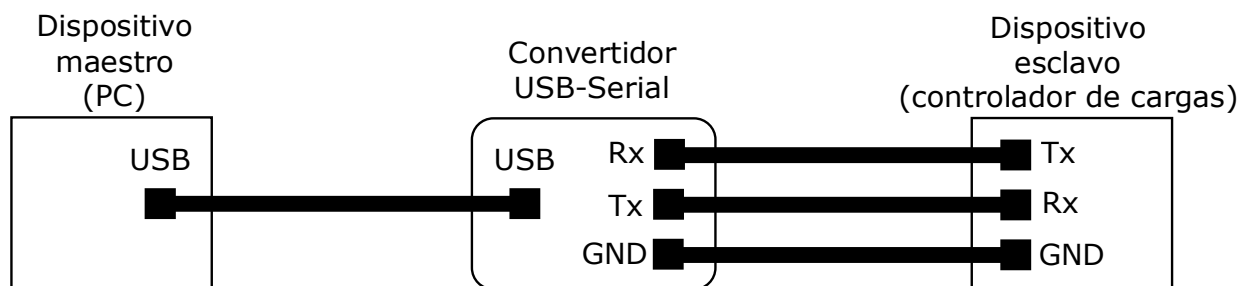


Figura B.3. Conexión del CDECA utilizada para hacer uso del puerto serial.

Para utilizar el puerto RS-485 del CDECA, es necesario modificar la posición de los jumpers 1 y 2. Para ello se puentean las terminales que se encuentran a la derecha de los jumpers 1 y 2.

Cuando se tiene esta configuración, las terminales que se usan para la comunicación son Rx+, Rx-, Tx+ , Tx- y GND. Como el CDECA transmite en modo diferencial, se usa un par de terminales para transmitir y otro para recibir, aunque se puede configurar para usar solo un par de terminales.

Son 2 los modos de operación del puerto RS-485, de dos líneas o de cuatro líneas. La figura B.4 muestra cómo se interconectan los dispositivos en configuración de 2 líneas, y la figura B.5, para la conexión de 4 líneas.

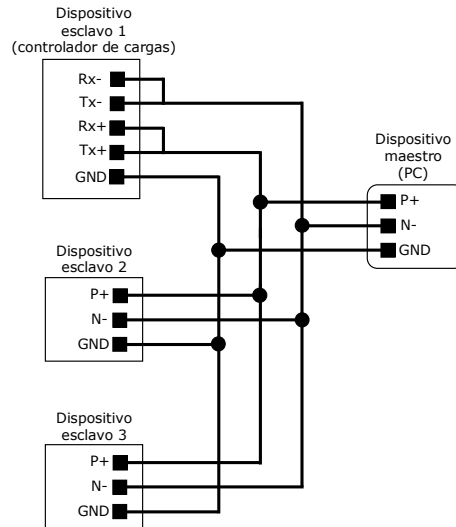


Figura B.4. Conexión del CDECA utilizada para hacer uso del puerto RS-485 de dos líneas.

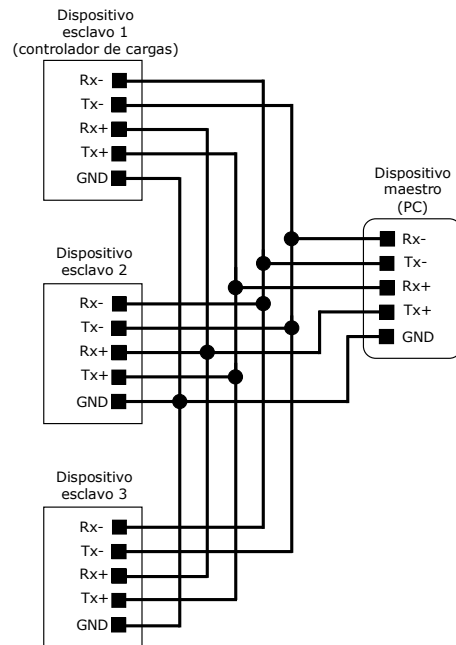


Figura B.5. Conexión del CDECA utilizada para hacer uso del puerto RS-485 de cuatro líneas.

B.1.2 Conexión de las cargas eléctricas.

Los relevadores de salida se encuentran en la posición de normalmente abierto, cuando se activan pasan de abierto a cerrado. La conexión de una carga eléctrica, alimentada con corriente alterna, se realiza como se muestra en la figura B.6. Las salidas en el CDECA están marcadas como #1, #2, #3 y #4 de la figura B.1. Cuando se manda la orden de cierre del relevador, éste cierra sus contactos y permite que fluya la corriente por la carga eléctrica.

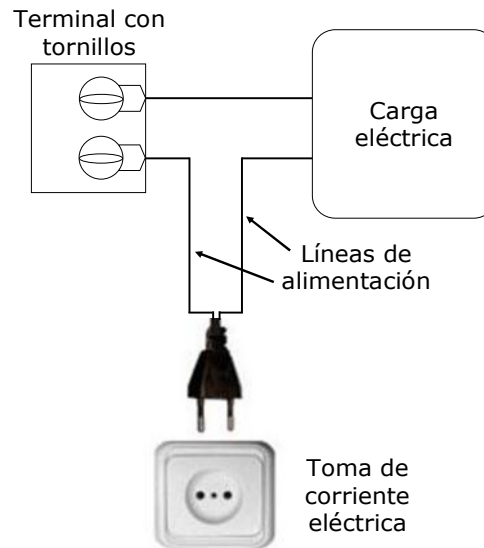


Figura B.6. Conexión de la terminal con tornillos y la carga eléctrica a controlar.

El módulo desmontable que controla las 4 cargas restantes se muestra en la figura B.7. Del CDECA recibe las 4 señales de activación con niveles TTL (de L1 a L4 en la figura B.1), la alimentación L6 (VCC=+5V) y tierra L5 (GND). Las cargas se conectan a los bornes marcados como #5, #6, #7 y #8 de la figura B.7.

B.1.3 Puerto de programación.

El puerto de programación del microcontrolador consta de 6 terminales tipo FCN6, como se muestra en la figura B.8. Fue implementado para poder realizar actualizaciones del firmware empotrado en el microcontrolador del dispositivo. Sus pines están distribuidos de tal manera que se pueden utilizar los programadores comerciales de la firma ATMEL como son: AVRISP y AVRISPMkII.

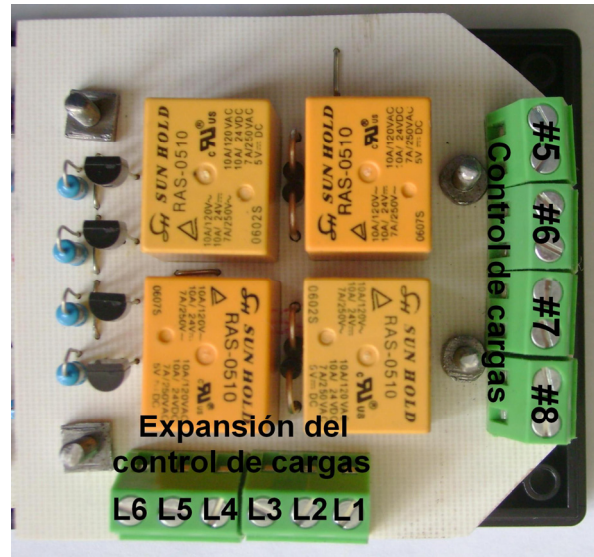


Figura B.7. Terminales móviles del CDECA.

En la figura B.8 se observa la distribución de las señales y el nombre de cada una de ellas. Para utilizar correctamente este puerto, es necesario saber el tipo de programador que se va a emplear. Esto es importante ya que el AVRISP necesita una fuente de alimentación externa para funcionar, caso contrario para el AVRISPMkII, ya que éste puede proporcionar alimentación al microcontrolador a programar.

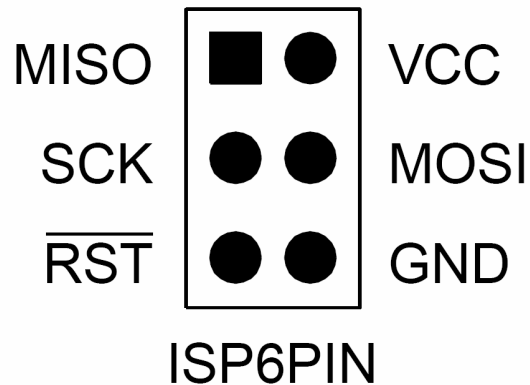


Figura B.8. Distribución de señales en el puerto de programación.

Para seleccionar el tipo de programador usado, se utiliza el jumper 3 de 2 terminales (figura B.1). Si el programador a utilizar es el AVRISP, es necesario que el jumper se encuentre colocado como se muestra en la figura B.1. En caso de que el programador sea el AVRISPMkII, el jumper 3 deberá ser retirado.

En la parte posterior del circuito impreso del CDECA, se encuentran enumerados los pines de éste, con el fin de que sirvan de guía al usuario y poder hacer un buen uso del puerto.

B.2 Diagramas del CDECA.

Con la finalidad de que en el futuro se requiera reparar el CDECA, o se necesite fabricar, en las siguientes secciones se muestran todos sus diagramas, así como la lista de componentes.

En la figura B.9 se muestra el diagrama esquemático del CDECA. En la figura B.10 y B.11 los diagramas de montaje del CDECA y su módulo desmontable respectivamente. En las figuras B.12 y B.13 el diagrama de pistas (layout) del CDECA y su módulo desmontable. En la tabla B.1 y B.2 se tiene el listado de los componentes del CDECA así como el del módulo desmontable.

B.2.1 Diagrama esquemático.

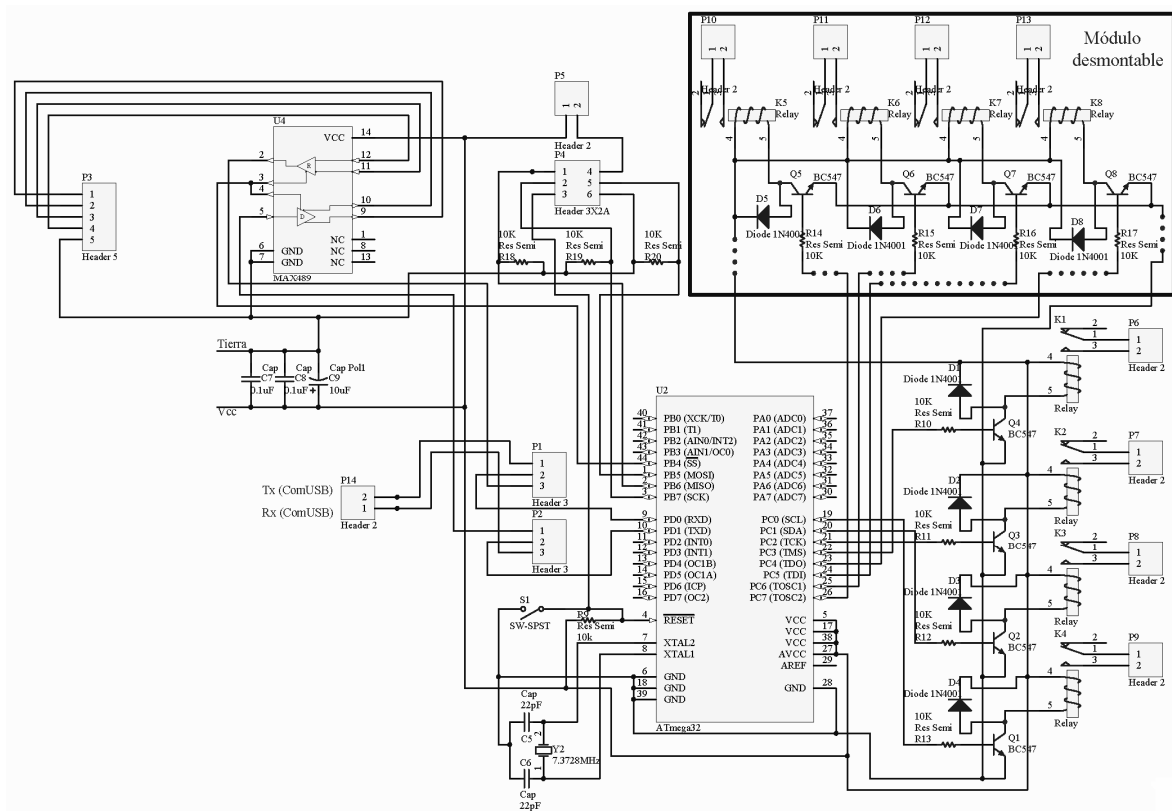


Figura B.9. Diagrama esquemático del CDECA.

B.2.2 Diagramas de montaje.

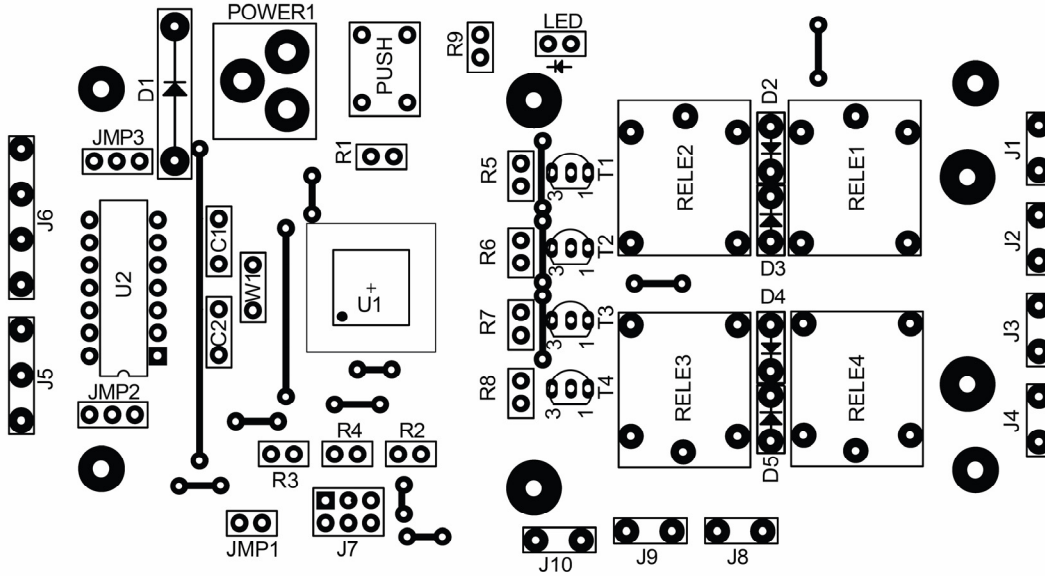


Figura B.10. Diagrama de montaje utilizado en la placa base del CDECA.

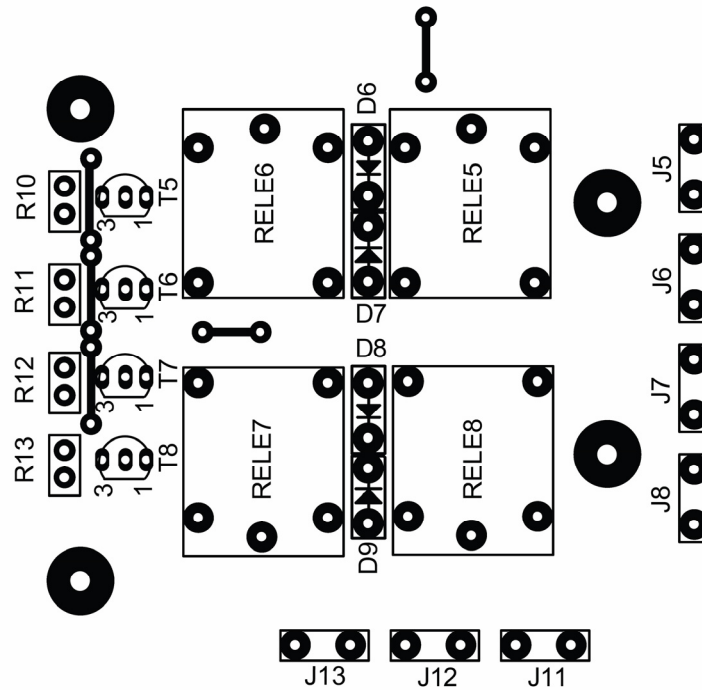


Figura B.11. Diagrama de montaje utilizado en el módulo desmontable.

B.2.2.1 Lista de componentes.

Tabla B.1. Lista de componentes utilizados en la placa base del CDECA.

Cantidad	Referencia	Descripción	Modelo	Precio por unidad	Proveedor
2	C1, C2.	Capacitor cerámico de 22 pF.	C22-500	\$1.739	AG Electrónica S.A de C.V.
1	D1.	Diodo rectificador de 3 AMP, propósito general.	MR500	\$1.740	AG Electrónica S.A de C.V.
4	D2, D3, D4, D5.	Diodo rectificador de 1 AMP, propósito general.	1N4001	\$0.434	AG Electrónica S.A de C.V.
6	J1, J2, J3, J4, J8, J9.	Terminal grande con 2 tornillos, para circuito impreso.	TRTG-02	\$3.480	AG Electrónica S.A de C.V.
1	J10.	Terminal chica con 2 tornillos, para circuito impreso.	TRT-02	\$2.610	AG Electrónica S.A de C.V.
1	J5.	Terminal grande con 3 tornillos, para circuito impreso.	TRTG-03	\$5.217	AG Electrónica S.A de C.V.
2	J6.	Terminal grande con 2 tornillos, para circuito impreso.	TRTG-02	\$3.480	AG Electrónica S.A de C.V.
1	JMP1.	Tira de 2 pines.	F36-S	\$2.610	AG Electrónica S.A de C.V.
2	JMP2, JMP3.	Tira de 3 pines.	F36-S	\$2.610	AG Electrónica S.A de C.V.
1	LED.	Led económico 5mm, super brillante.	E5/ROJ-SUPER	\$1.740	AG Electrónica S.A de C.V.
1	POWER1.	Jack para		\$5.000	AG

		alimentación.			Electrónica S.A de C.V.
1	PUSH.	Micro switch push, 4 terminales.	AU-101	\$1.740	AG Electrónica S.A de C.V.
8	R1, R2, R3, R4, R5, R6, R7, R8	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R10K ¼	\$0.348	AG Electrónica S.A de C.V.
1	R9.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R2.2K ¼	\$0.348	AG Electrónica S.A de C.V.
4	RELE1, RELE2, RELE3, RELE4.	Relevador compacto, 1P, 2T, 10amp.	RAS-0510	\$12.000	STEREN S.A de C.V
4	T1, T2, T3, T4.	Transistor de pequeña señal, npn.	BC337-B	\$2.610	AG Electrónica S.A de C.V.
1	U1.	Microcontrolador ATmega32, encapsulado TQFP.	ATmega 32	\$60.000	AG Electrónica S.A de C.V.
1	U2.	Convertidor RS-485 a TTL	MAX489	\$29.570	AG Electrónica S.A de C.V.
1	W1.	Cristal de cuarzo, 7.3728 MHZ.	C7.3728	\$6.956	AG Electrónica S.A de C.V.
1		Placa fenólica 1 cara.	PC-10X15	\$15.000	AG Electrónica S.A de C.V.
1		Gabinete de plástico.	GP-11	\$61.000	STEREN S.A de C.V

Tabla B.2. Lista de componentes utilizados en el módulo desmontable.

Cantidad	Referencia	Descripción	Modelo	Precio por unidad	Proveedor
4	D6, D7, D8, D9.	Diodo rectificador de 1 AMP,	1N4001	\$0.434	AG Electrónica S.A de C.V.

		propósito general.			
4	J5, J6, J7, J8, J11, J12, J13.	Terminal grande con 2 tornillos, para circuito impreso.	TRTG-02	\$3.480	AG Electrónica S.A de C.V.
8	R10, R11, R12, R13.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R10K ¼	\$0.348	AG Electrónica S.A de C.V.
4	RELE5, RELE6, RELE7, RELE7.	Relevador compacto, 1P, 2T, 10amp.	RAS-0510	\$12.000	STEREN S.A de C.V
4	T5, T6, T7, T8.	Transistor de pequeña señal, npn.	BC337-B	\$2.610	AG Electrónica S.A de C.V.

B.2.3 Diagramas de circuito impreso.

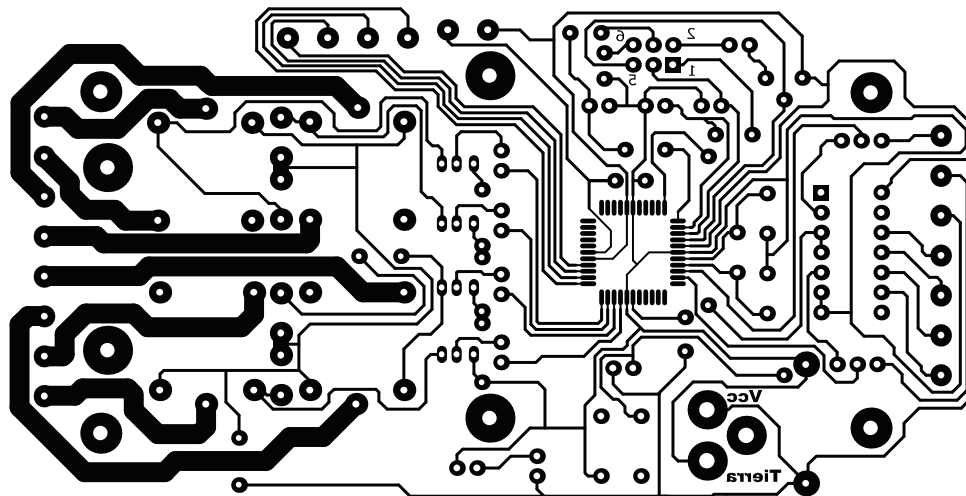


Figura B.12. Circuito impreso utilizado en la placa base del CDECA.

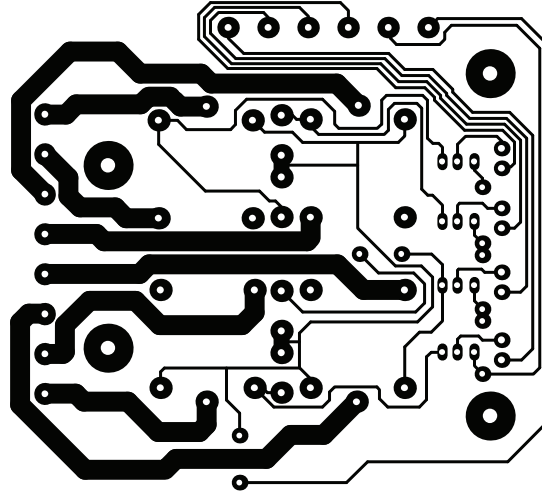


Figura B.13. Circuito impreso utilizado en el módulo desmontable.

Apéndice C. Manual de usuario del ComUSB

El controlador de comunicaciones seriales por USB (ComUSB) es un dispositivo cuyo objetivo es comunicar un microcontrolador u otro circuito integrado con salida serial TTL con una PC. Es la mejor opción para implementar un puerto USB a un proyecto sin tener que preocuparse por conocer el funcionamiento del protocolo USB.

La característica principal de este convertidor es que puede escoger el tipo de señal de salida, de los tres disponibles:

1. **Salidas en niveles TTL.** Con esta opción es posible conectar directamente el convertidor a circuitos integrados que manejen niveles TTL en su comunicación serial.
2. **Salidas en niveles RS-232.** Esta interfaz cumple completamente con los voltajes especificados por la norma EIA/TIA-232.
3. **Señales de control para manejar niveles RS-485.** Con esta opción se puede manejar el circuito integrado SP491 (o similares), el cual es un convertidor de nivel TTL a RS-485.

El ComUSB tiene como chasis de protección un gabinete color gris claro con dimensiones: 5 x 9.5 x 3.5 cm (ancho x largo x alto).

En la figura C.1 se muestra físicamente el convertidor ComUSB montado sobre la base del gabinete (lado de componentes). En ella se observan el conector de puerto USB, el RS-232, así como las terminales para el puerto RS-485 y las salidas a nivel TTL. La alimentación se le suministra externamente a través del conector marcado como alimentación.

En la figura C.2 se muestra la vista posterior del ComUSB, que corresponde a las pistas de conexión de todos los componentes y el manejador de USB, el CI FT232BM.



Figura C.1. Vista superior del ComUSB.

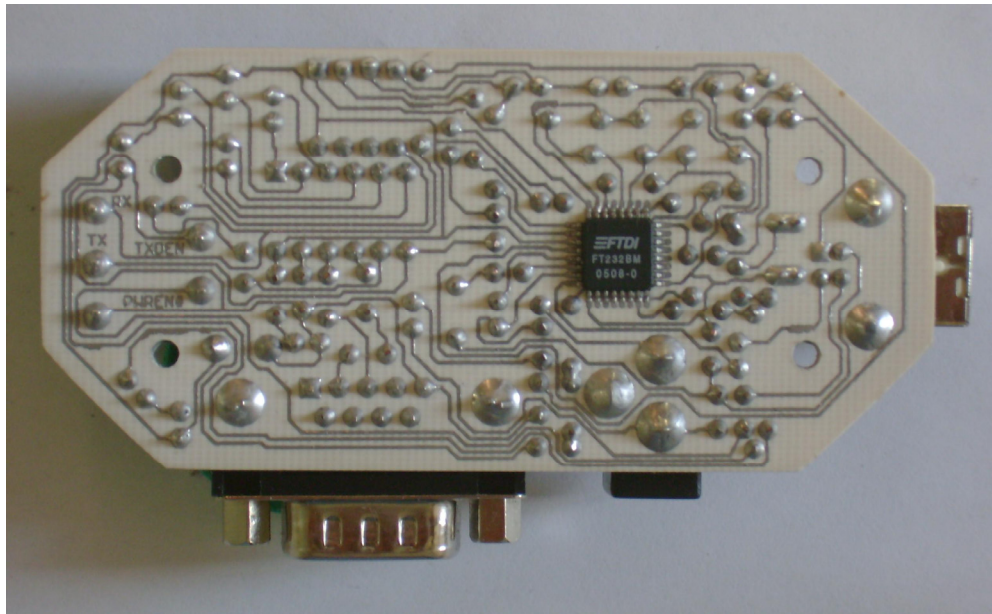


Figura C.2. Vista posterior del ComUSB (lado soldadura).

C.1 Modo de funcionamiento.

El estándar USB especifica que puede alimentar dispositivos conectados a este puerto, siempre y cuando demanden una corriente menor a 500mA con un voltaje de 5V. El ComUSB utiliza una fuente cuyo voltaje debe ser entre 4.4 y 5.25V.

En la figura C.3 se muestra la forma en la cual se conecta el ComUSB a la PC. Para realizarla se requieren de cuatro elementos (enumerados en la figura), que son:

1. Una PC que cuente con puertos USB disponibles.
2. El ComUSB.
3. Cable USB con una terminal macho tipo "C".
4. Una fuente de alimentación que proporcione entre 4.4 y 5.25 volts.

El cable usado para conectar el puerto USB a la PC, deberá contar con una terminal macho tipo "C", como la descrita en el capítulo 1. Al conectar a la PC el ComUSB, si es la primera vez que se hace, será necesario instalar los drivers para el sistema operativo usado (Windows o Linux).



Figura C.3. Conexión del ComUSB a la PC.

Los drivers utilizados para que el sistema operativo reconozca al ComUSB, se encuentra disponibles en la página del fabricante del controlador de USB usado [URL11]. En esta página están disponibles todos drivers de la familia de productos que maneja, por tal motivo es necesario asegurarse de que sean para el FT232BM.

Hay que aclarar que solamente la primera vez que se conecte ComUSB a alguna computadora, será necesario instalar los drivers. Para conexiones futuras, el sistema operativo de la PC lo reconoce en forma automática.

Para verificar que realmente el sistema operativo de la PC ha detectado el convertidor, simplemente hay que buscar en el administrador de dispositivos en el apartado "Puerto de comunicaciones (COM & LPT)". Esto servirá también para saber el número de COM que le ha sido asignado al convertidor. Esto solamente es válido para el sistema operativo Windows XP®.

Una vez que se han realizado los pasos anteriores, el módulo esta listo para ser utilizado. El puerto o salida a usar se configura moviendo el jumper que ComUSB tiene para este propósito.

C.1.1 Salidas en niveles TTL.

Este tipo de salida es conveniente para utilizarla con circuitos integrados. Para ello, es necesario verificar que el jumper marcado en la figura C.1 no se encuentre cortocircuitado. Las salidas correspondientes son Rx, Tx y GND, siendo Rx la señal de recepción, Tx la señal de transmisión y GND señal común; las tres pertenecientes al ComUSB.

C.1.2 Salidas en niveles RS-232.

Estas señales servirán para otorgar conectividad a dispositivos que requieran utilizar los niveles de voltajes marcados en la norma EIA/TIA-232. Cuando se quiera utilizar esta modalidad, el jumper deberá ponerse en corto circuito, ya que de no hacerlo el convertidor solo podrá transmitir datos.

C.1.3 Señales de control para manejar niveles RS-485.

El manejador de USB (el circuito integrado FT232BM), no maneja salidas con voltajes marcados en la norma EIA/TIA-485, pero proporciona señales de control para hacerlo. Para ello se utilizan las señales TXDEN y PWREN# marcadas en la figura C.1. Éstas tienen la finalidad de controlar circuitos integrados convertidores de nivel TTL a RS-485 como el MAX 489.

Para que se usen las señales TXDEN y PWREN#, en conjunto con las señales Rx, Tx y GND; es necesario que el jumper no esté cortocircuitado.

En la figura C.4 se muestra la forma en cómo se conectan las señales al circuito integrado MAX489. Para mayor información se recomienda leer la hoja de datos de los circuitos integrados FT232BM y MAX489, localizadas en el sitio de Internet de cada fabricante [URL11, URL19].

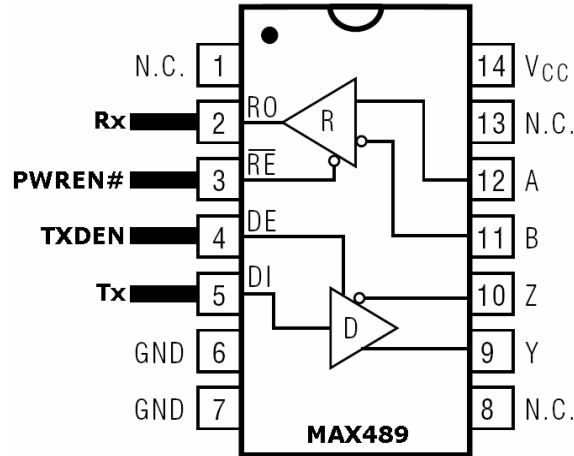


Figura C.4. Conexión utilizada para el manejo de niveles RS-485.

Con el fin de dejar una referencia a futuro para que se puedan fabricar más módulos ComUSB, o en caso de que se requiera realizar alguna reparación, en las siguientes secciones se dan todos los diagramas requeridos para ello.

En la figura C.5 se muestra del diagrama esquemático del ComUSB, en la figura C.6 el diagrama de montaje por el lado de los componentes del ComUSB. En la figura C.7 el diagrama de pistas (Layout) del circuito impreso, utilizado en la placa del convertidor ComUSB. El circuito impreso es de una sola cara. En la tabla C.1 se muestra la lista de componentes. Con precios del mes de Octubre del 2007, el costo de los componentes del ComUSB es de \$250 pesos, aproximadamente.

C.2 Diagrama esquemático.

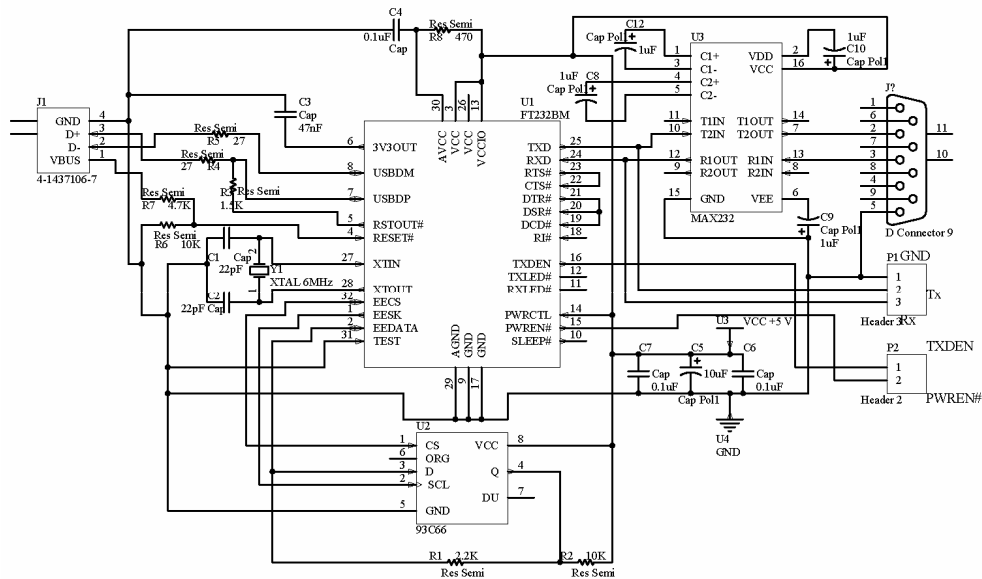


Figura C.5. Diagrama esquemático del ComUSB.

C.3 Diagrama de montaje.

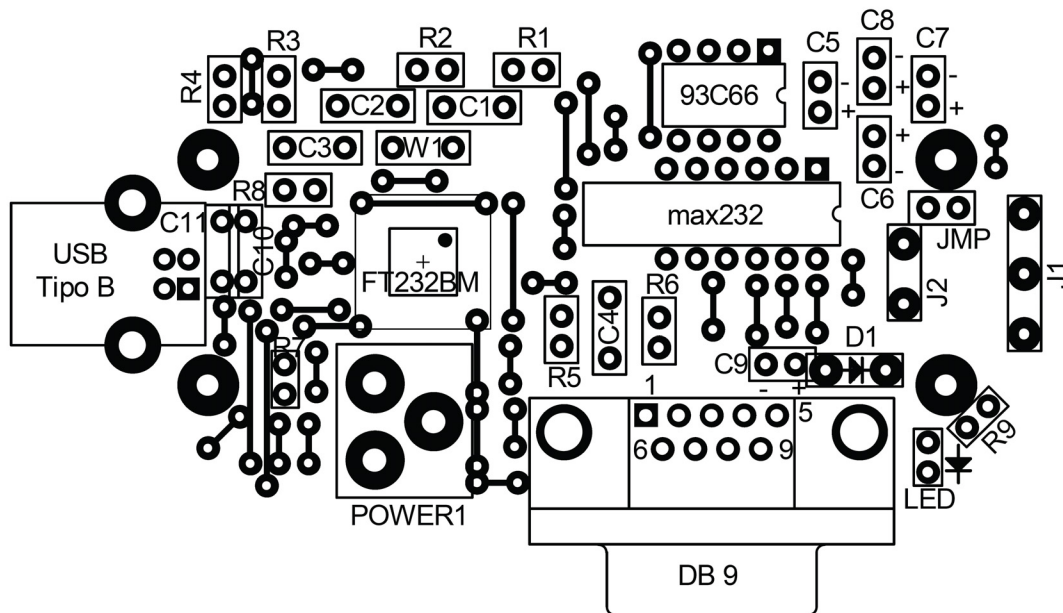


Figura C.6. Diagrama de montaje utilizado en la placa del ComUSB.

C.3.1 Lista de componentes.

Tabla C.1. Lista de componentes utilizados en la placa del ComUSB.

Cantidad	Referencia	Descripción	Modelo	Precio por unidad	Proveedor
2	C1, C2.	Capacitor cerámico de 22 pF.	C22-500	\$1.739	AG Electrónica S.A de C.V.
3	C3, C10, C11.	Capacitor cerámico de 0.1 μ F.	C.1-50	\$1.739	AG Electrónica S.A de C.V.
1	C4.	Capacitor cerámico de 33 pF.	C33-500	\$1.740	AG Electrónica S.A de C.V.
4	C5, C6, C7, C8.	Capacitor electrolítico de 1 μ F.	E1-63R	\$1.304	AG Electrónica S.A de C.V.
1	C9.	Capacitor electrolítico de 10 μ F.	E10-100R	\$0.870	AG Electrónica S.A de C.V.
1	D1	Diodo rectificador de 1	1N4001	\$0.434	AG Electrónica

		amp, propósito general.			S.A de C.V.
1	DB9	Conector DB9 macho para circuito impreso.	500-120	\$6.956	AG Electrónica S.A de C.V.
1	FT232BM	Convertidor USB a serial.	FT232B M	\$81.392	AG Electrónica S.A de C.V.
1	JMP	Tira de 2 pines.	F36-S	\$2.610	AG Electrónica S.A de C.V.
1	J1	Terminal chica con 3 tornillos para circuito impreso.	TRT-03	\$3.478	AG Electrónica S.A de C.V.
1	J2	Terminal chica con 2 tornillos para circuito impreso.	TRT-02	\$2.610	AG Electrónica S.A de C.V.
1	LED	Led económico 5mm, super brillante.	E5/ROJ-SUPER	\$1.740	AG Electrónica S.A de C.V.
1	MAX232	Interfase de alta velocidad.	MAX232	\$6.088	AG Electrónica S.A de C.V.
1	POWER1	Jack para alimentación.		\$5.000	AG Electrónica S.A de C.V.
2	R1, R3.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R10K ¼	\$0.348	AG Electrónica S.A de C.V.
1	R2.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R2.2K ¼	\$0.348	AG Electrónica S.A de C.V.
1	R4.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R4.7K ¼	\$0.348	AG Electrónica S.A de C.V.
1	R5.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R1.5K ¼	\$0.348	AG Electrónica S.A de C.V.

2	R6, R7.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R22 ¼	\$0.348	AG Electrónica S.A de C.V.
1	R8.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R470 ¼	\$0.348	AG Electrónica S.A de C.V.
1	R9.	Resistencia de carbón, 1/4 de watt, 5% tolerancia.	R220 ¼	\$0.348	AG Electrónica S.A de C.V.
1	USB tipo B.	Conector USB tipo B.	USBB/F90_DIP	\$6.956	AG Electrónica S.A de C.V.
1	W1.	Cristal de cuarzo, 6MHZ.	C6.0	\$6.956	AG Electrónica S.A de C.V.
1	93C66.	Memoria EEPROM serial.	AT93C66	\$5.218	AG Electrónica S.A de C.V.
1		Placa fenólica 1 cara.	PC-5x10	\$5.220	AG Electrónica S.A de C.V.
1		Gabinete de plástico.	GP-09	\$50.000	STEREN S.A de C.V

C.4 Diagrama de circuito impreso.

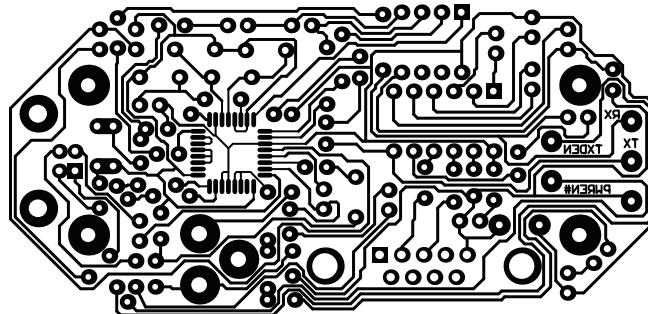


Figura C.7. Circuito impreso utilizado en la placa del ComUSB.

Apéndice D. Manual de usuario del SPCC

El software para el controlador de cargas (SPCC) es una herramienta creada para verificar el funcionamiento del CDECA. Para su elaboración se escogió el lenguaje de programación MATLAB[®] por ser fácil de utilizar y contener herramientas que permiten el uso de los puertos de comunicación con los que cuenta una PC, herramientas muy poco usadas por existir programas especializados en el uso de puertos de comunicación.

La función que realiza el SPCC es la de enviar peticiones o comandos al CDECA y recibir la cadena de respuesta que este genera para mostrarla al usuario de forma que esta sea entendible.

Para el envío de la información se utiliza algún puerto serial que se encuentre disponible en la máquina donde se este ejecutando el SPCC.

D.1 Modo de funcionamiento.

El funcionamiento del SPCC es muy fácil e intuitivo. Para poder ejecutar el SPCC en una PC es necesario tener instalado el entorno de desarrollo del lenguaje MATLAB[®].

Debido a que la instalación total del entorno de desarrollo es un poco robusta existe otra manera de ejecutar el SPCC en una PC que no tenga previamente instalado el entorno de desarrollo.

Esta segunda forma requiere instalar una librería que contiene todo lo necesario para poder ejecutar programas previamente compilados en una PC sin el entorno de desarrollo MATLAB[®] instalado. La librería mencionada se encuentra localizada en el CD que acompaña este documento de tesis.

D.1.1 Descripción de los componentes de la interfaz gráfica.

En la figura D.1 se muestra la pantalla que aparece al ejecutarse el archivo *interfaz.m* dentro del entorno de desarrollo o en su defecto el archivo *interfaz.exe* desde fuera del entorno.

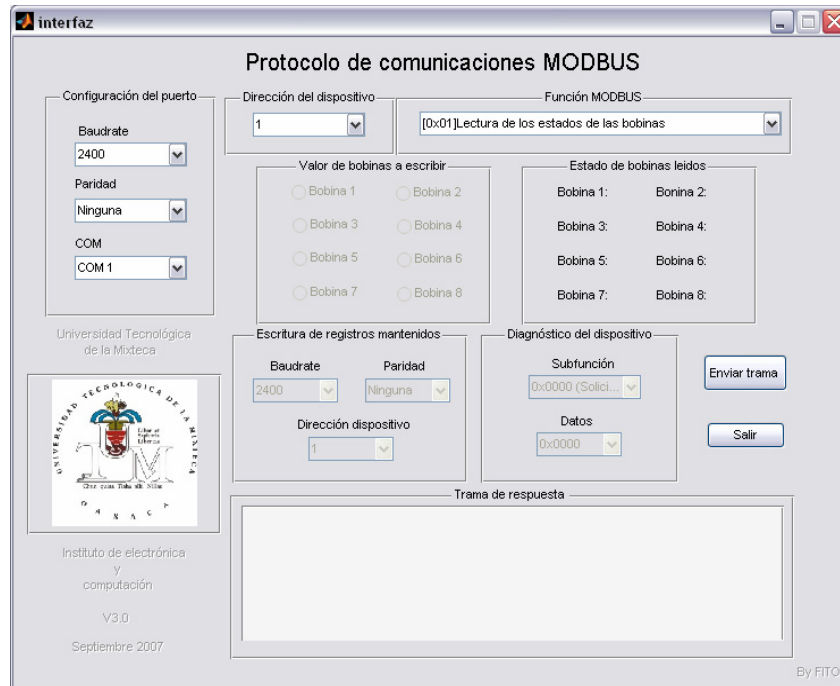


Figura D.1. Ventana principal del SPCC.

Para el envío de una función a ejecutar por el CDECA es necesario conocer los apartados que conforman la interfaz gráfica del SPCC, los cuales son:

- Configuración del puerto, se encuentran los menús para configurar el puerto serial de la computadora donde se está ejecutando el SPCC y por el cual se enviarán los datos.
- Dirección del dispositivo, sirve para designar la dirección del dispositivo esclavo al cual será dirigida la petición o función MODBUS.
- Función MODBUS, apartado en el cual se encuentran las funciones MODBUS que se pedirán ejecute el CDECA.
- Valor de bobinas a escribir, en él se podrán escoger las cargas que se encenderán o apagarán. Se activa al escoger la función 0x0F.
- Estado de bobinas leídas, apartado donde se mostrará el estado actual de las cargas. Es complemento de la función 0x01 ya que muestra la respuesta del CDECA al ejecutarla.

- Escritura de registros mantenidos, con los menús presentes en este apartado se puede cambiar la configuración del CDECA, ya sea el baudrate, la paridad o la dirección de esclavo. Se activa con la función 0x10.
- Diagnóstico del dispositivo, apartado que cuenta con las subfunciones de la función 0x08 soportadas por el CDECA. Se activa con la función 0x08 del menú del apartado "Función MODBUS".
- Trama de respuesta, apartado en el que se mostrará la información obtenida de la trama de respuesta una vez decodificada, esto para que sea más entendible al usuario del SPCC.

Se cuenta con dos botones "Enviar trama" y "Salir". El primero tiene la función de enviar la trama de petición (función MODBUS a ejecutar) al CDECA, todo esto una vez que se hayan elegido correctamente las opciones de la función a enviar; el segundo botón sirve para finalizar la ejecución del SPCC.

D.1.2 Envío de una petición al CDECA.

En la figura D.2 se muestra la pantalla obtenida del SPCC al ejecutar la función 0x2B, la cual pide la identificación del dispositivo al CDECA.

Se puede apreciar la configuración del puerto serial número 3 (COM3) de la PC, el cual enviará la información a una velocidad de 2400 BPS sin paridad, siendo 1 la dirección del esclavo al que será dirigida la trama de petición MODBUS.

En el apartado "Función MODBUS" se observa que la función escogida para que sea ejecutada por el CDECA es la 0x2B (identificación del dispositivo), la cual se enviará por el puerto serial 3 de la PC al momento de presionar el botón "Enviar trama".

Una vez que se ha enviado la función MODBUS al CDECA falta esperar que éste responda y se muestre la información ya decodificada en el apartado "Trama de respuesta".

Dado que en el ejemplo de la figura D.2 se manda a ejecutar la función 0x2B, la respuesta obtenida muestra el nombre de quien desarrolló el dispositivo, la versión del mismo y el código o clave con el que se identifica.

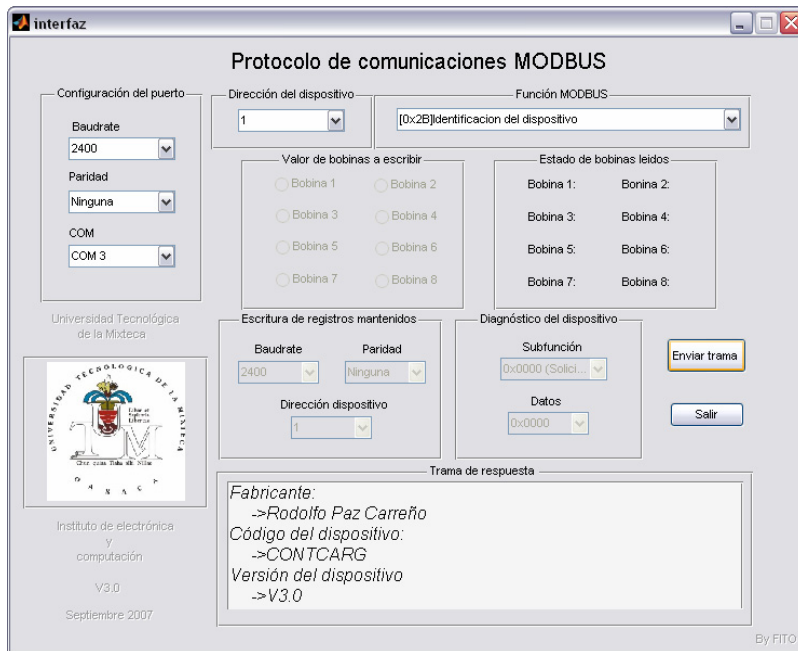


Figura D.2. Pantalla obtenida al mandar ejecutar la función 0x2B.

D.2 Posibles errores.

En la figura D.3 se muestra la carátula del SPCC al mandar ejecutar funciones al CDECA y estas no se ejecutaron por diversas causas.

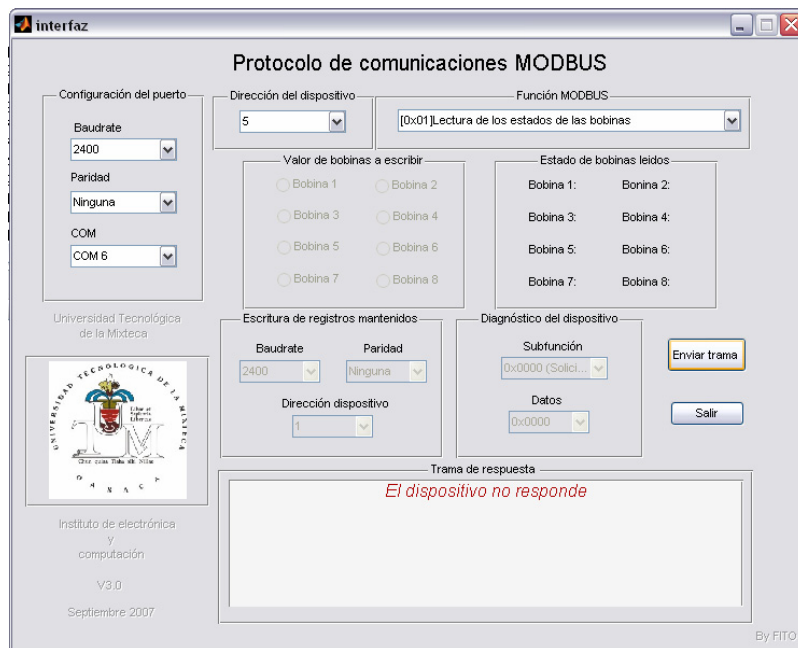


Figura D.3. Pantalla resultante al no recibir la contestación del CDECA.

Las posibles causas por las cuales el CDECA no responda pueden ser:

- El CDECA no se encuentra conectado a una fuente de voltaje.
- La dirección del apartado "Dirección del dispositivo" es distinta al del CDECA.
- El puerto serial por el cual se envió la información está configurado a una velocidad distinta a la que puede recibirlos el CDECA.
- La trama MODBUS enviada sea una petición de difusión.
- Los cables por los cuales se encuentran comunicados la PC y el CDECA se encuentran trozados (puerto RS-485). El ComUSB no se encuentra conectado a una fuente de voltaje o no cuenta con el cable USB para conectarse a la PC (puerto serial).

Las razones del por qué el CDECA no responde pueden evitarse si antes de hacer uso del dispositivo se revisan cada uno de los puntos mencionados anteriormente.

