



Universidad Autónoma de Querétaro
Facultad de Ingeniería, División de Investigación y Posgrado
Maestría en Ciencias de Ingeniería, Instrumentación y Control Automático

Análisis y desarrollo de una plataforma de comunicación entre dispositivos de control para un robot industrial de 6 ejes de libertad

TESIS

Que como parte de los requisitos para obtener el grado de
Maestro en ciencias en Instrumentación y Control Automático

Presenta:

Julio Alejandro Romero González

Dirigido por:

Dr. Juvenal Rodríguez Reséndiz

SINODALES

Dr. Juvenal Rodríguez Reséndiz
Presidente

Dr. Juan Carlos A. Jáuregui Correa
Secretario


Dr. Roberto Gómez Loenzo
Vocal


Dr. Edgar Rivas Araiza
Suplente


M. en C. José Marcelino Gutiérrez Villalobos
Suplente

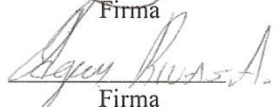
Dr. Aurelio Domínguez González
Director de la Facultad

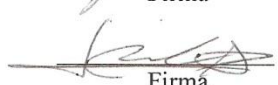
Dr. Irineo Torres Pacheco
Director de Investigación y
Posgrado


Firma


Firma


Firma


Firma


Firma

Centro Universitario
Querétaro, Qro.
MAY-2014

México

Resumen

Dentro de esta tesis se presenta el desarrollo de un prototipo para una plataforma de comunicación de arquitectura abierta que permita el manejo y gestión de datos en sistemas robóticos que se encuentran en desuso, los cuales puedan ser adaptados a nuevas tecnologías y aplicaciones, también se describen los métodos y aplicaciones empleados para logra obtener un prototipo flexible que permita a cada mecanismos ser autónomo, que tenga disponibilidad para acceso remoto, facilidad de integración de nuevos algoritmos de control o módulos de programas creados por el usuario, flexibilidad en la integración con un ordenador e interacción con el usuario, facilidad de integración con distintos servoamplificadores, finalmente se presenta el uso de la plataforma en una mesa de coordenadas, los resultados que se obtuvieron de las velocidades de comunicación, los retardos y colisiones que se presentan en el bus, también se redactan las conclusiones a las que se llegaron y en la sección de anexos los códigos empleados y diagramas.

Summary

Inside this thesis the development of a prototype is presented to a communication platform with open architecture that allows the handling and management of data in robotic systems that are not in use, which can be adapted to new technologies and applications are also described methods and applications used to manage to get a flexible prototype that allows each mechanism to be autonomous, having remote access availability, ease of integration of new control algorithms or software modules created by the user , flexible integration with a computer and interaction with the user, ease of integration with other servo amplifiers finally use the platform in a coordinate table presents the results that were obtained from the communication speeds , delays and collisions that occur on the bus , also the conclusions that were reached are written and section of appendices and diagrams used codes .

Dedicatorias

Dedico esta tesis a aquellos que han estado impulsándome en los momentos más difíciles de mi carrera.

Agradecimientos

Al término de esta etapa de mi vida, quiero expresar un profundo agradecimiento a quienes con su ayuda, apoyo, comprensión, amistad, compañía, ánimo, me han ayudado a cruzar con firmeza el camino de la superación, porque con su apoyo y aliento hoy he logrado uno de mis más grandes anhelos. Es cierto que cuando se presenta el momento de decir gracias, hacen falta palabras que realmente siente el corazón, agradezco de todo corazón a Dios y a mis Padres porque a través de ellos me concedió la vida en este mundo, y porque gracias a su apoyo y consejos han guiado mi camino. Así como a mis abuelos que para mí han sido como mis padres y quienes me han formado moralmente como ser humano, quiero agradecerles tantos sacrificios, esfuerzos de su vida para educarme, el amor con el que han entregado su vida para convertirme en un hombre de provecho. A mi tía quien con su ayuda, apoyo y comprensión me ha alentado y motivado a buscar un mejor futuro y edificación de mi vida, por ese cariño que siempre he recibido en los momentos buenos y malos de mi vida, hago este triunfo compartido. A mi mejor amiga a quien jamás encontraré la forma de agradecer su apoyo, comprensión, confianza, sonrisas, espero que comprendas que muchos de mis logros han sido por la fortaleza de tu apoyo, y que mis ideales son inspirados en tu nobleza, a mis amigos quienes directa o indirectamente han sido inspiradores. Finalmente pero no menos importantes a mi asesor y sinodales quienes se tomaron tiempo para revisar y guiarme en la escritura de esta tesis, a mis maestros por su dedicación para ayudar a formarme en un ámbito profesional y al Consejo Nacional de Ciencia y Tecnología (CONACyT) por haberme otorgado una beca para realizar mis estudios de posgrado.

ÍNDICE

Resumen	i
Summary	ii
Dedicatorias	iii
Agradecimientos	iv
Índice	v
Índice de cuadros	vi
Índice de figuras	vii
Capítulo 1. Introducción.....	1
Capítulo 2. Fundamentación teórica.....	2
2.1. Antecedentes.....	2
2.2. El modelo de referencia OSI.....	6
2.2.1. Niveles del modelo OSI.....	7
2.3. Ethernet.....	8
2.3.1. Definición.....	8
2.3.2. Control de colisiones.....	9
2.3.3. Direccionamiento.....	11
2.3.4. Formato de trama.....	12
2.4. Protocolo RS-485.....	13
2.4.1. Características básicas.....	13
2.4.2. Descripción del Bus.....	14
2.4.3. Simplex (Half-Duplex)	14
2.4.4. Full-Duplex.....	15
2.5. Profibus.....	16
2.5.1. Estructura de la red.....	17
2.5.2. Estructura del telegrama.....	18

2.5.3. Funciones de comandos.....	20
2.5.3.1. Delimitador de inicio (SD).....	20
2.5.3.2. Longitud del telegrama (LE y LEr)	20
2.5.3.3. Dirección de destino y dirección de origen.....	21
2.5.3.4. Código de función o control de trama.....	21
2.5.3.5. Puntos de acceso de servicio.....	22
2.6. Modbus.....	23
2.6.1. Modo ASCII.....	23
2.6.2. Modo RTU.....	24
2.6.3. Códigos de funciones.....	25
Capítulo 3. Metodología.....	26
3.1. Arquitectura de la plataforma.....	26
3.1.1. Interfaz de Usuario.....	27
3.1.2. Integración de datos.....	27
3.1.3. Capa de comunicación.....	27
3.1.4. Módulos de interfaz física.....	28
3.1.5. Comparación con el modelo OSI.....	29
3.2. Hardware.....	30
3.2.1. Unidad de Control Maestra (MCU).....	30
3.2.2. Controlador Ethernet.....	31
3.2.3. Transceptor RS-485.....	32
3.2.4. Transceptor Profibus.....	33
3.2.5. Módulos de interfaz física.....	34

3.3. Software.....	35
3.3.1. Diagrama de flujo para Modbus en el protocolo RS485.....	35
3.3.2. Diagrama de flujo para control para flujo de datos por TCP/IP.....	39
3.3.3. Diagrama de flujo para uso del protocolo Profibus.....	43
3.3.4. Diagrama de flujo para algoritmo CSMA/CD.....	45
3.3.5. Trama de datos para protocolo RS485.....	46
 Capítulo 4. RESULTADOS.....	 48
4.1. Conexión de la plataforma a la mesa de coordenadas.....	48
4.2. Conexión de la plataforma al Servo drive EDB.....	56
 Capítulo 5. CONCLUSIONES.....	 64
 Capítulo 6. APÉNDICE.....	 65
 LITERATURA CITADA.....	 86

Índice de Cuadros

TABLA 1.1 Formato de trama Ethernet.....	12
TABLA 2.1. Formato de telegramas.....	20
TABLA 2.2. Códigos para solicitud de telegramas.....	21
TABLA 2.3. SAP's usados en Profibus DP.....	22
TABLA 2.4. Código de funciones Modbus.....	25
TABLA 3.1. Comparación con el modelo OSI.....	29
TABLA 4.1. Datos de pruebas.....	55
TABLA 4.2. Datos obtenidos de RobotWorx.....	56
TABLA 4.3. Parámetros de comunicación.....	59
TABLA 4.4. Estructura de datos.....	60
TABLA 4.5. Búsqueda del servo drive.....	61
TABLA 4.6. Estado del servo drive.....	62

Índice de Figuras

FIGURA 2.1. Representación del modelo OSI.....	7
FIGURA 2.2. Topología de bus.....	14
FIGURA 2.3. Comunicación Half-Duplex.....	15
FIGURA 2.4. Comunicación Full-Duplex.....	15
FIGURA 2.5. Cabecera del telegrama de datos.....	19
FIGURA 2.6. Trama Modbus en modo ASCII.....	24
FIGURA 2.7. Trama Modbus en modo RTU.....	25
FIGURA 3.2. Arquitectura de la plataforma de comunicación.....	25
FIGURA 3.3. PIC18F97J60 como MCU.....	30
FIGURA 3.4. Interfaz basada en ENC28J60.....	31
FIGURA 3.5. Conexión física del controlador Ethernet.....	32
FIGURA 3.6. Configuración para operación <i>full-duplex</i>	32
FIGURA 3.7. Conexión física del MAX3089 e ISO1176T.....	33
FIGURA 3.8. Arquitectura de los módulos de interfaz física.....	34
FIGURA 3.9. Diagrama de bloques para transmisión Modbus.....	36
FIGURA 3.10. Diagrama de bloques para recepción de datos Modbus.....	38
FIGURA 3.11. Diagrama de flujo recepción TCP/IP.....	39
FIGURA 3.12. Diagrama de flujo transmisión TCP/IP.....	41
FIGURA 3.13. Diagrama de flujo de protocolo profibus.....	43

FIGURA 3.14. Diagrama de flujo de protocolo profibus.....	45
FIGURA 4.1. Mesa de coordenadas TX8.....	48
FIGURA 4.2. Servodrive ADS 50-10.....	49
FIGURA 4.3. Esquema de conexión de la plataforma de comunicación a los amplificadores.....	50
FIGURA 4.4. Módulos de interfaz física.....	50
FIGURA 4.5. Interfaz de usuario.....	51
FIGURA 4.6. Perfil de movimiento realizado por la mesa TX8.....	52
FIGURA 4.7. Ubicación de pic's auxiliares para obtención de muestras.....	53
FIGURA 4.8. Velocidad de transferencia RS232.....	53
FIGURA 4.9. Velocidad de transferencia RS485.....	54
FIGURA 4.10. Robot NACHI modelo SA160F-01.....	56
FIGURA 4.11. Servodrive EDB-50A.....	57
FIGURA 4.12. Diagrama de conexión RS485.....	58
FIGURA 4.13. Configuración velocidad de jogeo.....	64

CAPITULO 1. INTRODUCCIÓN

En la actualidad los servomecanismos se usan de manera extensa en la industria, en máquinas herramientas de control numérico y principalmente en robots, siendo este último un elemento indispensable en una gran parte de los procesos de manufactura, impulsados principalmente por el sector automotriz; mientras nuevas tecnologías y aplicaciones robóticas industriales surgen, mientras que los robots industriales que llevan mucho tiempo realizando labores enfrentan problemas en los procesos automatizados, por lo que cada vez grandes industrias se deshacen de máquinas en desuso, en tanto que pequeñas y medianas empresas no tienen acceso a esta tecnología, principalmente debido a altos costos tales como accesorios, tarjetas de control, interfaz humano-máquina y que cada fabricante utiliza protocolos propios para realizar la comunicación entre los dispositivos (Bracarense 1999).

Si bien las interfaces hombre-máquina han seguido una vía evolutiva en los sistemas de control buscando proporcionar al usuario la recolección de datos de manera automática así como la supervisión y monitorización de los sistemas de control en tiempo real, sin embargo en la actualidad arquitecturas cerradas están presentes en la totalidad de los robots industriales, las cuales han sido desarrolladas por corporaciones multinacionales dedicadas a la automatización de procesos, siendo este uno de los principales inconvenientes para la escalabilidad de estos sistemas, sin embargo se han estado desarrollando distintos sistemas de comunicación buscando adoptar una arquitectura estándar (Jos et al. 2003), por lo cual la investigación de este proyecto busca mediante la aplicación de la teoría y conocimientos en distintas áreas, desarrollar un prototipo que permita a nuevas tecnologías y aplicaciones robóticas ser adaptadas a máquinas que se encuentran en desuso en la industria.

CAPITULO 2. FUNDAMENTACIÓN TEÓRICA

2.1. Antecedentes

Cuando se habla de robots nos referimos a máquinas con características como repetibilidad en tareas, inteligencia (capacidad de tomar decisiones), movimientos articulados, adaptarse al medio ambiente (estructurados y no estructurados), contacto con el medio ambiente y máquinas inteligentes; en el campo de la robótica el problema del control de movimientos del brazo articulado es uno de los aspectos en los que se desarrollan más trabajos o investigaciones Martínez and Flugrad (1994), uno de los principales inconvenientes que enfrentan los investigadores es la imposibilidad de crear nuevos algoritmos de control sobre arquitecturas cerradas las cuales están presentes en la totalidad de los robots industriales actualmente en el mercado, en el estudio realizado por Morales-Velazquez et al. (2010), se propuso un sistema de comunicación que integra componentes de hardware y software que comparten un mismo protocolo. El sistema está diseñado para permitir la integración de las nuevas unidades de procesamiento de dos módulos de hardware y software. El sistema de MADCON (Multi-Agent Distributed CONTroller) abarca tres elementos: una plataforma de hardware, una plataforma de software, y un protocolo de conexión, dicha plataforma contienen dos elementos: la unidad de microprocesador (MPU) y el sistema de interconexión de los medios de comunicación (SIM). Una RDC (controlador) procesa los datos y utiliza el sistema de comunicación para enviar la información pertinente a otra RDC o para el usuario, el sistema que propone el autor fue basado en Field Programmable Gate Array (FPGA), ya que al usar un DSP (Digital Signal Processor) la plataforma de reconfiguración queda limitada. Oliveira and Guenther (2008) diseñaron un controlador con una estructura modular, cuya arquitectura de comunicación es en capas, la cual incluye una capa de tareas, una de integración, una de comunicación, una de interfaz y una física, dichas capas e incluyendo el software fueron desarrollados bajo una arquitectura abierta. Otros personajes se basan en la comunicación por módulos, en la cual integran un sistema de componentes a través de un módulo de

comunicación que pueda ser usado como el esqueleto para organizar todo el sistema, creando funciones de comunicación las cuales no se apeguen a una estructura jerárquica, es decir desarrollan un módulo de forma individual en el cual no definen una estructura de datos interna, este tipo de sistema, por lo que cada módulo llega a ser autónomo. Por otra parte Niu (2012), emplea un método de diseño de un convertidor USB-CAN basado en un microcontrolador PIC18F4580 y un chip CH372 de interfaz USB. Por medio del cual utilizando el controlador, un equipo puede tener acceso y controlar los equipos terminales de la red CAN (Controller Area Network) como un nodo.

Dentro del trabajo realizado por Xing, Jia, and Yanqianga (2011) se propone el uso de una interfaz SERCOS (Serial Real-time Communication System), sobre la tecnología de bus EtherCAT, en el cual estructura el protocolo de comunicación en un modo maestro-esclavo, en la cual el dispositivo maestro utiliza un estándar utilizando un FPGA para enviar y recibir tramas de EtherCAT, el modelo integra máquinas de estados para realizar la sincronización, la comunicación de datos de proceso y el acceso a través de servicio, este modelo de estructura Ethernet muestra comportamiento en tiempo real. Hongyu, Yong, and Na (2010), presenta un sistema distribuido de control digital de movimiento basado en SERCOS, por el cual transmite a través de fibra óptica, proporcionando al sistema inmunidad al ruido y alta velocidad, este sistema se compone de una PC y la tarjeta de interfaz SERCOS. El equipo esclavo comprende circuito DSP, el conductor y circuito de retorno. Lo anterior se aplica a una interfaz para controlar y supervisar el eje.

El sistema desarrollado por Draganjac et al. (2008), permite el modo de control remoto de un robot en el que los caminos de posición y la velocidad de retorno son procesados a través de software Matlab-Simulink en el cliente remoto (operador), en el cual se describe un control de robot accesible a través de Internet. El sistema se basa en una arquitectura cliente-servidor TCP/IP. En el que el sistema de servidor, se ejecuta en una plataforma integrada Ultimodule, controlando el movimiento del robot SCARA. La

aplicación basada en PC permite a los usuarios aplicar control de movimiento de trayectorias. La estructura modular de software que se adopta para el desarrollo de la arquitectura de controlador de movimiento libre es propuesta por Liu, Wang, and Fu (2008), donde realiza un análisis de la estructura de sistema universal de CNC, y extrae las características comunes como la base y características comunes del controlador en módulos de función. El desarrollo del proceso de configuración y el programa de aplicación propuesto se basa en la de modularidad, reutilización, configurabilidad, portabilidad, extensibilidad y escalabilidad requerida por el sistema de control de arquitectura abierta, aplicándolos a una máquina herramienta 5-ejes.

Con el fin de aprovechar al máximo los recursos del sistema y mejorar el rendimiento del equipo basado de control numérico (CNC) del sistema en Ethernet, que es muy necesaria para establecer el modelo de análisis de rendimiento y adopte el método de codiseño desde la perspectiva de control, comunicaciones e informática, el análisis de la arquitectura del sistema, las características de los parámetros temporales de control de lazo, el marco del modelo, para el cual Yan et al. (2011), propusieron un modelo de simulación para el análisis en tiempo real basada en Ethernet para el Sistema CNC. Este modelo incluye los efectos de temporización producidos por la plataforma de implementación en la simulación. En la cual la clave para establecer el modelo se realiza mediante el diseño del módulo de análisis de error y los nodos del controlador. De otra forma Xiao et al. (2007), propone una arquitectura abierta de control numérico computarizado (CNC) basado en Windows CE, que es una tarea en tiempo real del sistema operativo embebido. En la que el sistema CNC adopta una arquitectura maestro-esclavo y el maestro se aplica con Windows CE, la cual es un sistema embebido multitarea en tiempo real, mientras que el esclavo adopta una tarjeta de control de movimiento basado en un DSP y FPGA, la cual se en carga de duras tareas en tiempo real incluyendo interpolación, control de electromotor etc la investigación se da cuenta de la construcción de hardware y software de sistema de CNC.

Otros aspectos como el modelo de transparencia, la comunicación en tiempo real y determinista de Ethernet industrial es una necesidad urgente. La plataforma de comunicación basada en las especificaciones de fabricación de mensajes (MMS), TCP / IP y Ethernet es la tendencia del próximo sistema de comunicación industrial automatización. Por lo que Zhang, Liu, and Wang (2008), proponen un modelo de referencia basado en MMS + TCP / IP + Ethernet para alcanzar en tiempo real los datos de cambio de CNC en el entorno distribuido. Este modelo se construye mediante la tarjeta de control principal basado en microcontrolador ARM, controlador de movimiento (DSP basado y CPLD) y en tiempo real del sistema operativo y RTLinux interfaz gráfica de usuario de MiniGUI, analizan los factores del efecto de rendimiento de la comunicación en tiempo real, así como la comunicación de las tareas de prioridad Ethernet, el protocolo TCP / IP sobre Ethernet en tiempo real de los modelos de redes de comunicación y la programación orientada a eventos.

Dentro de los proyectos realizados en el alma mater Galindo (2001), presenta un desarrollo de una tarjeta electrónica basada en los controladores de arquitectura abierta, el cual fue implementado en un procesador digital de señales, con dicha tarjeta realizo experimentaciones sobre servomecanismos con motores de corriente continua (DC) y decodificadores de posición (encoders) incrementales, la que principalmente fue dedicada a realización y evaluación de algoritmos de control no convencionales en sistemas reales. Por otra parte Loenzo (2009), propone el desarrollo de un sistema CNC de maquinado con base en un proceso estandarizado de desarrollo de software, enfocándose en definir de una manera precisa los requerimientos del sistema CNC, de tal forma que se tenga un sistema de arquitectura abierta que se adapte al proceso de desarrollo del fabricante.

En la investigación realizada por Ortiz (2003), presenta el diseño de un controlador basado en una computadora personal para la generación de trayectorias y el control del movimiento de seis motores aplicados a un robot de cinemática paralela, implementando su diseño en un microcontrolador para llevar acabo la administración de

tareas y usándolo como interfaz y reportando los datos a la computadora a través del puerto paralelo. Rios (2004), desarrollo un sistema de control de posicionamiento de servomotores mediante la implementación de módulos en FPGA's, en los cuales incorpora la programación de las tareas de movimiento mediante una interfaz ISA (Interface Standar Adapter, Adaptador Estandar de interfaz), en una computadora personal típica.

2.2. *El modelo de referencia OSI*

El modelo de referencia OSI (*Open Systems Interconnection*) es el modelo principal para las comunicaciones por red. Aunque existen otros modelos, en la actualidad la mayoría de los fabricantes de redes relacionan sus productos con el modelo de referencia OSI, especialmente cuando desean enseñar a los usuarios cómo utilizar sus productos, el modelo OSI es un marco de trabajo estructurado en capas, cada una de las cuales ilustra una función de red específica. Esta división de las funciones de “networking” se denomina división en capas. Si la “networking” se divide en estas siete capas, se obtienen las siguientes ventajas:

- Divide la comunicación de red en partes más pequeñas y sencillas.
- Normaliza los componentes de red para permitir el desarrollo y el soporte de los productos de diferentes fabricantes.
- Permite a los distintos tipos de hardware y software de red comunicarse entre sí.
- Impide que los cambios en una capa puedan afectar las demás capas, para que se puedan desarrollar con más rapidez.
- Divide la comunicación de red en partes más pequeñas para simplificar el aprendizaje.



Figura 2.1. Representación del modelo OSI.

La Figura 2.1 presenta la estructura de capas que conforman el modelo OSI, la pirámide es uno de los modos que mejor ilustran la estructura de este modelo, en el que los datos con un formato bastante complejo pasan a convertirse en una secuencia simple de bits cuando alcanzan la capa de la red. A continuación se describen los niveles del modelo OSI.

2.2.1. Niveles del modelo OSI

1. Nivel físico. Esta capa proporciona el servicio de transmisión transparente de secuencias de bits sobre el medio físico, incluyendo la adecuada señalización, sincronización, multiplexión, recuperación de reloj, etc.
2. Nivel de enlace. Se ocupa de la transmisión ordenada de bloques de bits o tramas libres de error, incluyendo una forma de direccionamiento físico, control de flujo, control de acceso a la red, etc.

3. Nivel de red. Proporciona un servicio de para enviar paquetes de datos desde un origen hasta un destino que puede pertenecer a otra red, proporcionando además direcciones de red a la capa de transporte.
4. Nivel de transporte. Esta capa provee a los niveles superiores de un servicio de transferencia de información independiente de los detalles de la red, incluyendo la conversión de direcciones de red, secuenciación, corrección de errores, etc.
5. Nivel de sesión. Establece, gestiona y finaliza las conexiones entre los procesos o aplicaciones finales.
6. Nivel de presentación. Realiza conversiones en la representación interna de los datos, de manera que distintos equipos puedan trabajar con ellos.
7. Nivel de aplicación. Ofrece a los usuarios la posibilidad de acceder a los servicios del resto de capas. Los niveles cinco, seis y siete suelen a menudo fusionarse en un único nivel de aplicación que se comunica directamente con la capa de transporte.

2.3. Ethernet

2.3.1. Definición

Ethernet es un protocolo de nivel de enlace para redes de área local (LAN) basado en datagramas y definido en el estándar IEEE 802.32 (Man and Society 2012), de comunicaciones entre iguales (PTP o P2P: peer to peer) en el que no hay un control centralizado y todas las estaciones son tratadas por igual. Este estándar se basa en el trabajo realizado por Robert Metcalfe en el Xerox PARC, quien definió un protocolo de acceso al medio compartido que se conoce como CSMA/CD (Postel et al. 2009).

Existen distintas variantes de Ethernet que aparecen como suplementos al estándar original y que dependen de la tasa de transferencia máxima, del modo de transmisión y del medio físico. Actualmente hay definidas cuatro tasas de transferencia sobre distintos medios de transmisión:

- 10 Mbps. (Por ejemplo Ethernet 10Base-T, que define el transporte sobre cable de par trenzado).
- 100 Mbps (Fast Ethernet).
- 1000 Mbps (Gigabit Ethernet).
- 10 Gbps (10 Gigabit Ethernet).

Para garantizar la compatibilidad, los estándares IEEE 802.3 debían cubrir las necesidades de la Capa 1 y de las porciones inferiores de la Capa 2 del modelo OSI. Como resultado, ciertas pequeñas modificaciones al estándar original de Ethernet se efectuaron en el 802.3. El modelo ofrece una referencia sobre con qué puede relacionarse Ethernet, pero en realidad se implementa sólo en la mitad inferior de la capa de Enlace de datos, que se conoce como subcapa Control de acceso al medio (Media Access Control, MAC), y la capa física; este modelo se basa en una topología de bus con medio compartido, es decir, varias estaciones que pueden enviar datos al mismo tiempo. Por tanto se debe arbitrar un mecanismo de control de acceso al medio y resolver los problemas que conllevan los accesos simultáneos (colisiones). Las nuevas versiones se basan en comunicaciones full duplex sobre canales independientes, por lo que no pueden existir las colisiones, para ello es necesario un control que administre los envíos de datos para evitar y controlar las colisiones.

2.3.2. Control de colisiones

El mecanismo de control de colisiones de Ethernet es el CSMA/CD (Carrier Sense Multiple Access/Colision Detect) y se basa en escuchar si el medio está libre para empezar a transmitir y asegurarse que la señal transmitida no es alterada por otra señal. Una vez comprobado que el medio está libre únicamente hay que dejar pasar un tiempo equivalente

a 12 bytes antes de comenzar a transmitir. Esta pausa sirve para evitar que los datos transmitidos se concatenen con otra trama en la red y para dar tiempo a las estaciones a procesar una posible trama recibida y que vuelvan a la escucha.

Cuando dos estaciones transmiten a la vez se produce una “colisión” que supone una alteración de las señales enviadas. Esta alteración es detectada por las dos estaciones emisoras, que actúan de la siguiente manera:

1. Detienen la transmisión de la trama.
2. Envían una señal de atasco (Jam) durante el tiempo equivalente a 4 Bytes para que todas las estaciones se enteren de que ha habido colisión.
3. Ponen en marcha el mecanismo de retroceso exponencial binario para decidir cuánto tiempo esperan a retransmitir. Ese tiempo será aleatorio para evitar que las dos esperaren el mismo tiempo y vuelvan a colisionar.
4. Transcurrido ese tiempo, si el medio está libre, vuelven a intentar emitir.

Todas las estaciones que comparten el medio físico forman un dominio de colisión. Una estación compite por el medio con todas las de su dominio de colisión. No sólo con las que estén en su mismo cable, también con las que estén al otro lado de repetidores y concentradores (hubs), pero no con las que estén al otro lado de puentes (bridges), conmutadores (switchs) o encaminadores (routers).

El diámetro de un dominio de colisión es la distancia máxima entre estaciones. Esta distancia debe ser lo suficientemente pequeña para que en el tiempo que tarda en ir y volver la señal de un extremo a otro no se pueda transmitir una trama de tamaño mínimo T (2τ propagación en el diámetro máximo $< \tau$ transmisión de T).

2.3.3. Direccionamiento

Cada nodo tiene una dirección de nivel 2 única, llamada dirección MAC (Medium Access Control). Esta dirección tiene 48 bits (6 bytes) y se suele representar con 12 caracteres hexadecimales separados por “:” o por “-“ entre cada dos. Las direcciones son asignadas por los fabricantes, que se identifican por los primeros 24 bits. Una trama puede ir dirigida a:

- Una estación concreta (unicast). Lleva como dirección de destino la dirección MAC de la estación de destino.
- Todas las estaciones de su red (difusión o broadcast). Lleva como dirección de destino la dirección de difusión –todos los bits a uno- (FF:FF:FF:FF:FF:FF).
- Un grupo de estaciones que comparten una dirección especial (multicast). Estas direcciones especiales tienen un 1 en el último bit de su primer byte. Típicamente, son de la forma 01:xx:xx:xx:xx:xx.

La subcapa MAC de Ethernet tiene dos responsabilidades principales:

- a) Encapsulación de datos: incluye el armado de la trama antes de la transmisión y el análisis de la trama al momento de recibir una trama. Cuando forma una trama, la capa MAC agrega un encabezado y un tráiler a la PDU de Capa 3. La utilización de tramas facilita la transmisión de bits a medida que se colocan en los medios y la agrupación de bits en el nodo receptor.
- b) Control de Acceso al medio: controla la colocación de tramas en los medios y el retiro de tramas de los medios. Como su nombre lo indica, se encarga de administrar el control de acceso al medio. Esto incluye el inicio de la transmisión de tramas y la recuperación por fallo de transmisión debido a colisiones.

2.3.4. Formato de Trama

La trama en Ethernet puede tener entre 64 y 1518 bytes y tiene el siguiente formato:

Preámbulo	Delimitador de inicio de trama (SFD)	Dirección de destino	Dirección de origen	Tipo / Long. De trama	Datos	Relleno	Secuencia de chequeo de trama (FCS)	Pausa
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	0-1500 bytes	0-46 bytes	4 bytes	12 bytes

Tabla 1. Formato de trama Ethernet.

- Preámbulo: Secuencia de 7 bytes 10101010 para estabilizar el medio.
- Delimitador de inicio de trama (Start of Frame Delimiter): Byte 10101011 que indica que se va a iniciar la transmisión.
- Dirección de destino: es la dirección de destino.
- Dirección de origen: es la dirección de origen.
- Tipo/Long: es interpretado como:
 - Tipo de protocolo de nivel 3, si su valor es mayor de 1536 (formato original DIX29).
 - Longitud de la trama, si es menor que 1536 (formato IEEE 802.3).
- Datos: son los datos que lleva la trama.
- Relleno: existirá si no hay suficientes datos para que la trama tenga al menos 64 bytes.

- Secuencia de chequeo de trama (Frame Check Sequence): es un código de redundancia cíclico para comprobar que no ha habido errores.
- El final de la trama se detecta por el hueco equivalente a 12 bytes que todas las estaciones deben respetar. Es decir tras cada trama el emisor hace una pausa por el tiempo equivalente a enviar 12 bytes.

2.4. *Protocolo RS-485*

En 1983, la Asociación de Industrias Electrónicas (EIA) aprobó un nuevo estándar de transmisión balanceada llamado RS-485. El cual encontró a una amplia aceptación y uso en aplicaciones industriales, médicas y aplicaciones de consumo, RS-485 se ha convertido en la interfaz confiable de la industria (Axelson 2001). Está definido como un sistema en “bus” de transmisión multipunto diferencial, es ideal para transmitir a altas velocidades sobre largas distancias (35 Mbit/s hasta 10 metros y 100 kbit/s en 1200 metros) y a través de canales ruidosos, ya que reduce los ruidos que aparecen en los voltajes producidos en la línea de transmisión, la transmisión diferencial permite múltiples drivers dando la posibilidad de una configuración multipunto; al tratarse de un estándar bastante abierto permite muchas y muy diferentes configuraciones y utilizaciones (B&B 1997).

2.4.1. *Características básicas*

Al tratarse de un estándar bastante abierto permite muchas y muy diferentes configuraciones y utilizaciones, sus principales características de RS-485 son:

- ✓ Interfaz balanceada
- ✓ Operación multipunto desde una sola fuente de 5 V
- ✓ Rango de bus de -7 V a +12 V
- ✓ Hasta 32 unidades de carga
- ✓ 10 Mbps de velocidad de datos máxima (a 40 metros)
- ✓ 4.000 metros de longitud máxima del cable (a 100 kbps)

De manera integral, en esta sección se analizan los aspectos importantes de la norma RS-485 al centrarse en los siguientes temas: la topología de bus, las características del estándar, tipos de conexiones y colisiones de datos.

2.4.2. Descripción del Bus

El estándar RS-485 sugiere que sus nodos se pueden conectar en red en una conexión en cadena, también conocida como la línea compartida o la topología de bus (ver fig. 2.2). En esta topología, los dispositivos participantes, receptores y transceptores se conectan a una rama principal de cable a través de líneas de red cortas. El bus de la interfaz puede ser diseñado para la transmisión *half-duplex* o *full-duplex* (ver fig. 2.3 y 2.4 respectivamente).

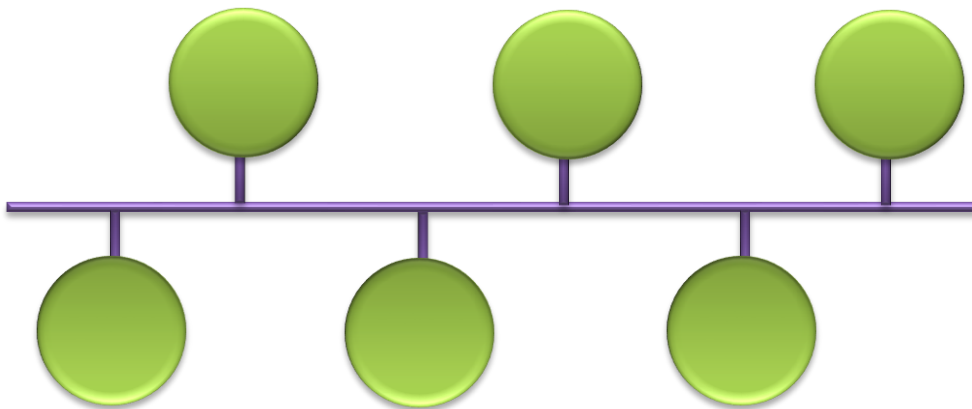


Figura 2.2. Topología de bus.

2.4.3. Simplex (Half-Duplex)

En una comunicación *half-duplex*, sólo se utiliza un par de señales (ver fig. 2.3), dicho término se refiere, a que puede ocurrir transmisión o recepción de información en diferentes momentos pero no al mismo tiempo. Tanto esta estructura como la *full-duplex* requieren una operación controlada de todos los nodos a través de señales de control de

dirección, como señales de habilitación Driver / Receiver, para asegurarse de que sólo un controlador está activo en el bus en cualquier momento.

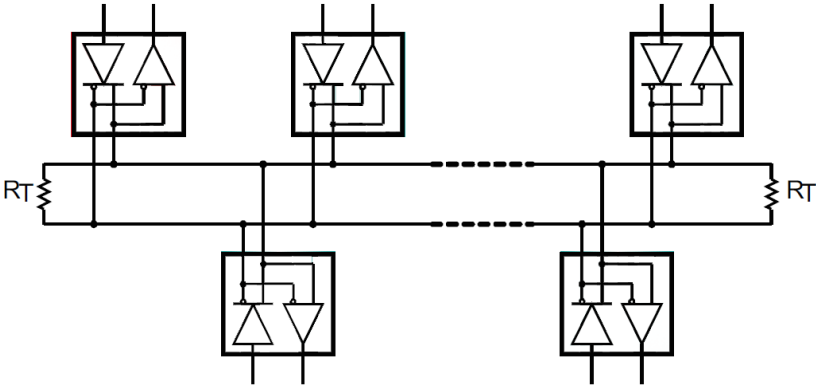


Figura 2.3. Comunicación Half-Duplex.

2.4.4. Full-Duplex

La implementación *full-duplex* requiere dos pares de señales, (cuatro hilos) y transceptores *full-duplex* con líneas de acceso de bus separados para el transmisor y el receptor. *Full-duplex* permite que un nodo transmita simultáneamente datos en un par, mientras que la recibe datos en el otro par.

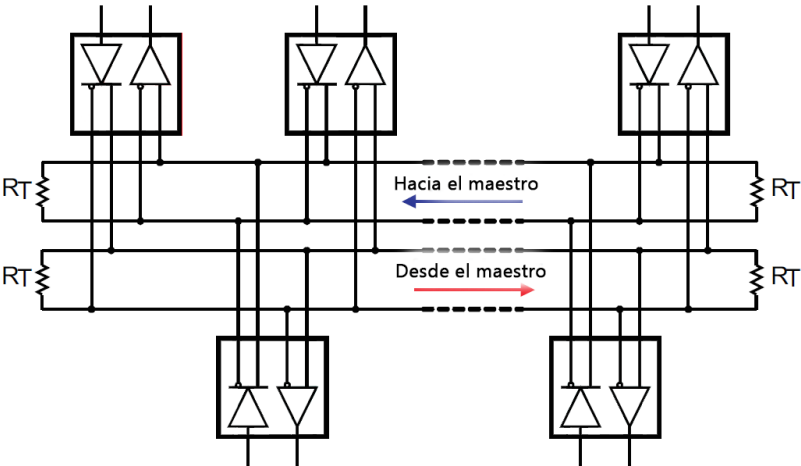


Figura 2.4. Comunicación Full-Duplex.

En el estándar RS-485 se ajustan sus conductores para proporcionar una salida diferencial de un mínimo de 1,5 V a través de una carga de 54 Ω , mientras que los receptores que conforman el estándar detectan una entrada diferencial a 200 mV. Los dos valores proporcionan un margen suficiente para una transmisión de datos fiable incluso bajo severa degradación de la señal a través del cable y los conectores. Esta solidez es la razón principal por la RS-485 es muy adecuado para la creación de redes de larga distancia en el medio ambiente ruidoso.

2.5. *Profibus*

Profibus también es implementado en la plataforma de comunicación ya que es un bus industrial abierto independiente de vendedores que sigue los estándares europeos EN 50170 y EN 5024 que aseguran tal condición, la organización que vela por este bus de campo es Profibus internacional (Nutzerorganisation 2013). Dentro de esta organización se encuentran inscritos más de 800 participantes de todo el mundo. Este es un bus que define todas las características de una red de comunicación serie industrial. Se utiliza como medio de intercambio de información y dispositivos distribuidos en campo, con Profibus los componentes de distintos fabricantes pueden comunicarse sin necesidad de ajustes especiales de interfaces (INCORPORATED 2002).

Profibus puede ser usado para transmisión crítica en el tiempo de datos a alta velocidad y para tareas de comunicación extensas y complejas. Esta versatilidad viene dada por las tres perfiles compatibles que componen a esta familia.

Profibus PA:

- Diseñado para automatización de procesos.
- Permite la conexión de sensores y actuadores a una línea de bus común incluso en áreas especialmente protegidas.
- Permite la comunicación de datos y energía en el bus mediante el uso de 2 tecnologías (norma IEC 1158-2).

Profibus DP:

- Optimizado para alta velocidad.
- Conexiones sencillas y baratas.
- Diseñada especialmente para la comunicación entre los sistemas de control de automatismos y las entradas/salidas distribuidas.

Profibus FMS:

- Solución general para tareas de comunicación a nivel de célula.
- Gran rango de aplicaciones y flexibilidad.
- Posibilidad de uso en tareas de comunicación compleja y extensa.

2.5.1. *Estructura de la red*

La tecnología de transmisión es la más usada en su área de aplicación ya que comprende aquellas aplicaciones donde prima su simplicidad, la velocidad de transmisión y lo barato de la instalación. Se usa un par diferencial con cable trenzado, previsto para comunicación semi-duplex, aunque también puede implementarse con fibra óptica y enlaces con estaciones remotas vía módem o vía radio. La velocidad de transmisión varía entre 9.6Kbits/s y 12Mbits/s, dependiendo del medio físico.

Al conectar varias estaciones, hay que comprobar que el cable de las líneas de datos no sea trenzado. El uso de líneas apantalladas es absolutamente esencial para el logro de una alta inmunidad del sistema en ambientes con emisiones altas de electromagnetismo (como en la fabricación de automóviles). El apantallamiento se usa para mejorar la compatibilidad electromagnética (CEM).

El elemento esencial del bus es el nodo. Profibus prevé la existencia de dos tipos de nodos:

- a) Activos: son nodos que pueden actuar como maestro del bus, tomando enteramente el control del bus.
- b) Pasivos: son nodos que únicamente pueden actuar como esclavos y, por tanto, no tienen capacidad para controlar el bus. Estos nodos pueden dialogar con los nodos activos mediante un simple mecanismo de pregunta-respuesta, pero no pueden dialogar directamente entre sí.

La topología puede ser simplemente en forma de bus lineal o en forma de árbol, en el que los repetidores constituyen el nodo de partida de una expansión del bus, el número máximo de nodos conectables a cada tramo del bus, sin necesidad de repetidores es de 32, mientras que el número máximo de nodos del bus es de 127.

No existe ninguna limitación en cuanto a poder configurar una estructura con buses anidados (un esclavo puede ser, a su vez, maestro de otro bus de nivel inferior), aunque deben considerarse como buses independientes, dado que el protocolo no permite direccionar desde arriba las estaciones de niveles inferiores.

2.5.2. *Estructura del telegrama*

Para la transmisión de datos Profibus se basa en la estructuración de un telegrama que puede contener 244 bytes de datos por mensaje, más 11 bytes de sobrecarga siendo un total de hasta 255 bytes. Esta sobrecarga se conoce como el telegrama de cabecera (ver fig. 2.5). Todas las cabeceras de telegramas son 11 bytes, a excepción de los telegramas Data_Exchange que tienen 9 bytes de información de cabecera (DSAP y SSAP). Tenga en cuenta que 12 bytes es una gran sobrecarga para un único mensaje y esto hace de ProfiBus

menos eficiente para pequeñas cantidades de datos. Sin embargo, desde hasta 244 bytes de datos pueden ocurrir por mensaje, y dado que se envían los datos de salida y los datos de entrada recibidos en un solo ciclo de mensajes, esto hace ProfiBus más eficiente cuando grandes cantidades de datos deben ser transferidos. Tenga en cuenta que un estado de inactividad de al menos 33 Tbits (tiempo de sincronización de bit) debe estar presente antes de cada mensaje de solicitud enviado, y todos los datos se transfieren sin espacios entre caracteres individuales.

Todos los intercambios de datos entre un maestro y esclavo se manejan en la cabecera del telegrama usando puntos de acceso de servicio (*Service Access Point, SAP*). Profibus DP utiliza SAP de 54 a 62, más el SAP por defecto (*Data_Exchange*).

SD	LE	LEr	SD	DA	SA	FC	DSAP	SSAP	DU	FCS	ED
1 b	1 b	1 b	1 b	1 b	1 b	1 b	1 b	1 b	1-32 b (o 1-244 b)	1 b	1 b

Figura 2.5. Cabecera del telegrama de datos.

Dónde:

SD	Delimitador de inicio de trama
LE	Longitud de datos de la red
Ler	Longitud del repetidor
SD	Delimitador de inicio
DA	Dirección de destino
SA	Dirección fuente
FC	Código de función
DSAP	Puntos de acceso de servicio de destino
SSAP	Puntos de acceso de servicio fuente
DU	Datos
FCS	Secuencia de verificación de trama
ED	Delimitador de fin de trama

2.5.3. Funciones de comandos

2.5.3.1. Delimitador de inicio (SD)

El delimitador de inicio identifica el inicio de un telegrama y su formato general. Profibus DP utiliza cuatro tipos de delimitadores de inicio (SD) para telegramas de petición y respuesta, más una quinta respuesta para un corto reconocimiento, como se muestra en la tabla 2.1.

Formato del telegrama	Valor (Hexadecimal)	Longitud del campo de datos (bytes)	Descripción
SD1	10	0	Una estación activa envía este telegrama para buscar nuevas estaciones activas en el bus después de que haya transcurrido el tiempo GAP
SD2	68	1 a 32	Se utiliza en el servicio SRD que permite que los datos de salida sean enviados y los datos de entrada que se recibirán en un solo ciclo de mensajes
SD3	A2	8	Este delimitador se utiliza para los telegramas de datos con una longitud fija de 8 bytes
SD4	DC	0	Este delimitador se usa entre 2 estaciones de bus activos para conceder derechos de acceso al bus
SC	E5	0	La trama de confirmación SC es un mensaje 1 byte que se puede utilizar para reconocer una solicitud de SDA (Sólo en PROFIBUS FMS) o reconocer negativamente una solicitud de SRD.

Tabla 2.1. Formato de telegramas.

2.5.3.2. Longitud del telegrama (LE y Ler)

Este byte especifica la longitud de un telegrama con la longitud de datos variables (es decir, los telegramas SD2), del byte DA hasta el final DU byte (el rango va DU + 5 bytes a 249 bytes), teniendo en cuenta que la longitud de la DU está generalmente limitada

a 32 octetos, pero la norma permite longitudes de hasta 244 bytes. LE se repite en el campo Ler para la protección de datos redundantes.

2.5.3.3. *Dirección de destino y dirección de origen*

El dispositivo maestro se dirige a un dispositivo esclavo específico mediante la colocación de la dirección del esclavo de 8 bits en el campo de la dirección del telegrama (Dirección de destino). Incluye su propia dirección en el campo Dirección SA (Dirección de origen). Las direcciones válidas son 0-127 (00H-7FH). La dirección 126 está reservada para los propósitos específicos y no se puede utilizar para el intercambio de datos del usuario. La dirección 127 se reserva como la dirección de difusión, que todos los dispositivos esclavos reconocidos en una red.

2.5.3.4. *Código de Función o control de la trama*

El código de función (FC) o control Frame especifica el tipo de telegrama (solicitud, respuesta, reconocimiento), tipo de estación (esclavo pasivo o activo o maestro), la prioridad, y el reconocimiento de telegramas (con o sin éxito) de la siguiente manera:

Códigos de función PROFIBUS DP para telegramas de solicitud	
4	SDN bajo (Enviar datos sin de acuse de recibo)
6	SDN alto (Enviar datos sin de acuse de recibo)
7	Reservado para Petición de datos de diagnóstico
9	Solicitar estado FDL con respuesta
12	SRD bajo (Enviar y los datos del petitorio con de acuse de recibo)
13	SRD alta (Enviar y los datos del petitorio con de acuse de recibo)
14	ID de solicitud con la respuesta
15	Solicitud del estatus del LSAP con respuesta

Tabla 2.2. Códigos para solicitud de telegramas.

2.5.3.5. Puntos de acceso de servicio

Los intercambios de datos se manejan en la cabecera del telegrama usando puntos de servicio de acceso (SAP, por sus siglas en ingles). El SAP dice qué datos van a ser transmitido o que función se va a realizar. Sólo los telegramas SD2 y SD3 incluyen campos de datos utilizan los bytes DSAP y SSAP. Recordemos que la transmisión SRD combina un mensaje de salida con una respuesta de entrada en un ciclo de mensaje. El encabezado del telegrama contiene un SSAP (punto acceso de servicio de origen) y/o DSAP (punto de acceso de servicio de destino) que indica los servicios que se ejecutarán.

La inclusión de una entrada DSAP o SSAP en una solicitud o una respuesta de telegrama se identifica estableciendo el bit más alto en el byte de la dirección del DA (Dirección de destino) o SA (Dirección de Origen), respectivamente. Basado en la detección de SAP's , cada estación es capaz de reconocer que datos se han solicitado y que datos de respuesta han sido suministrados. Profibus DP utiliza los SAP's 54 a 62 que se enumeran a continuación, además de la SAP de forma por defecto.

Por defecto SAP=0 Intercambio cíclico de datos (Write_Read_Data)	
SAP54	SAP-Master-a Maestro (Comunicación M-M)
SAP55	Cambio de Dirección de la estación (Set_Slave_Add)
SAP56	Leer entradas (Rd_Inp)
SAP57	Leer salidas (Rd_Outp)
SAP58	Comandos de control a un esclavo DP (Global_Control)
SAP59	Leer datos de configuración (Get_Cfg)
SAP60	Leer datos de diagnóstico (Slave_Diagnosis)
SAP61	Enviar datos de parametrización (Set_Prm)
SAP62	Verificar datos de configuración (Chk_Cfg)

Tabla 2.3. SAP's usados en Profibus DP.

2.6. *Modbus*

MODBUS es un protocolo estándar que puede gestionar una comunicación tipo cliente-servidor entre distintos equipos conectados físicamente con un bus serie. Este protocolo fue ideado para los PLCs Modicon (marca que ahora pertenece a Schneider Electric) en 1979, y con el tiempo se ha convertido en un protocolo muy empleado en las comunicaciones industriales. Las principales razones de ello son la sencillez del protocolo, versatilidad, y que sus especificaciones, gestionadas por la MODBUS Organization, son de acceso libre y gratuito. MODBUS es un protocolo de tipo Petición/Respuesta, por lo que en una transacción de datos se puede identificar al dispositivo que realiza una petición como el cliente o maestro, y al que devuelve la respuesta como el servidor o esclavo de la comunicación. En una red MODBUS se dispone de un equipo maestro que puede acceder a varios equipos esclavos. Cada esclavo de la red se identifica con una dirección única de dispositivo (Pefhany 2000).

Un maestro puede hacer dos tipos de peticiones a un esclavo: para enviar datos a un esclavo y espera su respuesta confirmación, o para pedir datos a un esclavo y espera su respuesta con los datos. Las peticiones de lectura y escritura que envía un maestro llevan asociado un código de función que el esclavo debe ejecutar. Según ese código, el esclavo interpretará los datos recibidos del maestro y decidirá qué datos debe devolver. Los códigos de función dependen de los dispositivos y de las tareas que estos pueden realizar.

2.6.1. *Modo ASCII*

Los datos se codifican como caracteres ASCII entre el “0” (30H) y el “9” (39H) y entre “A” (41H) y “F” (46H). Por ejemplo, si se requiere enviar el byte de valor FFH, se tiene que enviar la cadena “FF”, por lo que realmente se enviarían dos bytes: 46H y 46H. Además se utilizan 3 caracteres especiales. El carácter “:” (3AH) se emplea para marcar el comienzo de la trama y el par de caracteres no imprimibles “CR LF” (0DH, retorno de carro, y 0AH, salto de línea) se emplean como delimitador del fin de la trama.

Este formato tiene dos grandes ventajas. Primero, ofrece una facilidad de detección del principio y del fin de trama gracias a los campos de inicio y fin (caracteres “:”

y “CR LF”), con independencia de los tiempos de la transmisión del canal de comunicación. Segundo, permite trabajar con equipos de procesamiento lento sin tener que bajar la velocidad de comunicación siempre que tengan buffers de almacenamiento de los datos recibidos. Los inconvenientes son que requiere un mayor ancho banda que MODBUS RTU para el envío de la misma petición o respuesta, o visto de otra manera, para el mismo ancho de banda, el envío de una trama con ASCII es más lento que con RTU. La trama del modo ASCII se presenta en la fig. 2.6.

Inicio	Dirección esclavo	Función	Datos	LRC	CR	LF
3AH	2 caracteres	2 caracteres	Depende de la función	2 caracteres	ODH	OAH

Figura 2.6. Trama Modbus en modo ASCII.

2.6.2. *Modo RTU*

MODBUS RTU (Remote Terminal Unit) se caracteriza por que los bytes se envían en su codificación binaria plana, sin ningún tipo de conversión. Está inicialmente pensado para comunicaciones en bus serie. Como ventaja principal tiene el buen aprovechamiento del canal de comunicación, mejorando la velocidad de la transmisión de los datos. El inconveniente es que requiere una gestión de tiempos entre bytes recibidos para saber cuándo empiezan y terminan las tramas. Con la trama MODBUS RTU, la delimitación de la misma se realiza por intervalos de tiempo de caracteres de silencio, como muestra la Figura 14. Un carácter de silencio tiene la duración de un byte de datos enviado por el medio, pero no transporta datos, y su duración (T) depende de la velocidad (Vt) y del número bits que se usen para su codificación (N) según $T=N/Vt$. Según el estándar de MODBUS, para velocidades de hasta 19.200bps, el tiempo entre tramas debe ser como mínimo 3,5 veces la duración de un carácter, y para velocidades superiores se recomienda un tiempo fijo de 1,75ms. Por ejemplo, para una configuración del puerto serie de 19.200bps, con un bit de parada y un bit de paridad (11 bits en total, sumando el de inicio y 8 de datos) se tiene: $3,5 \cdot 11 / 19.200 = 2ms$.

La trama MODBUS RTU incorpora un código Cyclical Redundancy Check (CRC) de 16 bits (ver Figura 14) para poder detectar errores, que debe ser calculado por el emisor a partir de todos los bytes de la trama enviados antes del CRC, exceptuando los delimitadores. Para ello se usa un algoritmo específico, bien definido en la especificación de MODBUS serie. El receptor debe volver a calcular el código de igual forma que el emisor, y comprobar que el valor obtenido del cálculo es igual al valor presente en la trama para poder validar los datos.

Inicio	Dirección esclavo	Función	Datos	CRC	Final
3.5 caracteres de intervalo de espera	1 byte	1 byte	Depende de la función	2 bytes	3.5 caracteres de intervalo de espera

Figura 2.7. Trama Modbus en modo RTU.

2.6.3. Códigos de funciones

La siguiente tabla muestra las funciones más utilizadas en las peticiones y respuestas de MODBUS, con sus códigos

Código decimal	Código HEX	Función Tipo de datos	Tipos de datos
1	01H	Leer estado de marcas y salidas digitales (bobinas)	Bit
2	02H	Leer estado de entradas digitales	Bit
3	03H	Leer registros	16 Bits
4	04H	Leer entradas analógicas	16 Bits
5	05H	Forzar valor de una salida digital (bobina)	Bit
6	06H	Establecer valor de un registro	16 Bits
15	0FH	Forzar múltiples marcas o salidas digitales (bobinas)	Bit
16	10H	Establecer múltiples registros	16 Bits

Tabla 2.4. Código de funciones Modbus.

CAPÍTULO 3. METODOLOGÍA

La metodología empleada para el desarrollo del prototipo se organiza en tres secciones principales: En primer lugar se describe en la sección 3.2.1, la arquitectura funcional de la plataforma de comunicación y los elementos que la componen, en la sección 3.2 se presenta el *Hardware* y finalmente en la sección 3.3 el *software* que se compone de los algoritmos implementados.

3.1. Arquitectura de la plataforma

El objetivo principal del diseño ha sido el desarrollo de una plataforma de comunicación que permita implementar algoritmos para el control de datos en robots manipuladores, la estructura modular de la plataforma cuenta con una arquitectura de comunicación derivada del estándar internacional IEEE 802.3, basado en una topología de bus. La estructura de arquitectura está basada en capas de tareas, las cuales son presentadas en la Fig. 3.1.

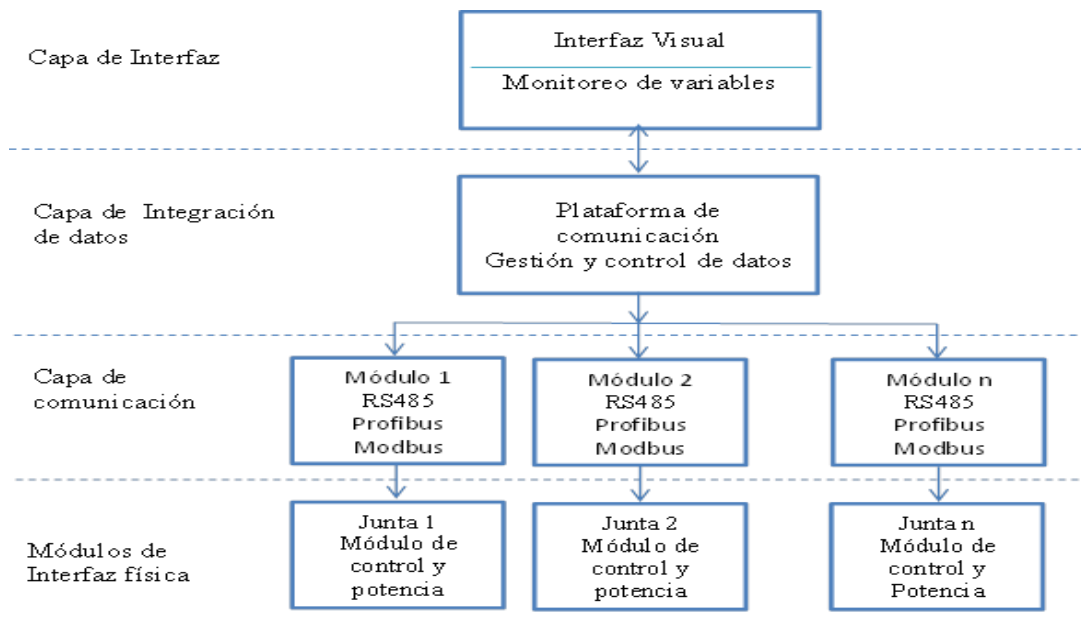


Figura 3.1. Capas de la arquitectura.

3.1.1. Interfaz de Usuario

En esta capa el usuario desarrolla el monitoreo de variables de posición y velocidad del manipulador, en la cual también se puede observar la información procedente de las leyes de control como las ganancias proporcionales, integrales y derivativas, y la información de la posición de los motores, la interfaz es desarrollada con lenguaje de programación visual gráfico LabVIEW.

3.1.2. Integración de datos

Sobre esta capa se realiza la organización de toda la información procedente de la capa superior para enviarlos a las capas inferiores. Se establecen los casos para incluir un nuevo módulo (servoamplificador), ya que será necesario añadir una estructura de control para este nuevo dispositivo, lo cual se realiza a través de una aplicación en la interfaz de usuario que gestiona las unidades de potencia y control, así como previene movimientos irregulares y que pongan en peligro el control sobre componentes de niveles inferiores, sobre esta capa se establecen interrupciones prioritarias para evitar que procesos de gran importancia queden en un estado de espera.

3.1.3. Capa de comunicación

La capa de comunicación controla la transferencia de datos que va dirigido a cada dispositivo que controla cada junta mediante alguno de los protocolos implementados en el prototipo, por medio del cual se realiza una interconexión a través de una topología de bus, con lo que se controlan los 6 dispositivos de las juntas, para poder transportar los datos a sus respectivos destinos se ha implementado el protocolo de ventanas deslizantes ya que permite la transmisión de múltiples segmentos de información antes de comenzar la espera para que el receptor le confirme la recepción de los segmentos, también se implementa el protocolo CSMA/CD (por sus siglas en inglés Carrier Sense Multiple Access with Collision Detection) o, en español, acceso múltiple con escucha de portadora y detección de colisiones para prevenir la colisión de datos (Beuerman and Coyle 1988).

3.1.4. Módulos de interfaz física

Sobre esta capa se implementan los dispositivos electrónicos necesarios para acoplar aquellos servoamplificadores que no cuentan con una interfaz de comunicación en su *hardware*, como es el caso de los servoamplificadores ADS 50/10. Como componente principal del prototipo y encargado de realizar las operaciones y procesos de la plataforma se tiene un procesador digital de señales PIC18F97J60 producido por Microchip que opera a 16 bits, a 48 MHz, el cual soporta el estándar IEEE 802.3. El firmware está basado en lenguaje de alto nivel C, se ha estructurado de forma modular, de manera que facilite al usuario realizar modificaciones necesarias de acuerdo a los requerimientos del proceso. La plataforma también se compone, módulos para interfaz Ethernet, RS485 y Profibus, se incorpora un módulo de programación para proporcionar un sistema abierto en la escritura del código del microprocesador, la composición de esta arquitectura se muestra en la fig. 3.2. Esta arquitectura fue elegida porque a través del bus RS232 se puede adicionar m'as transceptores que permitan la integración de más protocolos, como CAN, Interbus o SERCOS.

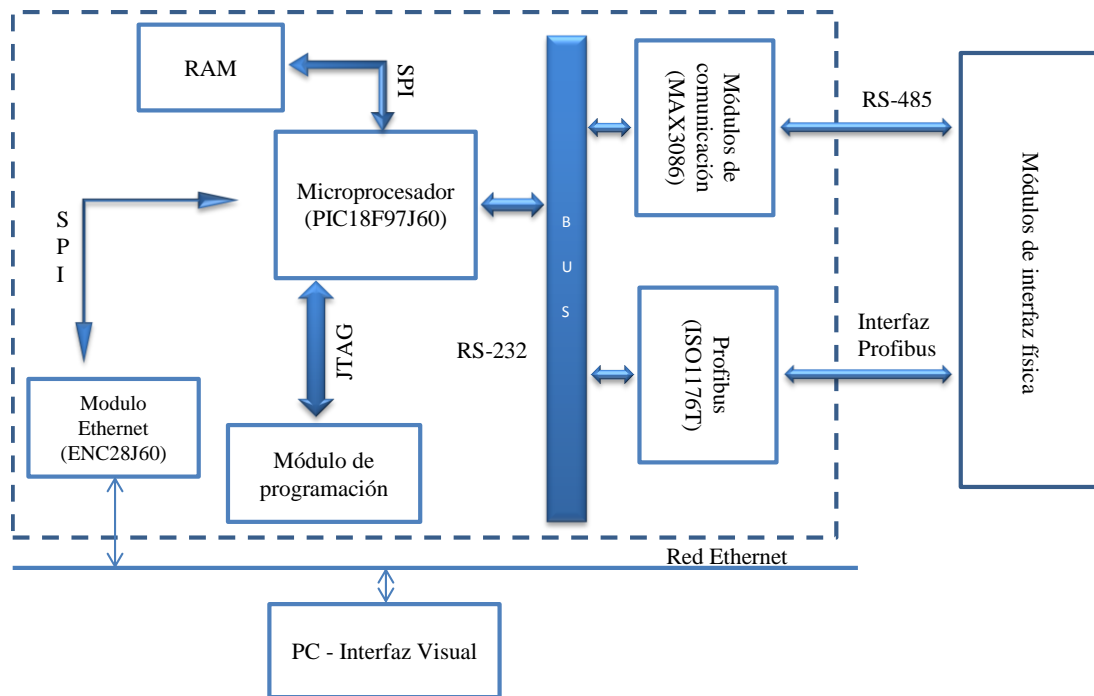


Figura 3.2. Arquitectura de la plataforma de comunicación.

3.1.5. Comparación con el modelo OSI

Para tener una guía de la estructuración de la plataforma de comunicación se tomó como referencia el modelo OSI, en la tabla 3.1 se describen los protocolos y métodos empleados para comunicar la plataforma con el usuario y los dispositivos esclavos (servo amplificadores), como se puede apreciar las capas 5,6 y 7 recaen en la interfaz de usuario que para este caso se usa LABView, pero esta puede ser remplazada por cualquier dispositivo que se pueda comunicar a la IP. 192.168.1.111 y enviar datos a través del puerto 80, ya que la interfaz solo es un medio para visualizar las operaciones que ejecuta el MCU.

Capa	Conexión Usuario – Plataforma	Conexión Plataforma – Dispositivos Esclavos	
7. Aplicación	LABView / HTTP	Nivel de Bit	
6. Presentación			
5. Sesión			
4. Transporte	TCP	Control de mensajes y gestión de datos por tramas	
3. Red	IP,ARP,ICMP		
2. Enlace	Ethernet	CSMA/CD, ACK	
1. Física		RS485 /MDB	Profibus /MDB

Tabla 3.1. Comparación con el modelo OSI.

En un análisis de la tabla 3.1, se puede apreciar que la plataforma cumple con el estándar del modelo OSI, ya que se programaron funciones que realizan los procesos de cada capa correspondiente.

Una vez comprendida la forma en que se compone la arquitectura física de la plataforma, en la sección nombrada *hardware* se describen los componentes físicos implementados en el prototipo y que ayudan a este a realizar las funciones destinadas a la gestión, comunicación y transporte de datos.

3.2. Hardware

3.2.1. Unidad de Control Maestra (MCU)

Como unidad de control maestra se ha seleccionado el microprocesador PIC18F97J60 que controla la transmisión y flujo de datos entre la interfaz y los servoamplificadores que manipulan los ejes del robot, este dispositivo contiene los algoritmos para el manejo de los protocolos MODBUS, PROFIBUS, RS-485, así como la construcción de tramas, algoritmos para el flujo de datos y los algoritmos para detección de colisiones. Se eligió este microcontrolador porque posee las siguientes características, opera a 16 bits, a 41.3 MHz, incluye memoria de programa de 128K y una memoria no volátil de 3904 bytes para almacenar información, además cuenta con 8192 bytes en buffer para la transmisión de datos en Ethernet, el cual soporta el estándar IEEE 802.3. En la fig. 3.3 se muestra el prototipo con los requerimientos mínimos para su funcionamiento.

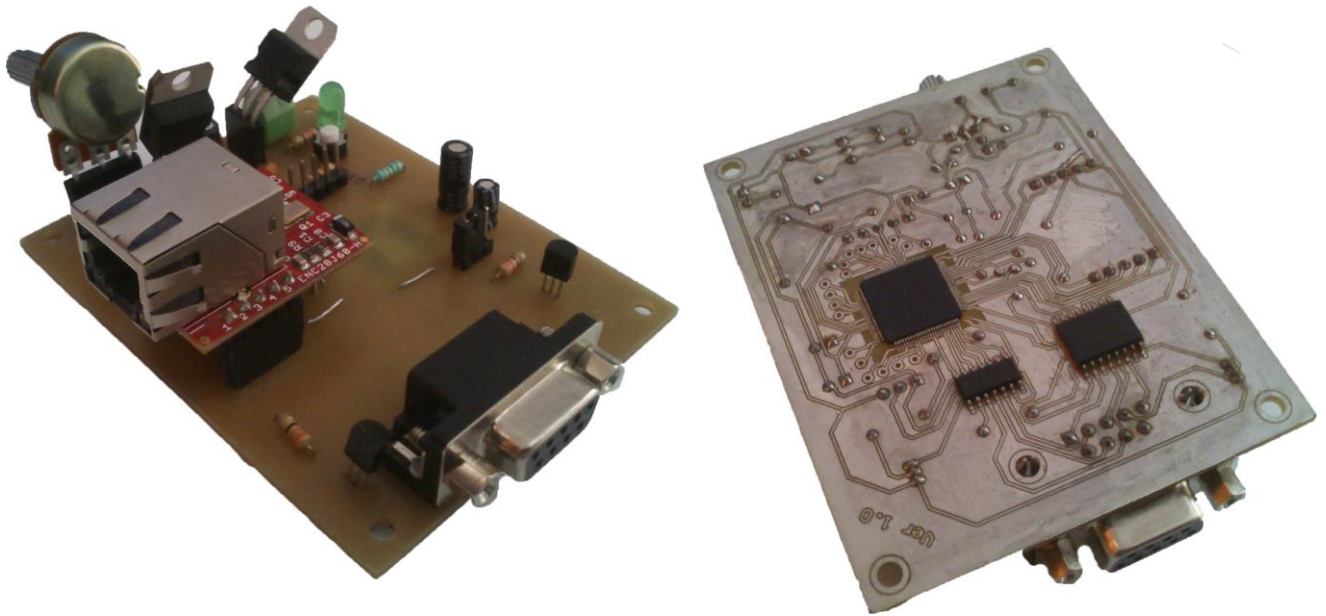


Figura 3.3. PIC18F97J60 como MCU.

3.2.2. Controlador Ethernet

Para el propósito de transmisión de datos y como etapa intermedia que permite realizar la conexión entre la interfaz de usuario y el prototipo de la plataforma de comunicación es un módulo controlador Ethernet (ENC28J60) del fabricante Microchip. Dispositivo que soporta los protocolos TCP/IP, FTP, UDP, HTTP, SMTP, entre otros. Este controlador permite una conexión Ethernet 10BaseT a una velocidad máxima de 10 Mb/s, soportando una comunicación tanto *half-duplex* como *full-duplex*. El fabricante Microchip ofrece una serie de instrucciones para el control de este módulo de forma libre; una interfaz SPI sirve como un canal de comunicación entre la unidad maestra de control (*MCU*, *PIC18F97J60*) y el controlador ENC28J60, esta conexión y los canales de transmisión de datos se muestran en la fig. 3.4, mientras que en la fig. 3.5 se muestra la conexión física en el prototipo.

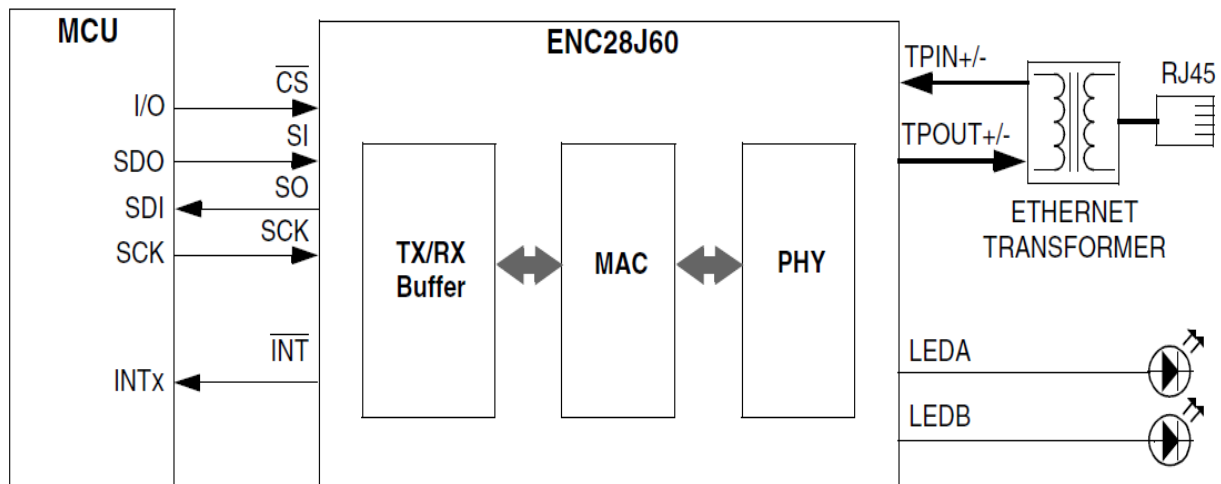


Figura 3.4. Interfaz basada en ENC28J60 (imagen tomada de ENC28J60 Datasheet).

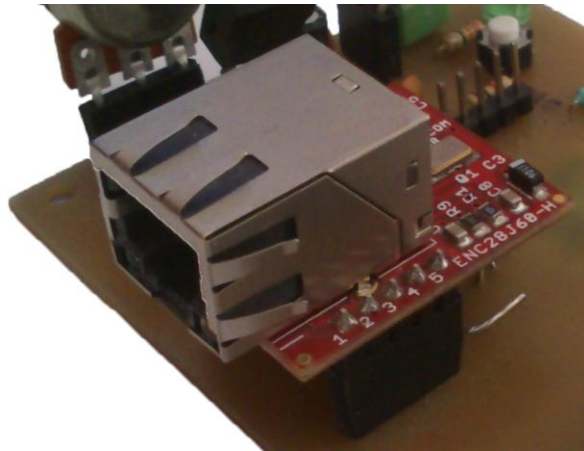


Figura 3.5. Conexión física del controlador Ethernet.

3.2.3. Transceptor RS-485

Para establecer una comunicación a través del protocolo RS-485, se usa el transceptor de alta velocidad MAX3089 que contiene módulos de transmisión y recepción de datos para comunicaciones RS-485/RS-422, con velocidad transmisión de datos libre de errores de hasta 10Mbps, permitiendo conectar hasta 256 dispositivos al bus con comunicaciones *half-duplex* o *full-duplex*, este dispositivo es ideal para la comunicación con servoamplificadores como el tipo EDB-30^a que se basa en este estándar de comunicación, la conexión entre dispositivos para comunicación RS-485 se muestra en la fig. 3.6.

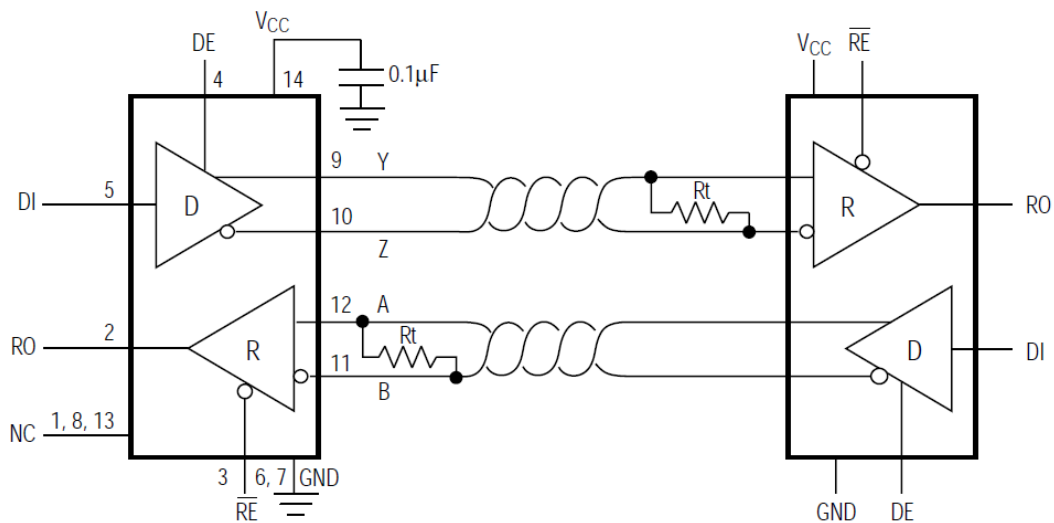


Figura 3.6. Configuración para operación *full-duplex* (imagen tomada de MAX3089 datasheet).

En la fig. 3.7 se muestra la conexión entre el transceptor para RS485 y profibus con el MCU

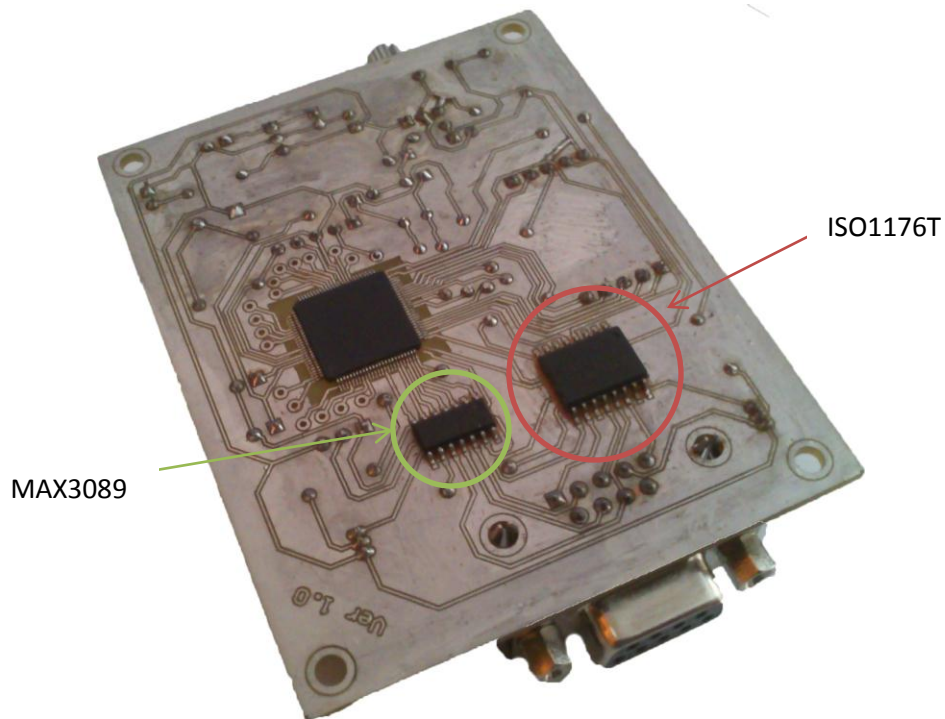


Figura 3.7. Conexión física del MAX3089 e ISO1176T.

3.2.4. *Transceptor Profibus*

El ISO1176T es un transceptor con línea diferencial aislada, ideal para largas líneas de transmisión debido a que el bucle de tierra permite que el dispositivo funcione con un rango de voltaje de modo común mucho más grande. Este circuito integrado ha sido diseñado para la comunicación bidireccional de datos en las líneas de bus de transmisión multipunto permitiendo velocidades de hasta 40 Mb/s, con reducción de riesgo en la corrupción de datos y daños en los circuitos de control de altos costos, su implementación se puede apreciar en la fig. 3.7.

3.2.5. Módulos de interfaz física

Estos módulos consisten en un microcontrolador que controla la recepción y envío de datos a través de los protocolos implementados, además cada módulo contiene un controlador PID que permite el control de posición sobre cada eje, un módulo para captura de datos del encoder y conversión grey a decimal, y un módulo de potencia para controlar motores que no están acoplados a servoamplificadores. La arquitectura de los módulos de interfaz física que hasta ahora han sido implementados en protoboard, dicha arquitectura se muestra en la fig. 3.8.

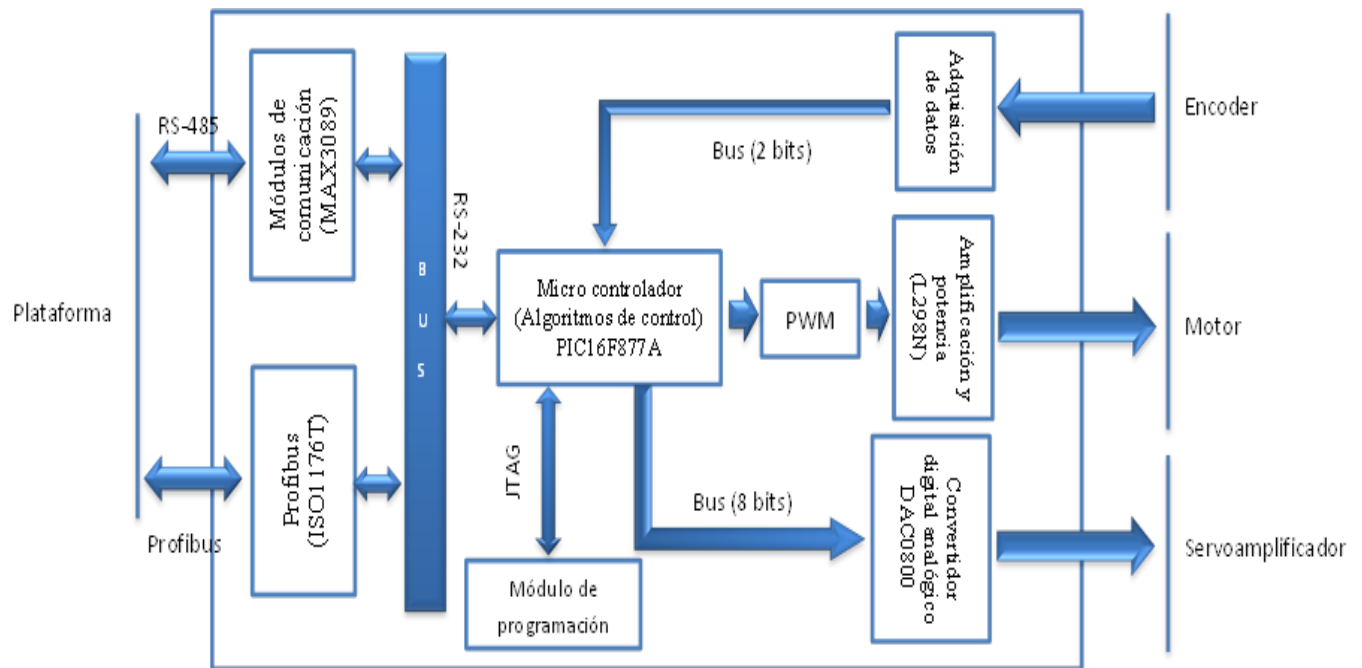


Figura 3.8. Arquitectura de los módulos de interfaz física.

3.3. Software

Se han desarrollado algoritmos para el manejo de los protocolos de comunicación Modbus en sus versiones RTU y ASCII, Profibus en su versión DP, RS-485, Ethernet, porque ofrecen diversas características como sencillez en las comunicaciones, versatilidad, alta aceptación en la industria, altas velocidades para transmisión de datos, manejo eficiente de colisiones, amplia conexión entre dispositivos actuadores, permiten intercomunicación entre diversas estaciones de trabajo entre muchas más, a continuación se anexan los diagramas de flujo y se descripción de los algoritmos implementados.

3.3.1. Modbus en el protocolo RS485

Modbus es un protocolo de comunicaciones estándar basado en la arquitectura maestro/esclavo. Existen dos variantes, Modbus RTU que es una representación binaria compacta de los datos y Modbus ASCII que es una representación legible del protocolo pero menos eficiente. Se ha empleado Modbus para el enlace de datos entre el MCU y los módulos de interfaz física.

En la fig. 3.9 se muestra el diagrama de bloques para transmisión de datos mediante Modbus, en el primer bloque de proceso llamado Modo ASCII o RTU, se selecciona el modo de operación de Modbus y se inicializan los dispositivos correspondientes, configura el canal de transmisión, velocidad, etc. El segundo bloque de proceso construye la trama de acuerdo a la configuración de Modbus (ASCII o RTU) y almacena los datos en una pila, en el bloque condicional se ejecuta la verificación del canal, para revisar que otro dispositivo no se encuentre enviando datos, en caso de que el canal de transmisión se encuentre ocupado se espera el tiempo de envío mayor o igual a $3 \frac{1}{2}$ caracteres, tomando en cuenta que a una velocidad de 10 Mbps se requieren 100 ns para transmitir un bit, tiempo que depende del estado del medio de transmisión (tipo de cable, longitud), en caso de que el canal se encuentre libre, en el bloque de proceso envío de trama

se transmiten todos los datos almacenados en la pila, y se verifica si ocurrió colisión de datos (proceso correspondiente al bloque condicional), si ocurrió alguna colisión se vuelve al bloque de retardo para comenzar el proceso, en caso contrario se espera a recibir el ACK (trama de respuesta), una vez recibido el ACK se finaliza la función de transmisión.

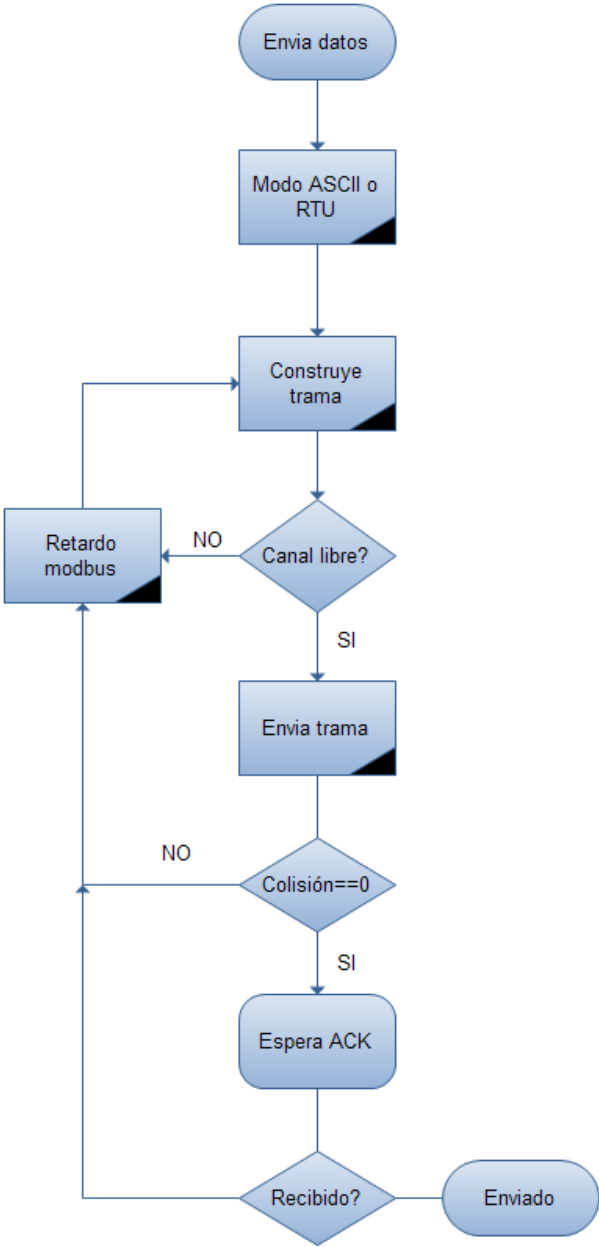


Figura 3.9. Diagrama de bloques para transmisión Modbus.

En la figura 3.10, se muestra el diagrama de bloques para recepción de datos mediante Modbus, en el primer bloque de proceso llamado esperar indicación, se espera a recibir alguna trama enviada por otro dispositivo, cuando se recibe, en el bloque de evento nombrado validar código de función se verifica que el código recibido corresponda a alguna de las funciones implementadas por Modbus (Ver sección 2.6.3), si no corresponde se activa el bloque de excepción 1, si la función es válida se verifica que la dirección de los datos sea la correspondiente a la dirección del dispositivo, en caso de que resulte invalida se activa el bloque de excepción 2 correspondiente a este proceso, pero si la dirección corresponde se verifica la función a realizar por el dispositivo en caso de no encontrarse implementada la función requerida se activa el bloque de excepción 3, cuando se ha completado este proceso se realiza una nueva verificación para determinar si la función fue ejecutada en caso de no ser así se activa el bloque de excepción 4, de otra forma se envía una trama de respuesta terminando con ello este proceso de recepción, si en durante el proceso se activó cualquier bloque de excepción se interrumpe el proceso y se finaliza enviando una trama con el código de error o excepción generado.

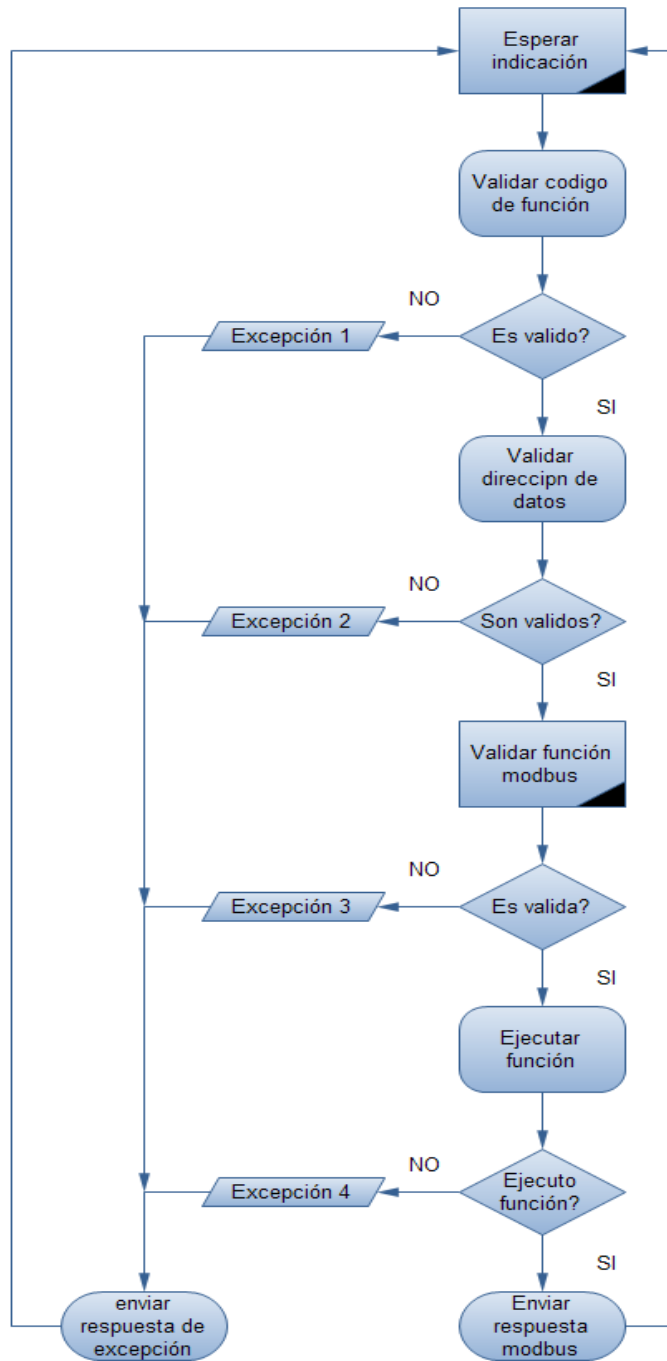


Figura 3.10. Diagrama de bloques para recepción de datos Modbus.

3.3.2. Diagrama de flujo del algoritmo de control para flujo de datos por TCP/IP

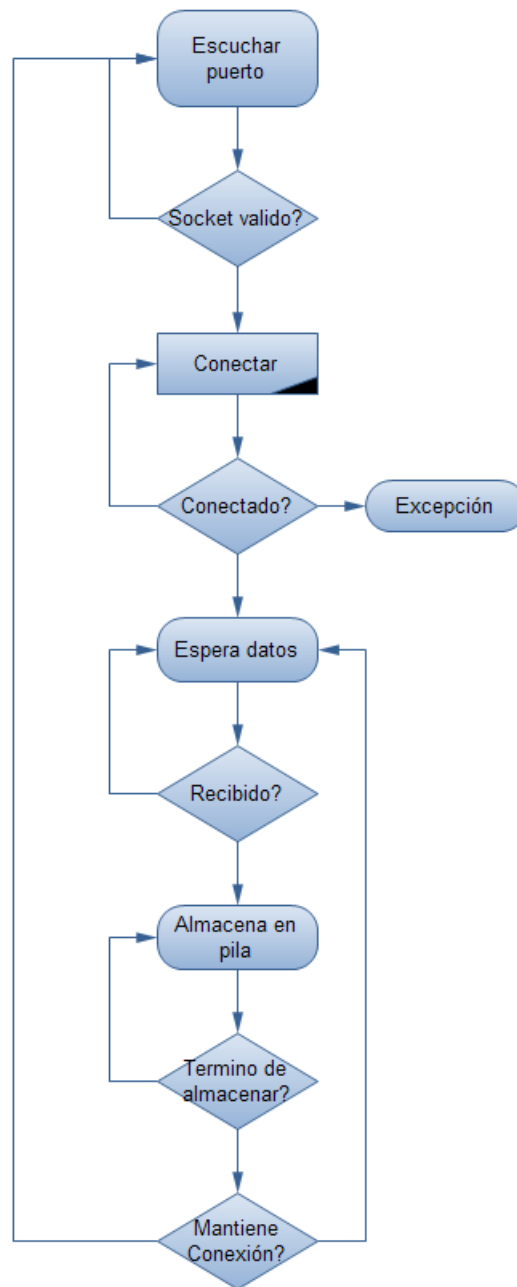


Figura 3.11. Diagrama de flujo recepción TCP/IP.

En la fig. 3.11, se aprecia el diagrama de recepción de datos por TCP, estos datos son recibidos del usuario y enviados a los dispositivos esclavos, en el primer bloque de procesos el algoritmo se mantiene escuchando un puerto especificado para este caso el puerto 80, correspondiente al protocolo de transferencia de hipertexto (HTTP), cuando entra una solicitud de conexión de algún dispositivo cliente (A través de la interfaz de usuario) se verifica el puerto por el cual el dispositivo cliente intenta realizar la conexión cuando se valida el puerto (bloque condicional llamado, socket valido?), en el siguiente bloque de proceso se realizan los procesos necesarios para establecer conexión con él dispositivo cliente, en caso de transcurrir cierto tiempo sin poder establecer conexión se ejecuta un bloque de excepción que cancela los intentos de conexión e inicia la escucha del puerto, en caso contrario, si se logra conexión se procede a recibir los datos (bloque de espera de datos), cuando se comienza a recibir datos estos comienzan a almacenarse en una pila, se verifica si la trama llego completa y se terminó de recibir los datos (bloque condicional nombrado termino?), en caso de no haber terminado sigue almacenando datos en la pila, finalmente se verifica si el dispositivo mantiene conexión en caso de ser así procede a ir al estado de espera de datos, en caso contrario vuelve al inicio de la secuencia escuchando por el puerto predestinado.

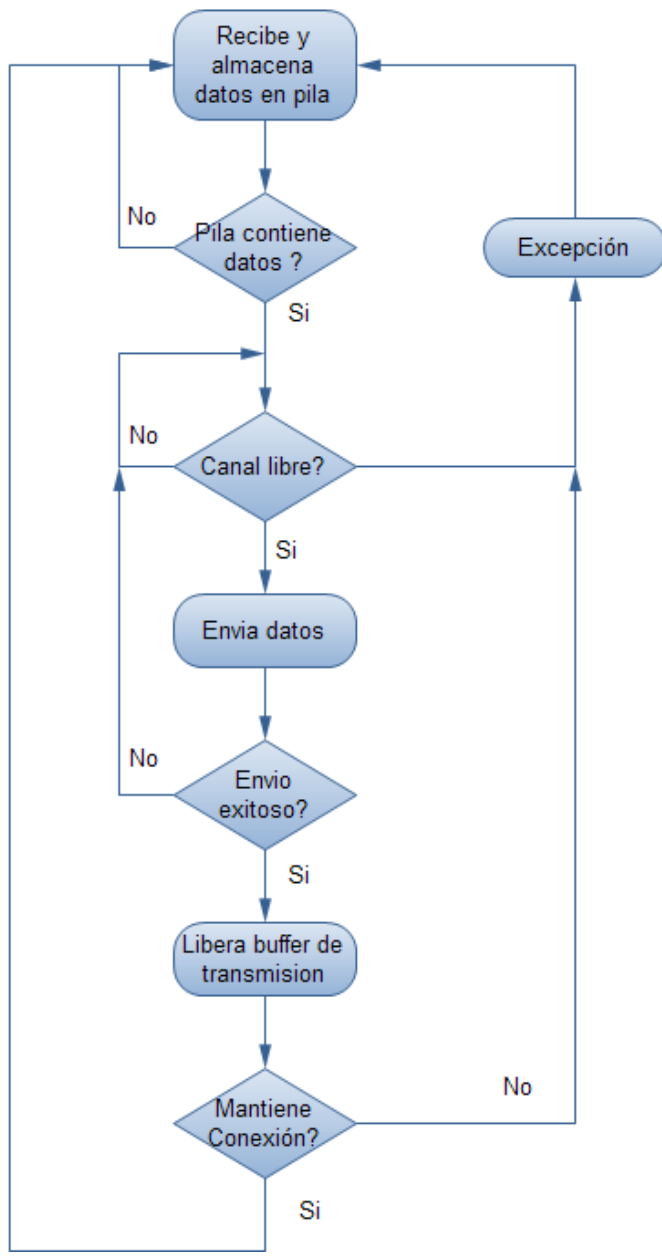


Figura 3.12. Diagrama de flujo transmisión TCP/IP.

El diagrama mostrado en la figura 3.12 muestra la transmisión de datos por TCP, los cuales son provenientes de los dispositivos esclavos y que son enviados al usuario, en el primer bloque todos los datos recibidos son almacenados en una pila para conservarlos y que nuevos datos no sobre escriban datos antiguos, cuando hay datos contenidos en la pila se activa una bandera que es verificada en el bloque condicional siguiente; donde se verifica si la trama de datos son coincidentes o son datos basura, si los datos no son los correctos se desechan y el proceso regresa al almacenamiento en pila, cuando los datos coinciden se verifica que el canal para transmitir se encuentre libre en el caso en que no se encuentre libre se seguirá verificando en un proceso en segundo plano para permitir que el programa siga con su proceso normal, cuando el canal está libre se envían los datos y se verifica si el envío fue exitoso, cuando no lo es el programa retrocede hasta la verificación del canal para una nueva transmisión, cuando es exitoso se libera el buffer de transmisión esto es importante para no almacenar datos basura en la pila de salida, finalmente se verifica que la conexión se mantiene, cualquier caso en que la conexión se haya perdido o después de varias respuestas negativas ante el canal libre se activa una excepción que verifica conexión y en caso de estar desconectado termina el proceso para realizar una nueva conexión.

3.3.3. Algoritmo para uso del protocolo Profibus

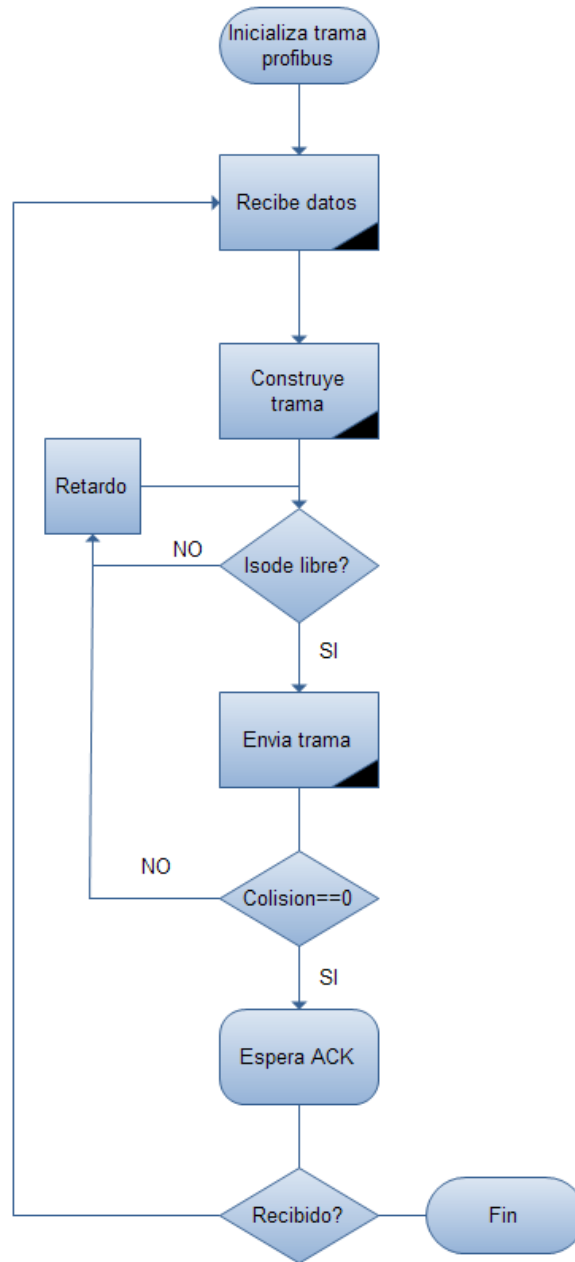


Figura 3.13. Diagrama de flujo de protocolo profibus.

Debido a que profibus solo se basa en las dos primeras capas del modelo OSI (enlace y física), se puede usar el algoritmo CSMA/CD para verificar que el dispositivo esclavo haya recibido los datos de forma correcta, en la fig. 3.13, describe la forma de enviar los datos con profibus, el primer bloque se encarga de inicializar la trama configurando el delimitador de inicio, dirección de destino y origen, delimitador de fin y códigos de función, el siguiente bloque almacena datos de forma temporal en una pila, lo que permitirá reenviar los datos almacenados en caso de no haberse realizado una transmisión exitosa, a continuación se construye la trama de acuerdo con el protocolo profibus, el transceptor que acondiciona los voltajes cuenta con un canal especial para verificar que algún otro dispositivo no se encuentre transmitiendo datos, en el bloque condicional se verifica si el canal se encuentra libre, si es así se envían los datos y se verifica que no ocurran colisiones, en caso contrario se genera un retardo antes de reintentar un nuevo envío, cuando se recibe el ACK de verificación el ciclo termina y el proceso queda dormido, hasta que haya nuevos datos para enviar, en caso de que no se reciba ACK se repite el proceso hasta que la transmisión se haya completado.

3.3.4. Diagrama de flujo Algoritmo CSMA/CD

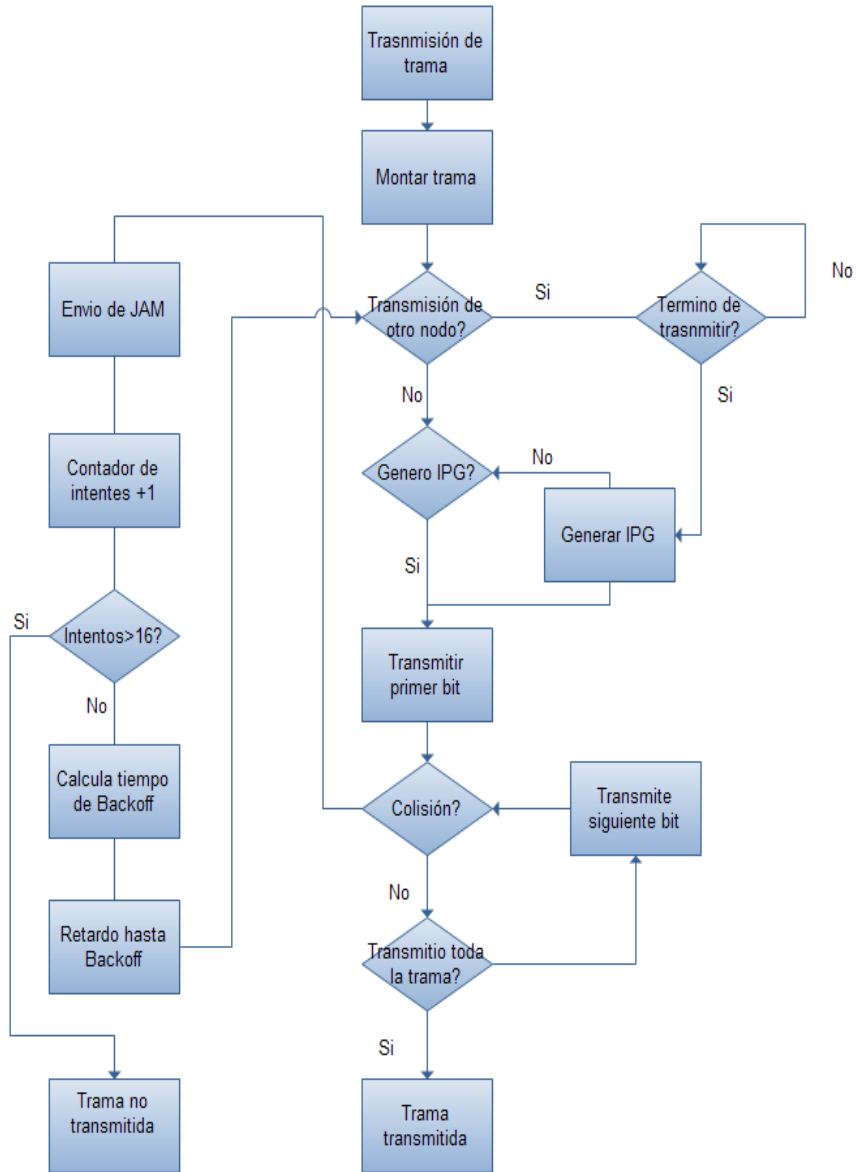


Figura 3.14. Diagrama de bloques del algoritmo CSMA/CD.

La figura 3.14, muestra el algoritmo CSMA/CD en el primer bloque de proceso se monta la trama de datos almacenándolos en una pila, se inicia el contador de intentos en 0, en el bloque condicional se verifica si algún dispositivo se encuentra transmitiendo datos, en caso de que otra estación este trasmitiendo datos se mantiene en espera hasta que las estación transmisora deje de transmitir, después del realizar esta verificación en el bloque de proceso llamado Generar IPG, se genera un retardo de n segundos aleatorios, al generarse este retardo se transmite el primer bit y se verifica que no haya colisión de datos y se trasmiten los bits restantes hasta haber transmitido la trama completa, en caso de que se genere alguna colisión de datos se detiene las transmisión y se envía una trama de JAM a las otras estaciones para indicar que ocurrió una colisión y que no transmitan datos, después de enviar la trama de JAM se aumenta el contador de intentos y se verifica que sean menor a 16, acto seguido se calcula el tiempo de retardo *Backoff* cuando ha ocurrido este retardo se prepara la trama para el intento de una nueva transmisión, en caso de que el número de intentos supere los 16 establecidos se termina este proceso indicando que se generó un error en la transmisión.

3.3.5. Trama de datos para protocolo RS485

Como es sabido el protocolo RS485 es el medio físico por el cual se transmiten los datos de forma serial, por lo cual carece de una estructura para su transporte, por lo que se definieron estructuras, funciones y una trama para organizar y transportar la información de manera correcta, la trama definida es la siguiente:

Preámbulo	Origen	Longitud	Paquete	LRC	Terminador
1 byte	1 byte	2 bytes	45 – 1530 bytes	1 byte	1 byte

Donde el preámbulo en lugar de usarlo como sincronizador de datos es usado para indicar el inicio de la trama y toma el valor 7EH. El segmento de paquete se divide de la siguiente forma:

DADR 1	LEN 1	DAT 1	TMR 1	...	DADR n	LEN n	DAT n	TMR n
--------	-------	-------	-------	-----	--------	-------	-------	-------

DADR: Dirección del dispositivo esclavo al cual se dirigen los datos.

LEN: Longitud de los datos para el respectivo dispositivo

DAT: Contiene los datos necesarios para los dispositivos, para esta tesis las ganancias K_p , K_d , K_i y el setpoint de un controlador proporcional, integral y derivativo.

TMR: Es un terminador interno que indica el termino de datos para los respectivos dispositivos.

LRC es una comprobación de redundancia longitudinal (Longitudinal Redundancy Check) es un método de detección de errores para determinar la exactitud de los datos transmitidos y almacenados y finalmente Terminador que es un byte asignado como 0AH que indica el final de la trama completa.

Capítulo 4. RESULTADOS

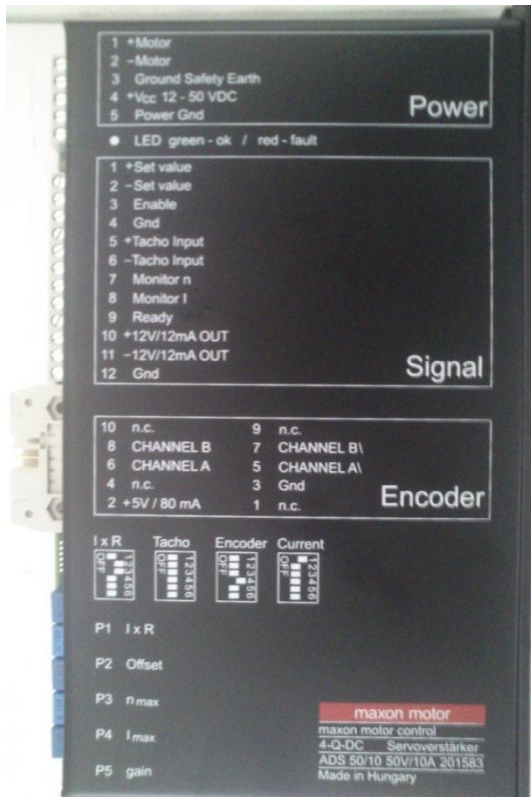
4.1. Conexión de la plataforma a la mesa de coordenadas

Para probar el funcionamiento de la plataforma es necesario un mecanismo con más de dos ejes, por lo que la mesa de coordenadas de la fig. 4.1 es ideal para realizar las pruebas requeridas ya que cuenta con tres ejes, este sistema actualmente está funcionando con una tarjeta de control de movimiento del modelo DMC-40x0 producido por Galil Motion Control.

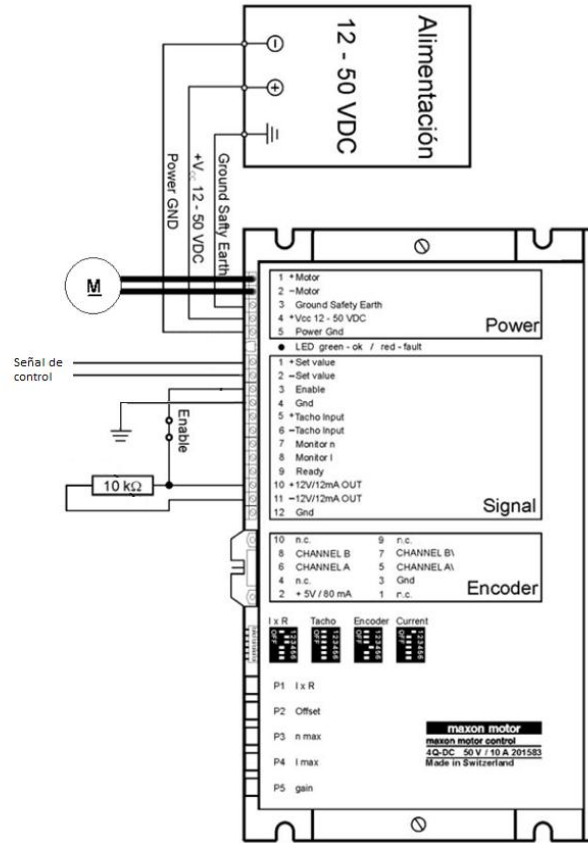


Figura 4.1. Mesa de coordenadas TX8.

Los servoamplificadores que se encargan de darle la potencia a sus motores son los ADS 50-10 producidos por maxon motor control (ver fig. 4.2-a), este tipo de servoamplificador recibe una señal de control diferencial de -10 a +10 VCD, las conexiones con los motores se muestran en la fig. 4.2-b.



a)



b)

Figura 4.2. Servoamplificador ADS 50-10.

A continuación se retiró la tarjeta galil y se conectó el prototipo desarrollado, conectando la interfaz RS485 con cada módulo de interfaz física, en la fig. 4.3 se muestra un esquemático de la conexión, mientras que en la fig. 4.4, se muestran los módulos de interfaz en protoboard uno por cada eje de la mesa. Las señales de los encoders de los servomotores se toman como retroalimentación para un lazo cerrado de un controlador PID implementado en cada microcontrolador, los cuales envían la señal de control que va dirigida a los servoamplificadores, esta señal controlará la velocidad y sentido de giro de cada motor.

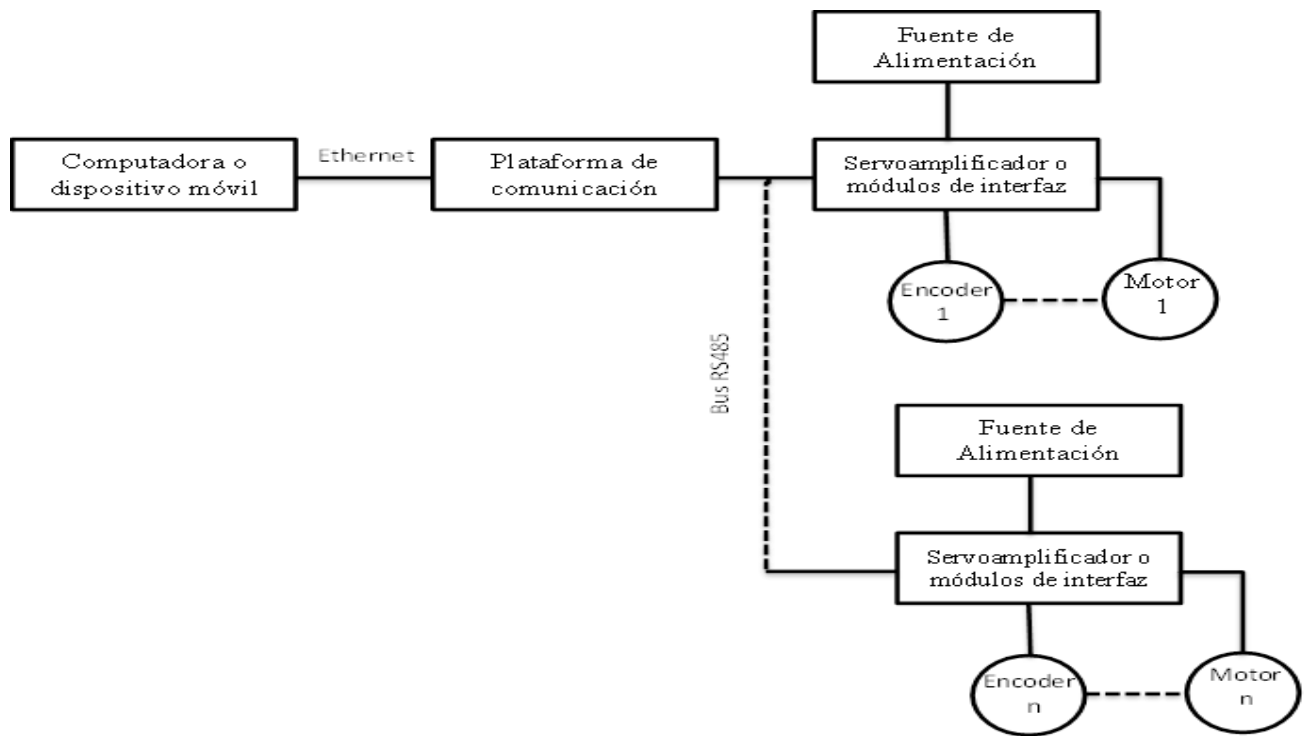


Figura 4.3. Esquema de conexión de la plataforma de comunicación a los amplificadores.

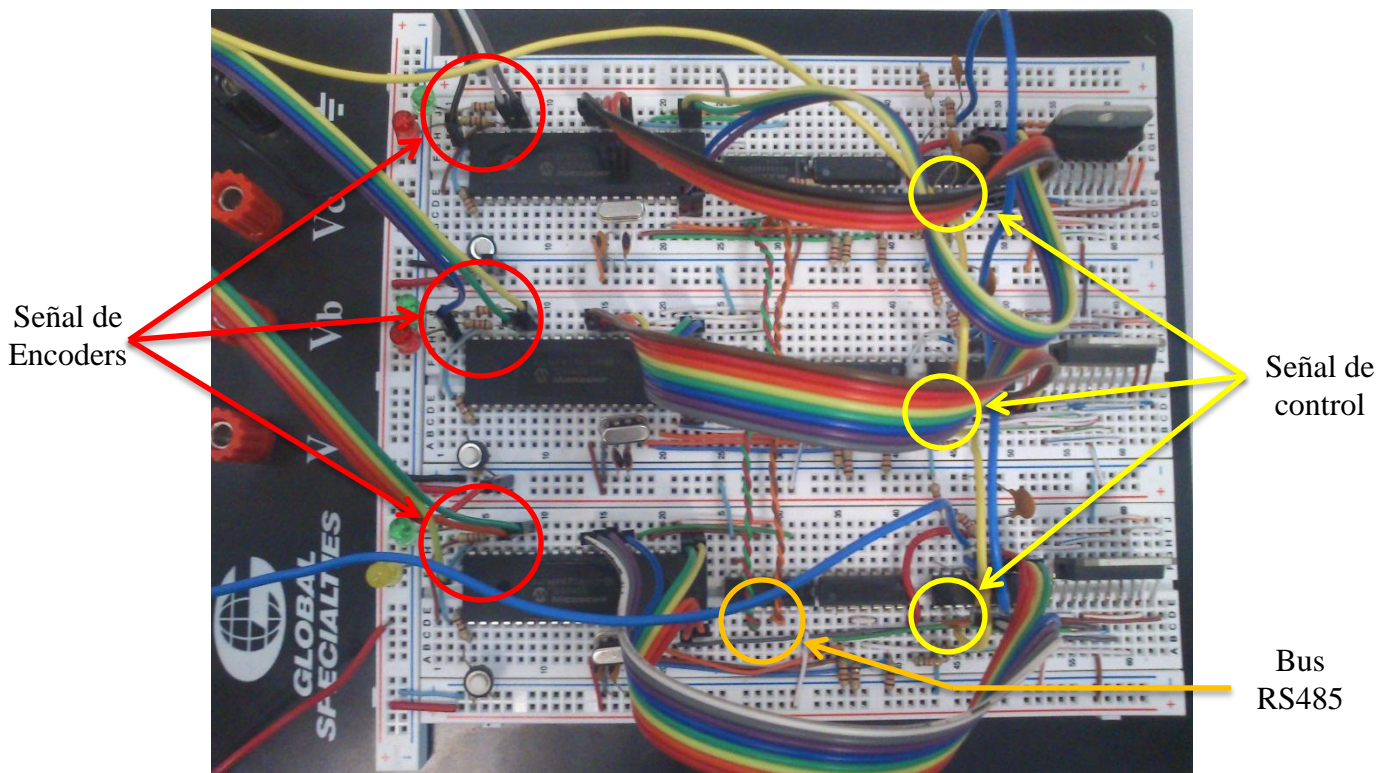


Figura 4.4. Módulos de interfaz física.

La interfaz para enviar las referencias o setpoint de posición a los dispositivos esclavos (módulos de interfaz física) fue desarrollada en LABView (ver fig. 4.5), ya que es un lenguaje de alto nivel que permite programación visual y una conexión remota mientras se genere un servidor y este se encuentre encendido, sobre esta interfaz se asigna la dirección IP (1) con la cual está identificada la plataforma de comunicación (192.168.1.111), de igual forma se asigna el puerto por el cual se comunica (2), para esta tesis se seleccionó el puerto 80, la dirección de cada dispositivo es asignada en (3), esta dirección se debe encontrar en el rango [000-255] y la referencia de posición en la sección (4), los demás campos son para monitoreo.

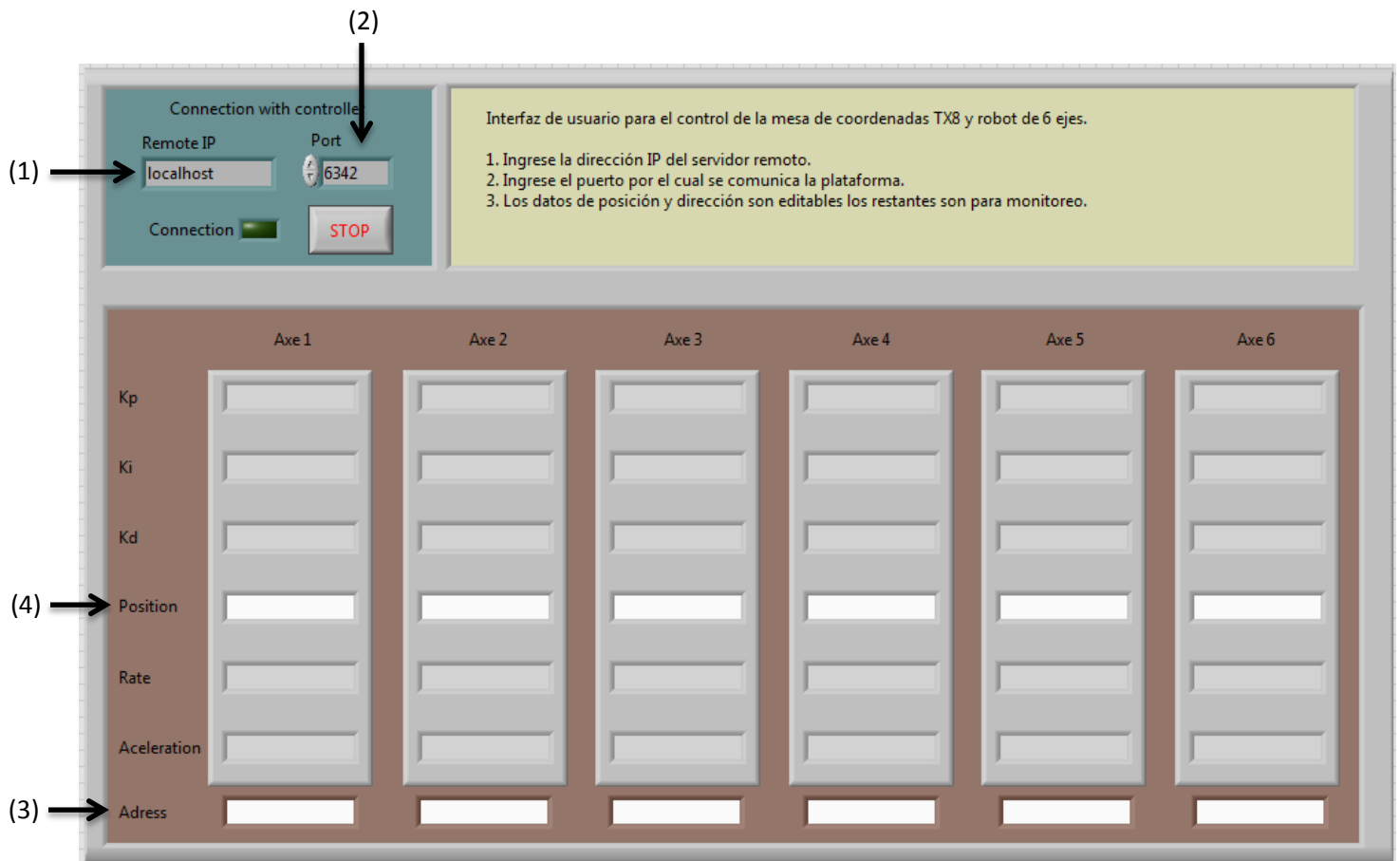


Figura 4.5. Interfaz de usuario.

Como pruebas se realizaron varios perfiles de movimiento simples como el que se muestra en la fig. 4.6, el seguimiento de la secuencia de las posiciones deseadas fueron seguidas de acuerdo a lo esperado, sin embargo se tuvieron algunos problemas con la parte mecánica de la mesa ya que no se modelo la inercia de los motores, debido a que esta fuera del alcance de esta tesis y por tanto pequeños cambios de movimiento no se producían solo aquellos en los que se requiriera mayor potencia.



Figura 4.6. Perfil de movimiento realizado por la mesa TX8.

Como el principal objetivo de este proyecto es el desarrollar una plataforma de comunicación, fue necesario realizar pruebas de velocidad de transmisión, colisión de datos, capacidad del bus, retardos y verificación de la llegada correcta de tramas. Para realizar las pruebas de velocidad de transmisión se desarrolló un algoritmo que midiera cada cierto tiempo la cantidad de bytes enviados a través del bus de transmisión y recepción, este programa se implementó en un pic auxiliar (18f4550) que tomaba muestras al bus cada 10 ms, después eran enviadas a una PC y almacenadas en un archivo para después ser graficadas, en total se tomaron 50 muestras en dos series, por lo cual se obtuvieron 500 ms de muestreo por serie, el diagrama de la fig. 4.7 muestra donde se colocaron los auxiliares.

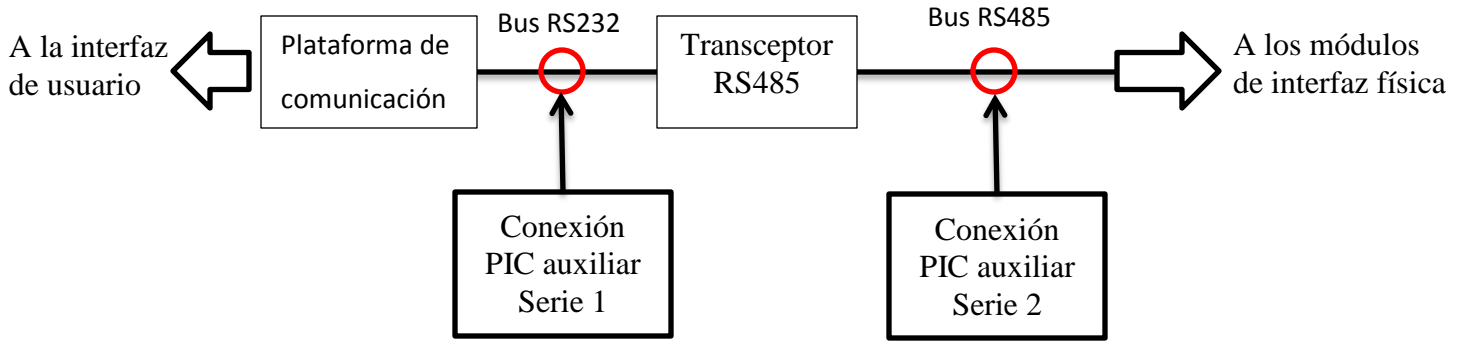


Figura 4.7. Ubicación de pic's auxiliares para obtención de muestras.

Serie de muestras 1

Una vez obtenidas las muestras que representan la cantidad total de bytes enviados cada 10 ms por parte de la plataforma de comunicación, se calculó la relación entre bytes y baudios, en la que 8192 baudios equivalen a 1 Kb real, por lo que se obtuvieron los datos de las velocidades estimadas presentados en la fig. 4.8. En teoría la velocidad de transferencia del MCU debería ser de 9600 baudios y la estimada oscila alrededor de los 9000 baudios.

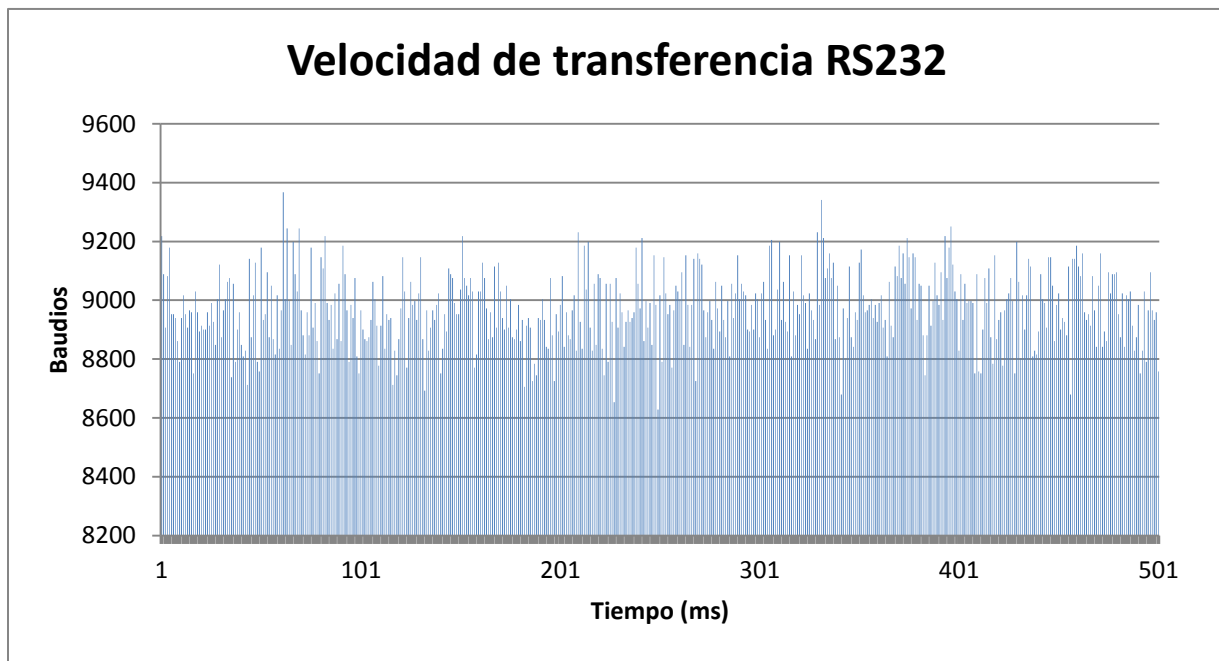


Figura 4.8. Velocidad de transferencia RS232.

Serie de muestras 2

La segunda serie de muestras fue tomada del bus de datos que comunica cada módulo de interfaz física, en las cuales muestra una compensación ya que el dispositivo empleado puede alcanzar una velocidad de hasta 10 Mb, cabe mencionar que la transferencia de datos se realizó para 3 ejes, con una trama de 255 bytes, los datos se grafican en la fig. 4.9.

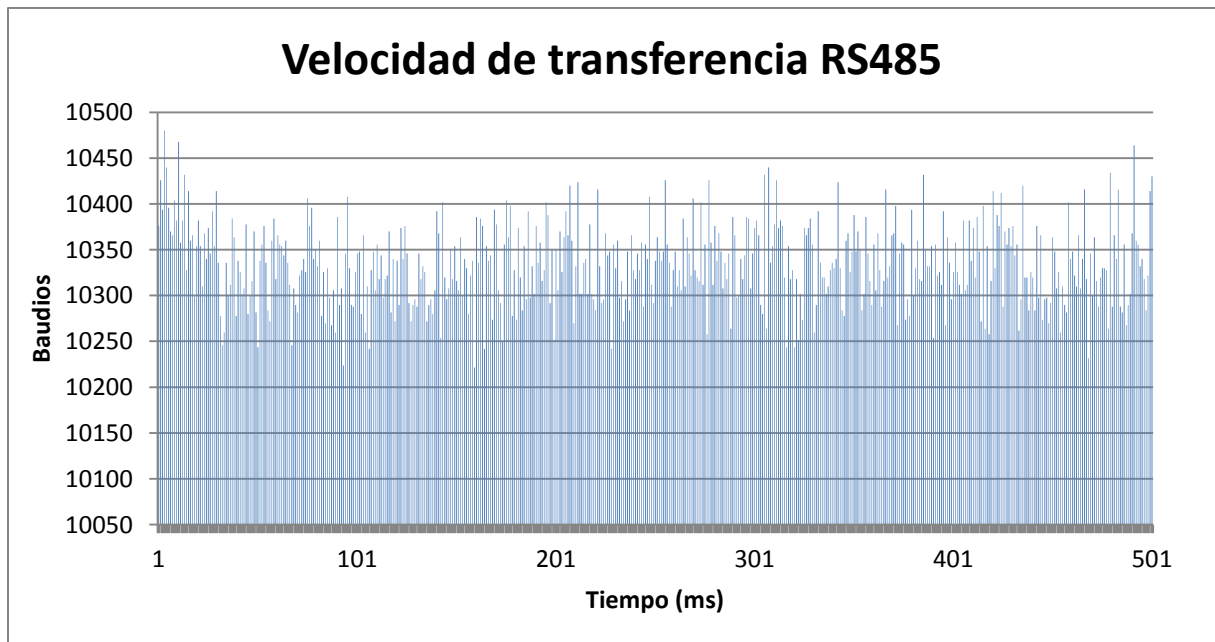


Figura 4.9. Velocidad de transferencia RS485.

Con estas velocidades de transmisión se obtuvieron los datos promedio que son presentados en la tabla 4.9, en los que se incluyen el tiempo de retardo entre dispositivos es el tiempo que tardan las tramas en llegar a su destino en una transmisión full-duplex, para obtener la velocidad de respuesta se evaluó el tiempo en que tardaba en enviarse y recibirse el ACK de cada trama a cada eje, promediando el mayor retardo de 1.94 ms.

Pruebas	1 eje	3 ejes
Velocidad de respuesta	0.31 ms – 0.85ms	1.94 ms
Frecuencia de transferencia	1.165 KHz – 3.199 KHz	5.14 KHz
Capacidad de datos	2.040 kbits (255 bytes)	6.120 Kbits (765 bytes)
		Dispositivo1: 41.2 us
Tiempo de retardo	41.2 us	Dispositivo2: 63.0 us
		Dispositivo3: 78.0 us
Colisiones	0	6

Tabla 4.1. Resultados de pruebas.

Para detectar el número de colisiones o pérdidas de tramas ocurridas se tomó en cuenta el ACK que se transmite en cada trama, de una serie de 2 Mb de datos transmitidos se perdieron 6 tramas durante las pruebas de transmisión, cabe decir que esta prueba se realizó con tramas completas y llenando el completamente la sección de datos reservada para cada dispositivo esclavo. Con las demás pruebas realizadas todos los ACK correspondientes llegaron sin atrasos o adelantos a otras tramas, por lo cual el método de anti-colisiones funciona de forma correcta, los retardos de tiempo medidos dependen del número de dispositivos con los que se desea comunicar y de la longitud de la trama a transmitir.

4.2. Conexión de la plataforma al Servo drive EDB

Como parte de las pruebas la plataforma se ha probado en el robot NACHI modelo SA160F-01 (ver fig. 4.10), este robot cuenta con 6 ejes los cuales cuentan con el rango de movimiento indicado en la tabla 4.2.

Eje	Rango de movimiento	Velocidad de movimiento
S	$\pm 150^\circ$	120 °/s (2.62 rad/s)
H	$+75^\circ - 55^\circ$	120 °/s (1.31 rad/s)
V	$+145^\circ - 115^\circ$	120 °/s (2.53 rad/s)
R2	$\pm 360^\circ$	160 °/s (6.28 rad/s)
B	$\pm 125^\circ$	160 °/s (2.18 rad/s)
R1	$\pm 360^\circ$	245 °/s (6.28 rad/s)

Tabla 4.2. Datos obtenidos de RobotWorx.



Figura 4.10. Robot NACHI modelo SA160F-01.

Este robot era manipulado por una tarjeta galil para generar el movimiento y la potencia es proporcionada por servo drives de la serie EDB, este tipo de servo drive (ilustrado en la fig. 4.11), maneja tres protocolos de comunicación RS232, RS422 y RS 485, con los cuales se puede realizar la modificación de referencia, supervisión de alarmas y estado del servo drive, modificación de parámetros; entre otras funciones.

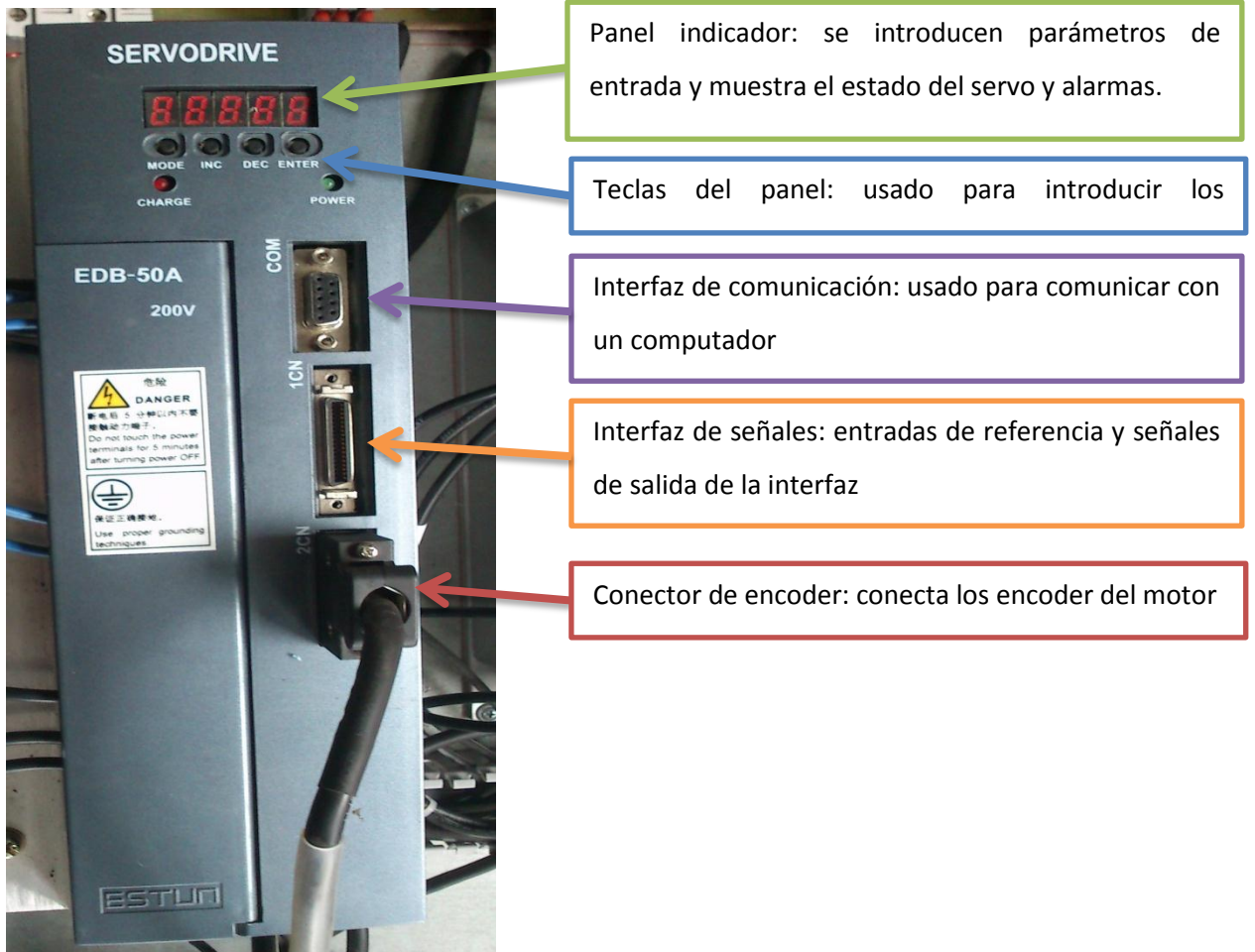


Figura 4.11. Servodrive EDB-50A.

Para probar la plataforma con este tipo de servo drives se usó el protocolo RS485 ya que con él cuenta la plataforma, el diagrama de conexión realizado se aprecia en la fig. 4.12.

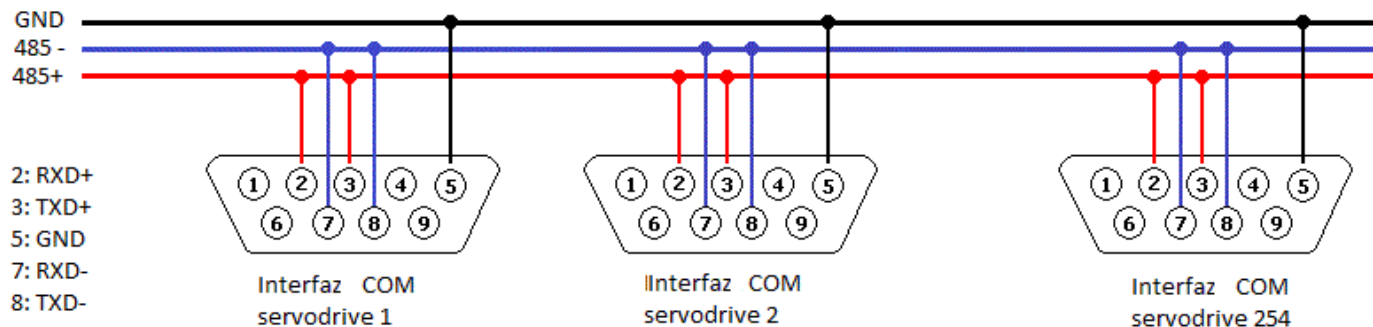


Figura 4.12. Diagrama de conexión RS485.

Una vez que se han realizado las conexiones con los servo drives se requieren realizar ciertos ajustes a cada servo, los cuales dependerán del tipo de protocolo, velocidad de transmisión, la forma de las tramas y el tipo de control, estos datos son mostrados en la tabla 4.3, la configuración de estos parámetros se realiza a través del panel indicador, una vez que se hayan configurado debe apagarse y volverse a encender el servo drive para que los cambios sean efectuados.

Parametros de comunicación RS-485, RS-232, RS-422			
Código	Descripción	Rango	Valor por defecto
Pn210	Dirección de servodrive	1 – 255	1
Pn211	Velocidades de comunicación:	0 – 2	1
	0 : 4800 bps		
	1 : 9600 bps		
	2 : 19200 bps		
Pn212	Forma del protocolo:	0 – 8	5
	0 : 7, N, 2 (Modbus,ASCII)		

	1 : 7, E, 1 (Modbus,ASCII)		
	2 : 7, O, 1 (Modbus,ASCII)		
	3 : 8, N, 2 (Modbus,ASCII)		
	4 : 8, E, 1 (Modbus,ASCII)		
	5 : 8, O, 1 (Modbus,ASCII)		
	6 : 8, N, 2 (Modbus,RTU)		
	7 : 8, E, 1 (Modbus,RTU)		
	8 : 8, O, 1 (Modbus,RTU)		
Pn213	Protocolos de comunicación:	0 – 2	2
	0 : RS232		
	1 : Protocolo RS-422/232 MODBUS		
	2 : Protocolo RS-485 MODBUS		
Pn216	Bit de control	0 – 255	0
	0 : controlado por interfaz externa		
	1 : controlado por comunicación		

Tabla 4.3. Parámetros de comunicación.

En total el servo drive cuenta con 222 códigos para el ajuste de parámetros y 42 códigos extra para la comunicación de datos y estado del servo drive, todos estos códigos son interpretados y manipulados por el microprocesador empleado en la plataforma, el cual también se encarga de la construcción de tramas y cálculo de redundancia longitudinal o cíclica. La construcción de las tramas que recibe el servo drive está conformada por la estructura que se muestra en la tabla 4.4.

Estructura de la trama	Descripción	Modo ASCII	Modo RTU
STX	Carácter de inicio	3Ah	Intervalo de espera igual a cuatro bytes de la velocidad de transmisión.
ADR	Dirección del servo drive	1 – 254 (2 – 4 bytes)	1 – 254 (1 byte)

CMD	Código de instrucción	1 byte	1 byte
DATA(n-1)	Dirección de acceso y contenido de datos	n-word=2n-byte, n < 12	
.....			
DATA(0)			
LRC	Código de verificación	1 byte	1 byte
End 1	Fin de código 1	0Dh	Intervalo de espera igual a cuatro bytes de la velocidad de transmisión.
End 0	Fin de código 2	0Ah	

Tabla 4.4. Estructura de transmisión de datos.

La trama de respuesta del servo drive es la misma que la de transmisión, con la excepción de la sección “*data*” que tiene la siguiente forma:

Bit 15	Bit 14	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ALM	RES	S-RDY	WAIT	COIN	AC-IN	PS-IN	PA-ST	TGON	N-OT	P-OT	

Dónde:

P-OT	Uso de entrada de rotación hacia adelante prohibida
N-OT	Uso de entrada de rotación de reversa prohibida
TGON	Salida de detección de rotación: 1 representa motor detenido
PA-ST	En control de posición: 1 representa estado de la alarma En control de velocidad: 1 representa exceso del torque
PS-IN	En control de posición: 1 representa que la señal es entrada En control de velocidad: 1 representa la velocidad puesta está más allá de los límites fijados
AC-IN	Representa que hay fuente de alimentación en la terminal
COIN	En control de posición: 1 representa que la posición de referencia ha sido completada. En control de velocidad: 1 representa que el motor alcanzó la velocidad deseada.
WAIT	Señal de espera del servo drive: 1 servo drive en espera
S-RDY	Servo drive listo
RES	No usado
ALM	Señal de alarma: 1 representa que una alarma ha ocurrido

Ahora que se ha comprendido la configuración y los formatos de tramas se ilustran algunos ejemplos de las tramas que se enviaron al servo drive para su funcionamiento y/o verificación de estado, debe considerarse que la transmisión seleccionada es MODBUS en modo ASCII.

	STX	ADR	CMD	Start data	Data	LRC	End 1	End 0	Descripción
Simbolo	:	01	03	1000	0001	EB	CR	LF	Trama de envío: Búsqueda del servo drive
HEX	3A	30 31	30 33	31 30 30 30	30 30 30 31	45 42	0D	0A	
Simbolo	:	01	03	02	0036	C4	CR	LF	Trama de respuesta: Respuesta del servo drive (servo encontrado)
HEX	3A	30 31	30 33	30 32	30 30 33 36	43 34	0D	0A	

Tabla 4.5. Búsqueda del servo drive.

En el ejemplo de la tabla 4.6 al microprocesador se le dio la orden de buscar el servo drive con la dirección 1 equivalente a 01 en hexadecimal, por lo cual la operación es de lectura; identificada con el código 03, la dirección a la que se buscará acceso es 1000 y se leerá 1 palabra por lo que el contenido de *Data* es 0001. Después de enviar dicha trama se recibió la respuesta identificada por el código 02, en el contenido “*Data*” se puede observar el valor 0036, este valor significa lo siguiente:

<i>Nibble</i> más significativo	0	Ninguna alarma detectada.
	0	No usado.
	3	Existe señal de la fuente de alimentación y señal de control de posición o velocidad.
<i>Nibble</i> menos significativo	6	Motor detenido y rotación de reversa prohibida.

En el ejemplo que se muestra en la tabla 4.6, se lee el estado en el que se encuentra el servo drive con dirección 01, por lo cual se leerá 1 palabra de la dirección 0901.

	STX	ADR	CMD	Start data	Data	LRC	End 1	End 0	Descripción
Simbolo	:	01	03	0901	0001	F1	CR	LF	Trama de envío: Estado del servo drive
HEX	3A	30 31	30 33	30 39 30 31	30 30 30 31	46 31	0D	0A	
Simbolo	:	01	03	02	0163	96	CR	LF	Trama de recepción: Respuesta del servo drive (estado en línea)
HEX	3A	30 31	30 33	30 32	30 31 36 33	39 36	0D	0A	

Tabla 4.6. Estado del servo drive.

El contenido del dato de la respuesta identificada por el código 02 es 0163 del cual cada valor significa lo siguiente:

<i>Nibble</i> más significativo	0	Ninguna alarma detectada.
	1	Servo drive en línea.
<i>Nibble</i> menos significativo	6	El motor alcanzó la velocidad deseada y existe señal de la fuente de alimentación.
	3	Motor encendido, rotación hacia adelante y hacia atrás prohibidas.

De forma de comprobar las tramas que se envían y reciben se conectó un transceptor MAX3086 al bus dirigido a los servo drives y en la terminal R0 (lectura) un analizador de señales mixtas para poder visualizar los datos enviados. En la fig. 4.13, se muestra la trama para cambiar la velocidad de Joggeo del servo drive 1, por lo que el código de operación cambia a 06 (escritura), y la velocidad que se desea es de 100 rpm valor correspondiente a 0064 en hexadecimal, el código de acceso es el Pn037

(<http://www.anaheimautomation.com/manuals/servo/>) por lo que la dirección de acceso es equivalente a 0025.

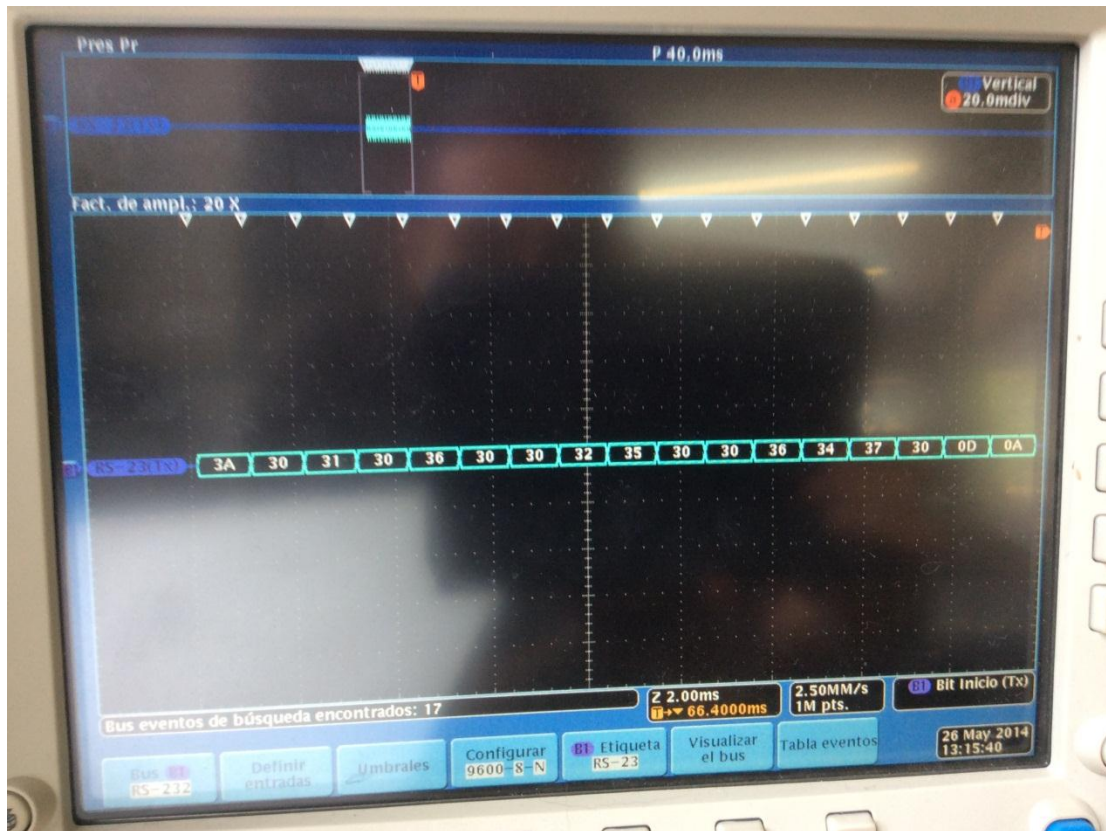


Figura 4.13. Trama de configuración velocidad de jogeo.

Las características tomadas de la transmisión de datos son las siguientes, las cuales solo representan los la transmisión y recepción de datos con los cuales funciona el servo drive, el cual tarda 45 ms en emitir una respuesta a una orden de solicitud.

Tiempo de retardo:	45 ms
Velocidad de transmisión:	9600 baudios
Capacidad de datos:	20 – 79 bytes
Colisiones:	0

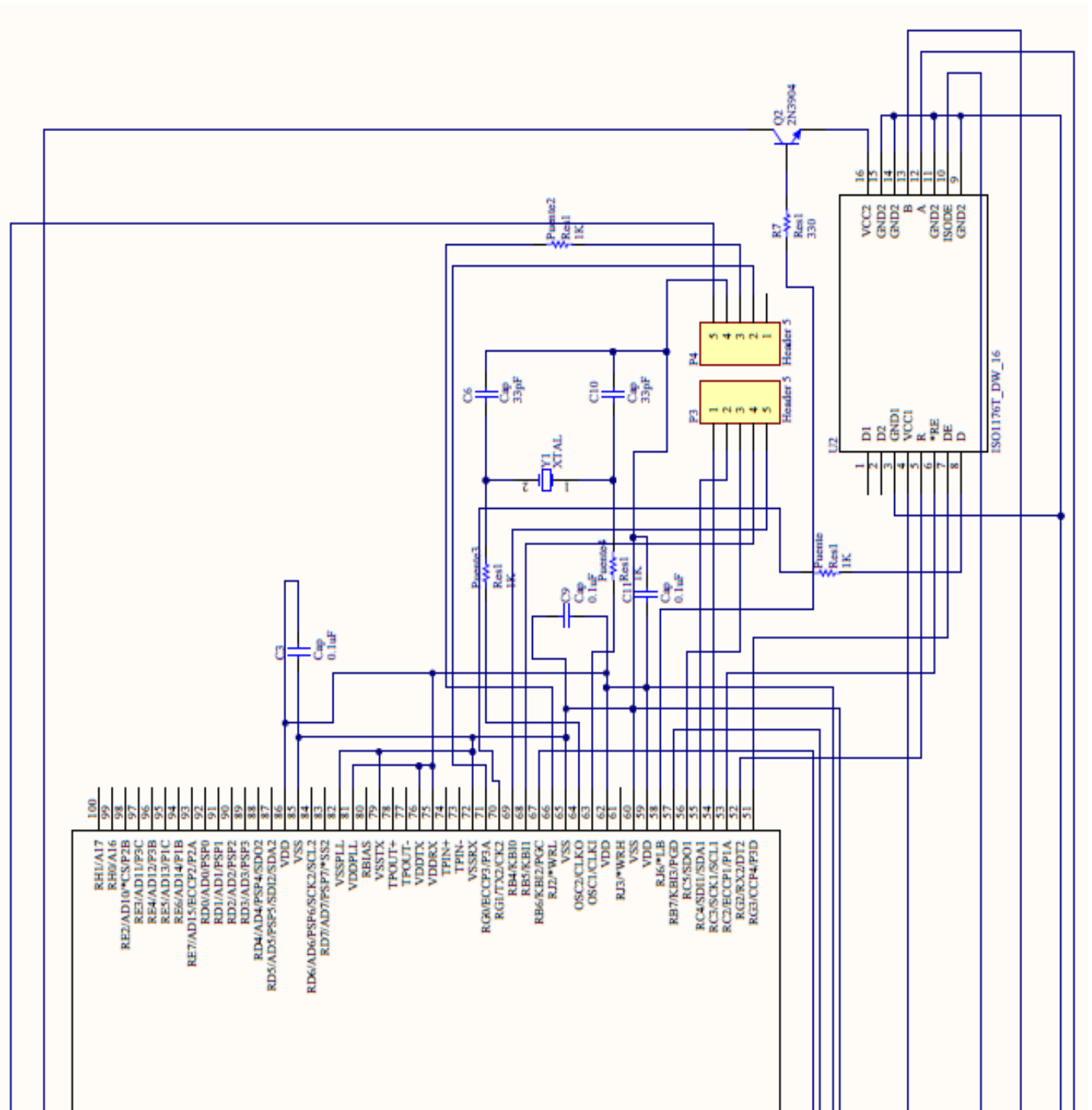
Capítulo 5. CONCLUSIONES

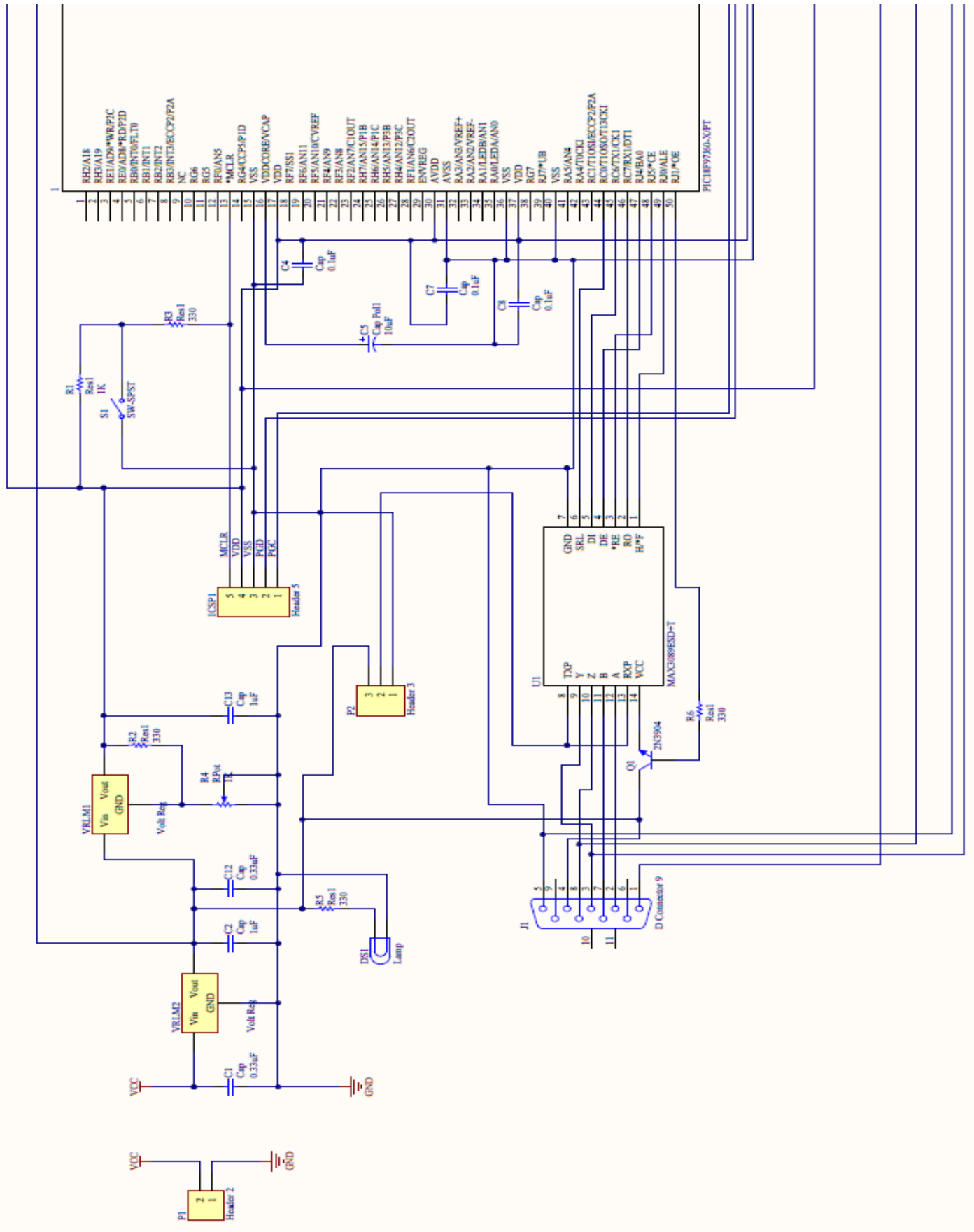
Al término de este proyecto se cuenta con un prototipo de una plataforma de comunicación en la que se usan protocolos de comunicación con estructuras de datos bien definidas, que permite la comunicación entre componentes electrónicos dedicados al servo control, de acuerdo a los datos obtenidos de las pruebas la plataforma permite a sistemas ser adaptados y reutilizados y transmitir datos a altas velocidades, siendo una plataforma flexible a la manipulación del software por el usuario, por lo que aplicar reingeniería a mecanismos que se encuentran en desuso es factible para que mecanismos como robots puedan ser reincorporados a labores.

Respecto a los algoritmos programados como CSMA/CD, tiene el problema en que a mayor cantidad de dispositivos mayor será el tiempo de espera de transmisión si se da el caso de tener colisiones, este puede ser mejorado implementando algún otro algoritmo, actualmente hay trabajos publicados en los que mejoran de manera eficiente la velocidad de algoritmo CSMA/CD, el algoritmo de ventanas deslizantes ha sido un método para solucionar este problema, pero este último consume mucha memoria libre cuando se envían datos superiores a 1 Mb y solo cuando se da el caso de que ocurra una colisión, de otra forma es una buena propuesta para solucionar este conflicto.

Capítulo 6. APÉNDICE

A continuación se muestra el esquemático de las conexiones físicas de la plataforma, lista de materiales, el PCB elaborados en Altium y finalmente los códigos implementados en el microprocesador.





Bill of Materials

<Parameter Title not found>

Source Data From: PCB_MCU.PrjPcb
 Project: PCB_MCU.PrjPcb
 Variant: None

Creation Date: 06/03/2014 10:04:54 p.m.
 Print Date: 41704 41704.92011

Footprint	Comment	LibRef	Designator	Description	Quantity
TQFP100_12x12MC	PIC18F97J80-X/PT	PIC18F97J80-X/PT	1	Imported	1
BAT-2	Cap	Cap	C1, C2, C4, C6, C7, C8, C9, C10, C11, C12, C13	Capacitor	11
Condensador	Cap	Cap	C3	Capacitor	1
BAT-2	Cap Pol1	Cap Pol1	C5	Polarized Capacitor (Radial)	1
PIN2	Lamp	Lamp	DS1	Incandescent Bulb	1
HDR1X5	Header 5	Header 5	ICSP1, P3, P4	Header, 5-Pin	3
DSUB1.385-2H9	D Connector 9	D Connector 9	J1	Receptacle Assembly, 9 Position, Right Angle	1
HDR1X2	Header 2	Header 2	P1	Header, 2-Pin	1
HDR1X3	Header 3	Header 3	P2	Header, 3-Pin	1
AXIAL-0.3	Res1	Res1	Puente, Puente2, Puente3, Puente4, R1, R2, R3, R5, R6, R7	Resistor	10
TO-92A	2N3904	2N3904	Q1, Q2	NPN General Purpose Amplifier	2
PCBComponent_1	RPot	RPot	R4	Potentiometer	1
SPST-2	SW-SPST	SW-SPST	S1	Single-Pole, Single-Throw Switch	1
21-0041B_14	MAX3089ESD+T	MAX3089ESD+T	U1	Imported	1
DW16	ISO1178T_DW_16	ISO1178T_DW_16	U2	Imported	1
389-03	Volt Reg	Volt Reg	VRLM1, VRLM2	Voltage Regulator	2
R38	XTAL	XTAL	Y1	Crystal Oscillator	1
					40

Approved	Notes


```

/*****
* Algoritmo para transmisión y recepción de datos por TCP
*****/

#include <18F97J60.h>
#fuses HS, NOWDT, NOPROTECT, NODEBUG
#use delay(clock=20000000)
#use rs232(stream=UART, rcv=PIN_C7,xmit=PIN_C6,baud=9600,bits=8,parity=N)
#include <stdlibm.h>

// Conexión entre el 18F97J60 y el Modulo ENC28J60
#define PIN_ENC_MAC_SO  PIN_C5 // Conectar con PIN MISO del ENC28J60.
#define PIN_ENC_MAC_SI  PIN_C4 // Conectar con PIN MOSI del ENC28J60.
#define PIN_ENC_MAC_CLK PIN_C3 // Conectar con PIN SCK del ENC28J60.
#define PIN_ENC_MAC_CS  PIN_G0 // Conectar con PIN CS del ENC28J60.
#define PIN_ENC_MAC_RST PIN_G1 // Conectar con PIN RST del ENC28J60.
#define PIN_ENC_MAC_INT PIN_B4 // Conectar con PIN INT del ENC28J60.
#define PIN_ENC_MAC_WOL PIN_B5 // Conectar con PIN WOL del ENC28J60.

//Protocolos a utilizar.
#define STACK_USE_MCPENC TRUE
#define STACK_USE_ARP  TRUE
#define STACK_USE_ICMP TRUE
#define STACK_USE_TCP  TRUE

#include "tcpip/stacktsk.c"
#include "lbhdr/mmemory.c"
#include "lbhdr/pic18F.h"
#include "lbhdr/HdrFrames.h"
#include "lbhdr/comCSMA.c"
#define DE PORTJ_RJ4
#define RE PORTJ_RJ5
#define SOCKET_PORT 80

fifo *package = NULL;
fifo *packageRX = NULL;
int getChar, flag = 0;
int8 HTTP_SPORT=INVALID_SOCKET_PORT;

// Microchip VendorID, MAC: 00-04-A3-XX-XX-XX
void MACAddrInit(void){
    MY_MAC_BYTE1 = 0x00;
    MY_MAC_BYTE2 = 0x04;
    MY_MAC_BYTE3 = 0xA3;

```



```

MY_MAC_BYTE4 = 0x00;
MY_MAC_BYTE5 = 0x00;
MY_MAC_BYTE6 = 0x01;
}

void IPAddrInit(void){
//Direccion IP
MY_IP_BYTE1 = 192;
MY_IP_BYTE2 = 168;
MY_IP_BYTE3 = 1;
MY_IP_BYTE4 = 111;
//Puerta de Enlace
MY_GATE_BYTE1 = 192;
MY_GATE_BYTE2 = 168;
MY_GATE_BYTE3 = 1;
MY_GATE_BYTE4 = 1;
//Mascara de Subred
MY_MASK_BYTE1 = 255;
MY_MASK_BYTE2 = 255;
MY_MASK_BYTE3 = 255;
MY_MASK_BYTE4 = 0;
}

void HTTPTask(void){
static enum{CONNECT = 0, WAIT_CONNECT = 1, STATUS_GET = 2,
STATUS_PUT = 3, DISCONNECT = 4} state = 0;
static TICKTYPE timeout_counter;
int cPkg;
char PKG_TX, PKG_RX;

if ( HTTP_SPORT == INVALID_SOCKET_PORT )
state = CONNECT;
else if ( !TCPIsConnected ( HTTP_SPORT ) )
state = WAIT_CONNECT;
else if ( TickGetDiff ( TickGet ( ) , timeout_counter ) > TICKS_PER_SECOND * 60 )
state = DISCONNECT;

switch ( state ){
case CONNECT:
HTTP_SPORT = TCPListen ( SOCKET_PORT );
if ( HTTP_SPORT != INVALID_SOCKET_PORT ){
state = WAIT_CONNECT;
timeout_counter = TickGet ( );
}
break;

```

```

case WAIT_CONNECT:
    timeout_counter = TickGet ( );
    if ( TCPIsConnected ( HTTP_SPORT ) )
        state = STATUS_GET;
break;

case STATUS_GET:
    cPkg = 0;
    LRC = 0;
    if ( TCPIsGetReady ( HTTP_SPORT ) ){
        DE = 1;
        while(TCPGet(HTTPSocket, &PKG_TX))
            addChip(PKG_TX, package);
        csmaCD();
        DE = 0;
    }

    state = STATUS_PUT;
break;

case STATUS_PUT:
    while( packageRX->length>0 ){
        PKG_RX = extractChip( packageRX );
        TCPput(HTTPSocket, PKG_RX.data_value);
    }
    TCPFlush(HTTPSocket);
    state = STATUS_GET;
break;

case DISCONNECT:
    if ( TCPIsPutReady ( HTTP_SPORT ) ){
        TCPDisconnect ( HTTP_SPORT );
        state = WAIT_CONNECT;
    }
break;
}
}

#int_rda
void rcv_data(){
    getChar = fgetc(UART);
    addChip( getChar,packageRX );
    if( getChar==0x0A ) flag = 1;
}

```

```

void main ( void){
    TRISJ &= 0xC0;
    OSCTUNE = 0xD0;
    OSCCON = 0x02;

    package = newFifo();
    packageRX = newFifo();

    MACAddrInit ( );
    IPAddrInit ( );
    StackInit ( );

    enable_interrupts(INT_RDA);
    enable_interrupts(GLOBAL);

    while ( TRUE ){
        StackTask ( );
        HTTPTask ( );
    }

    free(package);
    free(packageRX);
}

#include "comCSMA.h"

fifo *fifoTX = NULL;
fifo *fifoRX = NULL;
fifo *fifoENC = NULL;

/*****
* Function:    void csmaCD(void)
* Overview:    Implement process of CSMA/CD for transmit of data
*****/

void csmaCD( void ){
    float data_random;
    int cJAM;

    csma_frame(); // It will change for other protocols
    if( AttemptCounter<=16 ){
        if( !input_state(PIN_B3) && (end==0) ){
            delay_us(IPG);
            csma_send(); // It will change for other protocols
            end = 1;
        }
    }
}

```

```

    if( end==2 ){
        // Send JAM
        for( cJAM=0; cJAM<4; cJAM++ )
            putc( 0xAA );

        CW = CW*2;
        data_random = rand();
        enable_interrupts(INT_TIMER2);
        while( !csma_backoff(data_random) ){ };
        disable_interrupts(INT_TIMER2);
        countTimer2 = 0;
        end = 0;
    }
}

/*****
* Function:    void csma_frame(void)
* Overview:    Build data frame for send to serial communication
*****/

void csma_frame( void ) {
    int8 c, state_data = 0, checksum;
    int16 aux_data=0;

    if( *fifoTX.length>0 && state_data!=0 ){
        aux_data = extractChip(fifoTX);
        state_data = 1;
    }

    send_frame[4] = 0xEE;
    send_frame[7] = aux_data;
    send_frame[8] = (aux_data>>8);

    checksum=0x00;
    for( c=2;c<buf_length_frame;c++ ) checksum += send_frame[c];

    send_frame[buf_length_frame-1] = (0xFF-(checksum));
}

/*****
* Function:    void csma_send(void)
* Overview:    Send data frame to serial communication
*****/

void csma_send( void ){

```

```

int byte_frame;

for( byte_frame=0;byte_frame<buf_length_frame;byte_frame++ )
    putc( send_frame[byte_frame] );
}

/*****
* Function:    int1 csma_backoff( float data_rand )
* Overview:    Delay the data frames to send for an aleatory time
*****/

int1 csma_backoff( float data_rand )
{
    unsigned int32 Backoff_time;
    Backoff_time = (CW-1)*data_rand*IPG; // Backoff_time = [CW x rand()] x SlotTime

    if( countTimer2>=Backoff_time ){
        countTimer2 = 0;
        return 1;
    }

    else
        return 0;
}

/*****
* Function:    int1 csma_check_ack( void )
* Overview:    Check the ACK of data frames sent and verifies that the transfer was correct
*****/

int1 csma_check_ack( void ){
    int aux_checksum=0x00, c;

    for( c=2;c<buf_length_ack-1;c++ ) aux_checksum += RXAck[c];

    aux_checksum=(0xFF-(aux_checksum));

    // Check ACK
    if( aux_checksum==(RXAck-1) ){
        // Check errors
        if( RXAck[3]==0xDC ) return 1;
        else return 0;
    }

    else
        return 0;
}

```

```

typedef struct StackPROFIBUS{
    byte strDmr;
    byte netDatlen;
    byte lenRepeated;
    byte destAddr;
    byte srcAddr;
    byte fcnCode;
    byte DSAP;
    byte SSAP;
    byte FCS;
    byte endDmr;

    byte ftype;
}HEADER_PBUS;

// Header RS485

typedef struct StackStream485{
    byte PRM; // Preamble
    byte SADR; // Source address
    byte END;
}HEADER_stm485;

HEADER_stm485 strmdevice;

typedef struct setDEVICE{
    float Kp;
    float Ki;
    float Kd;
    int16 Sp;
}CNFG_SETDEVICE;

CNFG_SETDEVICE cnfgDVC;

/*****
* Function: void configFRM( void )
* Overview: Configure header of frame
*****/

void configFRM( byte preamble, byte end ){
    strmdevice.PRM = preamble;
    strmdevice.SADR = deviceID;
    strmdevice.END = end ;
}

```

```

/*****
* Function:   int32 StringHEX_TO_Decimal( char *string )
* Overview:   Convert a String HEX to decimal
*****/

```

```

int32 StringHEX_TO_Decimal( char *string ){
    int lenght, i, shift;
    int32 value = 0, aux;
    lenght = strlen( string )-1;

    for( i=0; i<=lenght; i++ ){
        shift = (lenght-i) * 4;
        if( (string[i]>=65 && string[i]<=70) )    aux = string[i]-0x37;
        else if( (string[i]>=48 && string[i]<=57) )    aux = string[i]-0x30;
        value |= (aux<<shift);
    }
    return value;
}

```

```

/*****
* Function:   int1 send_frames( fifo *fifoFrame )
* Overview:   Send a message over the RS485 bus
*****/

```

```

int1 send_frames( fifo *fifoFrame ){
    data auxChip;
    int pkg, aux_len[3];
    int32 length;
    byte lrc;

    while( fifoFrame->length!=0 ){
        fputc( strmdevice.PRM,RS232 );    // PRM
        auxChip = extractChip( fifoFrame );
        aux_len[0] = auxChip.data_value;
        auxChip = extractChip( fifoFrame );
        aux_len[1] = auxChip.data_value;
        auxChip = extractChip( fifoFrame );
        aux_len[2] = auxChip.data_value;
        length = StringHEX_TO_Decimal( aux_len );
        auxChip = extractChip( fifoFrame );
        if( auxChip.data_value!='0' )
            fputc( auxChip.data_value,RS232 );    // DADR

        fputc( strmdevice.SADR,RS232 );    // SADR
    }
}

```

```

//fputc( length,RS232 );          // LEN

for(pkg=0; pkg<length-1; pkg++ ){
    auxChip = extractChip( fifoFrame ); // PKG
    lrc = (BYTE) ((lrc+auxChip.data_value) & 0xFF);
    fputc( auxChip.data_value,RS232 );
}

lrc = (BYTE)((lrc^0xFF) + 1) & 0xFF; // LRC
fputc( lrc,RS232 );
fputc( strmdevice.END,RS232 );      // END
}
return TRUE;
}

/*****
* Function:    float GetData( fifo *fifoFrame )
* Overview:    Get data of frame
*****/

float GetData( fifo *fifoFrame ){
    data auxDatFRM;
    float data_frame;
    int lPackage, PKG, container[10];

    auxDatFRM = extractChip( fifoFrame ); // Length
    lPackage = auxDatFRM.data_value;

    for( PKG=48; PKG<lPackage; PKG++ ){
        auxDatFRM = extractChip( fifoFrame ); // Package
        container[PKG-48] = auxDatFRM.data_value;
    }

    data_frame = ( container!=NULL )? atof(container):NULL;
    if( container!=NULL ) data_frame = atof(container);
    else data_frame = NULL;
    return data_frame;
}

/*****
* Function:    int1 stractData( fifo *fifoFRM )
* Overview:    Extract data of frames
*****/

int1 extractData( fifo *fifoFrame ){
    data auxDatFRM;

```



```

auxDatFRM = extractChip( fifoFrame );
if( auxDatFRM.data_value==126 ){          // PREAMBLE
    auxDatFRM = extractChip( fifoFrame );
    if( auxDatFRM.data_value==deviceID ){ // DADR
        auxDatFRM = extractChip( fifoFrame ); // SADR
        sourceID = auxDatFRM.data_value;
        auxDatFRM = extractChip( fifoFrame ); // FULL LENGTH
        cnfgDVC.Kp = GetData( fifoFrame ); // Kp
        cnfgDVC.Ki = GetData( fifoFrame ); // Ki
        cnfgDVC.Kd = GetData( fifoFrame ); // Kd
        cnfgDVC.Sp = GetData( fifoFrame ); // Position
        auxDatFRM = extractChip( fifoFrame ); // LRC
        auxDatFRM = extractChip( fifoFrame ); // END
    }
}
return TRUE;
}

/*****
* Function:    fifo *newFifo( void );
* Overview:    Allocates space of memory for a stack of data
*****/

fifo *newFifo( void ){
    fifo *nFifo;
    nFifo = (fifo *)malloc( sizeof(fifo) );
    if( nFifo ){
        nFifo->length=0;
        nFifo->first = nFifo->last = null;
    }

    else return null;
    return nFifo;
}

/*****
* Function:    int1 addChip( int16 data_value, fifo *fifo1 );
* Overview:    Allocates space of memory for a stack of data
*****/

int1 addChip( int16 data_value, fifo *fifo1 ){
    chip *nChip;
    nChip = addData(data_value,0);

    if( nChip ){
        if( fifo1->length!=0 ){
            fifo1->last->next = nChip;

```

```

    }
    else        fifo1->first = nChip;

    fifo1->last = nChip;
    fifo1->length++;
    return 1;
}

else        return 0;
}

/*****
* Function:    chip *addData(int16 data_value, chip *next);
* Overview:    Allocates space of memory for a chip
*****/

chip *addData(int16 data_value, chip *next){
    chip *nChip;
    nChip=(chip *)malloc( sizeof(chip) );

    if( nChip ){
        nChip->data_chip = data_value;
        nChip->next = next;
    }

    else return null;
    return nChip;
}

/*****
* Function:    data structChip( fifo *fifo1 );
* Overview:    Free up space of memory in the stack
*****/

data extractChip( fifo *fifo1 ){
    data data_Chip;
    chip *tmp;

    tmp = fifo1->first;
    data_Chip.data_value = tmp->data_chip;
    fifo1->first = tmp->next;
    fifo1->length--;
    if( fifo1->length==0 )    fifo1->last = 0;
    free( ( chip *)tmp );
    return data_Chip;
}

```

```

/*****
* Function:    void configFRM( BYTE PRM, BYTE SFD )
* Overview:    Configure header of frame
*****/

void configFRM( BYTE PRM, BYTE SFD )
{
    HDR.PRM = PRM;
    HDR.SFD = SFD;
}

/*****
* Function:    BYTE valueLRC(BYTE *PKG, WORD LEN)
* Overview:    Longitudinal redundancy check
*****/

BYTE valueLRC(BYTE *PKG, WORD LEN)
{
    BYTE checksum = 0;
    int i;

    for( i=0;i<LEN;i++ )
        checksum = (BYTE) ((checksum+PKG[i] & 0xFF);

    checksum = (BYTE)((checksum^0xFF) + 1) & 0xFF );
    return checksum;
}

/*
* -----
* Order information - ASCII MODE
* -----
* STX Start character • ":" => (3Ah)
* ADR Communication address • • => 1 - byte contains 2 ASCII codes
* CMD Instruction code • • • => 1 - byte contains 2 ASCII codes
* DATA (n - 1) Datum content • • => n-word = 2n - byte contain n ASCII codes • , n <
12 •
* DATA (0)
* LRC Verifying code • • => 1 - byte contains 2 ASCII codes
* End 1 End code 1 • • • => (0Dh) < CR • >
* End 0 End code 0 • • • => (0Ah •) < LF • >
*
* -----
* Response information - ASCII MODE
* -----

```

```

* STX • ":"
* ADR • 1st b
* ADR 2nd b
* CMD 1st b
* CMD 2nd b
* Data number (counted as byte)
* Start data address (n bytes)
* Second data address (n bytes)
* LRC verifying 1st b
* LRC verifying 2nd b
* End 1 (0Dh) <CR >
* End 0 (0Ah) <LF >
*/
typedef STRUCT respInf
{
    int addressEDB;
    int command;
    int16 StDataAdr; // Start data address
    int16 SnDataAdr; // Second data address
}respInfoEDB;

respInfoEDB respEDB, sendEDB;

// Read response information - ASCII MODE
void GetEDBData( fifo *fifoFrame )
{
    data auxDatFRM;
    int lPackage, PKG, shortData[6];

    auxDatFRM = extractChip (fifoFrame); // STX

    if (auxDatFRM.data_value == 0x3A)
    {
        auxDatFRM = extractChip (fifoFrame); // 1st b
        shortData[0] = auxDatFRM.data_value;

        auxDatFRM = extractChip (fifoFrame); // 2nd b
        shortData[1] = auxDatFRM.data_value;

        respEDB.addressEDB = StringHEX_TO_Decimal (shortData); // Get address device
        that responds

        auxDatFRM = extractChip (fifoFrame); // 1st b
        shortData[0] = auxDatFRM.data_value;

        auxDatFRM = extractChip (fifoFrame); // 2nd b
        shortData[1] = auxDatFRM.data_value;
    }
}

```

```

respEDB.command = StringHEX_TO_Decimal (shortData); // Get operation command

for (PKG = 0; PKG < 4; PKG++) // Get start data Address or command of response
{
    auxDatFRM = extractChip (fifoFrame);
    shortData[PKG] = auxDatFRM.data_value;

    if (PKG == 1)
    {
        commad = StringHEX_TO_Decimal (shortData);

        if (commad == 2)
            PKG = 4;

        else
            command = 0;
    }
}

if (command == 0)
respEDB.StDataAdr = StringHEX_TO_Decimal (shortData);

for (PKG = 0; PKG < 4; PKG++) // Get datum
{
    auxDatFRM = extractChip (fifoFrame);
    shortData[PKG] = auxDatFRM.data_value;
}

respEDB.SnDataAdr = StringHEX_TO_Decimal (shortData);

auxDatFRM = extractChip (fifoFrame); // Get LRC
shortData[0] = auxDatFRM.data_value; // high nibble
auxDatFRM = extractChip (fifoFrame);
shortData[1] = auxDatFRM.data_value; // low nibble

auxDatFRM = extractChip (fifoFrame); // Get END1
auxDatFRM = extractChip (fifoFrame); // Get END0
}
}

// Send order information
int1 sendEDBData (int ReadOrWrite)

```

```

{
int LRC, frame[17];

frame[0] = 0x3A;          //STX

LRC = addressEDB>>4;     // communication Address
frame[1] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = addressEDB&&0x000F;
frame[2] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = ReadOrWrite>>4;    // Instruction code
frame[3] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = ReadOrWrite&&0x000F;
frame[4] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr>>12;     // Start data address
frame[5] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr>>8;
frame[6] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr>>4;
frame[7] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr&&0x000F;
frame[8] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr>>12;     // Second data address
frame[9] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr>>8;
frame[10] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr>>4;
frame[11] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

LRC = StDataAdr&&0x000F;
frame[12] = (LRC >= 0&&LRC <= 9) ? LRC + 48 : LRC + 55;

```

```

    GetLRC(frame);

    frame[15] = 0x0D;
    frame[16] = 0x0A;
}

void GetLRC(int *frame)
{
    BYTE ans = 0;
    int idx, dec=0;
    char aux[3];

    for( idx=1; idx<12; idx+=2 )
    {
        dec = (valueDEC(frame[idx])<<4)+valueDEC(frame[idx+1]);
        ans += dec;
    }

    ans = (BYTE)((ans^0xFF)+1) & 0xFF;

    dec = (ans>>4)&0x0F;
    frame[13] = (dec>=0&&dec<=9)? dec+=48 : dec+= 55;
    dec = ans&0x0F;
    frame[14] = (dec>=0&&dec<=9)? dec+=48 : dec+= 55;
}

unsigned int crc(int * frame, int length)
{
    int i,j;
    unsigned int crcReg=0xFFFF;

    While(length--)
    {
        crcReg ^= *frame++;

        for(j=0;j<8;j++)
        {
            if(crcReg & 0x01)
                crcReg =(crcReg >>1)^0xA001;

            else
                crcReg = crcReg>>1;
        }
    }
}

```

```
return crcReg;  
}
```

```
int valueDEC(int value)  
{  
    if (value >= 65&&value <= 70)  
        value -= 0x37;  
    else if (value >= 48&&value <= 57)  
        value -= 0x30;  
    return value;  
}
```


LITERATURA CITADA

- Axelsson, Jan. 2001. *1 Lakeview Research Programming and Circuits for RS-232 and RS-485: Links and Networks*. First Edit. PENRAM INTERNATIONAL.
- B&B. 1997. "RS422 and RS485 Application Note." *B&B Electronics* 1(October): 44.
- Beuerman, S.L., and E.J. Coyle. 1988. "The Delay Characteristics of CSMA/CD Networks." *IEEE Transactions on Communications* 36(5): 553–63.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1471>.
- Bracarense, Alexandre Queiroz. 1999. "Robot Retrofitting: A Perspective to Small and Medium Size Enterprises." *Mechanical and Electrical Engineering* 1: 11.
- Draganjac, I., V. Sesar, S. Bogdan, and Z. Kovacic. 2008. "An Internet-Based System for Remote Planning and Execution of SCARA Robot Trajectories." *2008 34th Annual Conference of IEEE Industrial Electronics*: 3485–90.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4758522>.
- Galindo J. C. (2001), Diseño y construcción de una tarjeta de control de arquitectura abierta. Universidad Autónoma de Querétaro
- Hongyu, Shi, Feng Yong, and Chen Na. 2010. "A Distributed Digital Motion Control System Based SERCOS." *2010 International Conference on Electrical and Control Engineering* (60474016): 48–52.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5631630> (September 12, 2013).
- INCORPORATED, ACROMAG. 2002. *Acromag ProfiBus/RS485 Network I/O Modules*. ed. ACROMAG INCORPORATED. Wixom, USA.
- Jos, Eduardo, Carlos Alberto, Carvalho Castro, and Alexandre Queiroz Bracarense. 2003. "SENSING FOR RETROFITTING OF AN INDUSTRIAL ROBOT." *IFAC* 1: 6.

- Liu, Yuan, Yong-zhang Wang, and Hong-ya Fu. 2008. "An Open Architecture Motion Controller for CNC Machine Tools." *2008 2nd International Symposium on Systems and Control in Aerospace and Astronautics*: 1–4.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4776146>.
- Loenzo R.A. (2009), Sistema de control numérico computarizado de arquitectura abierta, modular y portátil. Universidad Autónoma de Querétaro
- Man, L A N, and Computer Society. 2012. *2012 IEEE Standard for Ethernet*.
- Morales-Velazquez, Luis et al. 2010. "Open-Architecture System Based on a reconfigurable Hardware–software Multi-Agent Platform for CNC Machines." *Journal of Systems Architecture* 56(9): 407–18.
<http://www.sciencedirect.com/science/article/pii/S1383762110000354> (September 12, 2013).
- Niu, Junhao. 2012. "Design of USB-CAN Controller Based on PIC18F4580." *Procedia Engineering* 29: 329–33.
<http://linkinghub.elsevier.com/retrieve/pii/S1877705811065544> (September 12, 2013).
- Nutzerorganisation. 2013. *PROFIBUS, Process Field Bus*. www.profibus.com.
- Oliveira, Schneider De, and Raul Guenther. 2008. "EMBEDDED OPEN ARCHITECTURE ROBOTIC CONTROLLER." 3(2003): 308–15.
- Ortiz F.T. (2003), Diseño y construcción de un controlador en computadora para un robot paralelo de seis grados de libertad. Universidad Autónoma de Querétaro
- Pefhany, Spehro. 2000. "Modbus Protocol." *Penthon Media Inc* 5(January): 74.
- Postel, Jon, Steve Crocker, Vinton Cerf, and Creative Commons. 2009. s Creative Commons *Redes de Comunicaciones*. First Edit. Creative Commons.

- Rios R. A. (2004), Diseño y construcción de una tarjeta controladora de 3 ejes. Universidad Autónoma de Querétaro
- Xiao, Suhua et al. 2007. “An Open Architecture Numerical Control System Based on Windows CE.” 00: 1237–40.
- Xing, Hu, Huan Jia, and Liu Yanqianga. 2011. “Motion Control System Using SERCOS over EtherCAT.” *Procedia Engineering* 24: 749–53.
<http://linkinghub.elsevier.com/retrieve/pii/S1877705811055822> (September 12, 2013).
- Yan, Hehua et al. 2011. “Analysis Model for Ethernet-Based CNC Embedded Implementation.” *Procedia Engineering* 15: 448–53.
<http://linkinghub.elsevier.com/retrieve/pii/S1877705811015864> (September 12, 2013).
- Zhang, Xuhui, Qin Liu, and Sunan Wang. 2008. “Real-Time Communication Based on MMS+TCP/IP+ Ethernet for Embedded CNC.” *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*: 1–5.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4679269>.