Universidad Autónoma de Querétaro
Facultad de Ingeniería

# Implementación de protocolo Modbus RS-485 en sistema embebido basado en FPGA para red de sensores inerciales.

Tesis

Que como parte de los requisitos para obtener el
Grado de Maestro en Ciencias (Mecatrónica)

Presenta

Emmanuel Guillén García

Dirigido por:

Dr. Luis Morales Velázquez

Santiago de Querétaro, Qro, México, Octubre 2014.

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en ciencias (Mecatrónica)

# IMPLEMENTACIÓN DE PROTOCOLO MODBUS RS-485 EN SISTEMA EMBEBIDO BASADO EN FPGA PARA RED DE SENSORES INERCIALES.

**TESIS**

*Que como parte de los requisitos para obtener grado de*

Maestro en Ciencias

**Presenta:**
Emmanuel Guillén García

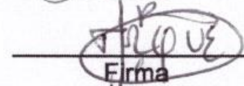
**Dirigido por:**
Dr Luis Morales Velázquez.

SINODALES.

Dr. Luis Morales Velázquez
Presidente

Firma

Dr. Roque Alfredo Osornio Ríos
Secretario


Firma

Dr. Jesús Rooney Rivera Guillén
Vocal

Firma

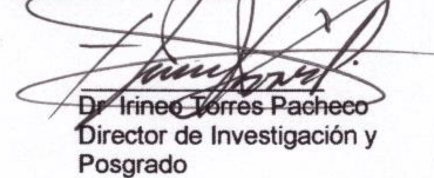
Dr. Jesús de Santiago Pérez
Suplente

Firma

Dr. Daniel Moriñigo Sotelo
Suplente

Firma

Dr Aurelio Domínguez González
Director de la Facultad



Dr Irineo Torres Pacheco
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Octubre, 2014
**México**

Actualmente, el monitoreo de procesos industriales o herramientas es de vital importancia para obtener un producto de calidad. El presente trabajo muestra una plataforma de monitoreo inercial con el protocolo Modbus, desarrollado como un sistema embebido basado en FPGA y apoyado de una herramienta informática (GUI) para el control del mismo. Este trabajo intenta satisfacer la necesidad de tener una herramienta de monitoreo que pueda trabajar en ambientes de tipo industrial con inmunidad al ruido y distancias de comunicación de por lo menos 10 metros. Este sistema se desarrolló siguiendo una estructura conjunta hardware-software para explotar lo mejor de cada uno. El hardware fue desarrollado bajo el lenguaje de descripción de hardware VHDL haciendo uso de las librerías IEEE y el software con lenguaje C estándar. Se siguió el estándar marcado por el protocolo Modbus en cuanto a la comunicación se refiere. El sistema desarrollado se probó en máquinas (torno y fresadora) con control numérico y sin control, obteniendo resultados satisfactorios con un monitoreo constante sin cortes y con distancias cercanas a los 300 metros, una velocidad de transferencia de entre 714 KHz y 2.5 MHz. En base a los resultados obtenidos, se puede concluir que el sistema creado cumple con los requerimientos planteados y deja espacio para posibles ajustes o mejoras proyectadas a futuro.

(**Palabras clave**: Modbus, Sistema embebido, FPGA, Monitoreo inercial)

Nowadays, monitoring industrial processes or working tools it's fundamental to obtain a quality product. This work shows an inertial monitoring platform using Modbus protocol, developed as an embedded system based on FPGA and supported by an informatics tool (GUI) to control the system. It is attempted to satisfy the needing of having a monitoring tool able to work in industrial environments with noise immunity and communication distances of at least ten meters. This system was developed following the hardware–software structure to get the best of each one. The hardware was developed under the VHDL hardware description language using IEEE libraries and C language was used to the software development. For communication between devices, the Modbus protocol was applied. The developed system was proved in machines with and without numeric control (lathe and milling machine), obtaining satisfactory results with a constant monitoring without breaks, distances close to 300 meters and a transfer velocity between 714 KHz and 2.5 MHz. Based on the obtained results, we can conclude that the developed system satisfy the set requirements and allows future adjustments or improvements.

(**Key words**: Modbus, FPGA, Embedded system, Inertial monitoring)

Mis principales agradecimientos son, a Dios por brindarme la paciencia y la motivación de terminar cada semestre del posgrado, a mis padres por otorgarme su apoyo tanto económico como moral a lo largo de este camino y a los profesores que se han preocupado por darme las herramientas necesarias para superar los retos que se puedan presentar en mi vida más adelante en especial a mi asesor Luis Morales que ha tenido mucha paciencia para guiarme en este trabajo. Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por otorgarme una beca y de esta manera poder realizar los estudios de posgrado para este trabajo. A la Universidad Autónoma de Querétaro por aceptarme en uno de sus programas de posgrado de calidad. Finalmente a P&G a través de Carlos Jaime Terán por compartir información técnica con la cual se pudieron validar algunas prestaciones del presente trabajo y a Tecno Turbo y Servicios Industriales a través de Carlos Alberto Vargas Hernández por facilitar sus instalaciones para realizar pruebas.

# Índice general

RESUMEN I

SUMMARY II

AGRADECIMIENTOS III

ÍNDICE GENERAL IV

ÍNDICE DE FIGURAS VI

ÍNDICE DE CUADROS VI

I. INTRODUCCIÓN 1

1.1. ANTECEDENTES 2
1.2. OBJETIVOS E HIPÓTESIS 4
1.3. JUSTIFICACIÓN 4
1.4. PLANTEAMIENTO GENERAL 5

II. FUNDAMENTACIÓN TEÓRICA 9

2.1 ESTADO DEL ARTE 9
2.2 INTERFAZ RS-485 10
2.3 PROTOCOLO DE COMUNICACIÓN MODBUS. 14
2.4 SISTEMA EMBEBIDO 19
2.5 EVALUACIÓN HEURÍSTICA 21
2.6 DISPOSITIVO DE LÓGICA PROGRAMABLE FPGA 24
2.7 LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL 26
2.8 LIBRERÍA GTKMM 26

III. METODOLOGÍA. 28

3.1. INTERFAZ RS-485 30
3.1.1. TRANSMISIÓN. 30

3.1.2. Recepción 34
**3.2. Firmware RS-485 del procesador 38**
**3.3 Protocolo Modbus 41**
**3.4 Interfaz gráfica de usuario 42**

**IV. PRUEBAS Y RESULTADOS 46**

**4.1. Pruebas y resultados de diseño 46**
4.1.1. Hardware 46
4.1.2. Enlace Hardware-Software (del sistema embebido) 47
4.1.3. Conexión con la PC 49
**4.2. Interfaz Gráfica de Usuario 50**
**4.3. Pruebas y resultados de comunicación 52**
**4.4. Pruebas y resultados de funcionamiento 54**

**V. CONCLUSIONES. 61**

**REFERENCIAS 63**

**APÉNDICE A, CÓDIGOS. 65**

**A.1 Módulo General. 65**
**A.2 Módulos de Transmisión. 67**
**A.3 Módulos de Recepción. 81**

**APÉNDICE B, ARTÍCULO 90**

# Índice de figuras


1. SISTEMA A IMPLEMENTAR. 5
2. DIFERENTES LÍNEAS (I-MICRO). 11
3. COMUNICACIÓN HALF DUPLEX (I-MICRO). 13
4. COMUNICACIÓN FULL DUPLEX (I-MICRO). 13
5. CICLO DE CONSULTAS (MODICON, 1996). 16
6. METODOLOGÍA GENERAL DEL SISTEMA. 29
7. PROCESO DE TRANSMISIÓN. 30
8. BLOQUE PARA LA TRANSMISIÓN. 31
9. MÓDULO DE TRANSMISIÓN RS485_TX 32
10. MÁQUINA SECUENCIAL PARA LA TRANSMISIÓN DE DATOS. 33
11. PROCESO DE RECEPCIÓN. 34
12. BLOQUE PARA LA RECEPCIÓN. 35
13. MÓDULO DE RECEPCIÓN RS485_RX 36
14. MÁQUINA SECUENCIAL PARA LA RECEPCIÓN DE DATOS. 37
15. RS485 EN SISTEMA EMBEBIDO 39
19. SIMULACIÓN DE ENVÍO Y RECEPCIÓN DE MENSAJE. 47
20. MENSAJE SOBRE LA CAPA FÍSICA. 48
21. COMPILADOR CON DATO RECIBIDO. 49
22. PRUEBA DE CONEXIÓN CON PC MEDIANTE LA INTERFAZ GRÁFICA. 50
23. INTERFAZ GRÁFICA DE USUARIO. 51
24. TOPOLOGIA DE RED. 53
25. DIAGRAMA DE PRUEBA EN FRESADORA. 55
26. ESCLAVOS EN FRESADORA. 55
27. GRÁFICAS DE DATOS. 56
28. DIAGRAMA EN CNC. 57
29. ESCLAVOS EN CNC. 58
30. GRÁFICAS DE DATOS. 59

# Índice de cuadros


1. FORMATO EN MODO ASCII. 17
2. FORMATO EN MODO RTU. 17
3. INTERRUPCIONES Y MACROS NECESARIAS PARA LA COMUNICACIÓN. 38
4. SENSORES Y MÓDULOS POSIBLES. 40
5. FORMATO DE DATOS GUARDADOS. 45
6. RESULTADOS DE COMUNICACIÓN. 54
7. COMPARACIÓN DE DATOS OBTENIDOS. 60

# I. INTRODUCCIÓN

Actualmente, una red de sensores es de vital importancia en procesos que necesitan ser constantemente monitoreados, esto, para mantener un control y conocer el funcionamiento actual de los instrumentos que están interactuando entre sí con el fin de obtener un producto final competitivo, ya sea a nivel industrial o académico. Nuestra institución no está exenta de las necesidades mencionadas y para cubrirlas se implementó una red para el control y monitoreo de diferentes sensores dentro de un área de trabajo de tipo industrial.

Para lograr la mejor comprensión del presente trabajo, se ha dividido el texto en varios capítulos con contenidos específicos. En el primer capítulo se abordan los antecedentes donde se mencionan trabajos que se han desarrollado en años anteriores tanto locales como fuera de la Universidad, además se plantea el objetivo general y los específicos, así como la hipótesis y justificación que dan pie al desarrollo de este trabajo y para terminar se encuentra el planteamiento general con el cual se pretende alcanzar los objetivos. El siguiente capítulo es la fundamentación teórica donde se presentan las múltiples herramientas (matemáticas, algoritmos, software, estándares) que fueron necesarias para completar este trabajo. El capítulo más importante es el dedicado a la metodología, donde se explica de manera detallada el procedimiento ejecutado para conseguir los objetivos en cada una de las etapas del trabajo. El penúltimo capítulo es para dar a conocer los resultados obtenidos a través del desarrollo de los casos de estudio, se discute si los objetivos fueron alcanzados y si la hipótesis planteada fue correcta. Finalmente se plasman las conclusiones obtenidas y el trabajo futuro que se considera pudiera llegar a realizarse.

## 1.1. Antecedentes

El protocolo Modbus fue desarrollado en 1979 por Modicon como un protocolo de comunicación entre dispositivos que es utilizado en múltiples aplicaciones con el fin de monitorear y programar dispositivos, establecer comunicación entre sensores e instrumentos, en aplicaciones RTU (*Remote Telemetry Unit, Unidad de Telemetría Remota*) y en diferentes proyectos donde sea necesario establecer comunicación para monitoreo, instrumentación y control (Modbus Organization, 2013).

En estrecha relación con la implementación del protocolo Modbus, se encuentran los instrumentos utilizados para medir las variables físicas de las que un proceso puede depender, un elemento necesario para efectuar estas mediciones es el sensor, que comúnmente puede ser empleado para medir temperatura, presión, posición, aceleración, entre otras variables. A nivel local se ha desarrollado tecnología implementando sensores, tal es el caso del trabajo de Lomelí (2012), donde implementó en FPGA (*Field Programmable Gate Array, Arreglo de Compuertas Programables en campo*), una unidad de preprocesamiento digital para giroscopios digitales de tres ejes tipo MEMS, esto con la finalidad de aumentar la exactitud en las señales del sensor para obtener la orientación de un robot. También Rivera et al. (2008) realizaron un algoritmo progresivo para el auto ajuste de sensores inteligentes con el objetivo de minimizar el error no lineal en la medición del sensor. En el plano internacional tenemos a Gervais (2011), que propuso una plataforma para sensores inteligentes, definida como la combinación entre el sensor y un procesamiento con una arquitectura dedicada.

Las redes de sensores de cualquier tipo, son recurrentemente utilizadas para el monitoreo de procesos industriales o investigaciones científicas para el desarrollo de nuevas tecnologías, un ejemplo es la desarrollada por Ghosh et al (2012), donde propusieron un modelo único y eficiente para redes de sensores, con el objetivo de crear carreteras senso-inteligentes, esto lo realizan mediante algoritmos, que calculan la forma en la que se deben desplegar los nodos de sensores en el camino y obtener así las

distintas variaciones de forma que puede tener el mismo. Meneses et al. (2007), propusieron una nueva arquitectura para redes de sensores inteligentes, manejada mediante eventos derivados de una compresión logarítmica de los datos dentro del sensor, esta arquitectura también provee la configuración y el monitoreo de datos para manejar el sistema.

Por otra parte, el protocolo Modbus es ampliamente utilizado por miles de industrias de todos los sectores para satisfacer sus necesidades de comunicación entre dispositivos, por esta razón, el desarrollo de nuevas tecnologías basadas en este protocolo se considera de utilidad. Actualmente, en el desarrollo de tecnologías para la implementación o estudio de redes industriales de herramientas, sensores y diferentes instrumentos, se utiliza Modbus como punto de partida o parte central del trabajo. Tal es el caso del trabajo propuesto por Rane et al. (2010), donde realizaron la arquitectura del protocolo Modbus en VHDL (*Very high speed integrated circuit Hardware Description Language, lenguaje descriptivo de circuitos integrados de muy alta velocidad*), para la administración remota de una red de dispositivos con sensores de temperatura y humedad, utilizan la interfaz UART y RS-232, también presentan los resultados de la simulación obtenida para la arquitectura diseñada. Otro caso es el trabajo de Hui et al. (2012), donde desarrollaron una aplicación para establecer comunicación de Gateway entre un sistema EPA (*Ethernet For Plant Automation, Ethernet Para La Automatización de la Planta*) y el protocolo Modbus por medio de la interfaz RS-485, para presentar una solución estable, segura y flexible en el proceso de control de una planta eléctrica. Por su parte Suhartono et al. (2012), construyeron una red de área local entre generadores eléctricos basándose en el protocolo Modbus, utilizando después la interfaz RS-485 para comunicar los generadores, y así, enviar los datos recopilados a un convertidor de señal para transmitir los datos a una PC (Computadora) mediante RS-232. Finalmente los datos son enviados a un servidor donde permanecen almacenados para su consulta utilizando un sitio web.

En los trabajos mencionados, es evidente la importancia que tiene en la actualidad el monitoreo de procesos y la medición de variables físicas de una manera precisa y eficiente, no solo para aplicaciones industriales sino también para la vida cotidiana de una persona. En este trabajo se pretende la integración de varios dispositivos estudiados por separado como sensores inerciales para medir el comportamiento en sistemas industriales, redes y protocolos de comunicación industrial que permitan el tránsito de información de un lugar a otro, todo esto implementado en un sistema embebido que permitan satisfacer las necesidades particulares de un sector industrial o un laboratorio de estudio, en éste caso en particular el diseño e implementación de una red flexible basada en FPGA, que permita integrar sensores propietarios y comerciales para aplicaciones de instrumentación y control.

Actualmente se cuenta con un sistema de monitoreo basado en el protocolo de comunicación USB. Se utiliza cable blindado con aislamiento galvánico de una longitud de 2 a 5 metros, cada cable tiene un filtro de ferrita para evitar interferencias. Por las características de la comunicación USB la velocidad de transferencia se encuentra en un margen de 1.5 a 12 Mb/s, posee un ancho de banda de 568 000 Bps (Bites por segundo) con un tiempo de retraso de 1ms (milisegundo) en la transferencia de cada dato. El sistema es controlado por una computadora Toshiba TECRA m9. Debido a que el USB fue concebido como un protocolo de oficina, funciona correctamente en ese tipo de ambiente, pero, al estar en contacto directo con una máquina de alta potencia se desconecta, estas máquinas pueden ser: motores de inducción, arrancadores, máquinas de soldar, etc.

Ante las características del sistema mencionado, es deseable desarrollar un nuevo sistema que incorpore elementos para un uso más industrial, las características deseadas son: mayor resistencia al ruido para no sufrir desconexiones frente a motores, herramientas de maquinado, etc., longitud del cable para comunicación de más de 10 metros, y un software para controlar la comunicación con los dispositivos conectados y almacenar los datos obtenidos en el monitoreo.

## 1.2. Objetivos e Hipótesis

El objetivo principal de este trabajo es, implementar el protocolo de comunicación industrial ModBus RS-485 en un sistema embebido propietario basado en FPGA para su aplicación en redes de sensores inerciales, como giroscopios y acelerómetros.

Con la finalidad de dividir el problema en partes más específicas se tienen los siguientes objetivos particulares.

- Desarrollar el módulo hardware para manejo de ModBus-485 utilizando VHDL.
- Integrar el sistema embebido a partir del procesador XQ16V7 y el módulo 485 para la implementación del protocolo ModBus.
- Desarrollar el manejador de firmware para el sistema embebido utilizando lenguaje C.
- Desarrollar el firmware de los microprocesadores en los sensores para la integración de la red utilizando lenguaje C.
- Desarrollar la interfaz de usuario para el control y monitoreo del sistema en lenguaje C mediante la librería gráfica GTKMM.
- Realizar pruebas de comunicación para verificar la operación de la red y medir el ancho de banda mediante pruebas de consultas a los sensores y tiempo de respuesta de la red.

## 1.3. Justificación

Este trabajo, surge de la necesidad de implementar un protocolo de comunicación robusto para integrar una red de sensores con mayor inmunidad al ruido que el sistema existente, agregando la funcionalidad de un sistema descentralizado; obteniendo así, una mayor flexibilidad en el arreglo topográfico de la red. Además, la implementación se

realizará en un sistema embebido, ya que en comparación con una unidad de procesamiento dedicado se obtiene un uso óptimo de recursos en cuanto al compromiso que se tiene con la velocidad de procesamiento y el costo de bloques lógicos. Otro punto importante, es, que el manejador del protocolo será un desarrollo propio construido de acuerdo a las necesidades que se tienen para la red y el sistema embebido, con esto se podrán reducir los costos en el mantenimiento técnico de la red y en la adaptación de nuevos dispositivos que se puedan necesitar a futuro.

## 1.4. Planteamiento general

La figura 1 muestra un diagrama a bloques del sistema que se pretende implementar en este trabajo, donde se puede ver los sensores inerciales, el sistema embebido y una PC con la que se establecerá la comunicación con el usuario.

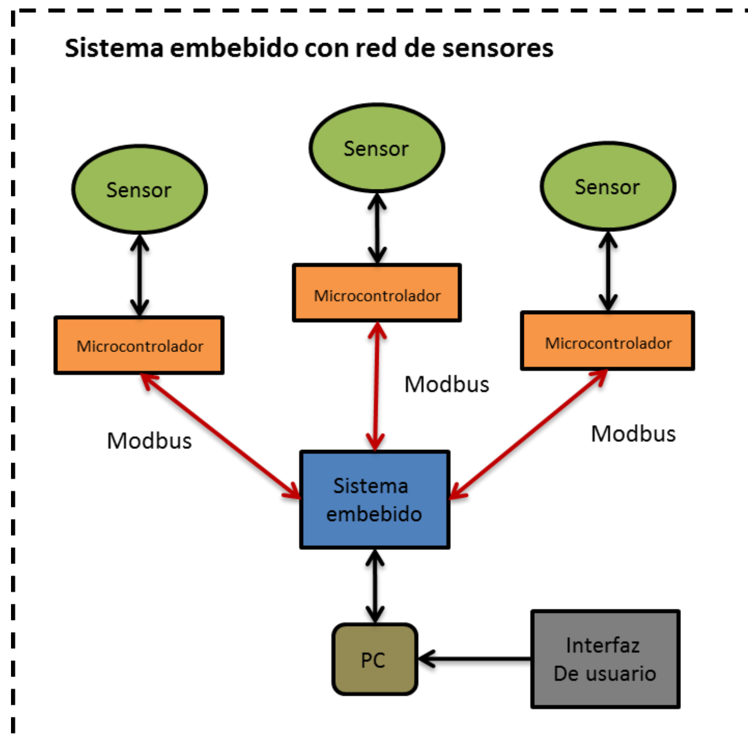
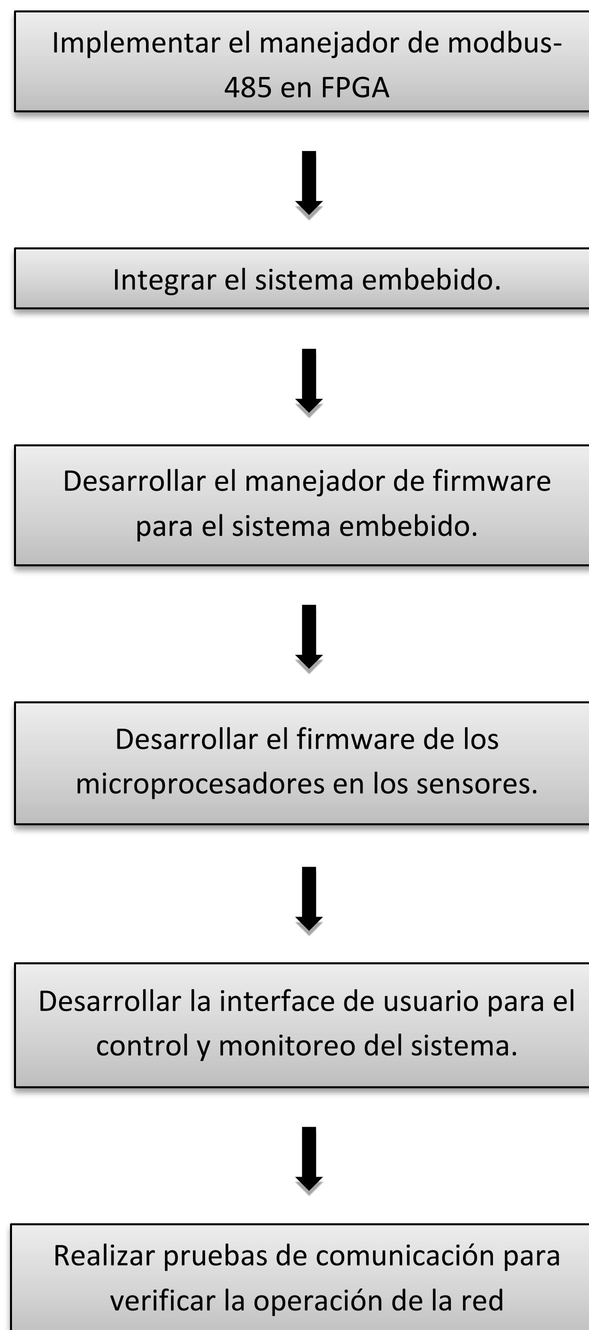



Figura 1. Sistema a implementar.

Para alcanzar el objetivo principal y comprobar la hipótesis planteada, se ha dividido el trabajo en varios pasos secundarios, con la finalidad de obtener un mejor panorama de las actividades a realizar, el siguiente diagrama muestra a grandes rasgos la metodología para realizar el trabajo propuesto.



Este trabajo pretende satisfacer la necesidad de una red de sensores flexible, con la suficiente inmunidad al ruido para operar de manera adecuada en un ambiente industrial tomando como base protocolos de comunicación pertinentes y sistemas embebidos propietarios. La estrategia para llevar a cabo el proyecto de investigación se plantea de varias etapas enlistadas a manera de tareas a continuación.

**a) Implementar el manejador de modbus-485 en FPGA.**

Para implementar el manejador, primero se deben de realizará los módulos que se encargarán de emular las tareas de la capa física del protocolo Modbus, dichas tareas son el encapsulado y desencapsulado de los paquetes con la información que viajará a través de la red, el reconocimiento de inicio y fin de transmisión y el desecho de información inservible. Los módulos se desarrollarán en VHDL utilizando el programa ISE de Xilinx y se implementará en un FPGA Spartan 3E de Xilinx.

**b) Integrar el sistema embebido.**

Esta tarea consiste en hacer las conexiones entre el procesador XQ16V7 que es un procesador propietario desarrollado en la UAQ y el manejador de Modbus.

**c) Desarrollar el manejador de firmware para el sistema embebido.**

Para esta parte se necesitará elaborar la parte lógica del Modbus en los procesadores, esto quiere decir, el código necesario para que se establezca la comunicación con el manejador de modbus, este se desarrollará en lenguaje C mediante el programa DevC o cualquier otro que se considere pertinente.

**d) Desarrollar el firmware de los microprocesadores en los sensores.**

Este firmware consiste nuevamente en código C para establecer la comunicación entre el sensor y el microcontrolador, donde el sensor será el dispositivo maestro y el procesador realizará las funciones de esclavo para la comunicación en la red.

**e) Desarrollar la interfaz gráfica de usuario para el control y monitoreo del sistema.**

La interfaz se creará utilizando la librería gráfica GTKMM en lenguaje C y consistirá en elaborar los elementos necesarios para que el usuario pueda monitorear la actividad dentro de la red, los elementos pueden ser cuadros de texto, botones, indicadores, etc. Se buscará cumplir con la mayoría de las heurísticas de diseño y se realizarán pruebas de usabilidad cuando se considere necesario.

**f) Realizar pruebas de comunicación para verificar la operación de la red.**

Finalmente se necesitará llevar a cabo diagnósticos de comunicación, para verificar la operación general de la red y medir el ancho de banda, esto mediante pruebas de consultas a los sensores y tiempo de respuesta de la red.

# II. FUNDAMENTACIÓN TEÓRICA

## 2.1 Estado del arte

Hasta los años 60, el control industrial se venía realizando mediante lógica cableada a base de relés electromecánicos. El desarrollo de la electrónica hizo posible la implantación de los dispositivos con microprocesador, también llamados Autómatas Programables o Controladores Lógicos Programables.

La creación de los PLCs (Programmable Logic Controller, Controlador Lógico Programable) eliminó el enorme costo que significaba el mantenimiento de un sistema de control electromecánico. Esto dio paso al desarrollo de un protocolo de comunicación que en 1979 fue llamado Modbus, originalmente utilizaba una interface de comunicación RS-232, pero con el tiempo cambio a RS-485 debido a que con este medio se pueden manejar distancias más grandes de cableado a mayor velocidad y es posible establecer redes multipunto. Debido a que este protocolo es de acceso libre y el usuario puede moldearlo de acuerdo a sus necesidades, lo que se traduce en menores costos de mantenimiento, actualmente Modbus es utilizado por la industria para la comunicación de sensores inteligentes, PLCs e instrumentos de laboratorio. Otras compañías han optado por el desarrollo de dispositivos que utilicen este medio para la comunicación y control entre los mismos, por ejemplo válvulas industriales y sistemas de válvulas de control. También es posible encontrar empresas que ofrecen capacitación y cursos de información para conocer las características y funcionamiento de Modbus para que el usuario pueda optimizar una red basada en PLCs.

## 2.2 Interfaz RS-485

El estándar RS-485 (TIA/EIA-485-A o RS-485) es compatible con las versiones anteriores de RS-422; sin embargo, está optimizado para aplicaciones de línea compartida o multipunto. La salida del controlador RS-422/485 puede estar activa (habilitada) o en tercer estado (inhabilitada). Esta capacidad permite que múltiples puertos sean conectados en un bus multipunto y sondeados selectivamente (i-micro).

Esta interfaz tiene muchas ventajas con respecto a RS 232, entre las cuales se mencionan:

**Bajo costo:** Los Circuitos Integrados para trasmitir y recibir son baratos y solo requieren una fuente de +5V para poder generar una diferencia mínima de 1.5v entre las salidas diferenciales. En contraste con RS-232 que en algunos casos requiere de fuentes dobles para alimentar algunos circuitos integrados.

**Capacidad de interconexión:** RS-485 es una interfaz multi-enlace con la capacidad de poder tener múltiples transmisores y receptores. Con una alta impedancia receptora, los enlaces con RS-485 pueden llegar a tener a lo máximo hasta 256 nodos.

**Longitud de Enlace:** En un enlace RS-485 puede tener hasta 1200 metros de longitud, comparado con RS-232 que tiene unos límites típicos de 50 a 100 pies.

**Rapidez:** La razón de bits puede ser tan alta como 10 Mega bits/ segundo.

La razón por la que RS-485 puede transmitir a largas distancias, es porque utiliza el balanceo de líneas. Cada señal tiene dedicados un par de cables, sobre uno de ellos se encontrará un voltaje y en el otro se estará su complemento, de esta forma, el receptor responde a la diferencia entre voltajes. La figura 2 muestra una línea balanceada y otra sin balanceo

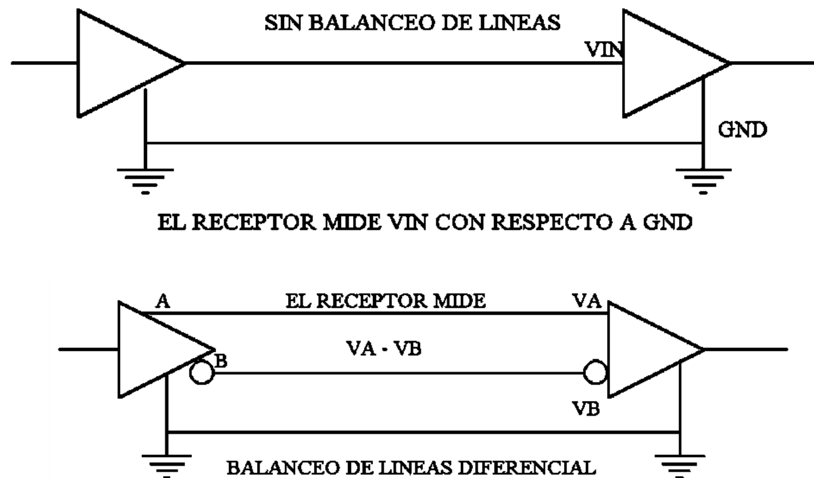



Figura 2. Diferentes líneas (i-micro).

En cuanto a las líneas balanceadas la TIA/EIA-485 designa a estas dos líneas como A y B. En el controlador TX, una entrada alta TTL causa que la línea A sea más positiva (+) que la línea B, mientras que un bajo en lógica TTL causa que la línea B sea más positiva (+) que la línea A. Por otra parte en el controlador de recepción RX, si la entrada A es más positiva que la entrada B, la salida lógica TTL será "1" y si la entrada B es más (+) que la entrada A, la salida lógica TTL será un "0" (i-micro).

**Requerimientos de voltaje.**

Las interfaces típicas RS-485 utilizan una fuente de +5 Volts, pero lo niveles lógicos de los transmisores y receptores no operan a niveles estándares de +5V o voltajes lógicos CMOS. Para una salida válida, la diferencia entre las salidas A y B debe ser al menos +1.5V. Si la interfaz está perfectamente balanceada, las salidas estarán desfasadas igualmente a un medio de la fuente de Voltaje.

En el receptor RS-485, la diferencia de voltaje entre las entradas A y B necesita ser 0.2V. Si A es al menos 0.2V más positiva que B, el receptor ve un 1 lógico y si B es al menos 0.2v más positivo que A, el receptor ve un 0 lógico. Si la diferencia entre A y B es menor a 0.2v, el nivel lógico es indefinido. Si esto ocurre habría un error en la transmisión y recepción de la información.

La diferencia entre los requerimientos del Transmisor y el Receptor pueden tener un margen de ruido de 1.3V. La señal diferencial puede atenuarse o tener picos de largo como de 1.3v, y aun así el receptor vera el nivel lógico correcto. El margen de ruido es menor que el de un enlace RS-232, no hay que olvidar que RS-485 maneja señales diferenciales y que cancela la mayoría del ruido a través de su enlace.

El total de corriente utilizada por un enlace RS-485 puede variar debido a la impedancia de los componentes, incluyendo los Transmisores, Receptores, cables y la terminación de los componentes. Una baja impedancia a la salida del transmisor y una baja impedancia en los cables, facilitan los cambios de nivel y asegura que el receptor vea la señal, no importa que tan larga sea la línea de transmisión. Una alta impedancia en el receptor decrementa la corriente en el enlace e incrementa la vida de la fuentes de voltaje.

La terminación de los componentes, cuando se utiliza tiene un gran efecto sobre la corriente en el enlace. Muchos enlaces con RS-485 tienen una resistencia de 120 ohms a través de las líneas A y B en cada extremo de la línea. Por lo tanto cada, enlace tiene dos terminales (i-micro).

**Comunicación en modo Half Duplex.**

El término Half Duplex en un sistema de comunicación se refiere, a que solamente en un tiempo determinado, el sistema puede transmitir o recibir información, sin embargo no lo puede hacer al mismo tiempo. En muchos enlaces del tipo RS-485 se comparte el BUS. La figura 3 muestra el esquema de una comunicación RS-485 en Modo Half Duplex.

Como se puede observar existe una línea de control en las entradas RE y DE, la cual habilita a los controladores en un solo sentido. Por lo tanto, se debe tener cuidado de no transmitir y recibir al mismo tiempo, ya que se podría crear una superposición de información (i-micro).

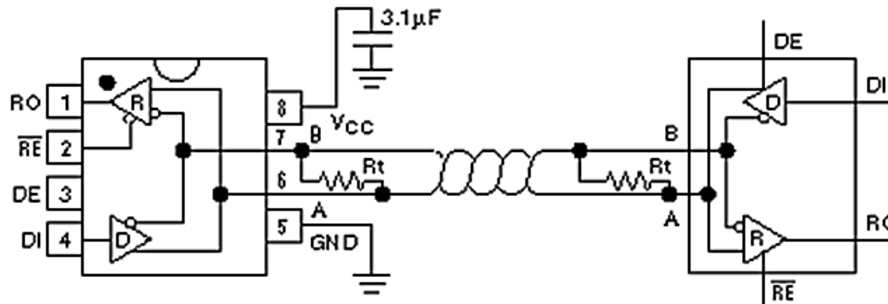



Figura 3. Comunicación Half Duplex (i-micro).

**Comunicación en modo Full Duplex.**

El término Full Duplex se refiere a que un sistema puede transmitir y recibir información simultáneamente. Bajo este concepto la interfaz RS-485 está diseñada para sistemas multipunto, esto significa que los enlaces pueden llegar a tener más de un transmisor y receptor, ya que cada dirección o sea Transmisión y Recepción tienen su propia ruta. La figura 4 muestra el esquema de una comunicación RS-485 en Modo Full Duplex (i-micro).

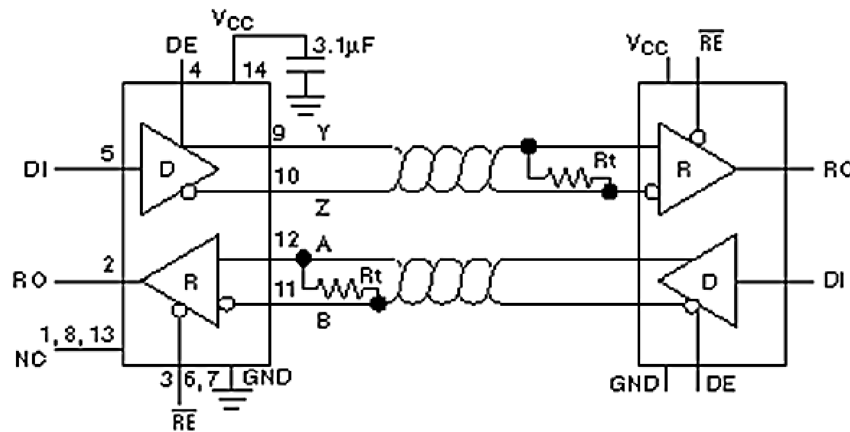



Figura 4. Comunicación Full Duplex (i-micro).

## 2.3 Protocolo de comunicación Modbus.

Modbus es un protocolo de comunicaciones publicado por Modicon en 1979. Que en ese tiempo, fue centrado principalmente en la comunicación de Controladores Lógicos Programables (PLC) fabricados por ellos y utilizados en la industria de la automatización. Modicon es actualmente una empresa propiedad de Schneider Electric y en 2004 el estándar Modbus fue trasladado a una organización sin fines de lucro, Modbus-IDA, cuyos miembros son principalmente usuarios y proveedores en la industria de la automatización.

El protocolo Modbus es una estructura utilizada para establecer comunicación de tipo maestro–esclavo entre dispositivos como PLCs, sensores, sensores inteligentes, equipos de cómputo, entre otros, dicha estructura define parámetros como:

- El formato de una petición hacia los dispositivos compatibles.
- El formato de respuesta de los dispositivos compatibles.
- La disposición y el contenido de los mensajes enviados dentro de la red.
- Como cada dispositivo debe de manejar la información recibida.
- El formato de los datos u otra información contenida en un mensaje.

Modbus es un estándar abierto al público y comúnmente utilizado en ambientes industriales, su funcionamiento tiene una base muy sencilla, donde, el dispositivo maestro pregunta y los esclavos responden o actúan en función de lo que el maestro requiera, solo este puede iniciar la comunicación y existirá una sola respuesta dependiendo del esclavo con el que se esté comunicando. Los dispositivos involucrados se encuentran dentro de una red de comunicación. (modbus organization, 2013) y (xmcarne, 2013).

**Mensajes dentro de la red Modbus.**

Los mensajes que circulan dentro de una red modbus, tienen una estructura definida para garantizar que los mensajes lleguen al dispositivo correcto y que este pueda

interpretarlos de manera correcta, dicha estructura está definida de la siguiente manera para los mensajes enviados desde el maestro:

- Una dirección que indica con que esclavo se desea comunicar. La dirección debe enviarse en el primer bite de datos del mensaje.
- Un código de función que define la acción que realizara el esclavo. La función debe de enviarse en el segundo bite de datos.
- Un bloque de datos que especifican la acción a realizar, por ejemplo, si el bloque de función indica que se debe de leer un registro con varias salidas, el bloque de datos indicara que salida es la que se requiere leer.
- Finalmente se envía un código de redundancia cíclica (CRC) o código de redundancia longitudinal (LRC) que permitirá al esclavo asegurar que el mensaje recibido es correcto.

Para el mensaje de respuesta la estructura que se debe emplear es similar a la mencionada anteriormente. Se debe de seguir lo siguiente:

- La dirección del dispositivo que está respondiendo.
- El código de la función solicitada.
- Los datos almacenados en las variables solicitadas, estos datos deben de ser enviados en bloques de bites según sea necesario.
- El código para comprobar que el mensaje recibido sea correcto.

La figura 5 muestra el ciclo de consultas en la comunicación.

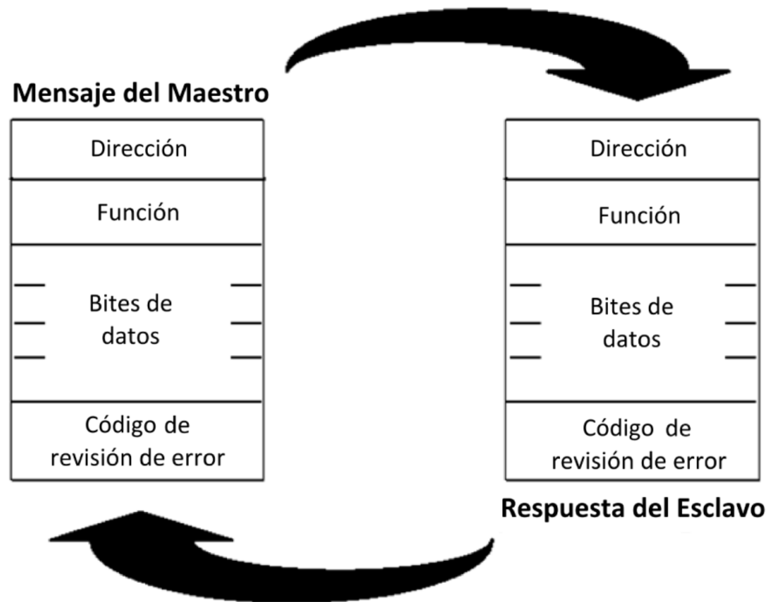



Figura 5. Ciclo de consultas (Modicon, 1996).

**Modos de transmisión serial.**

Los dispositivos pueden ser configurados para comunicarse en redes Modbus estándar utilizando alguno de los dos modos de transmisión: ASCII o RTU. Los usuarios seleccionan el modo deseado, junto con los parámetros de comunicación del puerto serie (velocidad de transmisión, modo de paridad, etc). El modo de transmisión y los parámetros de comunicación deben ser el mismo para todos los dispositivos en la red (Modicon, 1996).

Modo ASCII.

Cuando los dispositivos están configurados para comunicarse en utilizando ASCII (*American Standard Code for Information Interchange*, *Código Estándar Americano para el Intercambio de Información*), cada byte en un mensaje se envía como dos caracteres ASCII. La principal ventaja de este modo es que permite intervalos de tiempo de hasta un segundo que ocurren entre caracteres sin causar un error.

El formato para cada bite en modo ASCII se presenta en el Cuadro 1.

Cuadro 1. Formato en modo ASCII.

| Elemento. | Descripción. |
|---|---|
| Sistema de codificación | Hexadecimal, caracteres ASCII 0-9, A-F. Un caracter hexadecimal contenido en cada caracter ASCII del mensaje |
| Bits por Bite | 1 bit de inicio<br>7 bits de datos, el bit menos significativo se envía primero<br>1 bit de paridad; 0 bit para no paridad<br>1 bit de fin si se usa paridad; 2 bits si no se usa |
| Campo de chequeo de error | LRC (*Longitudinal Redundancy Check, Código de Redundancia Longitudinal*) |

Modo RTU.

Cuando los dispositivos están configurados para comunicarse en modo RTU (*Remote Terminal Unit, Unidad de terminal remota*), cada byte en un mensaje contiene dos caracteres en hexadecimal de 4 bits. La principal ventaja de este modo es que su mayor densidad de caracteres permite un mejor rendimiento de datos que ASCII para la misma velocidad de transmisión. Cada mensaje debe ser transmitido en un flujo continuo.

El formato para cada bite en modo RTU se presenta en el cuadro 2.

Cuadro 2. Formato en modo RTU.

| Elemento. | Descripción. |
|---|---|
| Sistema de codificación | Binario 8 bit, hexadecimal 0-9, A-F. Dos caracteres hexadecimales contenidos en cada campo de 8 bit del mensaje. |
| Bits por Bite | 1 bit de inicio<br>8 bits de datos, el bit menos significativo se envía primero<br>1 bit de paridad; 0 bit para no paridad<br>1 bit de fin si se usa paridad; 2 bits si no se usa |
| Campo de chequeo de error | CRC (*Cyclical Redundancy Check, Código de Redundancia Cíclica*) |

**Métodos de revisión de error.**

Las redes modbus utilizan dos tipos de métodos para la verificación de errores: comprobación de paridad (par o impar) esta es opcional y se aplica a cada carácter. Comprobación de trama (LRC o CRC) se aplica a todo el mensaje. Ambas comprobaciones son generadas por el dispositivo que transmite el mensaje y el que recibe recalcula los códigos con el mensaje recibido para asegurarse que el mensaje recibido sea correcto (Modicon, 1996).

Código de Redundancia Cíclica

En modo RTU, los mensajes incluyen un campo de comprobación de errores que se basa en el método de comprobación de redundancia cíclica (CRC). El campo CRC comprueba el contenido de todo el mensaje. Se aplica con independencia de cualquier método de paridad utilizado para los caracteres individuales del mensaje.

El CRC es un código de comprobación que se incluye al final de un mensaje para verificar que durante la transmisión no se ha modificado el contenido del mensaje original, es decir, una manera de detectar errores en la comunicación. El CRC no determina el origen del error, pero sí indica que ha habido uno.

Durante la comunicación con un dispositivo se producen dos cálculos del CRC, uno para el mensaje de envío y otro para el de respuesta. El CRC del mensaje de respuesta lo genera internamente el dispositivo esclavo, y el de envío es generado por el esclavo.

La idea del cálculo del CRC reside en considerar el mensaje como un extenso número binario. Este número entonces de divide por un cierto número, binario también, y el resto de esa división es el CRC.

El CRC no es único para un mensaje determinado, ya que según el divisor utilizado se obtendrá un resultado distinto, es decir, habrá un CRC por cada divisor utilizado. En realidad lo importante no es que se calcule de una forma u otra, sino que a la hora de comprobarlo, se sepa qué divisor se ha utilizado, y coincida el CRC que acompaña el mensaje con el CRC generado con ese mismo divisor, es decir, antes de enviar el mensaje se debe saber qué divisor utiliza el dispositivo de destino para comprobarlo.

El campo CRC contiene un valor binario de 16 bits, es calculado por el dispositivo de transmisión, que añade el CRC al final del mensaje. El dispositivo receptor vuelve a calcular un CRC durante la recepción del mensaje, y compara el valor calculado con el valor real que recibió en el campo CRC. Si los dos valores son diferentes, se marcará un error. Estos códigos se basan en el uso de un polinomio generador *G(X)* de grado *r*, y en el principio de que n bits de datos binarios se pueden considerar como los coeficientes de un polinomio de orden *n*-1. Por ejemplo, los datos 10111 pueden tratarse como el polinomio $x4 + x2 + x1 + x0$. A estos bits de datos se añaden *r* bits de redundancia de forma que el polinomio resultante sea divisible por el polinomio generador, sin generar resto. El receptor verificará si el polinomio recibido es divisible por *G(X)*. Si no lo es, habrá un error en la transmisión.

## 2.4 Sistema embebido

Un sistema embebido posee hardware junto con software embebido como uno de sus componentes más importantes. Es un sistema computacional dedicado para aplicaciones o productos. Puede ser un sistema independiente o parte de un sistema mayor, y dado que usualmente su software está embebido en ROM (*Read Only Memory, Memoria de Solo Lectura*) no necesita memoria secundaria como una computadora. Un sistema embebido tiene tres componentes principales:

a) Hardware.
b) Un software primario o aplicación principal. Este software o aplicación lleva a cabo una tarea en particular, o en algunas ocasiones una serie de tareas.
c) Un sistema operativo que permite supervisar las aplicaciones, además de proveer los mecanismos para la ejecución de procesos. En muchos sistemas embebidos es requerido que el sistema operativo posea características de tiempo real.

Es importante resaltar que el software que se ejecuta en un sistema embebido es diseñado bajo algunas restricciones importantes: (i) cantidades pequeñas de memoria, generalmente en el orden de los KB, (ii) capacidades limitadas de procesamiento, generalmente los procesadores poseen velocidades que no superan los MHz, (iii) la necesidad de limitar el consumo de energía en cualquier instante, bien sea en estado de ejecución o no (Pérez, 2009).

**Características de los sistemas embebidos.**

Los sistemas embebidos poseen ciertas características que los distinguen de otros sistemas de cómputo, a continuación se presentan las más importantes (Pérez, 2009).

- Funcionamiento específico. Un sistema embebido usualmente ejecuta uno o varios programas específicos de forma repetitiva. En contraste, un sistema de cómputo personal que ejecuta una amplia variedad de programas, como hojas de cálculo, juegos, etc.; además nuevos programas son añadidos frecuentemente. Por supuesto pueden haber excepciones, podría ocurrir que el programa del sistema embebido fuese actualizado a una nueva versión. Por ejemplo, un teléfono celular podría actualizarse de alguna manera.

- Fuertes limitaciones. Todos los sistemas de computación poseen limitaciones en sus métricas de diseño, pero en los sistemas embebidos son muy fuertes. Una métrica de diseño es una medida de algunas características de implementación, como: costo, tamaño, desempeño, y consumo de energía. Los sistemas embebidos generalmente deben ser poco costosos, poseer un tamaño reducido, tener un buen desempeño para procesar datos en tiempo real, y además consumir un mínimo de energía para extender el tiempo de vida de las baterías o prevenir la necesidad de elementos adicionales de enfriamiento.

- Reactivos y tiempo real. Muchos sistemas embebidos deben ser reactivos o reaccionar ante cambios en el ambiente, además de realizar algunos cálculos en tiempo real sin ningún retraso, es decir, se deben tener resultados en tiempos fijos ante cualquier eventualidad. Por ejemplo, el módulo de control de viaje de un automóvil continuamente monitorea la velocidad y los sensores de frenos, reaccionando ante cualquier eventualidad. Ante un estímulo anormal, el módulo de control debe realizar los cálculos de forma precisa y acelerada para garantizar la entrega de los resultados dentro de un tiempo límite, una violación en este tiempo podría ocasionar la pérdida del control del automóvil. En contraste, un sistema de escritorio se enfoca en realizar cálculos con una frecuencia no determinada y la demora de los mismos no producen fallas en el sistema.

## 2.5 Evaluación Heurística

La evaluación heurística (HE) es una técnica utilizada para analizar la facilidad de uso en las primeras etapas del diseño de una interfaz de usuario. Se desarrolló en Dinamarca con el objetivo de crear un método para analizar el diseño de interfaces de una manera informal, manejable y fácil de enseñar. Para llevar a cabo una evaluación heurística es necesario que varias personas analicen el diseño de interfaz de usuario para verificar si infringe alguna de las heurísticas, si lo hace es necesario modificar el diseño. El análisis puede estar en cualquier etapa de desarrollo del producto, puede ser tan preliminar como un borrador a lápiz del sistema, un prototipo parcial en algún paquete o librería de desarrollo o un sistema operacional completo. Sin embargo, conforme el sistema esté más avanzado, será más complicado arreglar los problemas de usabilidad, por lo tanto HE tiene más valor en los primeros borradores o prototipos. De manera general, varias personas harán evaluación HE de un diseño ya que las investigaciones han

demostrado que diferentes personas encontrarán diferentes infracciones. En un sistema sencillo (como el de un cajero automático que permite aproximadamente 6 funciones diferentes), probablemente cuatro o cinco examinadores serán suficiente para encontrar problemas de utilidad. Para sistemas más complicados se necesitan un número mayor de personas que evalúen el sistema, sin embargo, debido a que es una técnica muy simple de aprender a usar, no es difícil encontrar miembros del equipo de desarrollo que realicen la evaluación. Además, con que una o dos personas evalúen el sistema se encontrarán errores de usabilidad que una vez corregidos mejorarán el uso del sistema que se está diseñando (Icarnegie, 2008).

A continuación se encuentran algunas heurísticas importantes a considerar en el diseño de cualquier interfaz de usuario enumeradas en (Icarnegie, 2008).

1. Visibilidad del estatus del sistema.

   La computadora debe mantener al usuario informado de lo que está ocurriendo, las entradas que ha recibido la computadora, el proceso que se está llevando a cabo y los resultados del procesamiento. La computadora debe proveerle información al usuario de manera visual o a través de algún sonido. Si no es así, el usuario no tiene la información necesaria para entender lo que está sucediendo.

2. Comparación entre el sistema y el mundo real.

   El diseño de la interfaz del usuario debe utilizar los conceptos, lenguaje, y las convenciones del mundo real que sean familiares para el usuario. Esto significa que el programador del sistema debe comprender la tarea que el sistema debe desempeñar, desde el punto de vista del usuario. Los aspectos culturales se hacen relevantes para el diseño de sistemas computacionales globales.

3. Control y libertad del usuario.

   Permite que el usuario tenga control de la interacción. Los usuarios deben poder corregir sus acciones fácilmente, salirse de la interacción rápidamente en cualquier

momento, y no verse forzados a hacer una serie de acciones controladas por la computadora. Los usuarios cometerán errores y por lo tanto, necesitan una opción fácil para corregirlos.

4. Estándares y consistencia.

   La información que es igual debe verse igual (mismas palabras, iconos y posiciones en la pantalla). La información que es diferente debe ser expresada de manera diferente. Tal consistencia debe mantenerse dentro de la aplicación y de la plataforma. Los programadores deben conocer las convenciones de las plataformas.

5. Prevención de errores.

   El sistema debe prevenir que se cometan errores tanto como sea posible, por ejemplo, si hay una cantidad limitada de acciones legales en una parte de la aplicación, ayuda a los usuarios a seleccionarlas, en lugar de permitir que lleven a cabo una acción para luego decirles que cometieron un error. Esto se puede considerar como una parte de la primera heurística, Visibilidad del Estatus del Sistema (que entradas son aceptables para el sistema computacional), pero debido a que es tan importante y tan frecuentemente violada, merece su propia heurística.

6. Reconocer en vez de recordar.

   Se debe mostrar al usuario todos los objetos y acciones disponibles. No pedirle al usuario que se acuerde de la información de otra pantalla de la aplicación. Esta heurística es una aplicación directa de las teorías de la memoria humana. Es más fácil para alguien reconocer lo que tiene que hacer si existen ciertas pistas en el ambiente que le lleguen a la memoria de trabajo a través de la percepción y a través del conocimiento almacenado en la memoria de largo plazo.

7. Flexibilidad y eficiencia en el uso.

   El diseño debe contar con atajos en tecleado que permita que los usuarios expertos puedan acelerar su interacción (en lugar de siempre usar los menús e iconos con un mouse). Los usuarios expertos deben de poder adaptar la interfaz para que las acciones frecuentes se hagan con mayor rapidez.

8. Diseño estético y minimalista.

   Eliminar los elementos irrelevantes de la pantalla que sólo representan desorden y confusión para el usuario.

9. Como ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores.

   Los mensajes de error deben ser escritos en un lenguaje simple, deben explicar el problema dar consejo constructivo de como corregir el error. Una vez más, esto se puede considerar parte de la primera heurística, visibilidad del estatus del sistema, pero es tan importante e ignorada tan frecuentemente que merece su propia heurística.

10. Ayuda y documentación.

    Si el sistema no es extremadamente sencillo, va a ser necesario que cuente con sus opciones de ayuda y con una documentación adecuada. La ayuda y documentación debe estar siempre disponible, fácil de buscar y debe aportar consejos directos que sean aplicables a las tareas del usuario.

## 2.6 Dispositivo de lógica programable FPGA

En el nivel más alto, los FPGA son chips de silicio reprogramables, al utilizar bloques de lógica preconstruidos y recursos de ruteo programables, se puede configurar

estos chips para implementar funcionalidades de hardware personalizadas sin tener que utilizar un tablero o un cautín. Sólo se deberá desarrollar tareas de computación digital en software y compilarlas en un archivo de configuración o una escritura de bits que contenga información de cómo deben conectarse los componentes. Además, los FPGA son completamente reconfigurables y al instante cambian su funcionalidad cuando se compila una diferente configuración de circuitos. Anteriormente sólo los ingenieros con un profundo entendimiento de diseño de hardware digital podían trabajar con la tecnología FPGA, sin embargo, el aumento de herramientas de diseño de alto nivel está cambiando las reglas de programación de éstos, con nuevas tecnologías que convierten los diagramas a bloques gráficos, o hasta el código ANSI C, a circuitos de hardware digital. La adopción de chips FPGA en las industrias ha sido impulsada por el hecho de que combinan lo mejor de los ASIC (*Aplication Specific Integrated Circuit, Circuito integrado de Aplicación Específica*) circuitos Integrados para aplicaciones específicas, o *ASIC* por sus siglas en inglés y de los sistemas basados en procesadores, gracias a esto, ofrecen velocidades temporizadas por hardware y fiabilidad, pero sin requerir altos volúmenes de recursos para compensar el gran gasto que genera un diseño personalizado de ASIC. El silicio reprogramable tiene la misma capacidad para ajustarse de un software que se ejecuta en un sistema basado en procesadores, pero no está limitado por el número de núcleos de proceso disponibles. A diferencia de los procesadores, los FPGA llevan a cabo diferentes operaciones de manera paralela, por lo que éstas no necesitan competir por los mismos recursos. Cada tarea de procesos independientes se asigna a una sección dedicada del chip, y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan otros procesos (National Instruments, 2011).

## 2.7 Lenguaje de descripción de hardware VHDL

Existen diversos lenguajes descriptivos que han sido desarrollados en los últimos años, pero el más aceptado y con mayor difusión es el VHDL. Las características principales que hacen de VHDL el lenguaje universal de descripción de circuitos son el ser un lenguaje estándar definido como tal por el IEEE (específicamente en el estándar 1164), y que los proveedores del paquete tienen que seguir el estándar, haciendo que los diseños sean portátiles a cualquier plataforma (Romero, 2007). VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación. Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada, menos abstracta. Por tanto, la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño se va hacia el más bajo nivel de la jerarquía (Pardo, 1997).

## 2.8 Librería GTKMM

GTKMM es software libre distribuido bajo la Licencia Pública General Reducida de GNU (LGPL). Es la interfaz C++ oficial para la popular biblioteca de interfaz gráfica GTK+. Destacan las callbacks de tipo seguro, y un exhaustivo conjunto de widgets que son fácilmente extensibles a través de herencia. Puede crear interfaces de usuario ya sea con código o con el diseñador de interfaces gráficas Glade, usando Gtk::Builder. Las características principales de este software son:

- Puede utilizar herencia para derivar widgets personalizados.
- Manejadores de señales de tipo seguro, en C++ estándar.
- Polimorfismo.
- Uso de la biblioteca estándar C++, incluyendo cadenas, contenedores e iteradores.

- Internacionalización completa con UTF8.
- Manejo de memoria C++ completo
    - Composición de objetos.
    - Desasignación automática de widgets asignados dinámicamente.
- Uso completo de los espacios de nombres de C++.
- Sin macros.
- Multi-plataforma: Linux (gcc), FreeBSD (gcc), NetBSD (gcc), Solaris (gcc, Forte), Win32 (gcc, MSVC++ .Net 2003, 2005, 2008), MacOS X (gcc), otros.
- Software Libre, sin costo para desarrollos tanto propietarios como de código abierto.
- Discutido, diseñado e implementado en público.

GTKMM se libera en todas las plataformas bajo la licencia pública reducida GNU (LGPL) de esta manera puede usarse en bibliotecas compartidas en todo tipo de proyectos, ya sean abiertos o propietarios. Otros proyectos GNU pueden también integrarse y unirse de manera estática (gtkmm.org, 2013).

# III. METODOLOGÍA.

En esta sección se muestran a detalle los pasos y tareas a realizar para alcanzar los objetivos, dichas tareas están organizadas desde lo más fundamental, como la comunicación entre dispositivos a través del hardware, seguido del desarrollo del firmware para el sistema embebido y el protocolo de comunicación a seguir y terminando con el software necesario para monitorear y controlar los mensajes dentro de la red.

La metodología de manera general para realizar el presente trabajo se muestra en la figura 6 donde se tienen dos módulos principales (maestro y esclavo) en la plataforma de hardware-software. Por parte del dispositivo maestro, se encuentran los bloques de recepción de respuesta y transmisión de petición, los cuales contienen las descripciones VHDL de la interfaz de comunicación RS-485, además de los bloques de recepción de datos y generación de paquete para envío, que representan el firmare del procesador embebido en el sistema para la comunicación RS-485, además de comunicarse con la computadora para dar a conocer al usuario el estado de las comunicaciones. El módulo de esclavo es similar en sus componentes de hardware donde se maneja la comunicación RS-485 y también en el firmware para el análisis del dato y la generación de respuesta, solo que este módulo, incorpora un bloque para la comunicación con los sensores de los cuales se obtendrán las mediciones requeridas por el usuario.

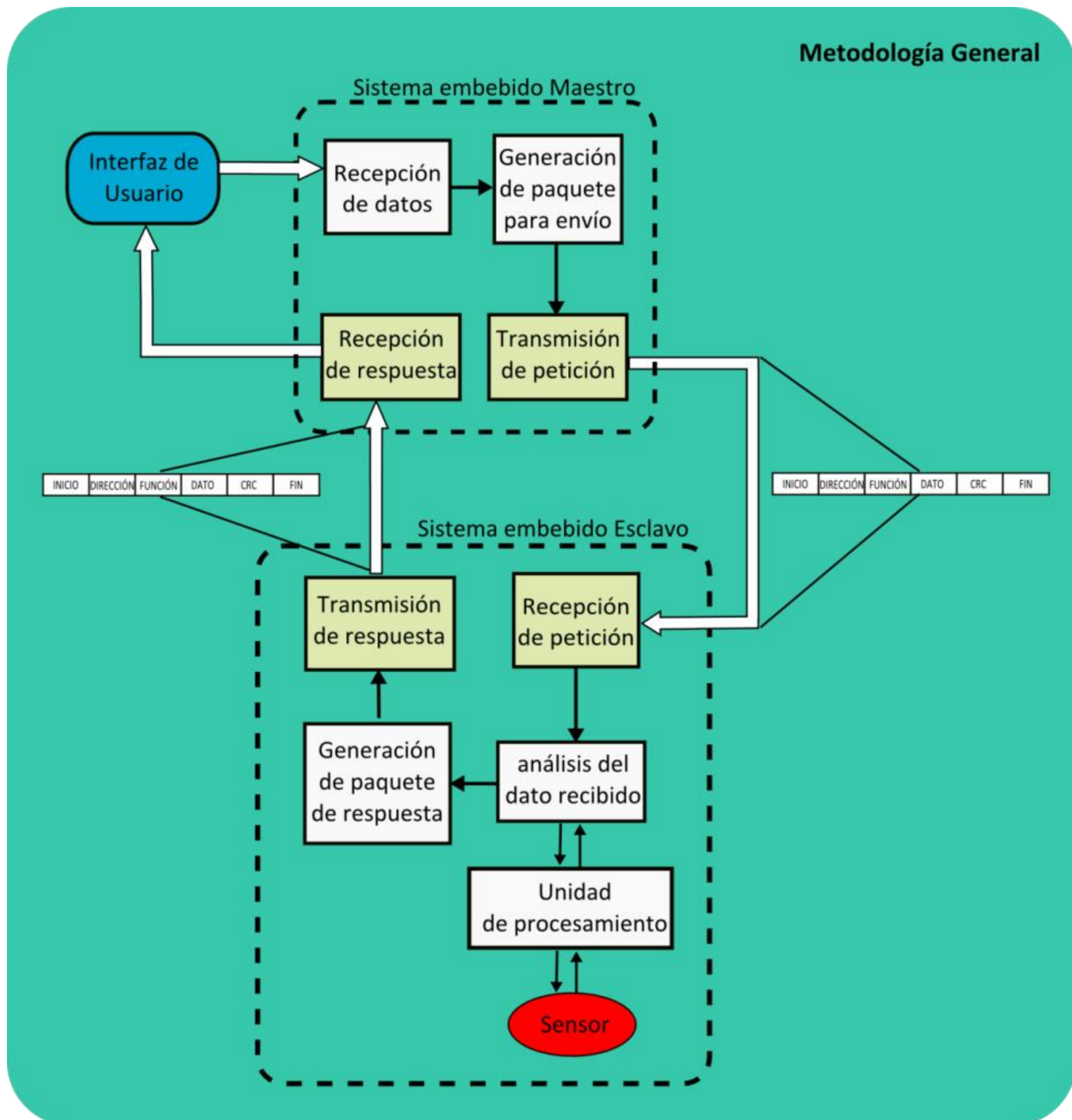



Figura 6. Metodología general del sistema.

Es importante señalar que la parte del sistema embebido esclavo se replicará asignando direcciones diferentes según sea necesario dependiendo del número de esclavos incorporados en el sistema.

## 3.1. Interfaz RS-485

Con la finalidad de establecer comunicación entre los dispositivos dentro de la red, es necesario desarrollar un manejador para el envío y recepción de datos. Para esto, se utilizó el circuito integrado ADM1485 y se crearon diferentes módulos de hardware, tanto para el envío como para la recepción de datos.

### 3.1.1. Transmisión.

Como se ha mencionado anteriormente la transmisión de un mensaje se debe de realizar en tramas consecutivas de 1 byte, donde cada una lleva la información para realizar alguna función en el dispositivo receptor. El diagrama de flujo del proceso de transmisión se muestra en la figura 7.

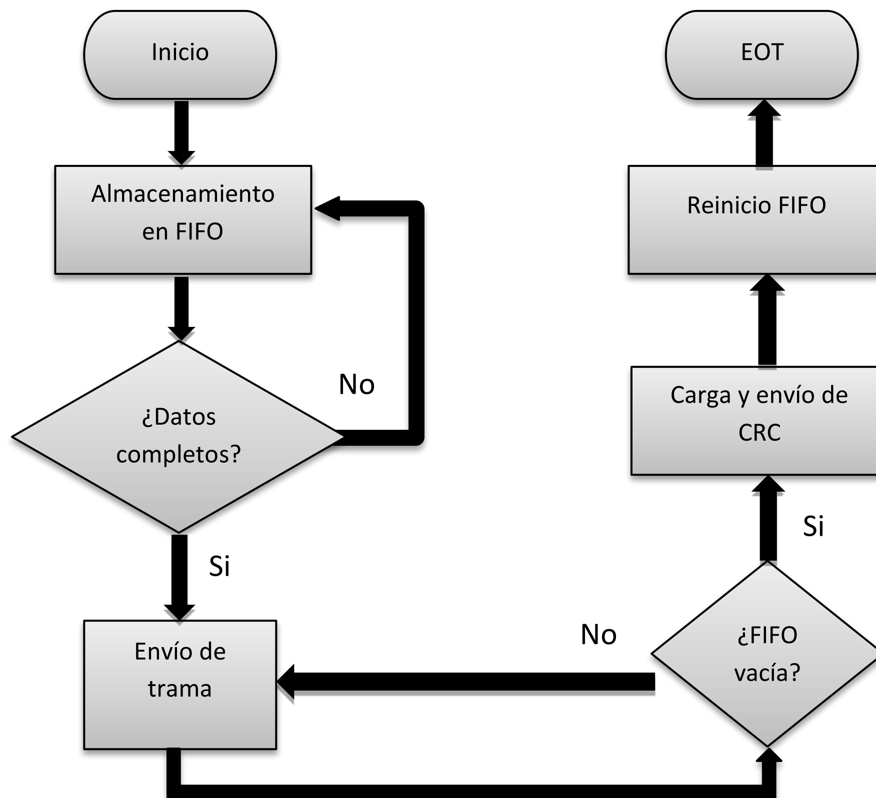



Figura 7. Proceso de transmisión.

El proceso de transmisión mostrado en el diagrama anterior comienza cuando se recibe la instrucción de inicio de transmisión, lo primero que se ejecuta es el almacenamiento de las tramas a transmitir en una estructura de tipo FIFO (*first in first out, Primero en llegar primero en salir*), cuando este proceso termina se comienza a enviar las tramas de datos. Una máquina de estados controla el proceso de envío, primero extrae la primera trama almacenada en la FIFO y la deposita en un bloque de hardware que se encarga de transmitir bit por bit la trama seleccionada, cuando este termina manda una señal de fin de trama a la máquina secuencial para que se continúe con el segundo dato almacenado, se continua de esta manera hasta que la FIFO sea vaciada y no queden tramas para extraer, cuando esto sucede el siguiente paso es cargar los primeros 8 bits del CRC en el bloque de transmisión y después de que sean enviados se cargan los últimos 8 bits, ya que se ha enviado la última trama del CRC se reinicia la FIFO y se activa una señal de fin de transmisión para indicar que envío de un mensaje se ha terminado. La figura 8 muestra el diagrama de bloques descriptivos de hardware para el proceso completo de transmisión.

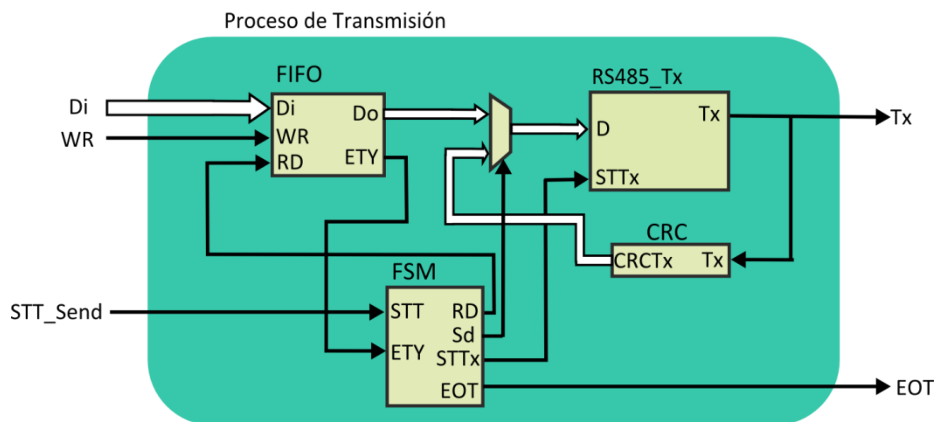



Figura 8. Bloque para la transmisión.

De la figura anterior, la señal (WR) se utiliza para almacenar el dato (Di) dentro de la FIFO, (STT_Send) indica cuando debe de comenzar con el proceso de envió mostrado a detalle en la figura 9, la señal (RD) se usa para extraer datos de la FIFO por (Do) y (STTx)

para habilitar el envío de una trama de bit en bit a través de (Tx). (ETY) le dice a la máquina secuencial que ya no hay más datos para transmitir, en ese momento con ayuda de la señal (Sd) se carga el CRC para su transmisión y una vez que este se ha terminado se activa la señal (EOT) que indica el final de la transmisión.

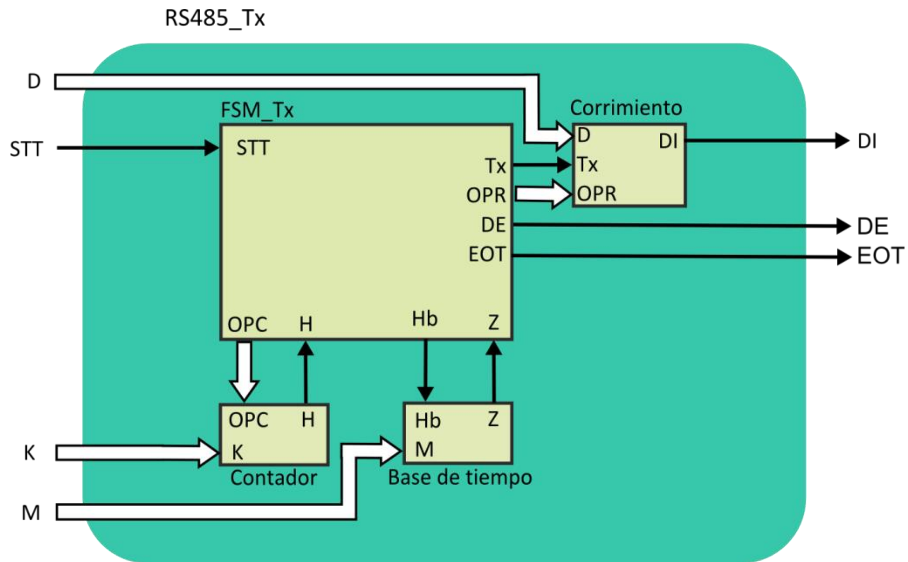



Figura 9. Módulo de transmisión RS485_Tx

En la figura 9 se puede observar cuatro bloques diferentes, donde, FSM_Tx es una máquina secuencial que controla el proceso de transmisión de datos y que podemos ver a detalle en la figura 10, esta máquina utiliza los bloques restantes de la siguiente manera. Al recibir la señal STT (inicio de transmisión) se activa una base de tiempo para regular la velocidad de transmisión en el sistema, el dato que determina la velocidad es ingresado por medio de la variable (M), al mismo tiempo se habilita un contador que se encarga de monitorear la cantidad de bits que se deben de enviar en el mensaje, la variable (K) contiene el número de bits a transmitir en una trama de datos, cuando se ha terminado el envío se genera una señal de fin de cuenta que es recibida por la máquina secuencial. El bloque de corrimiento recibe el dato que se desea enviar contenido en la variable (D), la señal (Tx) indica en qué momento se debe de enviar un bit a través de la salida (DI), (OPR) indica las operaciones que debe realizar el registro, las cuales pueden ser: cargar el dato en el vector de salida, limpiar el vector o colocar un bit en la posición de envío, los bits son

transmitidos empezando por la posición menos significativa del vector de salida. Ya que se ha terminado de enviar el dato, se genera la señal (EOT) que marca el final de la transmisión, la salida (DE) tiene como función habilitar el circuito LM1485 para transmisión.

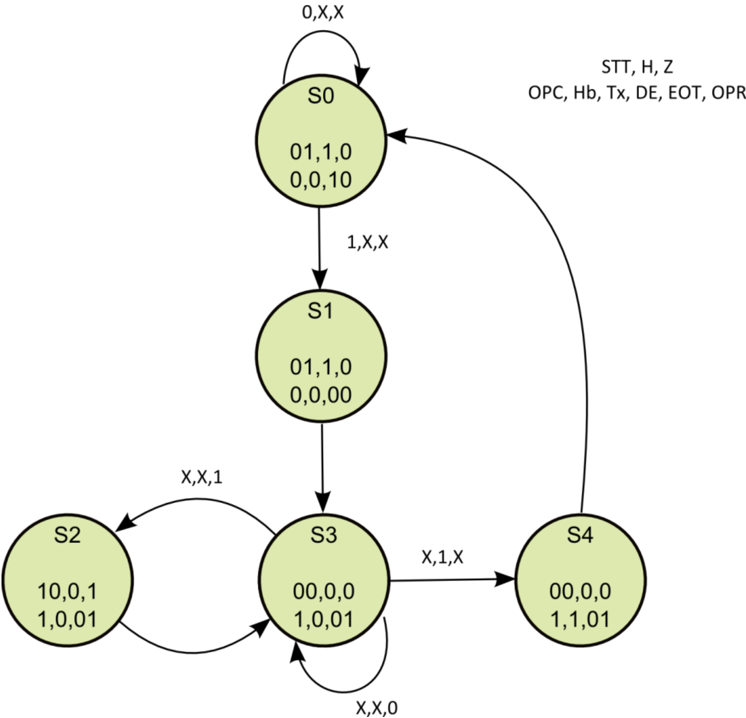



Figura 10. Máquina secuencial para la transmisión de datos.

La máquina secuencial de la figura 10 tiene la función de sincronizar los componentes del módulo RS485_Tx. El estado S0 es de espera y todas las salidas permanecen deshabilitadas, S1 se utiliza para cargar el número de bits a enviar en el módulo contador, S2 sirve para realizar un decremento en la cuenta de bits enviados, S3 mantiene la cuenta y espera a que la señal de la base de tiempo para la velocidad se active, para pasar a S2 y enviar un nuevo bit, finalmente en S4 se envía la señal (EOT) una vez que ha terminado la transmisión del dato.

### 3.1.2. Recepción

El proceso de recepción se muestra en la figura 11 donde se reciben las tramas consecutivas generadas por el transmisor y se almacenan para su posterior análisis

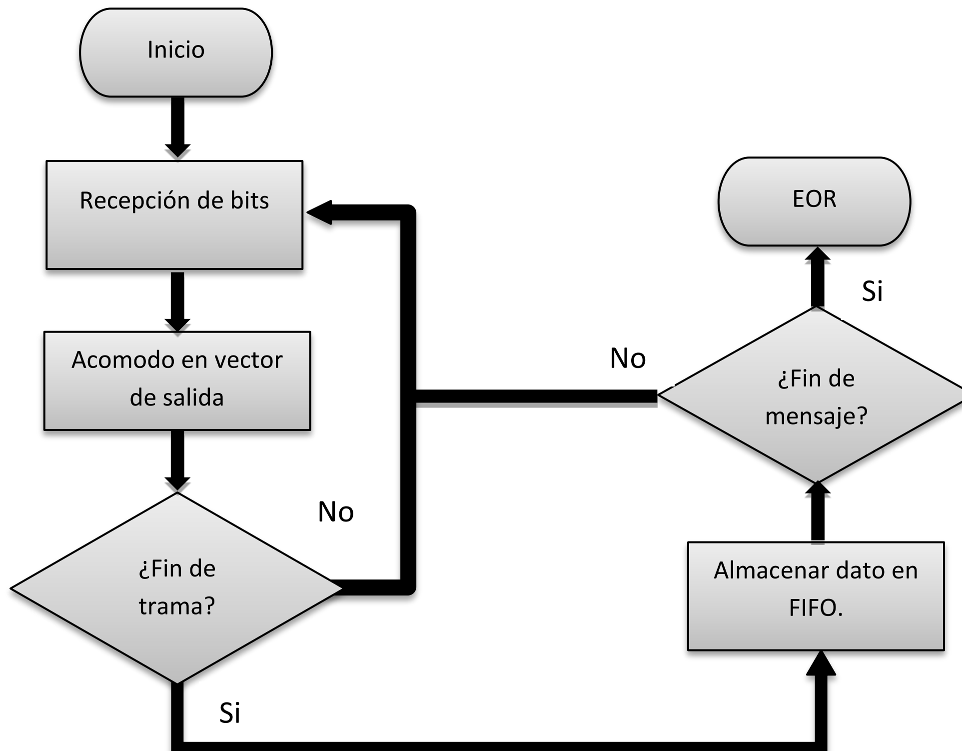



Figura 11. Proceso de Recepción.

La recepción comienza cuando se detecta el bit de inicio de una trama de tatos, se recibe cada bit de la trama y se colocan en un vector ordenándolos del menos al más significativo, una vez que se termina la recepción de una trama, el vector con 8 bits se almacena en una estructura de tipo FIFO y se espera a que llegue el bit de inicio de la siguiente trama o la señal de fin de mensaje, si llega el bit de inicio se continua con el proceso anterior hasta que se termine la recepción del mensaje, cuando esto ocurre se emite la señal de fin de recepción. La figura 12 muestra el diagrama de bloques descriptivos de hardware para el proceso completo de transmisión.

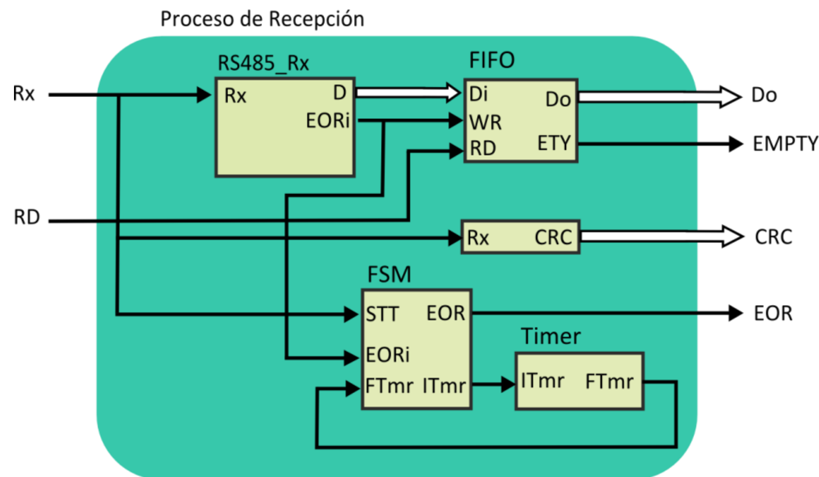


Figura 12. Bloque para la recepción.

La señal (Rx) es el medio por el cual llegan los bits de un mensaje al módulo de recepción mostrado en la figura 13, (EORi) indica el final de una trama y (D) es la salida de los datos que se almacenan en la FIFO por medio de (Di), (WR) se utiliza para indicar que un nuevo dato será almacenado. La señal (STT) le dice a la máquina en que momento llega un nuevo mensaje, (EORi) marca el final de recepción de una trama de 1 byte, (ITmr) se utiliza cada que se termina de recibir una trama para activar un contador que tiene como tarea medir el tiempo de espera entre cada trama recibida, si este tiempo excede a la suma de 3 veces la velocidad a la que se transmite 1 bit, la señal (FTmr) se activa y el sistema interpreta que el mensaje ha sido recibido por completo, en ese momento (EOR) marca el final de la recepción. La señal de salida (CRC) contiene el código de redundancia cíclica calculado en la recepción, esta señal solo se actualiza cuando un mensaje se ha terminado de recibir por completo.

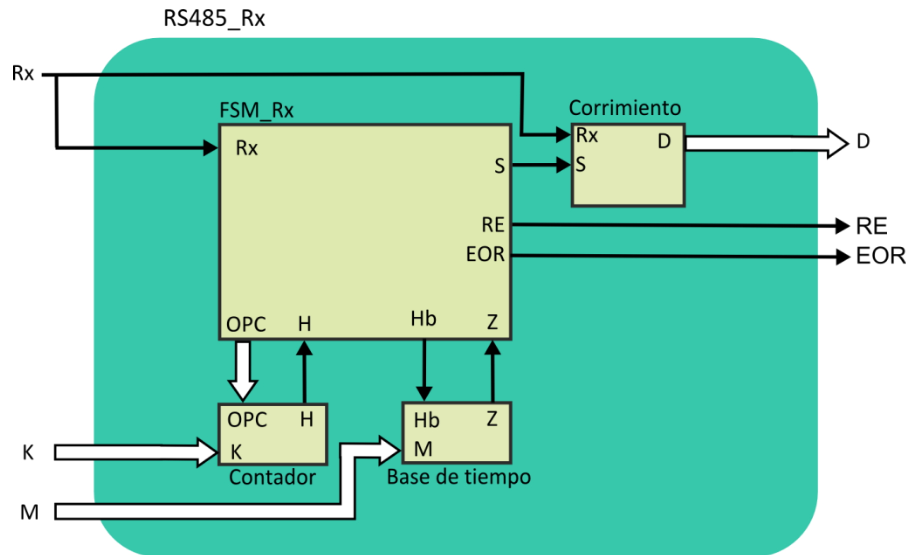



Figura 13. Módulo de recepción RS485_Rx

El módulo de recepción es muy similar al de transmisión, utiliza la máquina secuencial de la figura 14 que coordina el resto de módulos incluidos en la recepción, donde en el momento que se recibe un pulso positivo de la señal Rx, se activa un contador que monitorea la cantidad de datos que se están recibiendo, la variable (K) contiene el número de datos que se espera recibir, (OPC) controla la función del contador ya sea para mantener la cuenta, incrementarla o cargar el número (K), (H) es un pulso que indica el final de la cuenta. Se habilita también una base de tiempo que regula la velocidad de la recepción, (M) es el dato que indica esta velocidad, (Hb) es una señal de habilitación para iniciar la base de tiempo, (Z) es la señal con la cual se marca la velocidad de recepción. El módulo tiene como entrada los bits del dato recibido y los acomoda en un vector de salida (D) cada vez que la señal (S) lo indica. Las señales de salida RE y EOR tienen como función habilitar el circuito ADM1485 para recepción y generar la señal de fin de recepción respectivamente.

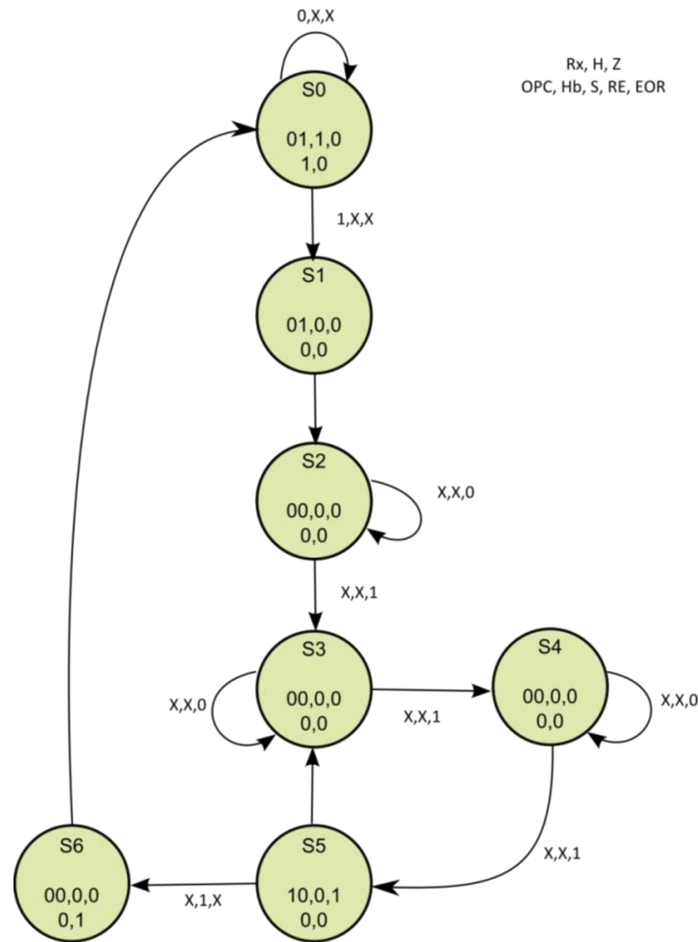



Figura 14. Máquina secuencial para la recepción de datos.

La máquina secuencial destinada a la recepción coordina los componentes del módulo RS485_Rx, utiliza 7 estados, el estado S0 es de espera y todas las salidas permanecen deshabilitadas, S1 se utiliza para cargar la cuenta en el módulo contador, S2 y S3 son estados de espera para que termine de ingresar el bit de inicio, una vez que se comienza con la recepción de los datos, S3 y S4 son estados de espera para sincronizar la recepción de los bits, S5 es el estado en el que se incrementa la cuenta de bits recibidos y se le habilita la salida (S) para que se almacene un bit en el vector de salida, final mente S6 envía la señal (EOR) que indica el final de recepción.

## 3.2. Firmware RS-485 del procesador

Este periférico es necesario para comunicar la interfaz de comunicación RS-485 con el procesador embebido dentro del FPGA, la figura 15 muestra la manera en que el periférico se incorpora al sistema embebido. Este periférico, consiste básicamente en la creación de una librería de C, donde se especifican los puertos de acceso al bus de datos en los cuales estará conectado el manejador, se definen también las señales que crearan interrupciones refiriéndose a algún evento importante en la comunicación, como el final de transmisión y recepción de datos, de igual manera, se programan las macros que podrán ser usadas para controlar el envío y recepción de datos dentro de la red, el cuadro 3 muestra una lista de las interrupciones y macros generadas.

Cuadro 3. Interrupciones y macros necesarias para la comunicación.

| Macro. | Función. |
|---|---|
| WR_FIFO(Dwr) | Almacena en la FIFO el dato contenido en la variable Dwr. |
| Inicio_TX | Inicia el proceso de transmisión de un mensaje |
| RD_pop | Habilita la FIFO para leer un dato. |
| RD_FIFO() | Lee el dato almacenado en la parte alta de la FIFO. |
| RD_FIN() | Verifica si los datos almacenados en la FIFO han sido leídos por completo. |
| leer_crc() | Lee el dato de crc generado durante la recepción. |
| **Interrupción** | **Función.** |

| | |
|---|---|
| EOR | Se genera cuando un dato se termina de recibir. |
| EOT | Se genera cuando un dato se termina de transmitir. |

Es necesario mencionar que las interrupciones son generadas en hardware durante la comunicación de los dispositivos y el software se encarga de capturarlas.

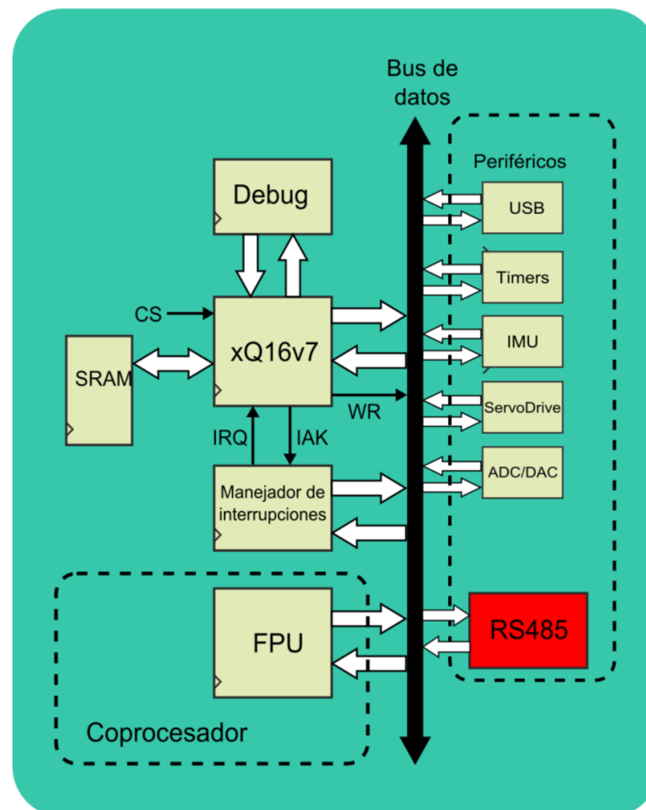



Figura 15. RS485 en sistema embebido

Este módulo tiene la capacidad de funcionar en 2 modalidades diferentes según sea necesario, puede trabajar como dispositivo maestro o como esclavo, ambas modalidades se dividen en dos partes, una maneja los datos que se reciben del hardware con la interfaz RS-485 y otra controla los datos que deben de ser enviados también por hardware.

La parte de transmisión en la modalidad de maestro recibe los datos que serán enviados en un mensaje, llena la FIFO con esos datos en el orden que especifica el protocolo Modbus e inicia la macro para empezar la transmisión. La parte de recepción espera la señal que indica que un mensaje ha llegado, verifica el CRC, si este indica que el mensaje llego correctamente, lee los datos almacenados en la FIFO y los transmite a la computadora.

La modalidad de esclavo siempre está a la espera de recibir un mensaje, cuando lo recibe, verifica el CRC y si el mensaje es correcto entonces lee los datos de la FIFO y los almacena en un vector, después analiza el campo del mensaje correspondiente a la dirección, si la dirección del mensaje es igual a la del dispositivo entonces actualiza las mediciones de los sensores en ese momento, después entra en una función donde se lee el campo de la función, si este campo contiene el valor que indica que se requiere la identificación del dispositivo, entonces se contesta con un mensaje para informar el tipo de sensores que están incorporados en la tarjeta y los módulos disponibles dentro de estos, el cuadro 4 muestra los dispositivos que pueden estar en una tarjeta

Cuadro 4. Sensores y módulos posibles.

| Sensor. | Módulos posibles |
|---|---|
| Acelerómetro | eje x, eje y, eje z |
| Giroscopio | eje x, eje y, eje z |
| Magnetómetro | eje x, eje y, eje z |
| Termómetro | tmp |

De no ser así, el campo de función especificara cual es el sensor del que se desea obtener información y el campo correspondiente al dato del mensaje indicará que módulo se desea leer. En base a todo esto se arma el mensaje de respuesta y se envía al dispositivo maestro.

## 3.3 Protocolo Modbus

Para este trabajo se implementó el protocolo Modbus en modo RTU utilizando el encapsulado de datos que sugiere MODICON el cual se puede ver en la figura 16.

| INICIO | DIRECCIÓN | FUNCIÓN | DATOS | CRC | FIN |
|---|---|---|---|---|---|
| T1-T2-T3-T4 | 8 bits | 8 bits | *n* x 8 bits | 16 bits | T1-T2-T3-T4 |

Figura 16. Entramado de datos Modbus.

Donde la primer bloque de bits, contiene la dirección del dispositivo esclavo con el cual se requiere tener comunicación, en el bloque de función se indica el sensor del cual se desea obtener la medición, el tercer bloque contiene la especificación de los datos que se desean obtener, como puede ser, un eje específico del sensor en un plano tridimensional (x, y, z) y el último bloque está reservado para enviar el código CRC.

El armado de estos paquetes se realiza con el firmware del RS-485 así como la interpretación de los mismos. El listado 1 muestra la función que se encarga de recibir las instrucciones de la Pc y acomodarlos para formar el mensaje, donde, se reciben los datos para el armado en las variables d0 a d2 a través de la función (on_USBRX), después ingresan como parámetros de la función (transmision) y ahí son acomodados uno a uno en el módulo de hardware para su envío, la macro (WR_FIFO) escribe el dato en la FIFO e (inicio_TX) comienza con la transmisión del mensaje, la función (usb_rxfree) libera la comunicación USB para que puedan seguir llegando datos.

Listado 1. Función para armado de mensaje Modbus.

```
void on_USBRX (void)
{
    read_usb(d0,d1,d2,d3);
    transmision(d0,d1,d2);
}

void transmision (int dd,int df, int ddt)
{
    WR_FIFO(dd);
    WR_FIFO(df);
    WR_FIFO(ddt);
    WR_FIFO(0x00);
    inicio_TX;
    usb_rxfree();
    led=1;
}
```

## 3.4 Interfaz gráfica de usuario

Con el fin de que el usuario pueda tener comunicación con el dispositivo maestro y con los esclavos, es necesario crear una GUI (Graphical User Interface, Interfaz Gráfica de Usuario). Para este propósito se utilizó el lenguaje de programación C y la librería GTKmm sobre la plataforma de diseño CodeBlocks, tratando de seguir las heurísticas de diseño para este tipo de desarrollos.

Las principales funciones del software desarrollado son: dar a conocer al usuario la cantidad de dispositivos conectados en la red y los elementos en cada uno (sensores), permitir realizar consultas individuales de cada elemento encontrado, también permite realizar un monitoreo constante de los sensores que el usuario considere pertinentes además de guardar los datos adquiridos en un archivo de texto para los fines que el usuario desee, la figura 17 muestra la arquitectura general del software para el monitoreo

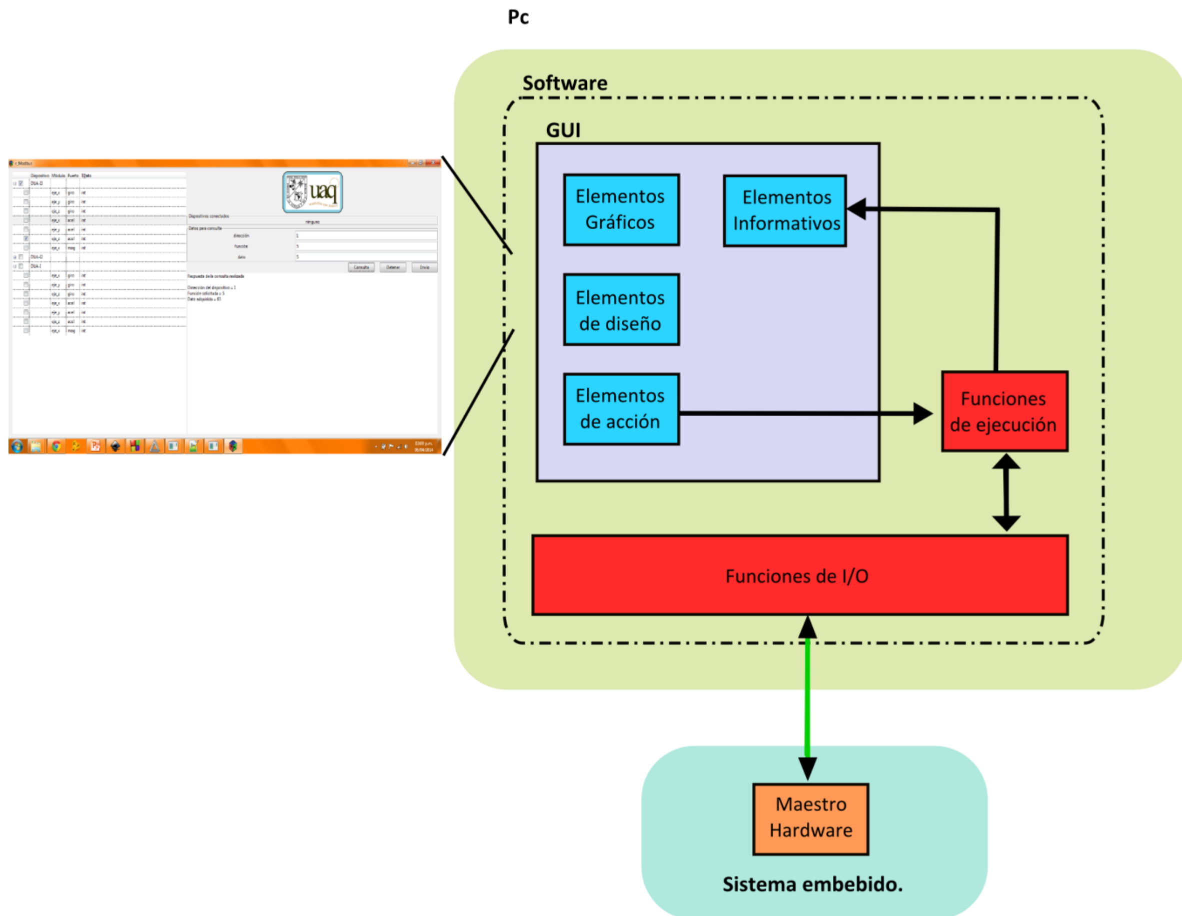



Figura 17. Diagrama general del software.

El software se compone de varios bloques, en primer lugar está la GUI que está formada por cuatro elementos principales. Los elementos gráficos que son aquellos que le dan la estética a la interfaz gráfica, como colores de fondo, estilo de las fuentes e imágenes o logotipos. Los elementos de diseño nos permiten establecer un orden de colocación de botones, cuadros de texto, etiquetas, imágenes y todo lo que sea necesario mostrar en pantalla, de esta manera se puede hacer más sencillo el manejo de la GUI para el usuario. Los elementos de acción son aquellos que interfieren directamente con el funcionamiento de la interfaz como botones, combo box, entradas de texto, cuadros de selección, diagramas de árbol, etc. Dichos elementos activan las funciones de ejecución que son básicamente el núcleo del software, toman los datos contenidos en los elementos de acción, y los utilizan como instrucciones para ejecutar rutinas, que controlan al sistema

embebido maestro durante la comunicación con los esclavos, tal y como lo muestra el diagrama de la figura 18.

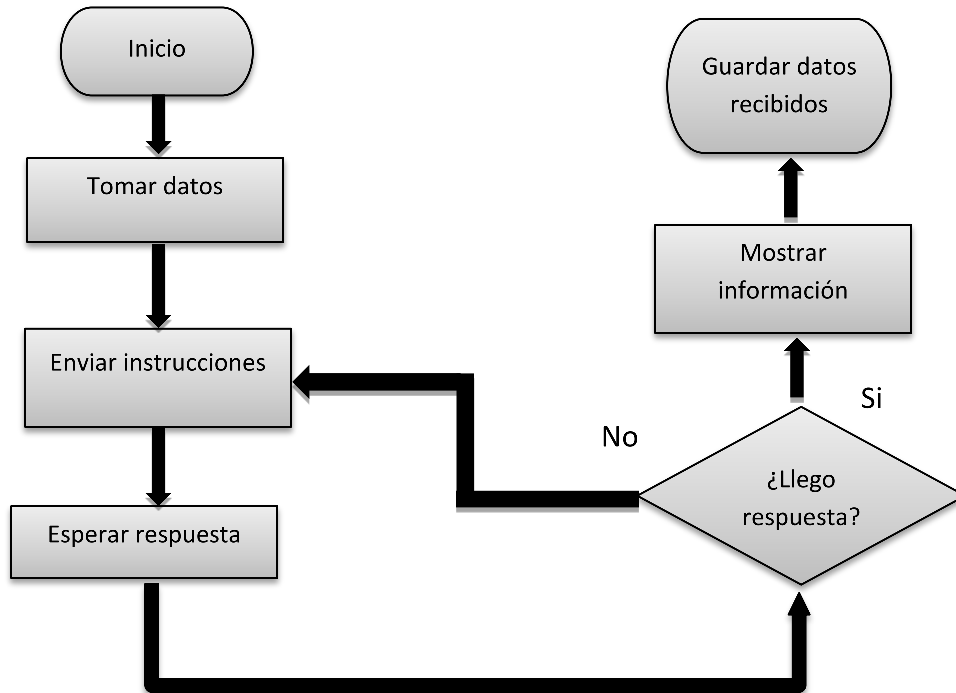



Figura 18. Rutina de control.

Se envían las instrucciones y se espera la respuesta, en cuanto llega se realizan varias acciones, como ajustar el dato recibido a las unidades correspondientes, para un dato del acelerómetro se debe de mostrar la respuesta en gravedades, los datos del giroscopio en grados por segundo, los del magnetómetro en Gauss y los del termómetro en grados centígrados. Después de esto, se muestran los datos en pantalla con ayuda de los elementos informativos, como etiquetas o cuadros de texto, para que el usuario advierta que el sistema está funcionando, y también, en caso de ser un monitoreo constante, se guarda cada dato recibido en un archivo de texto con el formato mostrado en el cuadro 5.

Cuadro 5. Formato de datos guardados.

| # Columna. | Descripción |
|---|---|
| 1 | Indica de que esclavo proviene el mensaje |
| 2 | Indica la función solicitada al esclavo |
| 3 | Los 8 bits más significativos del dato de respuesta |
| 4 | Los 8 bits menos significativos del dato de respuesta |
| 5 | Dato completo del mensaje en la unidades correspondientes |
| 6 | El número de la muestra |
| 7 | El segundo en el que se tomó la muestra después de haber iniciado el monitoreo |

El control del maestro se realiza con la ayuda de las funciones de I/O, las cuales utilizan la interfaz USB para enviar y recibir datos durante el proceso de envío y recepción de mensajes

# IV. PRUEBAS Y RESULTADOS

En esta sección se presentan los resultados obtenidos con diferentes pruebas, enfocadas a determinar el buen funcionamiento de los elementos más importantes para la comunicación dentro de la red, como: el hardware de envío y recepción, el enlace hardware-software del sistema embebido, el protocolo Modbus y la conexión con la Pc.

## 4.1. Pruebas y resultados de diseño

Las pruebas aquí mostradas están enfocadas a determinar el correcto funcionamiento y alcance del sistema desarrollado, se evalúan todas las partes de este por separado y en conjunto, como: sistema embebido (hardware y firmware), conexión con una PC y el software para el control de los mensajes.

### 4.1.1. Hardware

Antes de incorporar la descripción de hardware al FPGA, es importante simularla para comprobar que funcione como se espera, o advertir si existe algún error en el funcionamiento, la simulación, básicamente consiste en definir con valores controlados las señales de entrada para la descripción, y observar que la salida del sistema sea congruente con los valores ingresados. La figura 19 muestra la simulación de un mensaje transmitido por el dispositivo maestro y recibido por el esclavo. Las señales dentro del cuadro verde corresponden a los datos que deben de ser enviados en el mensaje, donde (STT) equivale a la instrucción de guardar el dato de la señal (DATA) dentro de la FIFO y (STT_Send) indica que el proceso de transmisión puede comenzar. La señal (DI) es la entrada del receptor y dentro de la llave en color negro se encuentra la recepción del

mensaje, (DR) muestra el acomodo de los bits recibidos para formar datos de 8 bits, en este caso se recibe un mensaje con 5 datos, que son (01,FF,00,61,F0) de este grupo de datos los últimos 2 corresponden al CRC calculado al momento de enviar el mensaje. Después de este proceso se emite la señal de fin de recepción. Se puede ver que los datos para transmitir dentro del recuadro verde son los mismos a los recibidos en la llave de color café, por lo tanto se interpreta que los módulos de transmisión y recepción funcionan correctamente

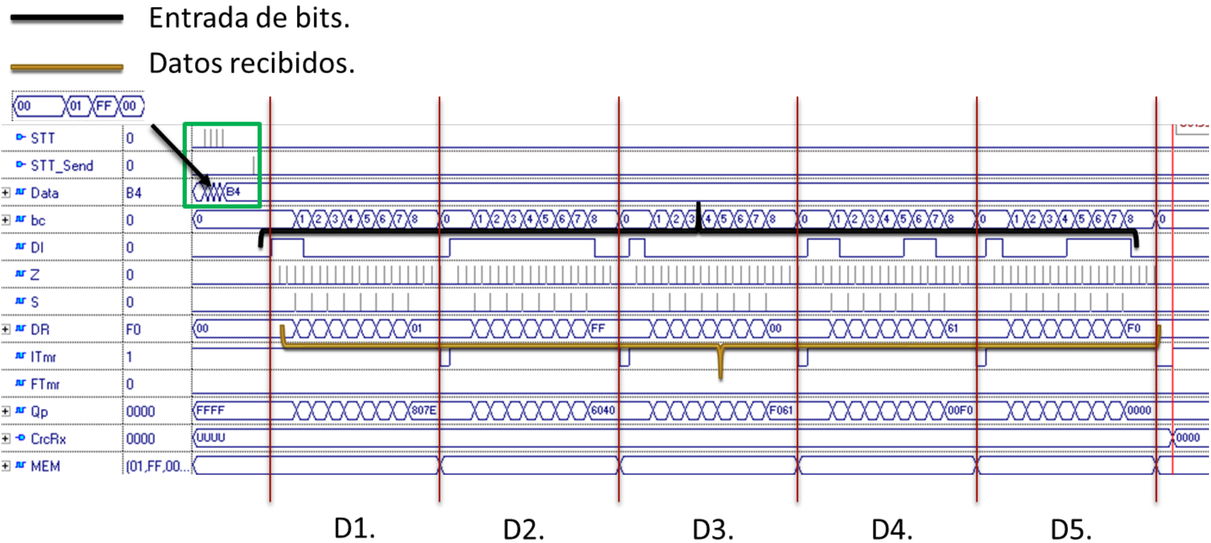



Figura 19. Simulación de envío y recepción de mensaje.

### 4.1.2. Enlace Hardware-Software (del sistema embebido)

Para estar seguros de que el firmware trabaja correctamente con el hardware en el sistema embebido, es necesario revisar los datos que se han recibido en un mensaje, utilizando el compilador de software instalado en la PC. Esta prueba es sencilla y basta con enviar un mensaje generado por el firmware del maestro hacia el esclavo, el listado 2 muestra la función para enviar un dato de forma manual.

Listado 2. Función para armado de mensaje Modbus.

```
int envio()
{
    WR_FIFO(0x01);
    WR_FIFO(0x01);
    WR_FIFO(0x10);
    WR_FIFO(0x00);// dato de ajuste
    inicio_TX;
    led=1;
}
```

En el listado se puede observar que se envió el mensaje (1, 1, 10) en hexadecimal, este mensaje pasa por el hardware y sale a la capa física de transporte (cable telefónico), en la figura 20 se puede visualizar el mensaje a través del cable, esta señal se adquirió con un analizador digital de señales, donde la línea café muestra el mensaje salido del maestro y la roja es el recibido por el esclavo, es importante señalar que el mensaje mostrado se envía bajo las especificaciones del protocolo Modbus.

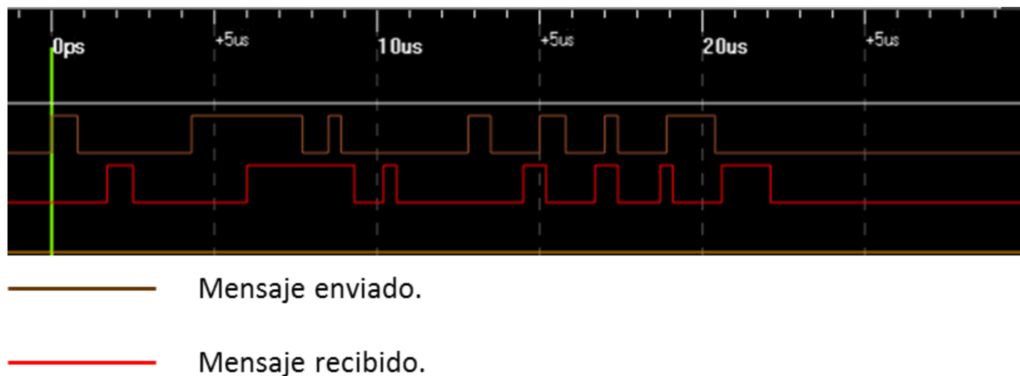



Figura 20. Mensaje sobre la capa física.

Cuando el mensaje llega al esclavo, es recibido por el hardware y pasado al firmware para que después sea tomado por el compilador, tal y como se puede ver en la figura 21 donde dir_0 es el primer dato del mensaje, fnc_0 el segundo y d_0 el tercero, también mediante esta prueba se puede observar que la variable crc_0 tiene un valor de '0' con lo cual el sistema comprueba que el dato recibido es correcto y ahora puede tomar la acción solicitada por los datos del mensaje

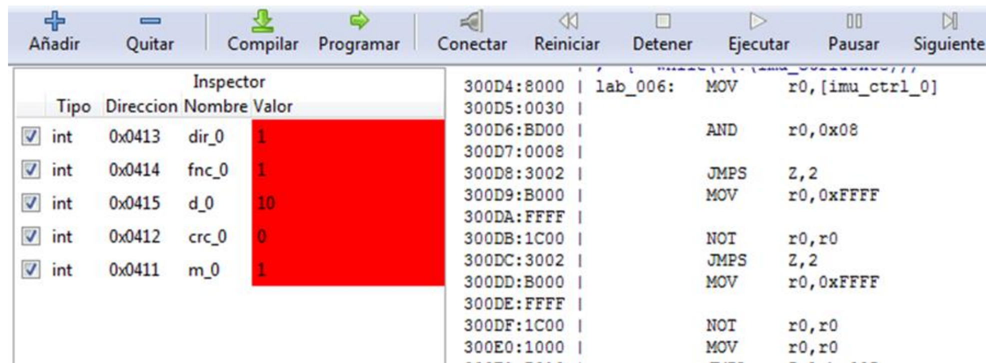


Figura 21. Compilador con dato recibido.

### 4.1.3. Conexión con la PC

La conexión con una computadora es importante para lograr la comunicación entre el dispositivo maestro y la interfaz gráfica, y de esta manera tener el control sobre las solicitudes hechas a los esclavos. Para verificar la conexión, se realiza una prueba con todo el sistema conectado y en funcionamiento, se corre el software de la GUI y se realiza la rutina de identificación, al recibir la respuesta de la figura 22a se comprueba que la comunicación funciona, ya que la rutina mencionada anteriormente envía un mensaje a cada dispositivo de la red, para que estos contesten con su nombre y sensores disponibles en ese momento. Dichos mensajes salen de la GUI al maestro y de este a los esclavos para después recibir la respuesta en el maestro y enseguida a la GUI. Para comprobar aún más la conexión se envía una consulta en específico como lo muestra la figura 22b, se solicita el dato del eje y del acelerómetro en el esclavo 2, de igual manera que la identificación el mensaje sigue la línea: GUI -> maestro -> esclavo -> maestro -> GUI. Cuando se recibe la respuesta con el dato adquirido (-0.0234) se comprueba la comunicación del maestro con la Pc además del funcionamiento de todo el sistema.

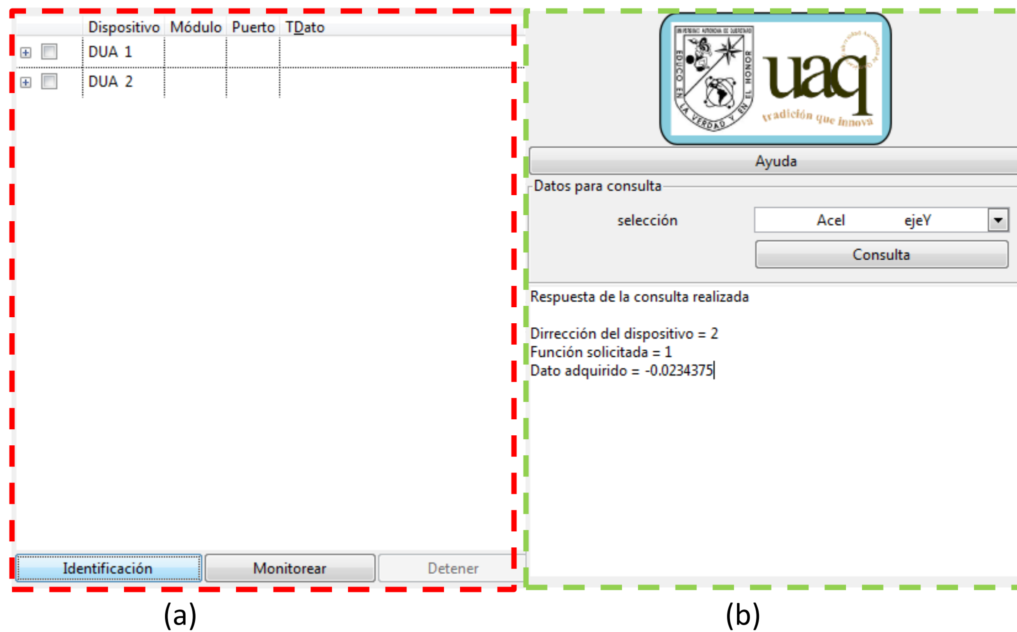
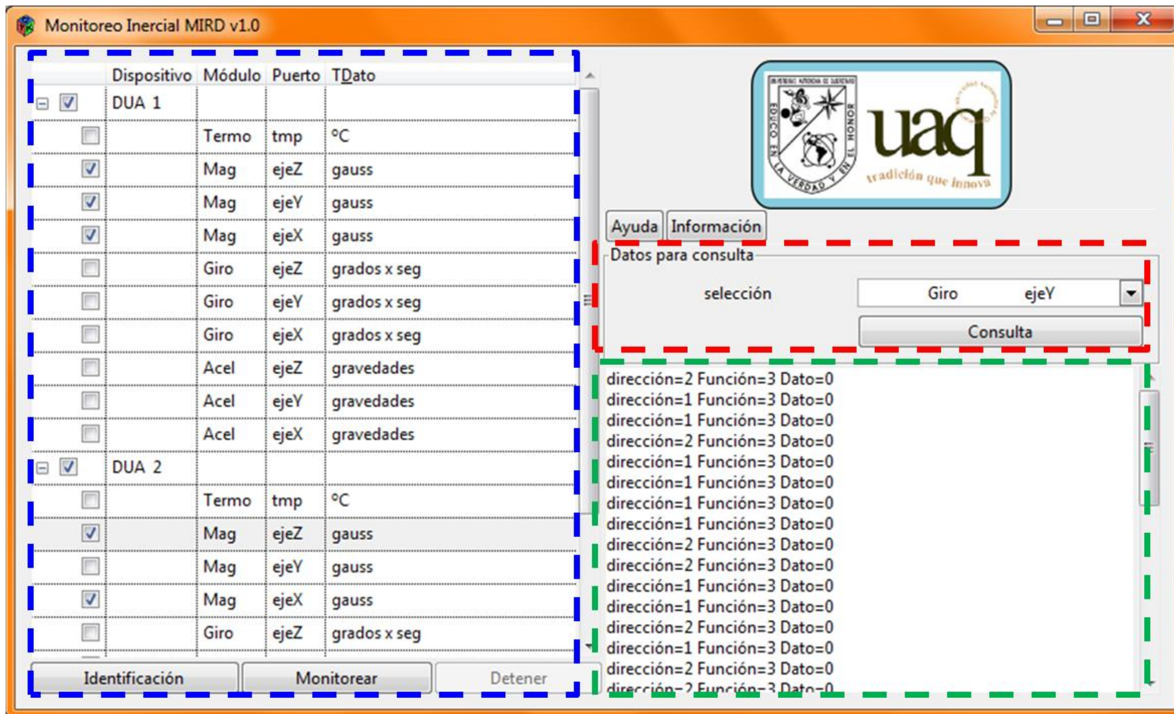



Figura 22. Prueba de conexión con Pc mediante la interfaz gráfica.

## 4.2. Interfaz Gráfica de Usuario

La GUI desarrollada es bastante sencilla y fácil de entender al momento de operarla, estéticamente se divide en tres elementos, como lo muestra la figura 23, uno del lado izquierdo de la interfaz que tiene la función de apoyar en el monitoreo constante de la red, otro en la parte superior derecha que se utiliza para consultas individuales a un dispositivo y un tercero en la parte inferior derecha que sirve como consola para mantener informado al usuario de las acciones realizadas por la GUI.




Monitoreo

Consulta

Consola

Figura 23. Interfaz Gráfica de Usuario.

En cuanto a la usabilidad de la GUI se cumplieron la mayoría de las heurísticas de diseño recomendadas, la número uno se cumple con la incorporación de la consola que mantiene al usuario informado de las operaciones en proceso, la numero dos se cumple ya que no existen botones con nombres extraños o símbolos ajenos a lo que el usuario pueda conocer. Debido a la simplicidad de la GUI, la heurística número 3 y 5 quedan cubiertas, ya que es fácil que el usuario corrija sus acciones además de prevenir errores, si seleccionó un elemento que no quería monitorear solo tiene que deseleccionarlo, si por el contrario olvido seleccionar uno, solo debe detener el proceso y seleccionar el elemento, además la parte de consultas individuales está limitada a los elementos dentro de la red en ese momento, no es posible ingresar direcciones manualmente evitando que se intente comunicar con un esclavo inexistente. Las palabras usadas para identificar un sensor o un esclavo conectado son las mismas para realizar un monitoreo o para hacer consultas

individuales, sin perder la individualidad de cada dispositivo, con esto se cumple la número 4. La heurística 6 indica que no se debe de pedir al usuario recordar datos de alguna ventana anterior, en este caso toda la información está en una sola ventana haciendo innecesario recordar algún dato. En la GUI propuesta no se tienen menús para correr algún proceso o seleccionar una herramienta no se consideró necesario agregar atajos en el teclado. La GUI es simple en todos sus aspectos, solo utiliza lo esencial para manejar el sistema, no hay botones o imágenes de más, ni información que no sea útil para el usuario, de esta manera se acata lo enunciado en la heurística 8. A pesar de lo sencillo de la interfaz gráfica se cuenta con opciones de ayuda y documentación en caso de que el usuario tenga alguna duda del funcionamiento o el diseño.

## 4.3. Pruebas y resultados de comunicación

Con la finalidad de comprobar que la comunicación entre los dispositivos se realice de manera fluida sin desconexiones o interrupciones de algún tipo, se realizaron pruebas de comunicación, donde se enviaban diferentes mensajes a todos los dispositivos dentro de la red, se esperaba la respuesta y se almacenaban los datos recibidos para su análisis.

Las pruebas se elaboraron utilizando la topología de red mostrada en la figura 24, con 3 esclavos conectados a una distancia de aproximadamente de 305 metros, usando cable telefónico estándar de 4 vías, plugs telefónicos RJ-11 y RJ-9, además de coples telefónicos y adaptadores de 3 jacks a un plug.

Las pruebas consisten en pedir la información de cada uno de los sensores de los esclavos, esto es: ejeY, ejeZ y ejeX del acelerómetro, giroscopio y magnetómetro, también el dato de temperatura del termómetro, dando un total de 10 solicitudes a cada esclavo por muestra, tomando en cuenta que son 3 esclavos y 100 muestras por prueba, tenemos un total de 3000 mensajes efectivos sin fallas. Es importante mencionar que para fines estadísticos cada prueba se realizó 40 veces y se probaron 5 velocidades de transmisión

diferentes (714 KHz, 1.4 MHz, 2.5 MHz, 3 MHz, 5 MHz). Para cada velocidad se aplicó el mismo procedimiento mencionado.

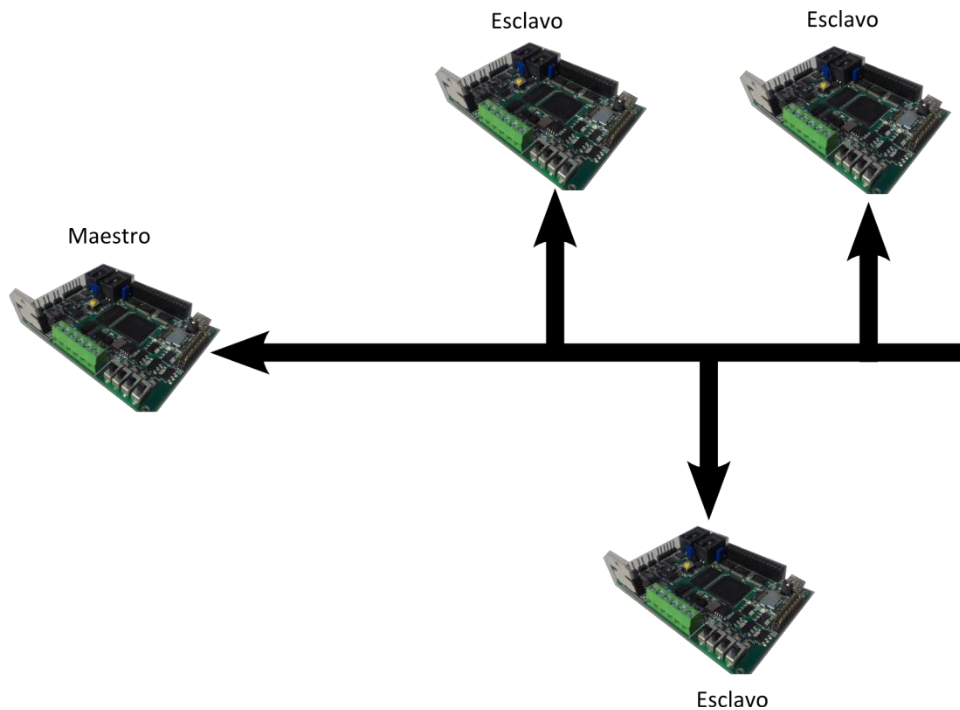



Figura 24.Topologia de red.

De cada una de las pruebas realizadas se obtuvo la cantidad de errores generados, entendiendo por error, que el CRC del mensaje no era el correcto y se tuvo que volver a solicitar el dato o que el software de la conexión con la Pc sufrió un retraso o perdió el dato solicitado, teniendo que hacer la solicitud nuevamente. Aunque existen los errores señalados es importante aclarar que la comunicación entre la red nunca se pierde ni resulta afectada a lo largo de todo el monitoreo. El cuadro 6 muestra los resultados con los errores generados para cada velocidad tanto en hardware como en software.

Cuadro 6. Resultados de comunicación.

| Velocidad. | Promedio err_S | Promedio err_H | % err_S | % errs_H | Tiempo promedio (s). | Ancho de banda B/s |
|---|---|---|---|---|---|---|
| 714 KHz | 246.65 | 0.075 | 8.2 | 0.0025 | 34.35 | 19,998 |
| 1.4 MHz | 286.52 | 0.025 | 9.55 | 0.0008 | 36.57 | 19,998 |
| 2.5 MHz | 271.75 | 0.025 | 9.05 | 0.0008 | 32.42 | 19,998 |
| 3 MHz | 261.27 | 0.05 | 8.7 | 0.0016 | 31.30 | 19,998 |
| 5 MHz | 305.22 | 10.35 | 10.17 | 0.345 | 37.85 | 19,998 |

Los datos mostrados en el cuadro son del total de las 40 pruebas realizadas para cada velocidad, las columnas con terminación en S representan los datos para software y los de terminación H en el sistema embebido, siendo los últimos los de mayor interés. El tiempo promedio, representa los segundos que tardó en hacerse cada prueba, de un total de 3000 mensajes enviados con respuesta recibida más los errores generados, se puede ver que va disminuyendo un poco con forme aumenta la velocidad, con excepción de los 5 MHz, esto es debido a que existe una mayor cantidad de errores promedio, lo cual ocasiona que se deban de repetir más mensajes y por esto el tiempo promedio es mayor.

## 4.4. Pruebas y resultados de funcionamiento

Estas pruebas se realizaron en la empresa Tecno Turbo y Servicios Industriales sobre herramientas de maquinado, un torno vertical CNC y una fresadora, se integraron 3 esclavos en la maquinaria y la longitud del cable fue de 305 metros.

En la fresadora se monitoreo el proceso de generación de un diente para un engrane helicoidal de acero al carbón 9840, se utilizó el acelerómetro, magnetómetro y giroscopio de cada esclavo, la figura 25 muestra un diagrama de la instrumentación en la máquina.

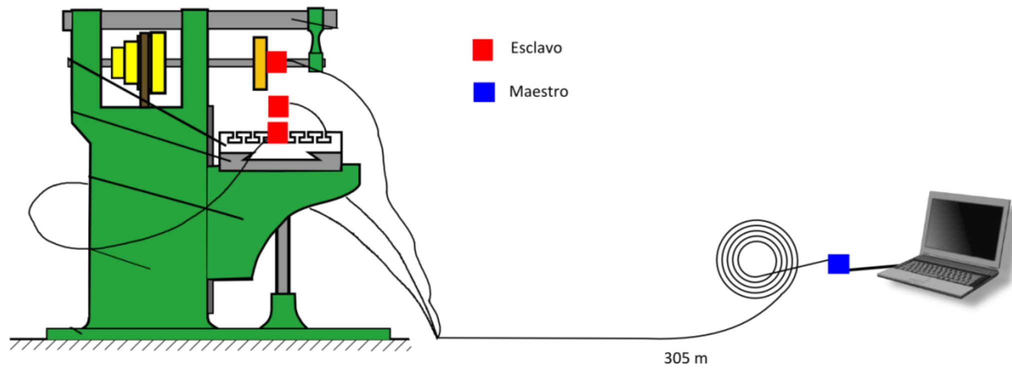



Figura 25. Diagrama de prueba en fresadora.

Los cuadros de color rojo muestran los lugares en donde se colocaron los esclavos y se pueden ver a detalle en la figura 26.

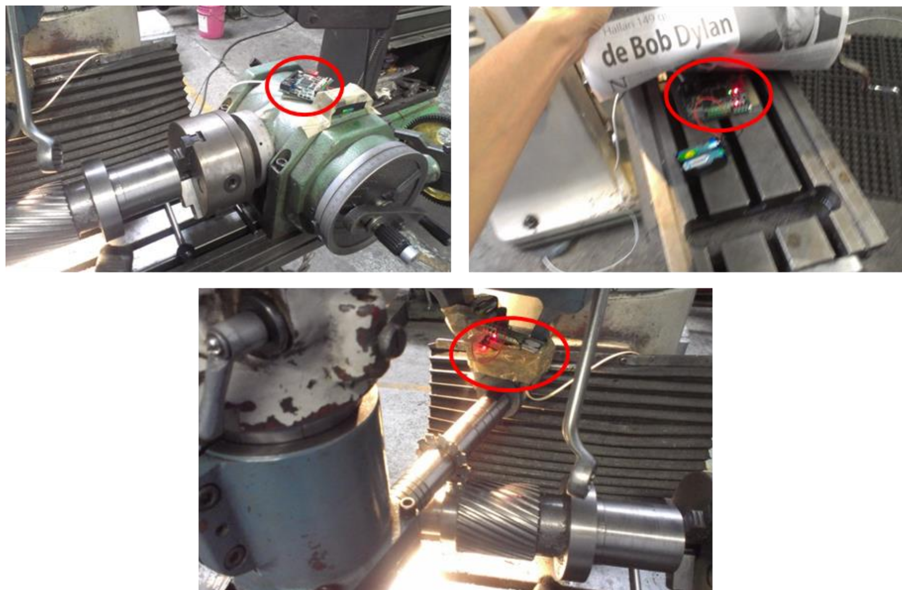



Figura 26. Esclavos en fresadora.

Algunos de los datos adquiridos a lo largo de proceso se muestran en las gráficas de la figura 27 donde se aprecia un monitoreo constante sin interrupciones.

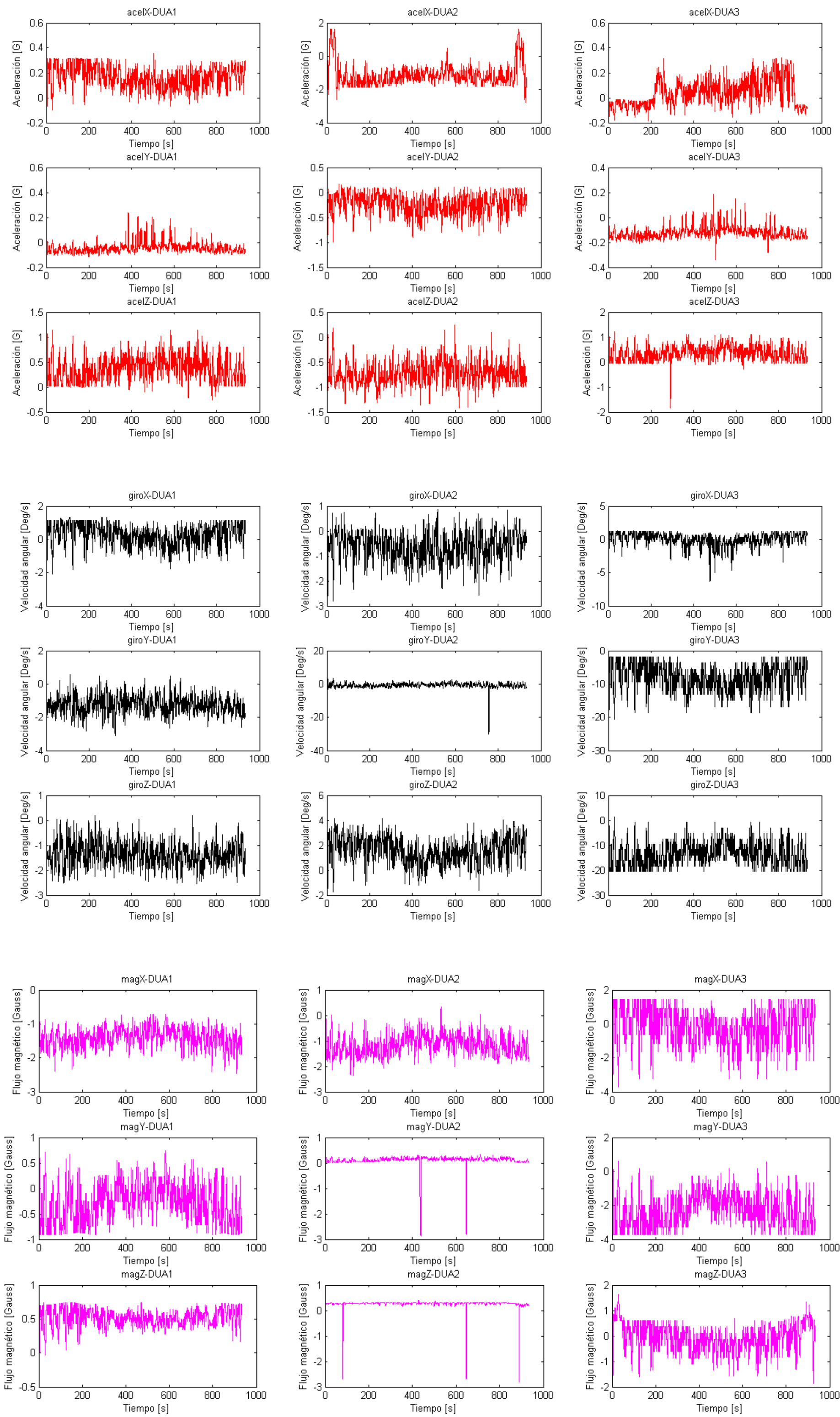



Figura 27. Gráficas de datos.

Las gráficas en color rojo representan los datos de los acelerómetros, se puede apreciar que los valores capturados son bastante constantes con ligeras variaciones, estos pequeños cambios se pueden interpretar como la variación en las vibraciones de la maquina a lo largo del maquinado del engrane, en el caso de la segunda grafica de color rojo se notan unas variaciones más marcadas al principio y al final de la muestra, esto puede ser ocasionado por el arranque y para de la maquinaria. Las gráficas de color negro son los datos de giroscopios, que también muestran ligeras variaciones, atribuibles a momentos torsionales en la maquina a lo largo del proceso. Finalmente las gráficas de color morado representan a los magnetómetros, que también presentan cierta variación visible que puede deberse a un incremento en el voltaje de los motores o en cualquier otra cosa que provoque un incremento en el campo magnético.

En la máquina CNC (torno vertical) se siguió el mismo procedimiento que en la fresadora, se montaron 3 esclavos en diferentes puntos y se monitorearon diferentes sensores, la diferencia es que en esta máquina no se encontraba ninguna pieza a maquinar, debido a esto se ejecutó un programa para el acabado de un pistón para flujo de gas, la figura 28 muestra el diagrama de cómo se instrumentó la maquinaria

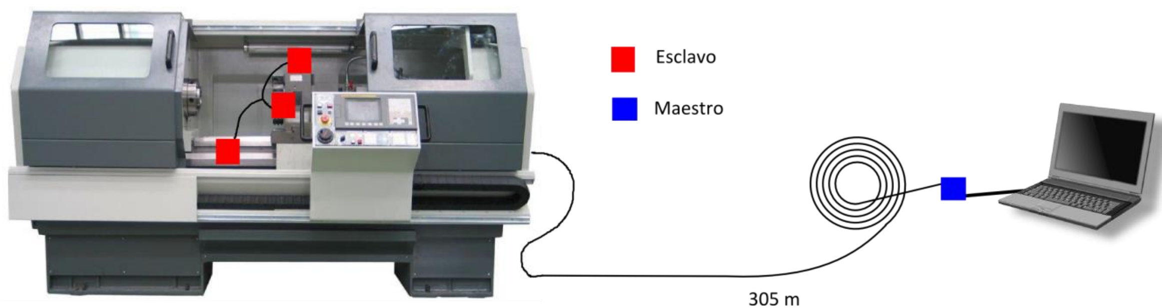



Figura 28. Diagrama en CNC.

Los cuadros de color rojo muestran los lugares en donde se colocaron los esclavos y se pueden ver a detalle en la figura 29.

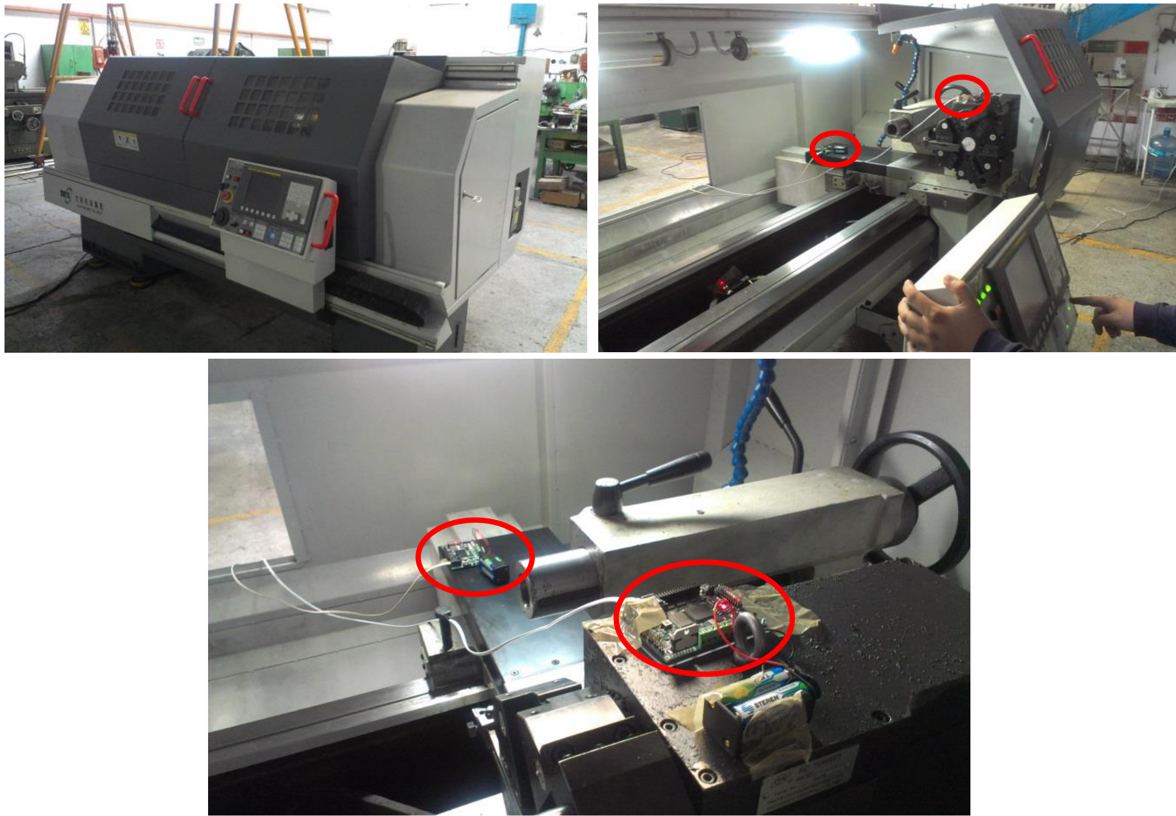


Figura 29. Esclavos en CNC.

Algunos de los datos adquiridos a lo largo de proceso se muestran en las gráficas de la figura 30 donde se aprecia un monitoreo constante sin interrupciones.

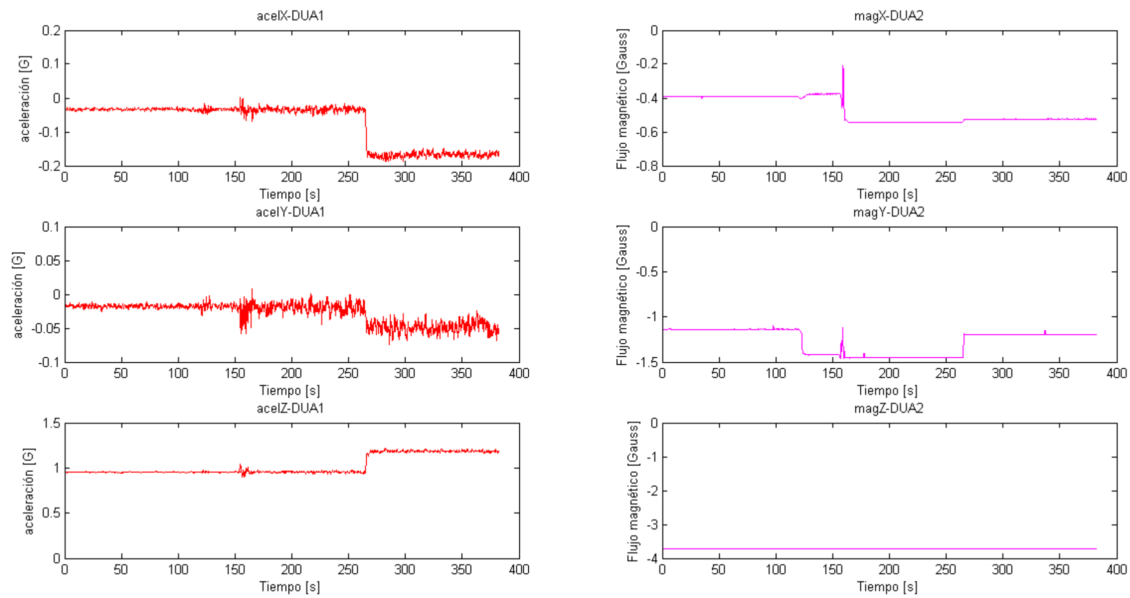



Figura 30. Gráficas de datos.

De la figura anterior las gráficas en color rojo representan los datos de diferentes ejes en los acelerómetros, donde se pueden observar algunos cambios muy marcados en el valor de los datos, estos cambios corresponden al desplazamiento de las partes móviles de la máquina, a lo largo del proceso de maquinado de alguna pieza en un plano (x, y). Las gráficas con una tonalidad morada representan las mediciones de diferentes magnetómetros, de igual manera que los acelerómetros se pueden notar cambios en los valores adquiridos, en este caso dichos cambios corresponden al accionamiento de los motores en la máquina, específicamente a los encargados de seleccionar la herramienta o desplazar la bancada. Las pruebas se realizaron con velocidades de 714 KHz y 2.5 MHz, se hiso así para probar la velocidad más baja y la más alta estable en ese momento, el tiempo promedio de actualización resultante de los datos es de 12 milisegundos.

Como resultado de todas las pruebas realizadas es posible realizar una evaluación general del sistema, contrastada con el trabajo existente y un sistema similar utilizado en la industria, montado para monitorear medidores de flujo micromotion de la marca Emerson. El cuadro 7 muestra los datos obtenidos.

Cuadro 7. Comparación de datos obtenidos.

| Sistema. | Distancia | Velocidad | Desconexión |
|---|---|---|---|
| Existente | 3m a 5m | 1.5 MHz | si |
| Industria | 150m | 153 KHz | no |
| Desarrollado | 305m | Hasta 5 MHz | no |

# V. CONCLUSIONES.

El presente trabajo de tesis muestra que es posible desarrollar herramientas enfocadas a tareas específicas, para ser usadas en procesos industriales o de laboratorio. Destacando la interacción entre hardware y software dentro de un sistema embebido con el objetivo de simplificar las tareas de desarrollo, influenciando de manera directa el costo computacional en el sistema completo.

El sistema creado tiene mejores prestaciones al existente, sobre todo en la parte de inmunidad al ruido y distancia de comunicación. Además fue creado utilizando los resultados de trabajos anteriores y plataformas de diseño libre u *open source* (código abierto), lo que le da a todo el trabajo la versatilidad de uso que se desee, ya que lo puede utilizar cualquier persona sin la necesidad de pagar alguna licencia.

En cuanto a las prestaciones del trabajo, se logró establecer comunicación en la red con una distancia mayor a 300 metros y con velocidades de 714 KHz, 1.2MHz, 2.5 MHz, hasta 5MHz, esta última con presencia constante de errores debido a la longitud del cable pero aún se considera funcional. Con una distancia de cable de 10 m se logra la comunicación a 10 MHz. Estos datos se obtuvieron con la conexión de 4 dispositivos en la red.

En mención a los resultados obtenidos, se concluye que son satisfactorios, ya que no solo se cumplieron los objetivos planteados, sino que se pudo validar el trabajo con organismos externos a la Universidad Autónoma de Querétaro, ya sea mediante comparaciones técnicas en cuanto a las prestaciones del presente trabajo como longitud de cable y velocidad, con sistemas similares operando en empresas como P&G o con trabajo directo en las instalaciones, usando la maquinaria industrial para realizar monitoreo con nuestros equipos en la empresa Tecno Turbo y Servicios Industriales.

También cabe mencionar que es posible realizar algunas mejoras en puntos específicos del trabajo, como, en los materiales utilizados en la capa física de transporte o en las modificaciones que se consideren pertinentes durante el uso de la GUI.

En relación con la publicación de congreso realizada, resulta satisfactoria la oportunidad de compartir conocimientos y dar a conocer el trabajo de la Universidad a lo largo del país, además de ser una experiencia enriquecedora tanto en lo profesional como en lo personal.

# Referencias


Alabau, A. 1992. Teleinformática y redes de computadores. Gráfiques 92, s.a. (2da. Ed.). Barcelona, España. 291p.

Dr. W. J. Buchanan, 2004. The Handbook of Data Communications and Networks. Springer US. 677 p.

Icarnegie Global Learning, 2008, Curso en diseño de interfaces, UAQ-SSD4-0801-30.

Gervais-Ducouret, S., 2011. Next smart sensors generation. Sensors Applications Symposium (SAS), 2011 IEEE 978-1-4244-8064-7/11 ©2011 IEEE.

Ghosh, S., 2012. Sensor Network Design for Smart Highways. IEEE Transactions On Systems, Man, And Cybernetics—Part A: Systems And Humans, Vol. 42, NO. 5, 1083-4427

Gtkmm.org, 2013. Interfaces C++ para GTK+ y GNOME.

Hui, L., 2012. Design and Application of Communication Gateway of EPA and MODBUS on Electric Power System. International Conference on Future Electrical Power and Energy Systems. Energy Procedia 17 ( 2012 ) 286 – 292.

Lomelí, H. U., 2012. Tesis Desarrollo e implementación de un sistema de giroscopios digitales Usando tecnología FPGA para monitoreo de la orientación de robots. Universidad Autónoma de Querétaro, Facultad de Ingeniería campus San Juan del Rio.

López, E., Ingeniería en microcontroladores Protocolo RS-485, i-micro 2-6.

Meneses, E., 2007. An Event-Triggered Smart Sensor Network Architecture. Industrial Informatics, 2007 5th IEEE International Conference on. 978-1-4244-0851-1/523-528

Modbus Organization (2013), Inc. About the protocol. http://www.modbus.org/faq.php.

Modicon (1996), Modbus Protocol Reference Guide, Modicon, Inc., Industrial Automation systems, Massachusetts 01845.

National Instruments, 2011. Tutorial, Introducción a la Tecnología FPGA.

Pardo, C., 1997. VHDL Lenguaje para descripción y modelado de circuitos. Universidad de Valencia, Ingeniería Informática.

Pérez, A. 2009. Sistemas Embebidos y Sistemas Operativos Embebidos. Centro de Investigación en Comunicación y Redes (CICORE). Universidad Central de Venezuela, Facultad de Ciencias.

Rane, P., 2010. Design of Modbus Controller Using VHDL for Remote Administrations of a Network of Devices. Emerging Trends in Engineering and Technology (ICETET). 2010 694 – 697.

Rivera, J., 2008. Improved Progressive Polynomial Algorithm for Self-Adjustment and Optimal Response in Intelligent Sensors. Sensors 2008, 8, 7410-7427; DOI: 10.3390/s8117410

Romero, R. 2007. Electrónica digital y lógica programable. Universidad de Guanajuato, Guanajuato, Gto.

Suhartono, D., 2012. Developing Controller Area Network Management Application Based on Modbus in Multi Generator Set Controller through Local Network and Internet. International Conference on Advances Science and Contemporary Engineering 2012 (ICASCE). Procedia Engineering 50 (2012) 426 – 435.

Xmcarne (2013). http://www.xmcarne.com/blog-tecnico/introduccion-modbus.

# Apéndice A, códigos.

## A.1 Módulo General.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity RS_485 is
  generic(
          n : integer := 8
     );
     port(
          RST : in  std_logic;
          CLK : in  std_logic;
          Vt  : in  std_logic_vector(15 downto 0);
          RO  : in  std_logic;
          ADDR: in  std_logic_vector(2 downto 0);
          WR  : in  std_logic;
          Di  : in  std_logic_vector(15 downto 0);
          EOR : out std_logic;
          EOT : out std_logic;
          DO  : out std_logic_vector(15 downto 0);
          RE  : out std_logic;                -- Habilitador del driver de entrada
          DE  : out std_logic;           -- Habilitador del driver de salida
          DIt : out std_logic
          );
end RS_485;

architecture Micro of RS_485 is

component Rs485_RO
     generic(
          n : integer := 8
     );
     port(
          RST  : in  std_logic;
          CLK  : in  std_logic;
          Vrx  : in  std_logic_vector(15 downto 0); -- Velocidad de recepción
          nbRx : in  std_logic_vector(15 downto 0); -- Número de bits a recibir
          Rx   : in  std_logic;
          RD   : in  std_logic;
          EOR  : out std_logic;
          RE   : out std_logic;
          EMPTY: out std_logic;
          Ddir : out std_logic_vector(7 downto 0);
          Dfnc : out std_logic_vector(7 downto 0);
          CrcRx: out std_logic_vector(15 downto 0);
```

```
			D    : out std_logic_vector(n-1 downto 0)
		);
end component;

component RS485_Tx is
		generic(
			n : integer := 8
		);
		port(
			RST : in  std_logic;
			CLK : in  std_logic;
			Vtx : in  std_logic_vector(15 downto 0);  -- Velocidad de transmisión
			STT : in  std_logic;                   -- Iniciar transmisión
			STT_Send: in std_logic;
			Ddir: in  std_logic_vector(7 downto 0);        -- Dirección del esclavo
			Dfnc: in  std_logic_vector(7 downto 0);        -- Función
			Data: in  std_logic_vector(n-1 downto 0); -- Dato a transmitir
			EOT : out std_logic;                   -- Fin de transmisión
			DE  : out std_logic;                   -- Habilitador del driver de salida
			DI  : out std_logic                          -- Manejador de entrada A o B
		);
end component;

signal DEi, REi: std_logic;
signal STT_PTx, STT_Send, RD_FIN, EOTm, EORm, Tx, RX_RD: std_logic;
signal DTx, DirTx, FncTx, DirRx, FncRx: std_logic_vector(7 downto 0);
signal nbRx, DcrcRx, Vrx, Vtx: std_logic_vector(15 downto 0);
signal DRx: std_logic_vector(n-1 downto 0);

begin


		Vtx <= Vt when Vt(0) = '1' else std_logic_vector(unsigned(Vt) + 1);
		Vrx <= '0' & Vt(15 downto 1);

		Modulo_00: RS485_Tx generic map(8) port map(RST, CLK, Vtx, STT_PTx, STT_Send, DirTx, FncTx,
DTx, EOTm, DEi, Tx);
		Modulo_01: RS485_RO generic map(8) port map(RST, CLK, Vrx, nbRx, RO, RX_RD, EORm, REi,
RD_FIN, DirRx, FncRx, DcrcRx, DRx);

		DIt <= Tx;
		DE  <= DEi;
  RE  <= DEi;
		EOR <= EORm;
		EOT <= EOTm;

		STT_PTx  <= Di(2) AND WR when ADDR="000" else '0';
		STT_Send <= Di(3) AND WR when ADDR="000" else '0';
		RX_RD    <= Di(4) AND WR when ADDR="000" else '0';
		DTx    <= (others=>'0') when RST='1' else Di(7 downto 0) when rising_edge(CLK) AND WR='1' AND
ADDR="001";
		DirTx <= (others=>'0') when RST='1' else Di(7 downto 0) when rising_edge(CLK) AND WR='1' AND
ADDR="010";
```

```
	FncTx <= (others=>'0') when RST='1' else Di(7 downto 0) when rising_edge(CLK) AND WR='1' AND
ADDR="011";

	Mux:process(RST,CLK)
	begin
		if RST='1' then
			Do <= (others=>'0');
		elsif rising_edge(CLK) then
			case ADDR is
				when "000"   => Do <= "00000000000" & RX_RD & STT_Send & STT_PTx &
EOTm & EORm;
				when "001"   => Do <= "00000000" & DTx;
				when "100"   => Do <= "00000000" & DRx;
				when "101"   => Do <= "000000000000000" & RD_FIN;
				when "110"   => Do <= "00000000" & FncRx;
				when others  => Do <= DcrcRx;
			end case;
		end if;
	end process;

end Micro;
```

## A.2 Módulos de Transmisión.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity RS485_Tx is
	generic(
		n : integer := 8
	);
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		Vtx : in  std_logic_vector(15 downto 0);  -- Velocidad de transmisión (baudios)
		STT : in  std_logic;                                          -- Iniciar transmisión
		STT_Send: in std_logic;
		Ddir: in  std_logic_vector(7 downto 0);         -- Dirección del esclavo
		Dfnc: in  std_logic_vector(7 downto 0);         -- Función
		Data: in  std_logic_vector(n-1 downto 0); -- Dato a transmitir
		EOT : out std_logic;                                          -- Fin de transmisión
		DE  : out std_logic;                    -- Habilitador del driver de salida
		DI  : out std_logic                            -- Manejador de entrada A o B
	);
end RS485_Tx;

architecture Tx of RS485_Tx is
```

```
component Divisor_M
	generic(
		n : integer := 16
	);
  Port (
		RST : in  STD_LOGIC;
    CLK : in  STD_LOGIC;
		H   : in  std_logic;
    M   : in  STD_LOGIC_vector(n-1 downto 0);
    Z   : out  STD_LOGIC
	);
end component;

component FSM_transmisor
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		STT : in  std_logic;
		H   : in  std_logic;
		Z   : in  std_logic;
		OPC : out std_logic_vector(1 downto 0);
		HC  : out std_logic;
		Tx  : out std_logic;
		DE  : out std_logic;
		OPR : out std_logic_vector(1 downto 0);
		EOT : out std_logic
	);
end component;

component Mb_cont
	generic (n : integer := 8
		);
	port(
		RST : in  std_logic ;
		CLK : in  std_logic ;
		OPC : in  std_logic_vector(1 downto 0);
		K   : in  std_logic_vector(n-1 downto 0);
		Z   : out std_logic;
		Cta : out std_logic_vector(n-1 downto 0)
		);
end component;

component Registro_CorrimientoTx
	generic (
	   n : integer := 8
		);
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		D   : in  std_logic_vector(n-1 downto 0);
		OPR : in  std_logic_vector(1 downto 0);
		Tx  : in  std_logic;
		DI  : out std_logic
```

```
        );
end component;

component reg_CorrimientocrcTx
        generic (
          n : integer := 8
                );
        port(
                RST : in  std_logic;
                CLK : in  std_logic;
                D   : in  std_logic_vector(n-1 downto 0);
                OPR : in  std_logic_vector(1 downto 0);
                Tx  : in  std_logic;
                DI  : out std_logic
        );
end component;

component MBus_CRC16
         port(
                 RST : in STD_LOGIC;
                 CLK : in STD_LOGIC;
                 OPR : in STD_LOGIC_VECTOR(1 downto 0);
                 Lcrc: in STD_LOGIC;
                 Di  : in STD_LOGIC;
                 Do  : out STD_LOGIC_VECTOR(15 downto 0)
           );
end component;

component Mb_FSM_Tx
        port(
                RST : in std_logic;
                CLK : in std_logic;
                STT : in std_logic;
                FTmr: in std_logic;
                EMPTY : in std_logic;
                EOT : in std_logic;
                STTx: out std_logic;
                ITmr: out std_logic;
                Sd  : out std_logic_vector(1 downto 0);
                Lcrc : out std_logic;
                RSRcrc: out std_logic;
                FLUSH: out std_logic;
                DE : out std_logic
        );
end component;

component Mb_FIFO
   generic(n : integer := 8; --! Input data number of bits
        w : integer := 10  --! Address number of bits
                        );
         port(
                 RST : in STD_LOGIC;                                    --! Reset
                 CLK : in STD_LOGIC;                                    --! Clock
```

```
			WR : in STD_LOGIC;								--!  Push   data   into
FIFO
			RD : in STD_LOGIC;								--!  Pop   data   from
FIFO
			FLUSH : in STD_LOGIC;							--! Flush current data
			FULL : out STD_LOGIC;							--! Full flag
			EMPTY : out STD_LOGIC;							--! Empty flag
			Di : in STD_LOGIC_VECTOR(n-1 downto 0);  --! Input data
			Do : out STD_LOGIC_VECTOR(n-1 downto 0)  --! Output data
		);
end component;

signal Z, Hc, H, Tx, bs, Lcrc, Dcrc, EOTi, DE0: std_logic;
signal FTmr, STTx, ITmr, RSTcrc: std_logic;
signal OPC, OPR, OPRcrc, OPRd, Sd: std_logic_vector(1 downto 0);
signal bc : std_logic_vector(3 downto 0);
signal CrcTx : std_logic_vector(15 downto 0);
signal D, Dfifo : std_logic_vector(7 downto 0);--(n+32 downto 0);

signal FULL, EMPTY, RD, FLUSH, Hdcrc, Htxcrc: std_logic;

begin

	modulo_00: Divisor_M		generic map(16)  port map(RST, CLK, Hc, Vtx, Z);			--
Controla la velocidad de transmisión
	modulo_01: FSM_transmisor			port map(RST, CLK, STTx, H, Z, OPC, Hc, Tx, DE0, OPR,
EOTi);
	modulo_02: Mb_cont			generic map(4)    port map(RST, CLK, OPC, "1001", H, bc);
					--Conteo de datos enviados
	modulo_03: Registro_CorrimientoTx generic map(8)   port map(RST, CLK, D, OPRd, Tx, bs);
				--Enviar datos
	modulo_04: MBus_CRC16			port map(RST, CLK, OPRcrc, Lcrc, Dcrc, CrcTx);
	    --Genera CRC
	modulo_05: Mb_FSM_Tx			port map(RST, CLK, STT_Send, FTmr, EMPTY, EOTi, STTx,
ITmr, Sd, Lcrc, RSTcrc, FLUSH, DE);  --Control de la transmisión
	modulo_06: Divisor_M		generic map(13)   port map(RST, CLK, ITmr, "1011001000000", FTmr);
--Establece el tiempo entre cada bite de datos(almenos 1T)
	modulo_07: Mb_FIFO		generic map(8,8) port map(RST, CLK, STT, RD, FLUSH, FULL, EMPTY,
Data, Dfifo);

	OPRd   <= OPR when rising_edge(CLK);
	EOT    <= FLUSH;
	RD     <= STT_Send or EOTi;
	Hdcrc  <= '1' when (bc > "0000" and bc < "1001") else '0';
  Dcrc   <= bs when Hdcrc = '1' else '0';
	Htxcrc <= TX when rising_edge(CLK);
  OPRcrc <= "11" when RST ='1' else RSTcrc & Htxcrc when Hdcrc = '1' else "00";
	DI     <= bs;

	SelD:process(Sd,CrcTx,Dfifo)
	begin
		case Sd is
			when "00"   => D <= Dfifo;
```

```
				when "01"		=> D <= CrcTx(7 downto 0);
				when others => D <= CrcTx(15 downto 8);
			end case;
		end process SelD;
end TX;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity Divisor_M is
		generic(
			n : integer := 16
		);
  Port (
			RST : in  STD_LOGIC;
    CLK : in  STD_LOGIC;
			H   : in  std_logic;
    M   : in  STD_LOGIC_vector(n-1 downto 0);
    Z   : out  STD_LOGIC
		);
end Divisor_M;

architecture Behavioral of Divisor_M is

signal Qn,Qp : std_logic_vector(n-1 downto 0);
signal Cero  : std_logic_vector(n-1 downto 0);
signal C     : std_logic;

begin
		Cero <= (others => '0');

		C <= '1' when Qp = Cero else '0';

		Qn <= std_logic_vector (unsigned(Qp)-1) when C='0' else M;

		Reloj: process(RST,CLK,M,Qn,H)
		begin
			if(RST='1') then
				Qp <= M;
			elsif(CLK'event and CLK='1') then
				if(H='0') then
					Qp <= Qn;
				else
					Qp <= M;
				end if;
			end if;
		end process Reloj;
		Z <= C;

end Behavioral;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FSM_transmisor is
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		STT : in  std_logic;
		H   : in  std_logic;
		Z   : in  std_logic;
		OPC : out std_logic_vector(1 downto 0);
		HC  : out std_logic;
		Tx  : out std_logic;
		DE  : out std_logic;
		OPR : out std_logic_vector(1 downto 0);
		EOT : out std_logic
	);
end FSM_transmisor;

architecture FSM of FSM_transmisor is

signal Qp, Qn : std_logic_vector(2 downto 0);

begin
	combinacional: process(STT,H,Z,Qp)
	begin
		case Qp is
			-- En espera de la señal de inicio
			when "000" =>
				if(STT='0') then
					Qn <= "000";
				else
					Qn <= "001";
				end if;
				OPC <= "01";
				Hc  <= '1';
				Tx  <= '0';
				DE  <= '0';
				EOT <= '0';
				OPR <= "10";
			when "001" =>
				Qn <= "011";

				OPC <= "01";
				Hc  <= '1';
				Tx  <= '0';
				DE  <= '1';
				EOT <= '0';
				OPR <= "00";
			when "010" =>
				Qn <= "011";

				OPC <= "10";
```

```
                        Hc  <= '0';
                        Tx  <= '1';
                        DE  <= '1';
                        EOT <= '0';
                        OPR <= "01";
                when "011" =>
                        if(H='1') then
                                Qn <= "100";
                        elsif(Z='1') then
                                Qn <= "010";
                        else
                                Qn <= "011";
                        end if;
                        OPC <= "00";
                        Hc  <= '0';
                        Tx  <= '0';
                        DE  <= '1';
                        EOT <= '0';
                        OPR <= "01";
                when "100" =>
                  if(Z='0') then
                                Qn <= "100";
                        else
                                Qn <= "101";
                        end if;
                        OPC <= "00";
                        Hc  <= '0';
                        Tx  <= '0';
                        DE  <= '1';
                        EOT <= '0';
                        OPR <= "01";
                when others =>
                  Qn <= "000";

                        OPC <= "01";
                        Hc  <= '1';
                        Tx  <= '0';
                        DE  <= '1';
                        EOT <= '1';
                        OPR <= "01";
        end case;
    end process combinacional;

    Qp <= (others => '0') when RST='1' else Qn when rising_edge(CLK);

end FSM;
```

---

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Mb_cont is
    generic (n : integer := 8
```

```
			);
		port(
			RST : in  std_logic ;
			CLK : in  std_logic ;
			OPC : in  std_logic_vector(1 downto 0);
			K   : in  std_logic_vector(n-1 downto 0);
			Z   : out std_logic;
			Cta : out std_logic_vector(n-1 downto 0)
			);
end Mb_cont ;

 architecture cont of Mb_cont is
 signal Qn ,Qp , Zr : std_logic_vector (n-1 downto 0);
 begin

		Cta <= Qp ;
		Zr <= (others => '0');
    Z <= '1' when Qp = K else '0';

		--! \ par Selector
		--! \ verbatim
		--! OPC | Qn
		--! ----------
		--! 00 | Qp Hold
		--! 01 | K Load
		--! 1x | Qp -1 Decrement
		--! \ endverbatim
		Mux : process (OPC ,Qp ,Zr)
		begin
			if OPC (1)= '1' then
				Qn <= std_logic_vector (unsigned(Qp)+1);
			elsif OPC (0)= '1' then
				Qn <= Zr;
			else
				Qn <= Qp ;
			end if ;
		end process ;

		Qp <= (others => '0') when RST = '1' else Qn when rising_edge(CLK);

 end cont;
```

---

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Registro_CorrimientoTx	is
		generic (
		   n : integer := 8
			);
		port(
			RST : in  std_logic;
			CLK : in  std_logic;
			D   : in  std_logic_vector(n-1 downto 0);
```

```
			OPR : in  std_logic_vector(1 downto 0);
			Tx  : in  std_logic;
			DI  : out std_logic
		);
end Registro_CorrimientoTx;

architecture Reg of Registro_CorrimientoTx is

signal Qp, Qn, DE : std_logic_vector(n+1 downto 0);

begin

		DE <= '0' & D & '1';
		combinacional: process(Tx,Qp)
		begin
			if(Tx='0') then
				Qn <= Qp;
			else
				Qn <= '0' & Qp(n+1 downto 1);
			end if;
			DI <= Qp(0);
		end process combinacional;

		Mux:process(RST,CLK,OPR)
		begin
			if RST='1' or OPR(1)='1' then
				Qp <= (others=>'0');
			elsif rising_edge(CLK) then
				case OPR(0) is
					when '0'    => Qp <= DE;
					when others => Qp <= Qn;
				end case;
			end if;
		end process;

end Reg;
```

---

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MBus_CRC16 is
		port(
			RST : in STD_LOGIC;
			CLK : in STD_LOGIC;
			OPR : in STD_LOGIC_VECTOR(1 downto 0);
			Lcrc: in STD_LOGIC;
			Di : in STD_LOGIC;
			Do : out STD_LOGIC_VECTOR(15 downto 0)
		 );
end MBus_CRC16;

architecture CRC16 of MBus_CRC16 is
signal Qn,Qp,G: std_logic_vector(15 downto 0);
```

```
begin

	Do <= Qp when Lcrc='1';-- else(others=>'1');

	G(0)  <= Qp(1) XOR Qp(0) XOR Di;
	G(1)  <= Qp(2);
	G(2)  <= Qp(3);
	G(3)  <= Qp(4);
	G(4)  <= Qp(5);
	G(5)  <= Qp(6);
	G(6)  <= Qp(7);
	G(7)  <= Qp(8);
	G(8)  <= Qp(9);
	G(9)  <= Qp(10);
	G(10) <= Qp(11);
	G(11) <= Qp(12);
	G(12) <= Qp(13);
	G(13) <= Qp(14) XOR Qp(0) XOR Di;
	G(14) <= Qp(15);
	G(15) <= Qp(0) XOR Di;

	Qn <= (others=>'1') when OPR(1)='1' else G;
	Qp <= (others=>'1') when RST='1' else Qn when rising_edge(CLK) AND (OPR(0)='1' OR OPR(1)='1');


end CRC16;
```

---

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mb_FSM_Tx is
	port(
		RST : in std_logic;
		CLK : in std_logic;
		STT : in std_logic;
		FTmr: in std_logic;
		EMPTY : in std_logic;
		EOT : in std_logic;
		STTx: out std_logic;
		ITmr: out std_logic;
		Sd  : out std_logic_vector(1 downto 0);
		Lcrc : out std_logic;
		RSRcrc: out std_logic;
		FLUSH :out std_logic;
		DE : out std_logic
	);
end Mb_FSM_Tx;

architecture FSM of Mb_FSM_Tx is

signal Qp, Qn : std_logic_vector(3 downto 0);

begin
```

```
combinacional: process(STT,EMPTY,FTmr,EOT,Qp)
begin
	case Qp is
		-- En espera de la señal de inicio
		when "0000" =>
			if(STT='0') then
				Qn <= "0000";
			else
				Qn <= "0001";
			end if;
			STTx <= '0';
			ITmr <= '1';  --desactivado en 1
			Sd   <= "00";
			Lcrc <= '1';
			RSRcrc <= '0';
			FLUSH <= '0';
			DE <= '0';
		when "0001" =>
			if(FTmr='0') then
				Qn <= "0001";
			else
				Qn <= "0010";
			end if;
			STTx <= '0';
			ITmr <= '0';
			Sd   <= "00";
			Lcrc <= '1';
			RSRcrc <= '0';
			FLUSH <= '0';
			DE <= '1';
		when "0010" =>
			Qn <= "0011";

			STTx <= '1';
			ITmr <= '1';
			Sd   <= "00";
			Lcrc <= '1';
			RSRcrc <= '0';
			FLUSH <= '0';
			DE <= '1';
		when "0011" =>
			if(EOT='1') then
				Qn <= "0001";
			elsif(EMPTY ='1')then
				Qn <= "0100";
			else
				Qn <= "0011";
			end if;
			STTx <= '0';
			ITmr <= '1';
			Sd   <= "00";
			Lcrc <= '1';
			RSRcrc <= '0';
```

```
            FLUSH <= '0';
            DE <= '1';
        when "0100" =>
            if(EOT='1') then
                Qn <= "0101";
            else
                Qn <= "0100";
            end if;
            STTx <= '0';
            ITmr <= '1';
            Sd   <= "01";
            Lcrc <= '0';
            RSRcrc <= '0';
            FLUSH <= '0';
            DE <= '1';
        when "0101" =>
            if(FTmr='0') then
                Qn <= "0101";
            else
                Qn <= "0110";
            end if;

            STTx <= '0';
            ITmr <= '0';
            Sd   <= "01";
            Lcrc <= '0';
            RSRcrc <= '0';
            FLUSH <= '0';
            DE <= '1';
        when "0110" =>
            Qn <= "0111";

            STTx <= '1';
            ITmr <= '1';
            Sd   <= "10";
            Lcrc <= '0';
            RSRcrc <= '0';
            FLUSH <= '0';
            DE <= '1';
        when "0111" =>
            if(EOT='1') then
                Qn <= "1000";
            else
                Qn <= "0111";
            end if;

            STTx <= '0';
            ITmr <= '1';
            Sd   <= "10";
            Lcrc <= '0';
            RSRcrc <= '1';
            FLUSH <= '0';
            DE <= '1';
```

```
                                        --STTx <= '0';
--                                      ITmr <= '1';  --desactivado en 1
--                                      Sd   <= "10";
--                                      Lcrc <= '0';
--                                      RSRcrc <= '0';
--                                      FLUSH <= '0';
--                                      DE <= '1';
                              when "1000" =>
                                        Qn <= "1001";

                                        STTx <= '0';
                                        ITmr <= '0';  --desactivado en 1
                                        Sd   <= "10";
                                        Lcrc <= '0';
                                        RSRcrc <= '0';
                                        FLUSH <= '1';
                                        DE <= '1';
                              when others =>
                                        if(FTmr='0') then
                                                  Qn <= "1001";
                                        else
                                                  Qn <= "0000";
                                        end if;

                                        STTx <= '0';
                                        ITmr <= '0';  --desactivado en 1
                                        Sd   <= "10";
                                        Lcrc <= '0';
                                        RSRcrc <= '0';
                                        FLUSH <= '1';
                                        DE <= '1';
                    end case;
          end process combinacional;

          Qp <= (others => '0') when RST='1' else Qn when rising_edge(CLK);

end FSM;
```

---

```
library IEEE;                                       --! Standard IEEE library
use IEEE.STD_LOGIC_1164.all;       --! Standard IEEE 1164 package
use IEEE.NUMERIC_STD.all;                    --! Standard IEEE numeric package

entity Mb_FIFO is
   generic(n : integer := 8; --! Input data number of bits
        w : integer := 10  --! Address number of bits
                              );
          port(
                    RST : in STD_LOGIC;                                   --! Reset
                    CLK : in STD_LOGIC;                                   --! Clock
                    WR : in STD_LOGIC;                                           --!   Push   data   into
FIFO
```

```
			RD : in STD_LOGIC;						--!	Pop	data	from
FIFO
			FLUSH : in STD_LOGIC;					--! Flush current data
			FULL : out STD_LOGIC;					--! Full flag
			EMPTY : out STD_LOGIC;					--! Empty flag
			Di : in STD_LOGIC_VECTOR(n-1 downto 0);  --! Input data
			Do : out STD_LOGIC_VECTOR(n-1 downto 0)  --! Output data
		);
end Mb_FIFO;

architecture Mb_FIFO of Mb_FIFO is
signal E,F : std_logic;
signal ADW,ADR : std_logic_vector(w-1 downto 0) := (others=>'0');
--! Define a new data type
type RAMtype is array (0 to (2**w)-1) of std_logic_vector(n-1 downto 0);
--! Memory array
signal MEM : RAMtype;
begin

	FULL  <= F;
	EMPTY <= E;


	MEM(to_integer(unsigned(ADW))) <= Di when rising_edge(CLK) AND WR='1';
	Do <= MEM(to_integer(unsigned(ADR))) when rising_edge(CLK) AND RD='1';

	F <= '0' when FLUSH='1' else '1' when std_logic_vector(unsigned(ADW)+1)=ADR else '0';
	E <= '0' when FLUSH='1' else '1' when ADW=ADR else '0';

	Counts: process(RST,CLK)
	begin
		if RST='1' then
			ADW <= (others=>'0');
			ADR <= (others=>'0');
		elsif rising_edge(CLK) then
			if FLUSH='1' then
				ADW <= (others=>'0');
				ADR <= (others=>'0');
			else
				if WR='1' AND F='0' then
					ADW <= std_logic_vector(unsigned(ADW)+1);
				end if;
				if RD='1' AND E='0' then
					ADR <= std_logic_vector(unsigned(ADR)+1);
				end if;
			end if;
		end if;
	end process;

end Mb_FIFO;
```

## A.3 Módulos de Recepción.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Rs485_RO is
	generic(
		n : integer := 8
	);
	port(
		RST  : in  std_logic;
		CLK  : in  std_logic;
		Vrx  : in  std_logic_vector(15 downto 0);
		nbRx : in  std_logic_vector(15 downto 0);
		Rx   : in  std_logic;
		RD   : in  std_logic;
		EOR  : out std_logic;
		RE   : out std_logic;
		EMPTY: out std_logic;
		Ddir : out std_logic_vector(7 downto 0);
		Dfnc : out std_logic_vector(7 downto 0);
		CrcRx: out std_logic_vector(15 downto 0);
		D    : out std_logic_vector(n-1 downto 0)
	);
end Rs485_RO;

architecture Rx of Rs485_RO is

component Divisor_M
	generic(
		n : integer := 16
	);
  Port (
		RST : in  STD_LOGIC;
    CLK : in  STD_LOGIC;
		H   : in  std_logic;
    M   : in  STD_LOGIC_vector(n-1 downto 0);
    Z   : out  STD_LOGIC
	);
end component;

component FSM_Receptor
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		Rx  : in  std_logic;
		H   : in  std_logic;
		Z   : in  std_logic;
		OPC : out std_logic_vector(1 downto 0);
		Hb  : out std_logic;
```

```
			S   : out std_logic;
			RE  : out std_logic;
			EOR : out std_logic
		);
end component;

component Registro_CorrimientoRx
		generic (n : integer := 8
			);
		port(
			RST : in  std_logic;
			CLK : in  std_logic;
			S   : in  std_logic;
			Rx  : in  std_logic;
			DR  : out std_logic_vector(n-1 downto 0)
		);
end component;

component Mb_cont
		generic (n : integer := 8
			);
		port(
			RST : in  std_logic ;
			CLK : in  std_logic ;
			OPC : in  std_logic_vector(1 downto 0);
			K   : in  std_logic_vector(n-1 downto 0);
			Z   : out std_logic;
			Cta : out std_logic_vector(n-1 downto 0)
			);
end component;

component MBus_CRC16
		 port(
			 RST : in STD_LOGIC;
			 CLK : in STD_LOGIC;
			 OPR : in STD_LOGIC_VECTOR(1 downto 0);
			 Lcrc: in STD_LOGIC;
			 Di  : in STD_LOGIC;
			 Do  : out STD_LOGIC_VECTOR(15 downto 0)
		   );
end component;

component Mb_FIFO
   generic(n : integer := 8; --! Input data number of bits
	   w : integer := 10  --! Address number of bits
				   );
		 port(
			 RST : in STD_LOGIC;				--! Reset
			 CLK : in STD_LOGIC;				--! Clock
			 WR : in STD_LOGIC;					--!  Push  data  into
FIFO
			 RD : in STD_LOGIC;					--!  Pop  data  from
FIFO
```

```
                FLUSH : in STD_LOGIC;                                   --! Flush current data
                FULL : out STD_LOGIC;                                   --! Full flag
                EMPTY : out STD_LOGIC;                                  --! Empty flag
                Di : in STD_LOGIC_VECTOR(n-1 downto 0);  --! Input data
                Do : out STD_LOGIC_VECTOR(n-1 downto 0)  --! Output data
        );
end component;

component Mb_FSM_Ro
        port(
    RST   : in std_logic;
                CLK   : in std_logic;
                EORp  : in std_logic;
                FTmr  : in std_logic;
                Rx    : in std_logic;
                EMPTY : in std_logic;
                ITmr  : out std_logic;
                RSTcrc: out std_logic;
                EOR   : out std_logic;
                FLUSH : out std_logic
        );
end component;

signal Z, S, H, Hb, EORi, EORf, Lcrc, Dcrc, ITmr, FTmr, FULL, EPTY, FLUSH, RSTcrc : std_logic;
signal OPC, OPRcrc   : std_logic_vector(1 downto 0);
signal bc : std_logic_vector(3 downto 0);
signal DR : std_logic_vector(7 downto 0);

begin

        Modulo_00: Divisor_M          generic map (16) port map(RST, CLK, Hb, Vrx, Z);
        Modulo_01: FSM_Receptor                        port map(RST, CLK, Rx, H, Z, OPC, Hb, S, RE, EORi);
        Modulo_02: Registro_CorrimientoRx generic map (8)  port map(RST, CLK, S, Rx, DR);
        Modulo_03: Mb_Cont            generic map (4)  port map(RST, CLK, OPC, "0111", H, bc);
        Modulo_04: MBus_CRC16                                                          port
map(RST, CLK, OPRcrc, Lcrc, Dcrc, CrcRX);
        Modulo_05: Mb_FSM_Ro                                                  port  map(RST,
CLK, EORi, FTmr, Rx, EPTY, ITmr, RSTcrc, EORf, FLUSH);
        Modulo_06: Divisor_M          generic map (13)  port map(RST, CLK, ITmr, "1011001000001",
FTmr);          --Establece el tiempo entre cada bite de datos(almenos 1T)
        Modulo_07: Mb_FIFO            generic map(8,8) port map(RST, CLK, EORi, RD, FLUSH, FULL, EPTY,
DR, D);

        EOR    <= EORf;
  Dcrc   <= Rx when S = '1' else '0';
        OPRcrc <= "11" when RST ='1' else RSTcrc & S;
        Lcrc   <= EORf;
        EMPTY  <= EPTY;

end Rx;
```

---

```
library    IEEE;
use IEEE.std_logic_1164.all;
```

```
entity FSM_Receptor is
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		Rx  : in  std_logic;
		H   : in  std_logic;
		Z   : in  std_logic;
		OPC : out std_logic_vector(1 downto 0);
		Hb  : out std_logic;
		S   : out std_logic;
		RE  : out std_logic;
		EOR : out std_logic
	);
end FSM_Receptor;

architecture FSM of FSM_Receptor is
signal Qn, Qp : std_logic_vector(3 downto 0);
begin
	Maquina: process(Rx,Z,H,Qp)--RDD
	begin
		case Qp is
			-- Espera cambio de la señal Rx (bit de inicio)
			when "0000" =>
				if(Rx='0') then
					Qn <= "0000";
				else
					Qn <= "0001";
				end if;
				OPC <= "01";
				Hb	<= '1';
				S   <= '0';
				RE	<= '1';
				EOR <= '0';
			when "0001" =>
				Qn <= "0010";-- ajuste

				OPC <= "01";
				Hb	<= '0';
				S   <= '0';
				RE	<= '0';
				EOR <= '0';
			when "0010" =>
				if(Z='0') then
					Qn <= "0010";
				else
					Qn <= "0011";
				end if;
				OPC <= "00";
				Hb	<= '0';
				S   <= '0';
				RE	<= '0';
				EOR <= '0';
```

```
when "0011" =>
            if(Z='0') then
                        Qn <= "0011";
            else
                        Qn <= "0100";
            end if;
            OPC <= "00";
            Hb         <= '0';
            S   <= '0';
            RE         <= '0';
            EOR <= '0';
when "0100" =>
            if(Z='0') then
                        Qn <= "0100";
            else
                        Qn <= "0101";
            end if;

            OPC <= "00";
            Hb         <= '0';
            S   <= '0';
            RE         <= '0';
            EOR <= '0';
when "0101" =>
            if(H='1') then
                        Qn <= "0110";
            else
                        Qn <= "0011";
            end if;

            OPC <= "10";
            Hb         <= '0';
            S   <= '1';
            RE         <= '0';
            EOR <= '0';
when "0110" =>   --bit de paro
   if(Z='1') then
                        Qn <= "0111";
            else
                        Qn <= "0110";
            end if;
            OPC <= "00";
            Hb         <= '0';
            S   <= '0';
            RE         <= '0';
            EOR <= '0';
when "0111" =>
  if(Z='1') then
                        Qn <= "1000";
            else
                        Qn <= "0111";
            end if;
```

```
					OPC <= "00";
					Hb	<= '0';
					S   <= '0';
					RE	<= '0';
					EOR <= '0';		--fin bit de paro
				when "1000" =>
				   if(Z='1') then
						Qn <= "1001";
					else
						Qn <= "1000";
					end if;
					OPC <= "00";
					Hb	<= '0';
					S   <= '0';
					RE	<= '0';
					EOR <= '0';
				when "1001" =>
				  if(Z='1') then
						Qn <= "1010";
					else
						Qn <= "1001";
					end if;

					OPC <= "00";
					Hb	<= '0';
					S   <= '0';
					RE	<= '0';
					EOR <= '0';
				when others =>
				  Qn <= "0000";

					OPC <= "01";
					Hb	<= '1';
					S   <= '0';
					RE	<= '1';
					EOR <= '1';
		end case;
	end process maquina;

	Qp <= (others => '0') when RST='1' else Qn when rising_edge(CLK);

end FSM;
```

---

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Registro_CorrimientoRx	is
	generic (n : integer := 8
		);
	port(
		RST : in  std_logic;
		CLK : in  std_logic;
		S   : in  std_logic;
```

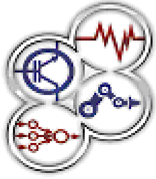
```
		Rx  : in  std_logic;
		DR  : out std_logic_vector(n-1 downto 0)
	);
end Registro_CorrimientoRx;

architecture Reg of Registro_CorrimientoRx is
signal Qp, Qn : std_logic_vector(n-1 downto 0);
begin
	combinacional: process(Rx,S,Qp)
	begin
		if(S='0') then
			Qn <= Qp;
		else
			Qn <= Rx & Qp(n-1 downto 1);
		end if;
		DR <= Qp;
	end process combinacional;

	Qp <= (others => '0') when RST='1' else Qn when rising_edge(CLK);

end Reg;
```

---

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mb_FSM_Ro is
	port(
		RST   : in std_logic;
		CLK   : in std_logic;
		EORp  : in std_logic;
		FTmr  : in std_logic;
		Rx    : in std_logic;
		EMPTY : in std_logic;
		ITmr  : out std_logic;
		RSTcrc: out std_logic;
		EOR   : out std_logic;
		FLUSH : out std_logic
	);
end Mb_FSM_Ro;

architecture FSM of Mb_FSM_Ro is

signal Qp, Qn : std_logic_vector(2 downto 0);

begin
	combinacional: process(Rx,EORp,FTmr,EMPTY,Qp)
	begin
		case Qp is
			when "000" =>
				if(Rx='0') then
					Qn <= "000";
				else
					Qn <= "001";
```

```
			end if;
			ITmr   <= '1';  --desactivado en 1
			RSTcrc <= '0';
			FLUSH  <= '0';
			EOR    <= '0';
		when "001" =>
			Qn <= "010";

			ITmr   <= '1';
			RSTcrc <= '0';
			FLUSH  <= '1';
			EOR    <= '0';
		when "010" =>
			if(EORp='0') then
				Qn <= "010";
			else
				Qn <= "011";
			end if;
			ITmr   <= '1';
			RSTcrc <= '0';
			FLUSH  <= '0';
			EOR    <= '0';
		when "011" =>
			if(FTmr='1') then
				Qn <= "100";
			elsif(Rx = '0') then
				Qn <= "011";
			else
				Qn <= "010";
			end if;
			ITmr   <= '0';
			RSTcrc <= '0';
			FLUSH  <= '0';
			EOR    <= '0';
		when "100" =>
			Qn <= "101";

			ITmr   <= '1';
			RSTcrc <= '0';
			FLUSH  <= '0';
			EOR    <= '1';
		when "101" =>
			if(EMPTY='0') then
				Qn <= "101";
			else
				Qn <= "110";
			end if;
			ITmr   <= '1';
			RSTcrc <= '0';
			FLUSH  <= '0';
			EOR    <= '0';
		when others =>
			Qn <= "000";
```

```
                    ITmr   <= '1';
                    RSTcrc <= '1';
                    FLUSH  <= '1';
                    EOR    <= '0';
        end case;
    end process combinacional;

    Qp <= (others => '0') when RST='1' else Qn when rising_edge(CLK);

end FSM;
```





# Apéndice B, artículo

## Protocolo Modbus 485 aplicado a monitoreo inercial con microprocesador embebido xQ16v7 en FPGA

E. Guillén García, L. Morales Velázquez, J Jesús de Santiago Pérez, Carlos Andrés Pérez Ramírez, Roque A. Osornio Rios.

**Resumen:** En este trabajo se emplearon diferentes herramientas de software y hardware para implementar una red de sensores inerciales con un microprocesador embebido propietario basado en FPGA. Se desarrollan diferentes módulos en VHDL con tareas específicas, como: la comunicación serial RS485, la verificación por redundancia cíclica (CRC), y la interconexión con el bus de datos del procesador, además del protocolo Modbus en modo RTU. También fue necesario crear librerías de software con el fin de incorporar macros para definir los datos que serán transmitidos por la red y establecer los parámetros de comunicación que indica el protocolo Modbus. Finalmente, como medio físico de transporte se utiliza cable telefónico de 4 vías y conectores de cuatro terminales RJ-11. Al integrar todos estos elementos se consiguió establecer comunicación dentro de la red con un ancho de banda de 7.7 Kb/s por lectura completa de un sensor incluyendo el encapsulado estándar marcado para Modbus.

Palabras Clave: FPGA, VHDL, Sensores inerciales, Microprocesador embebido.

Emmanuel Guillén García. imanolgg@hotmail.com
Luis Morales Velázquez. lmorales@hspdigital.org
J Jesús de Santiago Pérez jjdesantiago@hspdigital.org
Carlos Andrés Pérez Ramírez cperez@hspdigital.org
Roque A. Osornio Rios raor@uaq.mx
Universidad Autónoma de Querétaro, Facultad de Ingeniería, Campus San Juan del Rio, Sanjuan del Rio Querétaro, 76807

**Abstract**: In this work different software and hardware tools are used to implement an inertial-sensors-based network, using a property embedded microprocessor based on FPGA, to achieve this it was necessary to develop some VHDL modules with specific tasks like: RS485 serial communication, cyclic redundancy check (CRC) and interconnection between processor data bus and Modbus protocol in RTU mode. In addition, it was needed to develop software libraries to include macros with the data to transmit that define and establish communication parameters for the Modbus protocol. Finally, telephonic cable and RJ-11 connectors are used as a physic layer. Within the integration of these elements inside the communication net it was successfully to reach a 7.7 Kb/s bandwidth for a complete read of one sensor.

Keywords: FPGA, VHDL, inertial sensors, embedded microprocessor.

### Introducción

Los procesos de producción que actualmente son necesarios en un ambiente industrial o de laboratorio, requieren del constante monitoreo y control de las variables físicas que intervienen en dicho proceso, es por esta razón que se considera pertinente la implementación de redes de sensores mediante las cuales sea posible el monitoreo y control de las variables involucradas. Estas redes deben de estar sujetas a normas específicas para su correcta operación, surgiendo así protocolos que establecen pasos a seguir para alcanzar dichos

objetivos. Uno de los protocolos de comunicación más utilizados para este propósito es ModBus por su flexibilidad y resistencia al ruido, fue desarrollado en 1979 por Modicon como un protocolo de comunicación entre dispositivos. Ejemplos de estas tecnologías los encontramos [1] donde propusieron una nueva arquitectura para redes de sensores inteligentes, manejada mediante eventos derivados de una compresión logarítmica de los datos dentro del sensor, o en [2] que propusieron un modelo único y eficiente para redes de sensores, con el objetivo de crear carreteras senso-inteligentes, utilizando ModBus encontramos que en [3] realizaron la arquitectura del protocolo Modbus en VHDL (*Very high speed integrated circuit Hardware Description Language, lenguaje descriptivo de circuitos integrados de muy alta velocidad*) para la administración remota de una red de dispositivos utilizando la interfaz RS-232 para la comunicación de la red, también en [4] construyeron una red de área local entre generadores eléctricos basándose en el protocolo Modbus, utilizando después la interfaz RS-485 para comunicar los generadores, se puede encontrar también el caso de [5] donde implementó en FPGA (*Field Programmable Gate Array, Arreglo de Compuertas Programables en campo*) una unidad de pre procesamiento digital para giroscopios digitales de tres ejes tipo MEMS, esto con la finalidad de aumentar la exactitud en las señales del sensor para obtener la orientación de un robot. Además en [6] realizaron un algoritmo progresivo para el auto ajuste de sensores inteligentes con el objetivo de minimizar el error no lineal en la medición del sensor.

La implementación del protocolo Modbus-485 en una plataforma embebida en FPGA mediante el uso del microprocesador xQ16v7 que se presenta en este trabajo, resulta diferente de implementaciones similares anteriormente mencionadas debido a la integración de un sistema embebido que maneja la interacción entre hardware y software facilitando las tareas más redundantes y potenciando la capacidad de cálculo del sistema en conjunto con la interfaz de comunicación RS485.

## Procesador.

El Procesador utilizado para esta aplicación es el xQ16v7 el cual se desarrolló con la finalidad de potenciar el uso de los FPGA en aplicaciones donde se exploten la alta potencia de cálculo de los FPGAs y la rapidez de desarrollo de software. El xQ16v7 es un procesador de 16 bits pensado como una plataforma para construcción de sistemas embebidos personalizados de alto desempeño, este procesador es RISC de 16 bits de arquitectura Harvard modificada con estructura pipe-line de 6 estados con cuatro ciclos de reloj por instrucción, cuenta con 16 registros internos de 16 bits, es capaz de direccionar hasta 256KB de memoria, considera hasta 1024 puertos para periféricos, cuenta con un servicio de interrupción rápido, además de cinco modos de direccionamiento de datos. La Figura 1 muestra un diagrama simplificado de la arquitectura el xQ16v7, que incluye una unidad aritmética-lógica (ALU), un conjunto de registros, un registro de instrucción, un decodificador de instrucciones, un generador de direcciones y un contador. En la arquitectura del xQ16v7 no se incluye una unidad de microcontrol (MCU) ya que la codificación de instrucciones es tal que el flujo del sistema es controlado por el contador de 4 estados.

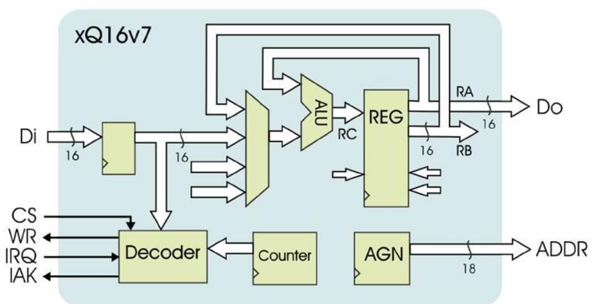



Fig. 1 Arquitectura del procesador.

## Interfaz RS485.

La implementación de RS485 está dividida en dos partes, la primera consiste en desarrollar los elementos que estarán dentro del sistema embebido como el módulo de VHDL para manejar la comunicación RS-485 en Hardware, la figura 2 muestra el diagrama de bloques del módulo construido para la recepción del dato, donde se tiene (RO) como la señal de entrada proveniente del exterior, (D) es el dato obtenido de la recepción, (EOR) indica que se ha terminado de recibir el dato y (M) es un valor constante que se utiliza para determinar la velocidad de

recepción de los datos. Por otra parte en la figura 3 podemos observar el diagrama del módulo construido para la transmisión de un dato.

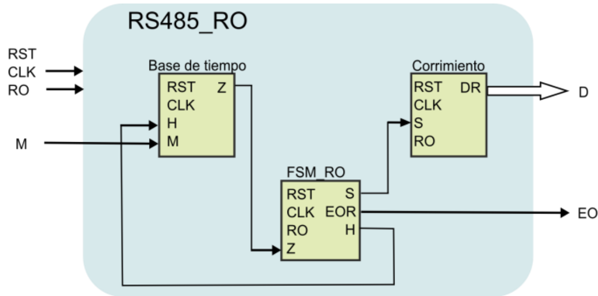



Fig. 2 Módulo para la recepción en hardware.

De manera similar a la recepción tenemos una entrada (D) que define el dato que se desea transmitir, (DI) es la señal que sale del sistema para transmitir, la señal (STT) se utiliza para iniciar la transmisión de un dato, (EOT) indica el final de la transmisión, (M) tiene la misma función que el módulo de recepción.

## Firmware del sistema.

Para continuar con la implementación de la interfaz es necesario realizar el firmware para el manejo de la transmisión y recepción que estará dentro del sistema embebido y que funcionara como periférico dentro de la arquitectura del procesador.

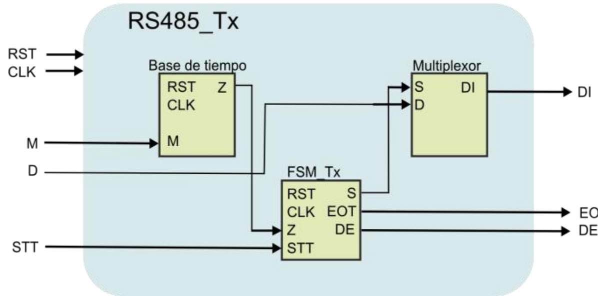



Fig. 3 Módulo para la transmisión en hardware.

Este firmware consiste en la creación de una librería en lenguaje C, la cual contiene las direcciones de acceso para el procesador, las macros que serán utilizadas para la comunicación, los valores de inicio para el procesador y las interrupciones de sistema que sean necesarias para la transmisión y recepción de los datos. La figura 4 muestra el sistema embebido completo incluyendo la interfaz RS485 creada.

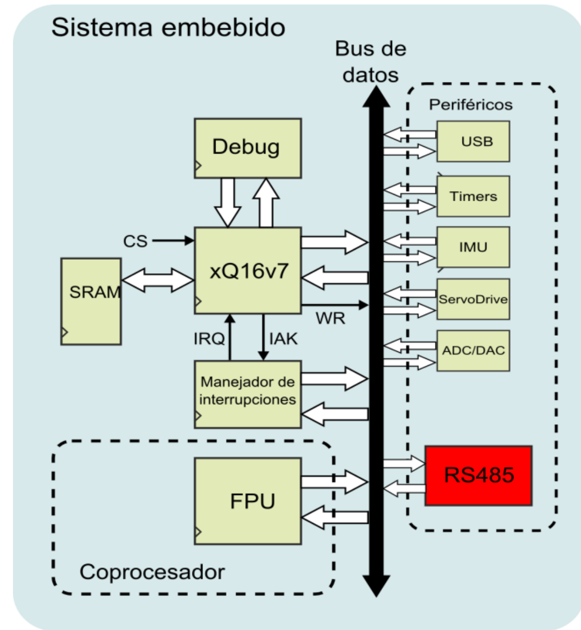



Fig. 4 Sistema embebido.

La segunda parte de la implementación, consiste en crear el firmware que estará en el microprocesador y que tendrá como función el monitoreo de los sensores inerciales para la comunicación con el sistema embebido, este elemento es desarrollado en lenguaje C y su función principal es recibir las instrucciones que emita el sistema embebido y tomar la medición del sensor solicitada, después de esto se responde a la petición con el dato adquirido, La figura 5 muestra el planteamiento general del sistema completo.

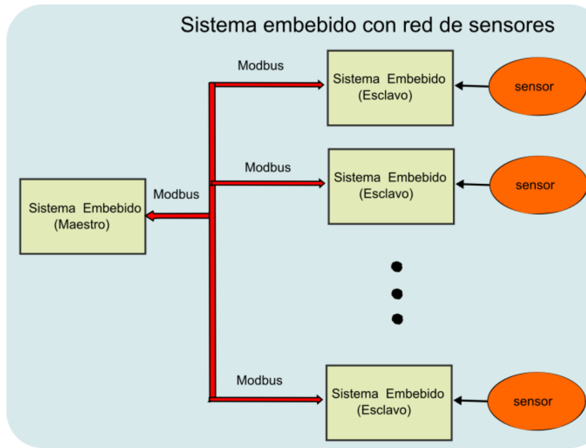



Fig. 5 Sistema implementado.

## Protocolo Modbus.

Para este trabajo se implementó el protocolo Modbus en el modo RTU (*Remote Terminal Unit, Unidad de Control Remota*) utilizando el encapsulado de datos que sugiere MODICON [7] que se puede ver en la figura 6.

| START | ADDRESS | FUNCTION | DATA | CRC CHECK | END |
|---|---|---|---|---|---|
| T1–T2–T3–T4 | 8 BITS | 8 BITS | *n* x 8 BITS | 16 BITS | T1–T2–T3–T4 |

Fig. 6 Entramado de datos.

Donde la primera trama de datos contiene la dirección del dispositivo esclavo con el cual se requiere tener comunicación, en la trama de función se indica el dato que es necesario leer, la tercera parte contiene la especificación de los datos que se desean obtener y la última trama está reservada para enviar el código CRC (*Cyclical Redundancy Check, Verificación por Redundancia Cíclica*) este dato es un código de detección de errores que se obtiene por medio de una división de polinomios calculado al momento de transmitir la trama del dato [7], se utiliza para que el dispositivo receptor pueda comprobar si el mensaje recibido es correcto, esto se realiza en ambos lados de la comunicación, el dispositivo transmisor calcula el CRC y se envía en la última trama de la comunicación el receptor recalcula el CRC al recibir el dato, si es igual al código recibido quiere decir que la comunicación se realizó de manera adecuada, de lo contrario responde con un código de error para indicar que el dato no se recibió correctamente.

## Resultados

Para probar el funcionamiento de la red y del microprocesador embebido, se realizó la implementación del sistema en un FPGA Spartan-3E de Xilinx, con capacidad de 1600k de compuertas lógicas y una frecuencia de reloj de 48MHz.

La tabla 1 muestra la capacidad consumida del dispositivo en la implementación del sistema completo.

**Tabla 1**: Recursos utilizados en el FPGA Spartan-3E.

| Utilización Lógica | Usado | Disponible | % Uso |
|---|---|---|---|
| Registros | 4298 | 29504 | 14 |
| Número de LUTs | 7314 | 29504 | 24 |
| Número de Slices | 5042 | 14752 | 34 |
| Número de Bloques I/O | 27 | 173 | 15 |
| Número de BUFGMUXs | 1 | 24 | 4 |

La figura 7 muestra la plataforma utilizada en como dispositivo maestro de la red y con la cual se realizaron las pruebas de comunicación, en dichas pruebas se utilizó un dispositivo maestro y varios esclavos enviando diferentes solicitudes de lectura a los datos de un acelerómetro en las posiciones x, y, z. Recibiendo la respuesta sin errores encontrados por el código CRC.

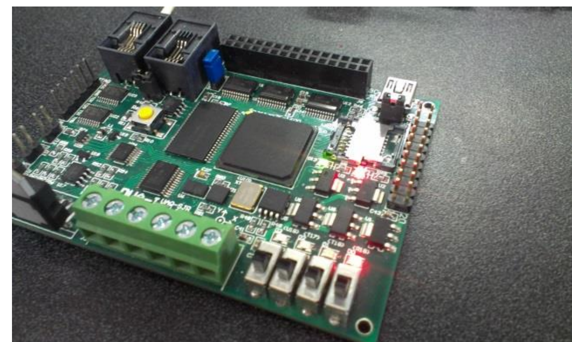
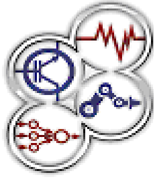


Fig. 7 Plataforma para sistemas embebidos DUA-I

La red implementada opera con un ancho de banda de 7.7 Kb/s por lectura completa de un sensor, que comprende la transmisión de 144 bits de los cuales 24 corresponden al dato del sensor requerido. En caso del acelerómetro equivale al muestreo de los tres ejes (x, y, z) enviando por





cada eje una trama de datos completa, incluyendo dirección, función, dato y CRC.
También es posible la incorporación de esclavos según los requerimientos del proceso a monitorear.
La figura 8 muestra los sensores inerciales incorporados en la tarjeta DUA-I utilizados en este trabajo, el giroscopio es el dispositivo L3G4200D tipo MEMS de 3 ejes, también se empleó el componente LSM303DLHC que es un dispositivo MEMS que integra un acelerómetro de tres ejes, un magnetómetro de tres ejes y un sensor de temperatura.

Fig. 8 Sensores inerciales (acelerómetro y giroscopio)

## Conclusiones.

Este trabajo presenta la implementación de un protocolo de comunicación industrial aplicado a monitoreo inercial con un microprocesador embebido. Se utiliza un FPGA Spartan-3E y la plataforma DUA-I para realizar las pruebas prácticas aprovechando las ventajas de los dispositivos lógicos programables y los IPcore propietarios.
A diferencia de otros trabajos reportados previamente en aplicaciones similares como el realizado por Mrs. Prachi Rane [5], En este trabajo se muestra una comunicación con interfaz RS485 la cual proporciona mayor alcance en la extensión del cableado e inmunidad al ruido en ambientes industriales; además, el sistema embebido permite explotar la alta potencia de cálculo del FPGA y la rapidez del desarrollo de software.
Adicionalmente, cabe resaltar que el uso de dispositivos propietarios permite implementar un sistema a medida y de bajo costo.
Como parte del trabajo futuro, se contempla la creación de un sistema integral que incluye una interfaz gráfica de usuario para la administración de la red.

E. Meneses, et all, "An Event-Triggered Smart Sensor Network Architecture. Industrial Informatics", 5th IEEE International Conference on, 2007, 978-1-4244-0851-1/523-528.

S. Ghosh, et all, "Sensor Network Design for Smart Highways", no. 5, vol. 42 in IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, 2012

P. Rane, "Design of Modbus Controller Using VHDL for Remote Administrations of a Network of Devices", Emerging Trends in Engineering and Technology (ICETET), pp.694 – 697, 2010.

D. Suhartono, "Developing Controller Area Network Management Application Based on Modbus in Multi Generator Set Controller through Local Network and Internet", International Conference on Advances Science and Contemporary Engineering, pp. 426 – 435, (ICASCE). Procedia Engineering, 2012.

Rivera, J., 2008. Improved Progressive Polynomial Algorithm for Self-Adjustment and Optimal Response in Intelligent Sensors. Sensors 2008, 8, 7410-7427; DOI: 10.3390/s8117410.

Lomelí, H. U., 2012. Tesis Desarrollo e implementación de un sistema de giroscopios digitales Usando tecnología FPGA para monitoreo de la orientación de robots. Universidad Autónoma de Querétaro, Facultad de Ingeniería campus San Juan del Rio.

Modicon, "Modbus Protocol Reference Guide", North Andover, Massachusetts, 1996

**Emmanuel Guillén García.**
Ingeniero en Computación egresado de la Universidad Autónoma de Querétaro en 2011. Actualmente se encuentra estudiando para obtener el grado de Maestro en Ciencias en la Universidad Autónoma de Querétaro, Querétaro.

**Luis Morales Velázquez.**
Ingeniero en Electrónica egresado de la Universidad de Guanajuato. Obtuvo el grado de Maestría en Instrumentación y Control además de un Doctorado en la Universidad Autónoma de Querétaro. Es reconocido por CONACYT como investigador.

**J Jesús de Santiago Pérez.**
Dr. J. Jesús de Santiago Pérez. Obtuvo la Licenciatura y Maestría en Matemáticas Aplicadas así como el Doctorado en Ingeniería por parte de la Universidad Autónoma de Querétaro, Actualmente se encuentra trabajando como

profesor investigador en la Facultad de Ingeniería de la Universidad Autónoma de Querétaro.

**Carlos Andrés Pérez Ramírez.**
Ingeniero en Comunicaciones y Electrónica por la Universidad de Guanajuato, Salamanca, México, en el 2013. Actualmente se encuentra estudiando para obtener el grado de Maestro en Ciencias en la Universidad Autónoma de Querétaro, Querétaro.

**Roque A. Osornio Rios.**
Recibió el grado de Ingeniero por parte del Instituto Tecnológico de Querétaro, Querétaro, México. Y los grados de Maestro en ingeniería y Doctor por parte de la Universidad de Querétaro, Querétaro, México, en 2007. Es un investigador nacional CONACYT.