



**Escuela Técnica Superior de Ingeniería**

**Universidad de Huelva**

Grado en Ingeniería Informática

**Trabajo Fin de Grado/Máster**

Modelos VHDL de protocolos de comunicaciones I

Juan Carlos Hidalgo Caballero

[Fecha]

“Don't ever, for any reason, do anything to anyone for any reason ever, no matter what, no matter where, or who, or who you are with, or where you are going, or where you've been... ever, for any reason whatsoever...” Michael scott.

# ABSTRACT

In a world where more and more people are operating remotely, communication protocols are of utmost importance. This is where the content of this paper comes into play in order to understand the low-level layer operation of the MODBUS and CANBUS communication protocols.

MODBUS nowadays is mostly used in industrial environments, in extractive industries and even in home automation installations due to its simplicity and high performance compared to others.

Secondly, we have CANBUS, which is quicker and more robust, with both high fault tolerance and resolution. Due to its reliability, it is used in the automotive, agricultural and military machine industries, among others.

The goal of this project is to get to know both protocols in depth and implement them in non-conventional platforms, such as FPGAs, in order to offer solutions in specific systems that require real-time responses.

# RESUMEN

En un mundo en el que cada vez hay menos cosas que no estén conectada entre sí, tienen muchísima importancia los protocolos de comunicaciones y ahí es donde entra en juego el contenido de este trabajo para comprender el funcionamiento a bajo nivel de los protocolos de comunicaciones MODBUS y CANBUS.

MODBUS es un protocolo que a día de hoy se utiliza más que nada en entornos industriales, en instalaciones extractivas de recursos e incluso en instalaciones domóticas debido a su sencillez y altas prestaciones con comparación con otros.

Por otro lado tenemos CANBUS, que se trata de un protocolo muy veloz y robusto con alta intolerancia a fallos y resolución de estos. Debido a su fiabilidad es usado en la automoción, maquinaria agrícola, vehículos militares, etc.

El objetivo de este trabajo es conocer a fondo ambos protocolos e implementarlos en plataformas no convencionales, como son las FPGA, para ofrecer soluciones en sistemas específicos que requieran respuestas en tiempo real.

AGRADECIMIENTOS

Índice general

[ABSTRACT 4](#_Toc105433501)

[RESUMEN 6](#_Toc105433502)

[Índice de imágenes 11](#_Toc105433503)

[Índice de tablas 12](#_Toc105433504)

[Glosario 13](#_Toc105433505)

[1. Introducción 14](#_Toc105433506)

[Objetivos 14](#_Toc105433507)

[1. MODBUS 19](#_Toc105433508)

[1.2 CAPA FISICA 20](#_Toc105433509)

[1.3 CAPA DE ENLACE 23](#_Toc105433510)

[1.3.1 MODBUS RTU 23](#_Toc105433511)

[1.3.2 MODBUS ASCII 23](#_Toc105433512)

[1.3.3 MODBUR TCP 24](#_Toc105433513)

[1.3.4 TRAMA MOD BUS 24](#_Toc105433514)

[2.3 CAPA APLICACIÓN 28](#_Toc105433515)

[2.4 Diagrama de Estados 29](#_Toc105433516)

[2.5 SIMULACION 35](#_Toc105433517)

[3 CAN-BUS 37](#_Toc105433518)

[3.5 CAPA FISICA 37](#_Toc105433519)

[3.6 CAPA DE ENLACE 39](#_Toc105433520)

[3.6.1 TRAMA DE DATOS 41](#_Toc105433521)

[3.6.2 TRAMA REMOTA 43](#_Toc105433522)

[3.6.3 TRAMA DE SOBRECARGA 44](#_Toc105433523)

[3.6.4 TRAMA DE ERROR 44](#_Toc105433524)

[3.7 Seguridad del protocolo 45](#_Toc105433525)

[3.8 SIMULACIÓN 46](#_Toc105433526)

[4 COMPARACION 47](#_Toc105433527)

[5 CONCLUSIONES 47](#_Toc105433528)

[Bibliografía 49](#_Toc105433529)

# Índice de imágenes

[Ilustración 1 Placa desarrollo ARTY A7 15](#_Toc105442013)

[Ilustración 2 Placa de desarrollo Ardunio UNO 16](#_Toc105442014)

[Ilustración 3 Interfaz RS485 16](#_Toc105442015)

[Ilustración 4 Módulo MCP2515 17](#_Toc105442016)

[Ilustración 5 Protoboard 17](#_Toc105442017)

[Ilustración 6 Cables de protoboard 18](#_Toc105442018)

[Ilustración 7 USB tipo B 18](#_Toc105442019)

[Ilustración 8 USB tipo A 19](#_Toc105442020)

[Ilustración 9 Kit elegoo 19](#_Toc105442021)

[Ilustración 10 Ejemplo sistema MODBUS 20](#_Toc105442022)

[Ilustración 11 Conexión Daisy-chain 21](#_Toc105442023)

[Ilustración 12 Niveles de tensión línea MODBUS 22](#_Toc105442024)

[Ilustración 13 Comunicación Full-duplex 22](#_Toc105442025)

[Ilustración 14 Comunicación Half-duplex 23](#_Toc105442026)

[Ilustración 15 Diagrama de estados del envío protocolo serie VHDL 31](#_Toc105442027)

[Ilustración 16 Diagrama de estados del mensaje 32](#_Toc105442028)

[Ilustración 17 Diagrama de estados del recibo de respuesta 33](#_Toc105442029)

[Ilustración 18Diagrama de estados de envío de respuesta 34](#_Toc105442030)

[Ilustración 19Diagrama de estados recibo de petición 35](#_Toc105442031)

[Ilustración 20 Conexiones prototipo MODBUS 37](#_Toc105442032)

[Ilustración 21 Reducción cableado en un vehículo con CANBUS 38](#_Toc105442033)

[Ilustración 22 Topología CANBUS 38](#_Toc105442034)

[Ilustración 23 Niveles de Tensión Línea CAN 40](#_Toc105442035)

[Ilustración 24Trama CAN de datos 44](file:///F:\PARA%20ENTREGA\Memoria2.docx#_Toc105442036)

[Ilustración 25 Trama CAN remota 44](#_Toc105442037)

[Ilustración 26 Conexiones Prototipo CANBUS 47](#_Toc105442038)

# Índice de tablas y figuras

Tabla 1 Valores de tensión MODBUS …………………………………………………………………………………….20

Tabla 2 Trama MODBUS………………………………………………………………………………………………….…….23

Tabla 3 Instrucciones MODBUS……………………………….…………………………………………………………….24

Tabla 4 Pseudocódigo LRC……………………………………………………………………………….…………………….25

Tabla 5 Ejemplo de Trama MODBUS…………………………………………………………………………………..….25

Tabla 6 Ejemplo falla LRC……………………………………………….15

Tabla 1 Trama MODBUS…………………………………………………………….15

# Glosario

MODBUS Modicon Bus

CAN Controled Area Network

LRC Longitudinal Redundance check

CRC cyclic redundance check

# Introducción

La importancia de la industria y los automóviles son innegables en nuestra sociedad, no la podemos concebir sin estos dos pilares debido a la riqueza económica, cultural y social que han aportado al mundo sobre todo en los últimos 50 años. Tanto es así que si echamos un vistazo al mercado podemos ver un incremento bastante sustancial en la cantidad de dispositivos controlables por el usuario desde la consola central de los vehículos, ajuste de frenada y suspensiones, comportamiento del motor a distintos regímenes de uso… y todo ello digitalmente y desde un único punto. Algo parecido sucede en empresas metalúrgicas, fabricas refinerías, etc. Que ya todo se controla desde una única sala

Al principio las empresas automovilísticas construían sus vehículos con dispositivos mecánicos y electro-mecánicos y las empresas del entorno industrial (minería, metalurgias, petroleras, fabricas…) tenían que conseguir sus objetivos también con el uso de dispositivos mecánicos, pero con la llegada de estos dos protocolos todo se revolucionó, ya que estos protocolos les permitían tener un mayor control de la producción, de la seguridad, de la logística… haciéndolo todo mucho más eficiente.

Por un lado tenemos MOD-BUS que fue desarrollado por Modicon en 1979 para sus PLCs conectados en serie, pero que se convirtió en un estándar en la industria debido a la facilidad de implementación y desarrollo, está pensado desde el principio para aplicaciones industriales en las que se necesitaba conocer el estado de muchos sensores y controlar diferentes actuadores desde un único punto sin la necesidad de desplazarse físicamente al lugar donde estos están situados.

El otro lugar que nos ocupa es CAN-BUS, inventado por BOSH en 1982 pero que fue lanzado oficialmente en 1986 en el congreso de la Sociedad de Ingenieros Automotrices en Detroit. Este protocolo nació como una solución para la comunicación entre varias CPUs pudiéndose convertir en un sistema multinúcleo según la configuración que queramos poner. El funcionamiento de los vehículos actuales sería muy diferente si este protocolo no existiera, ya que permite a la ECU tener conocimiento de todo lo que pasa en el vehículo, desde ABS electrónico o controles de tracción hasta la administración del combustible o las baterías de una manera mucho más eficiente a como se hacía antes.

## Objetivos

Los objetivos de este trabajo son los siguientes:

1. Comprender el funcionamiento de ambos protocolos de comunicación conociendo las diferencias entre ellos y por qué se usan en ámbitos distintos.
2. Codificación de ambos en VHDL y C++ para construir un sistema Maestro-Esclavo en el caso del protocolo MOD BUS y un sistema multimaestro para CANBUS y comprobar el funcionamiento de ambos.

Para alcanzar dichos objetivos voy a contar con los siguientes materiales que servirán para demostrar el funcionamiento de ambos protocolos.

PLACA DE DESARROLLO ARTY A7

Hará las funciones de maestro en nuestro sistema, dará las instrucciones y escuchará las respuestas con los valores de los sensores y actuadores. Se trata de una FPGA con un reloj de 100 Mhz. De esta placa haremos uso de los pines de entrada y salida del shield de arduino que tiene incorporado para comunicarnos con el resto de nodos de la red, los pulsadores, los switches y los leds.

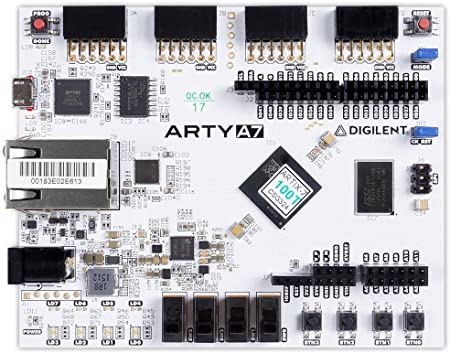


Ilustración 1 Placa desarrollo ARTY A7

PLACA DE DESARROLLO ARDUINO ATMEGA 328 UNO

Será el esclavo de nuestros prototipos, usará sus pines analógicos y digitales para ver el estado de los sensores y regular a los actuadores y para comunicarse con el maestro.



Ilustración 2 Placa de desarrollo Ardunio UNO

INTERFACES RS485

Componente que se encarga de dar a los valores lógicos sus correspondientes niveles de tensión en el bus de comunicación.

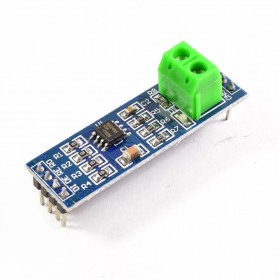


Ilustración 3 Interfaz RS485

Módulo MCP2515

Este se trata de un componente que nos va a facilitar las tareas correspondientes a la capa física y la capa de enlace del protocolo CAN.



Ilustración 4 Módulo MCP2515

PROTOBOARD

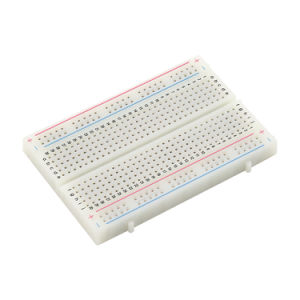


Ilustración 5 Protoboard

Cables para protoboard:



Ilustración 6 Cables de protoboard

Cable USB tipo B

Cable que vamos a usar para dar corriente al Arduino y para leer lo que le llega por el puerto serie



Ilustración 7 USB tipo B

Cable USB tipo A

Cable que vamos a usar para dar corriente a la FPGA, para preparar las peticiones y para comprobar si manda bien los mensajes.



Ilustración 8 USB tipo A

Kit Elegoo

Conjunto de sensores y actuadores que simularán un entorno industrial o algún componente electrónico de un vehículo



Ilustración Kit elegoo

# MODBUS

Como ya he comentado en la introducción MODBUS se trata de un protocolo creado por Modicon (que hoy día es Schneider Electric) en 1979 para su gama de PLC’s por líneas serie, pero que se terminó convirtiendo en un estándar en las comunicaciones en procesos industriales. Este éxito e impacto del protocolo se debe a varios factores. El primer motivo y más importante es que es abierto, cualquiera puede copiar el código y ajustarlo a sus necesidades e incluso comercializarlo sin tener que pagar los derechos de autor a Schneider Electric. Otro de los motivos es que la red se monta sobre protocolos serie RS232/RS485/RS422, que son muy sencillos de entender, rápidos y no suponen un costo considerable en los proyectos. Si un nodo esclavo falla no se altera el funcionamiento del resto de nodos.

Un sistema que se monte sobre MODBUS consta de dos elementos principales, uno que solicita la información (maestro) y otro que capta la petición y responde (esclavo). Tenemos por tanto un sistema Maestro-Esclavos que nos va a permitir centralizar todo el proceso desde un único punto físico del complejo industrial ahorrando a la empresa tiempo, dinero y ganando en seguridad para los trabajadores ya que no tendrán que desplazarse hacia un ventilador extractor de gases y polvo para cambiar su velocidad por ejemplo con el peligro que eso supone.

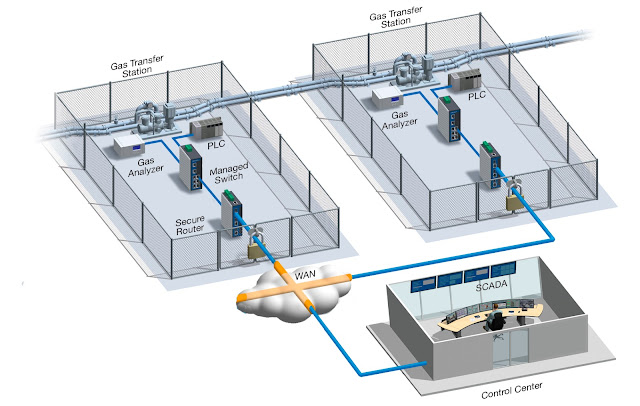


Ilustración 10 Ejemplo sistema MODBUS

Aun así el uso de MODBUS no solo se ciñe a entornos industriales, es más, cada dia se incrementan el número de hogares que incluyen este protocolo en sus instalaciones domóticas ya que la mayoría de los fabricantes construyen sus componentes pensando en la compatibilidad con MODBUS pudiendo combinar sus ventajas con las ventajas de otros protocolos como PROFIBUS que también es muy usado en plantas de biomasa, petroquímicas, etc.

## CAPA FISICA

Lo primero que hay que decidir es cuál es el medio más adecuado para nuestro proyecto ya que MODBUS nos ofrece varias posibilidades entre ellas las más importantes y comunes son Ethernet y por puerto serie con RS232. Para este proyecto se ha decidido utilizar RS485 porque RS232 está pensado para conexiones punto a punto, no para sistemas con más de 2 nodos. Además una de las principales características por las que MODBUS se ha vuelto tan popular es porque usa una topología en bus, la norma TIA-485 recomienda Daisy-chain como se puede observar en la Ilustración 10, esto es que todos los terminales van conectados a la misma línea permitiendo reducir cableado, simplificando al máximo el diseño y mejorando el rendimiento de los componentes que forman la red y su modularidad.

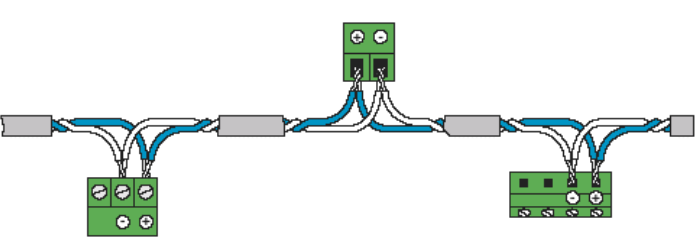


Ilustración 11 Conexión Daisy-chain

Esta línea que acabamos de nombrar, sobre la que van todos los componentes tiene que cumplir una serie de requisitos para que no haya interferencias o retardos producidos por todos los aparatos eléctricos que puede haber en el entorno en que se está montando el sistema que suelen ser naves industriales, instalaciones mineras, petroleras, etc. Los cables de la línea deben estar apantallados y trenzados para evitar esas interferencias producidas por campos electromagnéticos y además las señales son diferenciales para añadir todavía más robustez al sistema.

La especificación TIA-485 establece que una señal corresponderá al 1 binario cuando la diferencia de tensiones entre la línea A y la línea B sea menor que -200mV, (Voa–Vob < −200 mV), mientras que el 0 lógico, por otro lado, se dará cuando la diferencia de tensiones sea mayor que -200 mV, (Voa–Vob > −200 mV). Con esta regla nuestra señal tendrá la siguiente forma según se encuentre en estado ‘1’ o ‘0’.

|  |  |
| --- | --- |
| Valor lógico | Diferencia de tensión (Voa - Vob ) |
| 0 | > -200mV |
| 1 | < -200mV |

*Tabla 1 Valores de Tensión MODBUS*

Durante la trasmisión de información los cambios de los valores es una línea RS485 se ven como en la *Ilustración 11.*

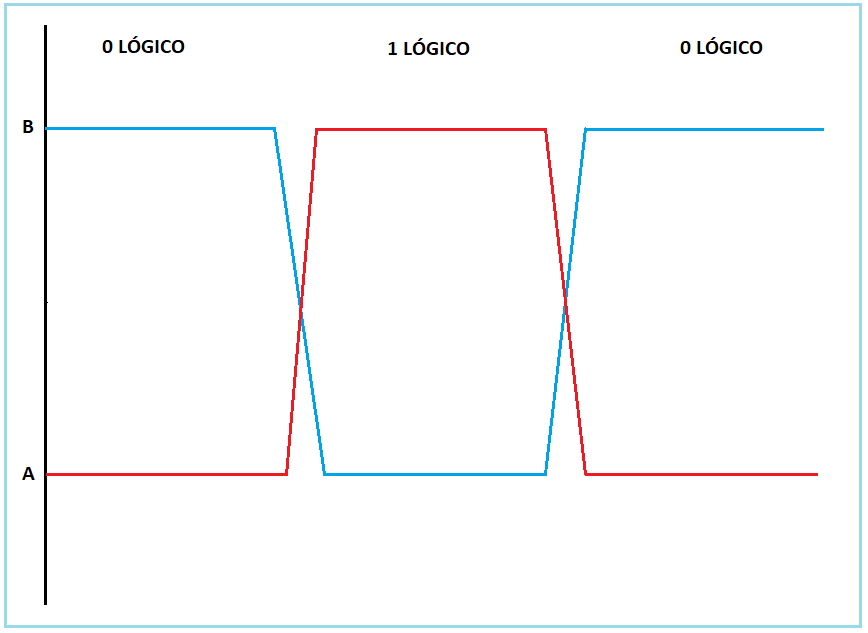


Ilustración 12 Niveles de tensión línea MODBUS

La longitud máxima del cable que se permite en una comunicación sobre RS485 es de 1200 metros lo que nos permitiría trasmitir a una velocidad de 100kbits/s. Según la especificación, para calcular la velocidad de trasmisión máxima en función de la distancia, el producto resultante de multiplicar la distancia (en metros) por la velocidad (en bits/s) debe ser menor que 10^8.

El límite de terminales que se permiten conectar a la red con módulos rs485 es de 32 dispositivos y tendremos que tener en cuenta este dado a la hora de diseñar la red.

Para este trabajo la comunicación va a ser Half-Duplex por lo que solo necesitaríamos una línea con un par (A, B). Si las necesidades del usuario requirieran Full-Duplex tendríamos que añadir una línea más al sistema y 2 interfaces RS485 por cada nodo. La *Ilustración 12* y la *Ilustración 13* muestran cómo serían las conexiones para ambos modos de comunicación. Con RS232 si sería posible tener una conexión Full-Duplex sin necesidad de añadir componentes pero teniendo en cuenta que con RS232 solo podríamos tener 2 nodos.

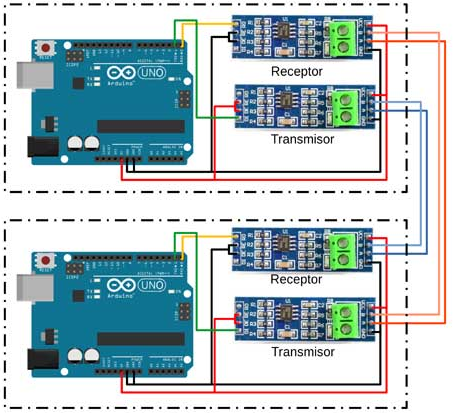


Ilustración 13 Comunicación Full-duplex

*ma*

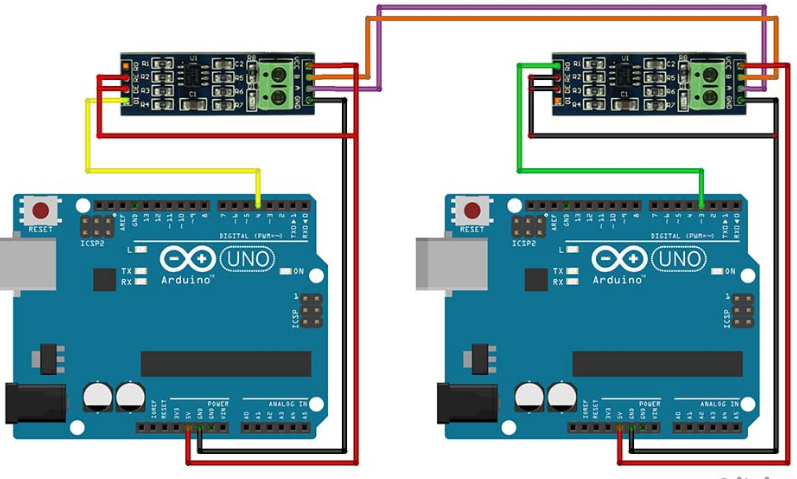


Ilustración 14 Comunicación Half-duplex

## CAPA DE ENLACE

Esta es la parte en la que más nos vamos a focalizar ya que en esta capa se encuentran los mecanismos necesarios para iniciar y finalizar el mensaje, tener un control del origen y el destino de este, calcular el código de verificación de integridad del mensaje y de guardar todos los datos que nos lleguen en distintos registros para su posterior recopilación por la capa de aplicación.

Antes de seguir hay que definir con que versión del protocolo nos vamos a quedar y en función de nuestra decisión las tramas tendrán un contenido u otro. Estamos hablando de los 3 principales modos que tiene MODBUS:

### MODBUS RTU

Se trata de la versión más extendida, utilizada por diferentes softwares de adquisición de datos como son SCADA y HMI, que se tratan de los programas de adquisición y gestión de datos más populares del entorno industrial.

Este modo funciona sobre protocolo serie como puede ser la línea que forma una red sobre RS485.

En este modo la información será tratada puramente en binario y es de vital importancia incluir un campo en la trama que nos sirva para detectar errores en la comunicación. En el caso de MODBUS RTU se utilizará CRC (Código de Redundancia Cíclica).

### MODBUS ASCII

Es bastante similar a MODBUS RTU, las principales diferencias son que los dispositivos en vez de manejar información puramente en binario, se utilizan caracteres ASCII para manejar los datos de los mensajes y que el método de comprobación de errores que hay que utilizar es LRC (verificación de redundancia longitudinal).

Al igual que en MODBUS-RTU, aquí el medio se trata de un bus serie y tendremos que hacer uso de RS485.

Este va a ser el modo que se ha elegido para el desarrollo del prototipo porque hace que la codificación sea más sencilla.

### MODBUS TCP

Esta es una implementación más reciente que utiliza una red TCP/IP conectándose a una red por el puerto 502. Como las capas inferiores de TCP ya proporcionan su propio checksum (método de comprobación de errores e integridad) no es necesario que tengamos un campo de comprobación de errores en nuestra trama, ni LRC ni CRC. Es la versión que más se está extendiendo porque permite tener muchos más dispositivos y que sea más sencillo la incorporación de estos a la red, permite el monitoreo y control en tiempo real de todo el sistema, etc.

### TRAMA MODBUS

La trama especificada por Modicon, y que será en la que nos basaremos para este proyecto, para MODBUS ASCII es la siguiente:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INICIO | DESTINO | CÓDIGO FUNCION | DATOS | LRC | FIN |
| 1 carácter | 2 caracteres | 2 caracteres | n caracteres | 2 caracteres | 4 caracteres |

*Tabla 2 Trama MODBUS*

1. Inicio: Las tramas empiezan con el carácter ASCII “:” que en hexadecimal corresponde a “3A” y este carácter no podrá ser utilizado para nada más.
2. Destino: Dirección de esclavo destino. En MODBUS las direcciones de los esclavos van desde la 1 hasta la 247 (aunque hay que tener en cuenta que por las limitaciones físicas de la red ,mencionadas en el capítulo 1.1 de este documento, solo podremos conectar 32 dispositivos). Para enviar una petición el maestro coloca la dirección del esclavo a quien va dirigida la petición y cuando el esclavo responda colocará su propia dirección, asó el maestro podrá saber de qué esclavo viene el mensaje. La dirección 0 está reservada para mensajes tipo broadcast y este mensaje no tiene asociada ninguna respuesta por parte de los esclavos, incluso depende del tipo de MODBUS implementado es posible que no admita mensajes broadcast. En nuestra configuración la dirección tiene que estar comprendida entre 0x00 y 0XF7.
3. Código Función: Este campo contiene las distintas instrucciones que ejecutaran los esclavos. Según la función recibida se realizarán operaciones sobre registros o sobre bits (coils). Estas son las funciones imprescindibles en todas las distintas versiones del protocolo pero se pueden implementar tantas como sean necesarias según las especificaciones concretas de la versión o las necesidades del usuario.

|  |  |  |
| --- | --- | --- |
| Código Función (HEX) | Acción | Nombre de la tabla |
| 01 | Lectura múltiple | Salidas Discretas |
| 02 | Lectura múltiple | Entradas Discretas |
| 03 | Lectura múltiple | Salidas Analógicas |
| 04 | Lectura múltiple | Entradas Analógicas |
| 05 | Escritura simple | Salidas Discretas |
| 06 | Escritura simple | Salidas Analógicas |
| 15 | Escritura múltiple | Salidas Discretas |
| 16 | Escritura múltiple | Salidas Analógicas |

*Tabla 3 Instrucciones MODBUS*

Según el código de función que aparezca en el mensaje, los esclavos tendrán que hacer operaciones de lectura o escritura en las tablas de registros que tienen almacenadas:

* Salidas Discretas: Contienen los valores que se asignan a señales de salida del sistema que pueden tener valor 0 o 1.
* Entradas Discretas: almacena valores que provienen de señales de entrada de sensores que envían o 0 o 1.
* Salidas Analógicas: Se almacenan valores que recogerán los actuadores del sistema como puede ser un led con distintas intensidades, un servomotor, etc.
* Entradas Analógicas: Esta tabla guarda valores analógicos que provienen de sensores como un sensor de distancia o de humedad por ejemplo.

Estos códigos serán enviados únicamente por el maestro ya que el esclavo al que vaya dirigido el mensaje, cuando vaya a responder colocará un 1 en el bit más significativo conforme se vayan encontrando errores.

Cuando el esclavo responda colocará en este campo otros códigos que indican si ha habido errores colocando un ‘1’ en el bit de más peso, a continuación el código de función y en el campo DATOS se pondrá el código de error, y si no se producen errores se introduce el mismo código de función que le llegó.

1. DATOS: Del contenido de este campo se encarga la capa de aplicación y se explicará más adelante en este documento, por ahora solo hace falta saber que el contenido dependerá tanto del tipo de instrucción como la naturaleza del dispositivo, si es maestro o esclavo.
2. LRC: Son las siglas en inglés de verificación de redundancia longitudinal y es el código generado en MODBUS ASCII para que los terminales comprueben la integridad del mensaje. Concretamente LRC se trata de una verificación que se calcula usando el siguiente algoritmo que no es más que hacer el complemento a dos de la suma de todos los bytes del mensaje sin tener en cuenta el byte de INICIO y los dos bytes de FIN:

LRC = 0

PARA CADA BYTE B HACER

LRC = LRC + B

FIN PARA

LRC = LRC XOR 0x"FF"

LRC = LRC + 1

*Tabla 4 Pseudocódigo LRC*

A continuación y con la finalizad de entenderlo mejor se va a realizar la traza del cálculo de un LRC de 8 bits para una trama de petición del maestro. El protocolo especifica que el tamaño del campo LRC es de 16 bits y así es como se ha implementado en este trabajo pero para este ejemplo se han reducido los tamaños para que sea más sencillo su entendimiento:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INICIO  8 bits | DESTINO  8 bits | CÓDIGO FUNCION  8 bits | DATOS  n x 8 bits | LRC  8 bits | FIN  16 bits |
| 3A | 0A | 01 | 04 A1 00 01 | ------------- | 0D 0A |

*Tabla 5 Ejemplo de trama MODBUS*

Los bytes de inicio y de fin los ignoramos para el cálculo del LRC.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

LRC ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

0A ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

LRC = LRC + 0A ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

01 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

LRC = LRC + 01 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

04 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

LRC = LRC + 04 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

A1 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

LRC = LRC + A1 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

00 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

LRC = LRC + 00 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

01 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

LRC = LRC + 01 ->

FIN BUCLE

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

FF ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

LRC = LRC XOR FF->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

LRC = LRC + 1 ->

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

LRC ->

LRC= 0x”4F”

El maestro colocará el valor “4F” en el campo LRC y el esclavo tiene que comparar el LRC que le ha llegado del maestro con el que el mismo ha generado al leer el mensaje. Si ambos coinciden entonces procede a ejecutar la instrucción del maestro, en caso contrario el esclavo no ejecuta ninguna acción y envía un mensaje con el código de error correspondiente a fallo en LRC y el maestro tiene que responder con el mensaje anterior.

Aun así la comprobación mediante LRC tiene un problema y es que distintos mensajes pueden dar lugar al mismo código y por lo tanto si en un mensaje se produjeran 2 errores cabría la posibilidad de que el sistema lo dé por válido. Las siguientes tuplas son un ejemplo de este problema que tiene este método. El maestro puede enviar el dato 61A8 y durante la trasmisión justamente los bits de menos peso de cada byte cambian de valor y dan una lectura distinta sin embargo el cálculo del LRC sigue dando el mismo resultado con lo que el esclavo dará por válida la información.

|  |  |  |
| --- | --- | --- |
| Binario (dato enviado) | Hexadecimal | LRC (Hexadecimal) |
| 01100001 10101000 | 61A8 | 20 |
| Binario (dato recibido) | **Hexadecimal** | **LRC (Hexadecimal)** |
| 01100000 10101001 | 60A9 | 20 |

*Tabla 6 Ejemplo falla LRC*

1. Campo Fin: Todas las tramas en MODBUS ASCII terminan con un retorno de carro y alimentación de línea cuyos caracteres ASCII corresponden con el par de bytes “0D” y “0A” respectivamente.

En esta capa también se ha incorporado un mecanismo para detectar destinos inalcanzables que podrían suceder o porque nos hemos equivocado seleccionando al esclavo y hemos seleccionado uno que no está en nuestro sistema, o porque el esclavo al que queremos acceder se ha caído. Esto lo vamos a hacer con un temporizador de respuesta, si el esclavo no responde en menos de 1 segundo y medio el maestro volverá a hacer la petición, así hasta un máximo de 3 veces. Finalizados los 3 intentos si no hay respuesta, el maestro desiste.

## 2.3 CAPA APLICACIÓN

Esta capa se encarga de gestionar el contenido del campo DATOS de la trama, o dicho de otra manera, la información que tramita esta capa va encapsulada en el campo DATOS de la trama. En caso de estar trabajando con MODBUS-Ethernet también entraría aquí la parte de servicios de red.

Esta capa tendría que ser completada con una aplicación de escritorio por ejemplo en el que se nos vayan mostrando los valores de los sensores y actuadores en tiempo real pero como el trabajo no se centra en esta parte vamos a usar un terminal serie para ver y comprobar los valores.

Para nuestra implementación podremos tener los siguientes valores dependiendo tanto de la instrucción que se trate como de quien produce el mensaje, si es un maestro o un esclavo.

Dirección de Memoria: dirección de memoria en la que empieza el dato o datos que queremos leer o modificar.

Cantidad de registros o coils: contiene el número de datos a los que queremos acceder

Contador de bytes: contiene el número de bytes de datos que hay a continuación que hay que leer.

Data: podrá tener o bien el valor, o los valores, que queremos escribir sobre uno o varios registros o bien los valores que se han leído de dichos registros.

## 2.4 Diagrama de Estados

El apartado que nos ocupa ahora va a explicar cómo se ha procedido con la lógica interna del protocolo. Primero se verán los 3 procesos principales del maestro y después cómo los esclavos envían y reciben la información respectivamente.

Para la parte del maestro realmente hay más procesos que sirven para gestionar el reloj del sistema, adquirir los datos desde el puerto serie o el proceso que vamos a usar para comprobar la respuesta del esclavo pero son completamente independientes del protocolo

Maestro:

* *Diagrama 1* (protocolo serie MAESTRO): Este es el diagrama de flujo del protocolo serie. Inicialmente estamos en el estado WAITING y para comenzar la comunicación tendremos que pulsar y soltar el botón de envío, momento en el que nos vamos al estado START que es donde se envía el bit de inicio. Posteriormente, en el estado DATA, enviaremos bit a bit el dato almacenado en el registro char\_tx (cuyo valor es modificado en otro proceso).Cuando el ultimo bit de datos se haya trasmitido colocamos el bit de STOP en el bus y se selecciona cual va a ser el siguiente byte cambiando el valor de la variable tx\_char\_counter. Cuando se envíe el último byte del mensaje saltamos al estado RESPONSE\_DELAY, en el que vamos a esperar durante un tiempo preestablecido la respuesta del esclavo, si la respuesta no llega trasmitimos el mensaje de nuevo, así hasta un número concreto de intentos. Tanto si hay respuesta como si no, volveremos al estado inicial WAITING. Ilustración 15.
* *Diagrama 2* (Datos a enviar y cálculo de LRC): Los cambios de estado de este diagrama vienen condicionados por el diagrama anterior ya que cambiaremos según el valor de la variable tx\_char\_counter. Además en los bytes que procedan, se calculará el valor LRC para enviarlo cuando llegue el momento. La secuencia siempre va a comenzar asignándole a char\_tx el valor de la constante INICIO que es x”3A”. Siguiendo después por la dirección del esclavo al que acceder, la instrucción que queremos ejecutar, la dirección de memoria sobre la que queremos ejecutar la instrucción, los datos que queramos escribir si es que es el caso y por último el LRC y los bytes de fin. . Ilustración 16.
* *Diagrama 3* (Recepción de datos en el maestro): En este diagrama se detalla el funcionamiento del protocolo serie para la recepción de datos, y almacenamiento de estos en sus correspondientes registros. Inicialmente estamos en IDLE y permanecemos aquí mientras estemos recibiendo un ‘1’. Cuando llegue un 0 al pin TX saltamos al estado START y esperaremos un cierto número de ciclos de reloj para colocarnos en la mitad de la recepción del bit y asegurarnos que no tomamos un valor erróneo. Pasado este tiempo llegamos al estado DATA en el cual, guardaremos en un registro auxiliar llamado char\_rx el byte que nos lega por el puerto serie. Al terminar de recibir el byte legamos al estado STOP donde guardaremos el byte en un registro que será distinto cada vez, según el valor de rx\_char\_counter. Tras esto, llegamos al estado CLEANUP que es para dejar un ciclo del protocolo serie de espacio para después volver al estado inicial IDDLE. Ilustración 17.

Esclavo:

* Diagrama 4 (Envío de respuesta del Arduino): La respuesta se produce cuando tras leer la instrucción no se han detectado errores. Mientras no se termine de enviar el mensaje ponemos a ‘1’ el pin 2 que es el pin de activación del módulo RS485. Después se envían los bytes que ha solicitado el maestro uno a uno calculando a su vez el LRC. Cuando enviemos el último Byte, la variable de control FIN\_ENVIO se pone a TRUE y el bucle finaliza poniendo al Arduino en un estado de escucha del canal esperando nuevos mensajes. Ilustración 18.
* Diagrama 5 (Recepción de datos en el esclavo): Es el proceso principal del Arduino, utilizamos la función serialEvent que se ejecuta automáticamente cada vez que recibe algún dato por el puerto serie. Empezamos en el estado InicioLeerTrama y si el dato que nos llega es x”3A” significa que está llegando un mensaje nuevo. En el siguiente estado comprobamos que el mensaje va destinado a ese esclavo concreto, tanto si es porque la dirección coincide con la propia del esclavo como si la dirección es 0. El resto de la ejecución es similar al procedimiento del diagrama 4, vamos leyendo los datos, calculando el LRC y preparando los valores que enviarán en la respuesta. Ilustración 19.

PROTOCOLO SERIE DEL MAESTRO PARA EL ENVÍO DE DATOS

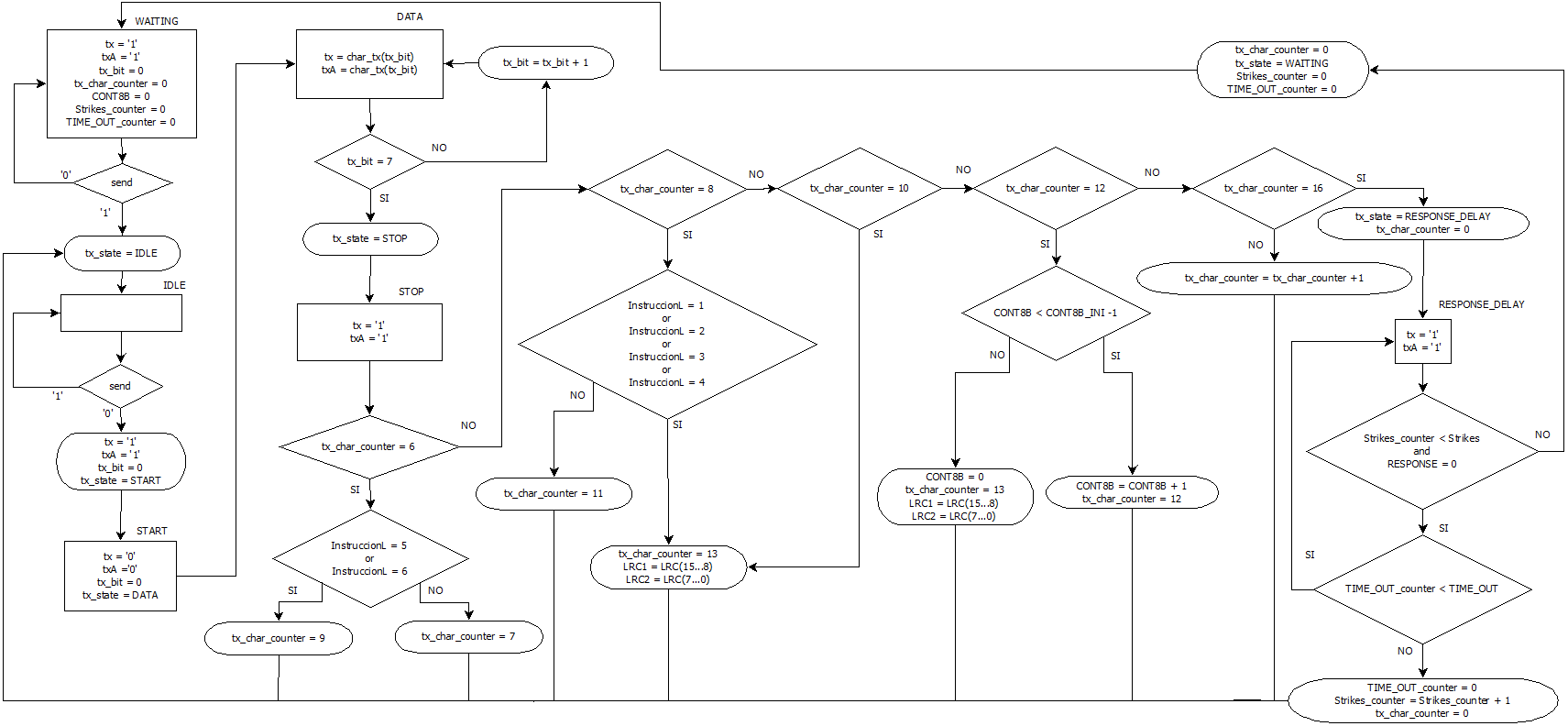


Ilustración 15 Diagrama de estados del envío protocolo serie VHDL

MÁQUINA DE ESTADOS DEL MAESTRO QUE DECIDE EL VALOR QUE SE VA A ENVIAR POR EL PUERTO SERIE Y CALCULA EL LRC PARA EL ENVÍO.

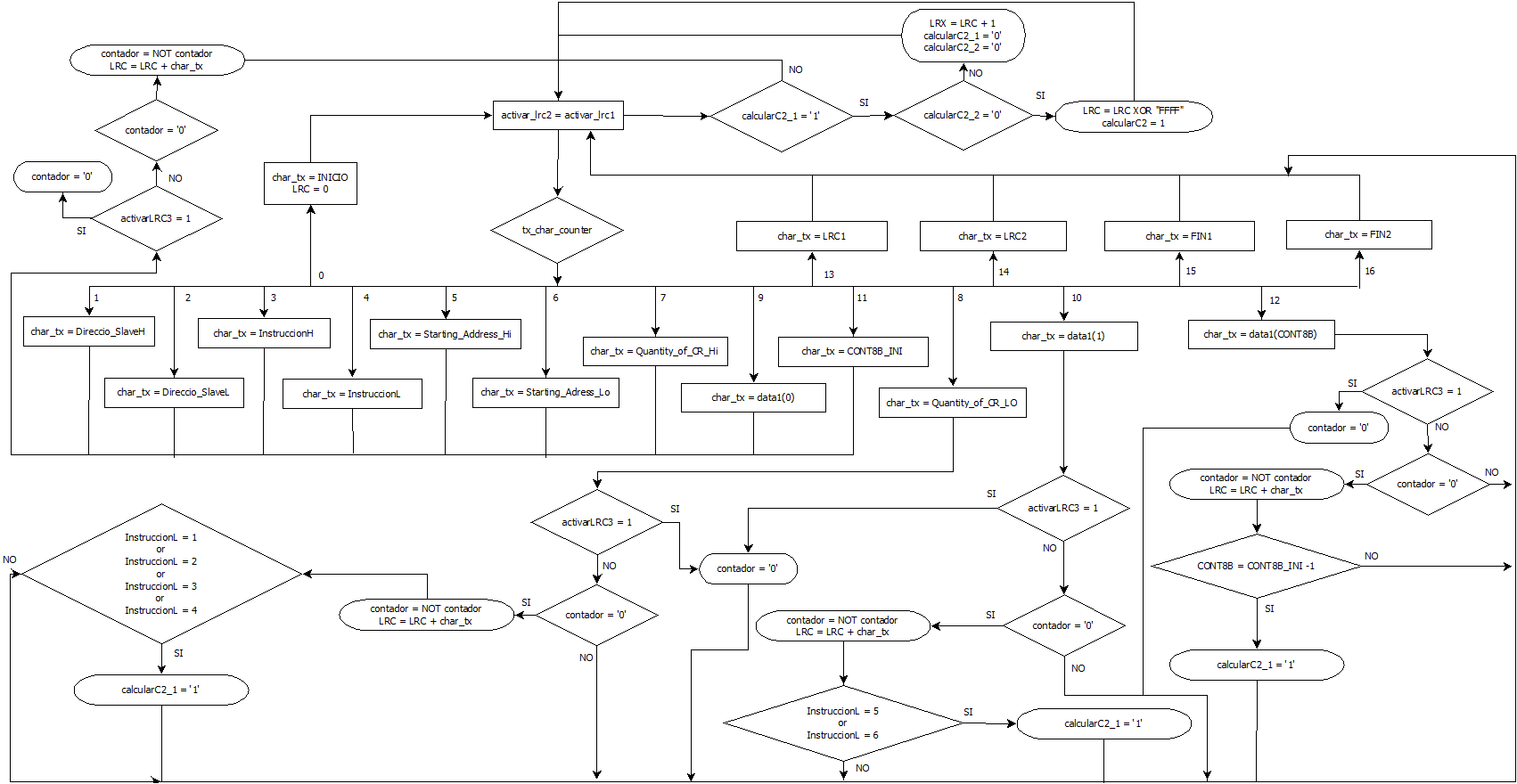


Ilustración 16 Diagrama de estados del mensaje

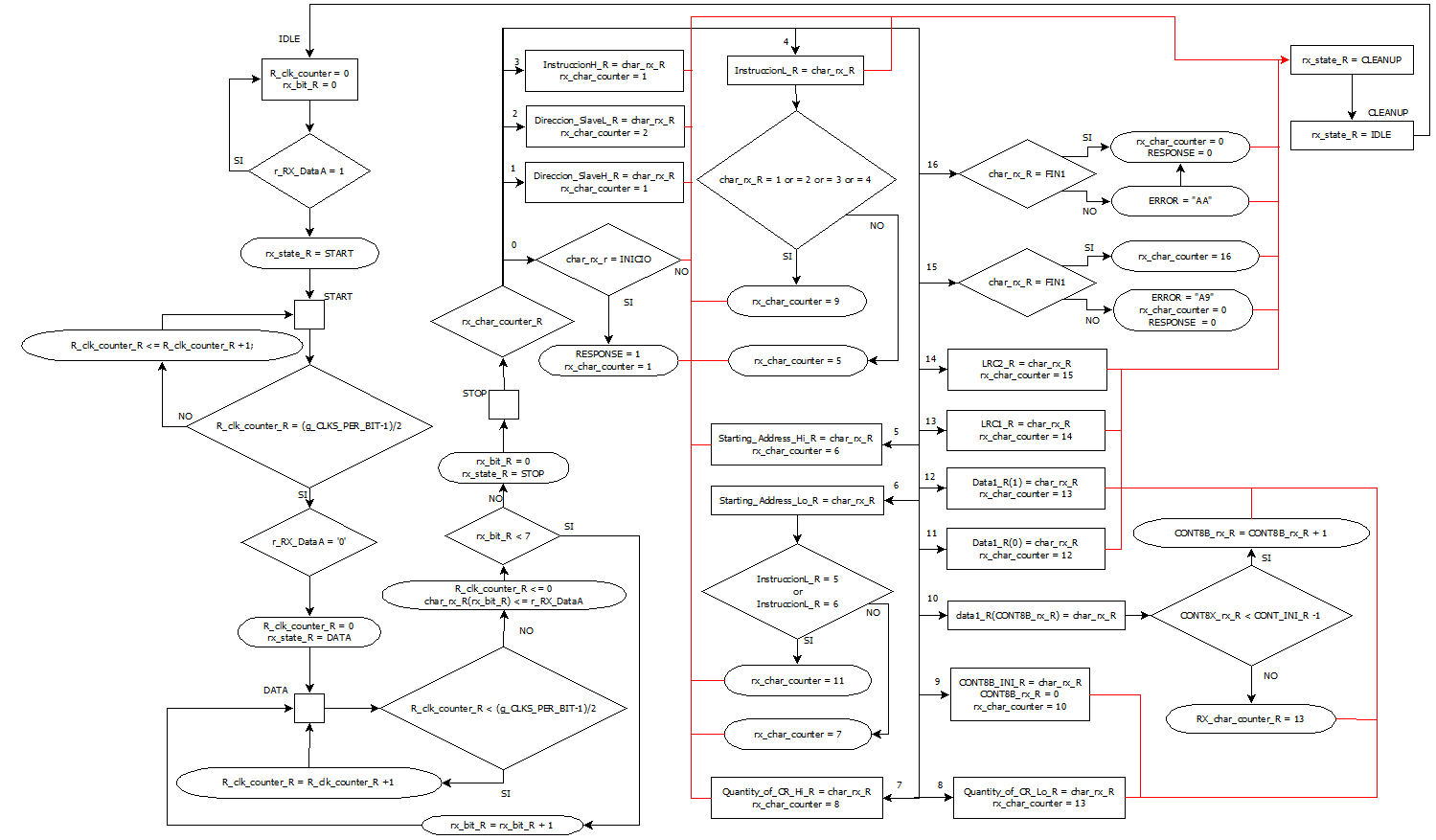
RECEPCION DE DATOS EN EL MAESTRO

Ilustración 17 Diagrama de estados del recibo de respuesta

ENVÍO DE RESPUESTA DEL ESCLAVO.

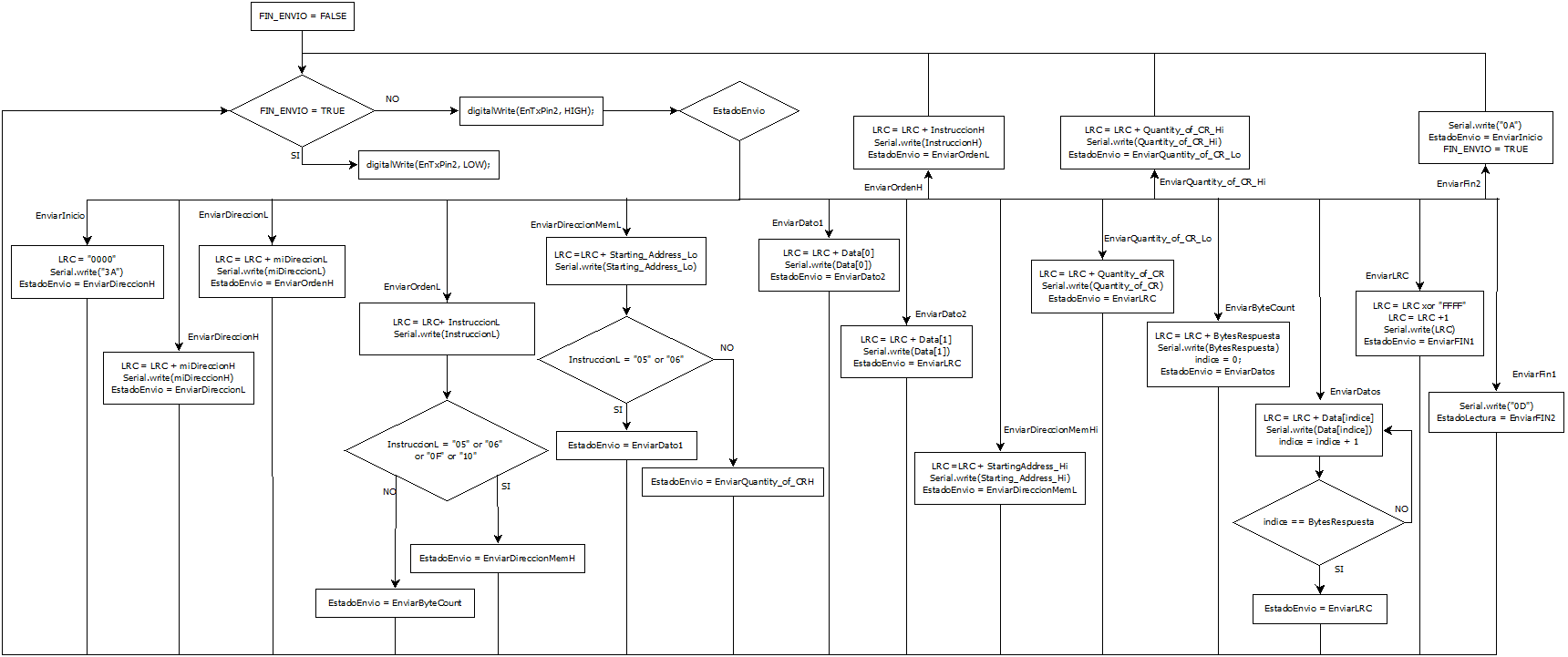


Ilustración 18Diagrama de estados de envío de respuesta

RECIBO DE DATOS EN EL ESCLAVO

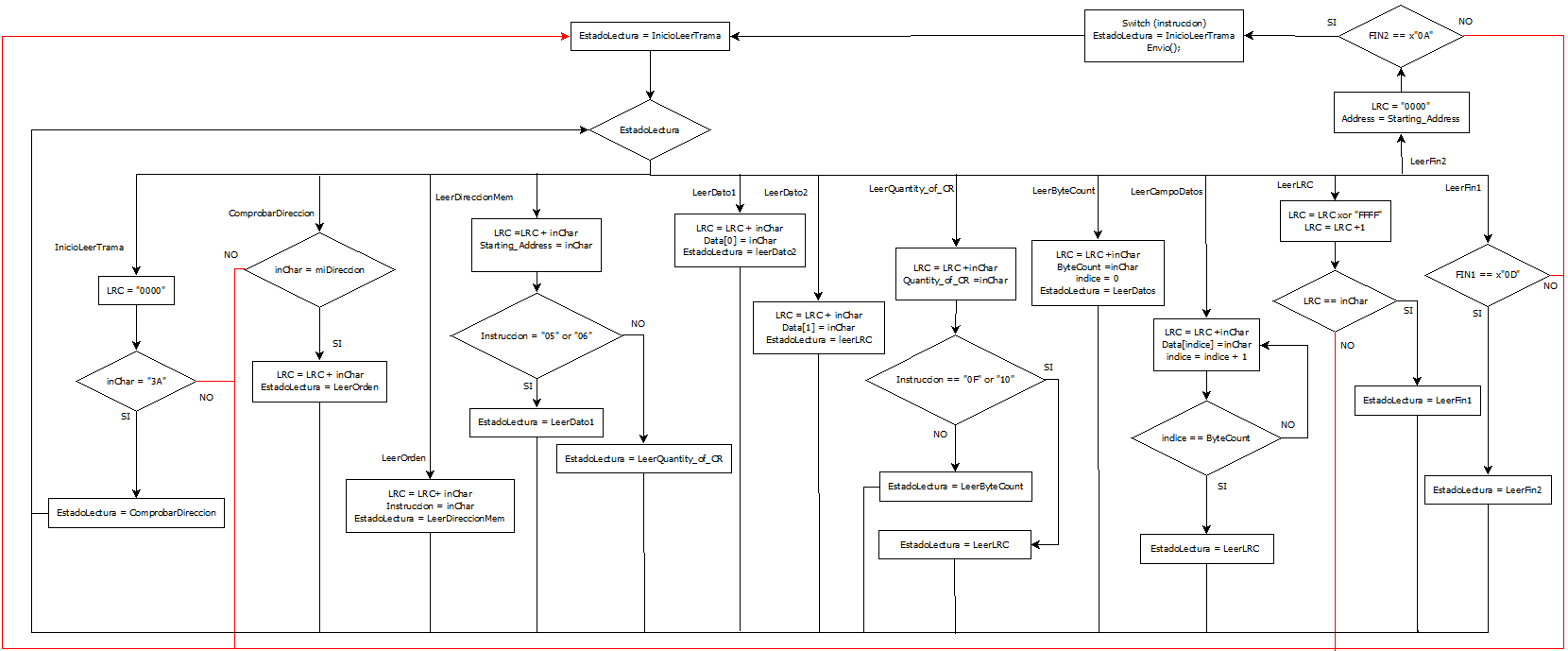


Ilustración 19Diagrama de estados recibo de petición

## SIMULACION

Ahora que ha quedado explicado el funcionamiento del protocolo podemos proceder a ejecutar simulaciones en el prototipo que se ha montado.

Este es el esquema físico de conexiones, y para realizar las pruebas se realizaran diferentes peticiones tanto de lectura de sensores como escritura en actuadores y podremos comprobar si el funcionamiento es correcto mirando los leds de la FPGA.

Necesitaremos tambien tener conectada nuestra FPGA al ordenador porque el primer paso es comunicarle por puerto serie que es lo que queremos hacer. Una de las mejores herramientas para realizar esta tarea es Realterm, que es un monitor de puertos serie que nos permite escribir nuestros propios valores que seran los que recogera la FPGA para su posterior envío.

Abrimos el entorno y lo primero que tenemos que hacer es seleccionar por que puerto vamos a comunicarnos, en mi caso la FPGA es el puerto 5, y seleccionamos la velocidad en 9600 baudios, sin bit de paridad, datos de 8 bits, 1 bit de parada y sin control de hardware. Luego en la pestaña display elegimos como queremos que se muestre la informacion, en mi caso he elegido hexadecimal ya que si cogemos ASCII habrá algunos caracteres que el monitor no los puede mostrar porque son caracteres de control del sistema. El resto de parámetros los dejamos tal y como están.

Los sensores y actuadores que estaran leyendo los registros van a ser los siguientes:

Sensor de distancia por ultrasonido: es un sensor que nos proporciona un valor analógico que para poder almacenarlo en un byte se reescala para que los valores queden comprendidos entre 0 y 255. El valor se almacena en la direccion 3 del registro Analog Input.

Joystick Analógico: Este componente tiene 2 partes, una parte será analógica y otra digital. La parte analógica seran las coordenadas (x, y) del joystick y la parte digital será un pulsador que tiene incorporado. El valor del eje “X” se almacena en la direccion 0 del registro Analog Input, la corrdenada “Y” en la direccion 1 del mismo registro, y al pulsador se le ha asignado la direccion 3 de memoria del registro Coils Input.

Módulo RGB: Cada led del módulo simulará un actuador que recibe valores comprendidos entre 0 y 255. Cada uno se iluminará independientemente con una intensidad proporcional a l valor que esté leyendo del resigtro analógico. Los valores se almacenan en el registro Analog out y sus direcciones son las que siguen: Rojo en la direccion 0, verde en la direccion 1 y azul en la direccion 2.

Senson de posicion: Es un sensor que cuando se encuentra en posicion vertial envía ‘1’ y en el resto de posiciones se encuentra en estado ‘0’. El valor lo almacena en la posicion 5 del registro Coils Input

Pulsador: Un sencillo pulsador que envia un valor alto cuando se encuentra activado. Se le ha asignado la direccion 4 del registro Coils Input.

La primera prueba será una lectura de multiples registros de entrada, que es el registro del joystick y del sensor ultrasonido. Solicitaremos los valores de los registros 1, 2 y 3.

En realterm pondremos la siguiente secuencia de bytes en la ventana SEND y una vez escrito clicaremos en el botón send ASCII:

0x00 0x04 0x00 0x04 0x00 0x01 0x00 0x03

Para enviar el mensaje a nuestro esclavo pulsaremos el boton numero 1 de nuestra FPGA y al instante obtenemos la respuesta en los leds.

Una vez haya recibido la respuesta, se iluminaran los leds con la información, la fila de arriba es la parte alta del byte, y la fila de abajo es la parte baja del byte. Tengo que modificar un par de cables

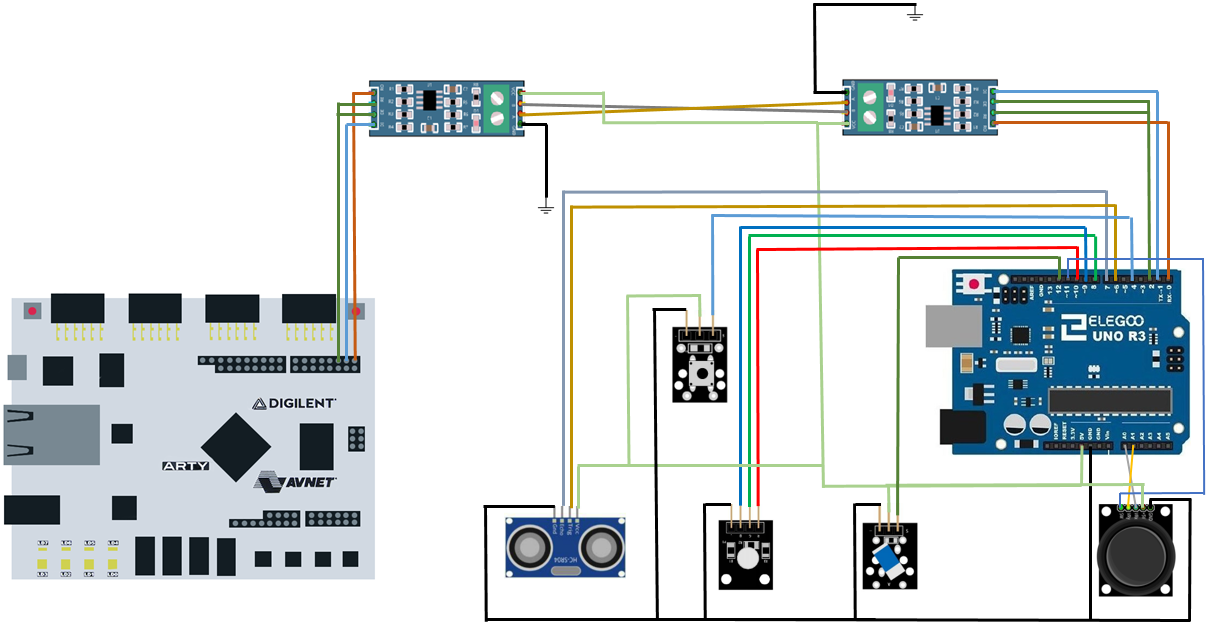


Ilustración 20 Conexiones prototipo MODBUS

# CAN-BUS

La segunda parte de este trabajo se trata de implementar un protocolo que también es usado en sistemas de adquisición de datos pero que a diferencia de MOD-BUS este no es usado en procesos industriales sino que va incorporado a cada vehículo actual producido, desde automóviles compactos hasta avionetas, barcos, autobuses e incluso maquinaria agrícola. Este protocolo, CAN-BUS cuyas siglas en ingles responden a Controlled Area Network, o red de área controlada en español, fue desarrollado por Bosch en 1986 ante la necesidad de reducir cableado, proteger ante fallos y hacer el diseño más sencillo con respecto a cómo se gestionaba la electrónica, lo que antes eran todo conexiones directas ahora se ha reducido a dos sencillos cables con lo que tenemos un bus principal y conectar a ese bus todos los sensores, actuadores y controladores necesarios. Tan solo un año después de la publicación, Intel y Philips lanzaron al mercado los primeros controladores CAN para vehículos y BMW se convirtió en la primera marca en montar este sistema en sus vehículos.

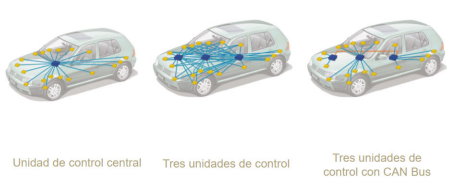


Ilustración Reducción cableado en un vehículo con CANBUS

Este se trata de un protocolo con topología bus multimaestro con prioridades, esto es que cualquier dispositivo dentro de la red podrá adjudicarse el uso del bus para hacer peticiones al resto de dispositivos pero.

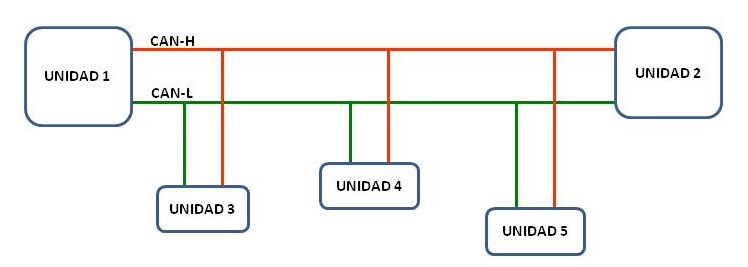


Ilustración 22 Topología CANBUS

Este protocolo se define dentro del conjunto de estándares ISO 11898 y define las capas física y de enlace en sus distintos tipos

## CAPA FISICA

CAN-BUS se trata de un protocolo con topología en bus de dos cables trenzados que pueden estar apantallados o no y que por convenio se decidió llamarlos CAN-High y CAN- Low. Las señales de estos cables son diferenciales que, de igual manera que con el protocolo anterior, nos van a ayudar a protegernos contra posibles fallas del sistema.

Para esta parte del trabajo nos ayudaremos del componente MCP2515, que es una placa que tiene 2 circuitos integrados, uno de ellos tiene el mismo nombre que el componente y es el controlador CAN, y el otro componente es un transceptor TJA1050T. De la parte física se va a encargar este transceptor que es quien va conectado al bus, y el controlador CAN se va a comunicar con nuestra FPGA y con nuestro Arduino mediante un puerto SPI. Más adelante se incluye un esquema de conexiones en el que se verá cómo se produce la comunicación entre la FPGA y el Arduino.

Según los niveles de tensión podremos estar en dos estados distintos: dominante y recesivo, siendo el estado dominante el equivalente al 0 lógico y el estado recesivo al 1 lógico. Cuando la diferencia de tensiones esté comprendida entre -0.5 y 0.05 voltios nos encontraremos en estado recesivo y cuando la diferencia entre ambas señales este comprendida entre 1.5 y 3-5 voltios se dice que el bus se encuentra en estado dominante.

Lo anteriormente nombrado se cumple para el envío, pero el receptor tiene más rango para determinar cuándo se encuentra en un estado u otro. El receptor detecta un estado recesivo cuando la diferencia nos da un valor entre -1 y 0.5 voltios y por otro lado detecta cuando se encuentra en estado dominante en el momento que la diferencia de tensión se comprende entre 0.9 y 5 voltios. Esto se hace así para que los ruidos electromagnéticos que se cuelen no influyan en nada.

{

Emisor:

{

Receptor:

En la siguiente imagen extraída de la hoja de datos del componente MCP2515 se ve claramente cómo funcionan los cambios de estado.

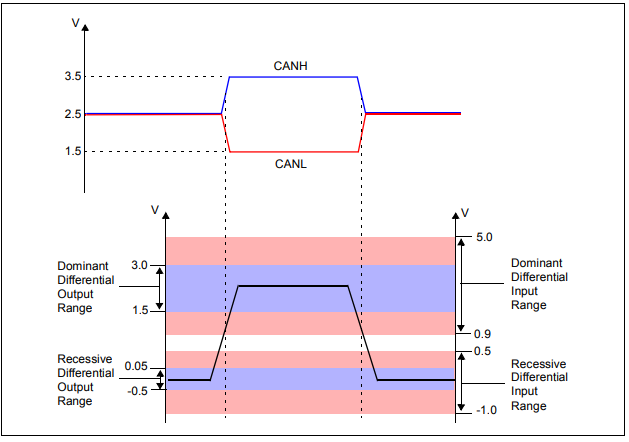


Ilustración 23 Niveles de Tensión Línea CAN

Según las limitaciones físicas del ambiente podremos disponer de varios tipos de buses para el protocolo CAN:

-Líneas de 40 metros de longitud máxima: podremos alcanzar velocidades de transmisión de hasta 1Mbit/s

-Para distancias de hasta 1km podemos conseguir una tasa de transferencia de 50Kbit/s para el máximo rango de distancia y de 125Kbit/s para distancias intermedias.

Cuando se comienza un proyecto de cableado en un vehículo siempre nos encontraremos con 2 buses can, uno de ellos es de alta velocidad para los sistemas críticos como gestión de inyección electrónica, trasmisión, , mecanismos de seguridad activa y pasiva

## CAPA DE ENLACE

Vamos a ver ahora como el protocolo gestiona los bits recibidos y también como se administran los distintos dispositivos para acceder al medio compartido, ya que como aquí podemos tener varios maestros es probable que en algún punto haya dos o incluso más que quieran adjudicarse el control del bus. También vamos a ver un mecanismo que sirve tanto para sincronizar el mensaje como para prevenir de algunos errores gracias al bit de stuff.

CSMA/CD-CR: Es el protocolo que se utiliza para detectar colisiones y poner en marcha procedimientos que servirán para solventarlos y es el acrónimo de:

* Carrier Sense (CS): Los dispositivos que comparten el medio (en este caso un bus) solo podrán hacer uso de el cuando se encuentre totalmente libre. Para ellos todos los dispositivos están escuchando.
* Multiple Access (MA): Como son varios los dispositivos que pueden tener un acceso simultaneo al medio es necesario este mecanismo.
* Collision Detection (CD): Se entiende que las colisiones son naturales y no se pueden evitar entonces lo que sucede en caso de colisión es que ambos dispositivos comienzan a trasmitir en el mismo instante de tiempo, y como comparan cada bit que envían con cada bit recibido les dará un valor que no coincide y ahí entra el mecanismo de resolución del conflicto.
* Collision Resolution: que cuando se produce una colisión ambos nodos van a detectarlo y cada uno se impondrá un tiempo de espera que dependerá del identificador del mensaje que, que como se expondrá más adelante, sirve para dar prioridad a los mensajes. El nodo que consigue hacerse con el control del medio comienza a trasmitir y el otro nodo detectará que el medio está ocupado y tendrá que esperar a que el primer nodo termine y después competir contra otros nodos en caso de que los haya.

<https://oa.upm.es/48054/8/TFM_ADRIAN_MARTINEZ_REQUENA.pdf>

pagina 23

Stuff bit: Es un bit que se utiliza para que emisor y receptores no de desincronicen. Cuando el emisor coloca 5 bits consecutivos del mismo estado (dominante o recesivo) el siguiente bit será del estado contrario a la fuerza y lo mismo para el receptor, leerá 5 bits del mismo estado y lo siguiente que espera encontrarse es un bit del estado contrario, si esto no es así se produce un error. Este conteo solo se realiza en las tramas de datos y en las tramas remotas y se le aplica a todos los campos excepto al campo EOF que delimita el final de la trama. Esta operación se realiza sobre todos los campos de las tramas de datos y remotas hasta el campo CRC, este incluido. Desde el delimitador de CRC en adelante no se tiene en cuenta.

Primero se van a exponer las distintas tramas que tenemos disponibles. Estas son:

* Trama de Datos: Se utiliza para el intercambio de información de sensores entre nodos.
* Trama de Remota: Tramas que se envían para solicitar información al nodo que indiquemos. El nodo receptor responderá con una trama de datos.
* Trama de Error: Son enviadas por nodos que detectan errores en el mensaje.
* Trama de Sobrecarga: Cuando un nodo se encuentra sobrecargado envía una trama de sobrecarga por el busque provoca un retardo extra entre las tramas para que al nodo le dé tiempo a procesar la información.
  + 1. TRAMA DE DATOS

Es el tipo de tramas más común y sirve para enviar y recibir los datos de los sensores y actuadores y cuenta con los siguientes campos:

* Start Of Frame (SOF): Las tramas comienzan con un bit dominante (0) para que todos los nodos se sincronice. El bus se encuentra en reposo, estado recesivo (1) y cuando un noto trasmite todos detectan un cambio de flanco y saben que alguien ha empezado a enviar información.
* Identificador: Sirve tanto para identificar al nodo como para darle una prioridad al mensaje. Según el formato, estándar o extendido, este campo podrá tener 11 o 29 bits respectivamente y sabremos cuando es un tipo u otro por el valor del campo IDE que aparece más adelante. Una trama con un identificador más bajo tendrá más prioridad. Si alguna trama se trasmite en formato extendido primero se mandan los 11 bits más significativos, a continuación se envían los bits RTR Y EID que se explican a continuación y después se mandan el resto de bits del identificaros, es decir, los 18 bits restantes.
* Petición de transmisión remota (RTR): sirve para saber si se trata de una trama de datos o una trama remota. Este campo ocupa un solo bit, dominante (0) nos indicaría que se trata de una trama de datos y recesivo (1) nos indica que es una trama remota o de petición de información.
* Bit de extensión de identificador (EID): como su nombre indica, se utiliza para seleccionar el formato extendido o formato base. La longitud de este campo es de 1 bit, cuando en este campo tengamos de valor ‘0’ quiere decir que estamos en una trama de formato estándar o base pero si aparece un ‘1’ tendremos una trama de datos con formato extendido y significaría que a continuación vienen el resto de bits de identificación que serían 18 bits.
* Bit reservado: El uso de este bit se reserva al fabricante en caso de que necesite incluir alguna información extra.
* Data Length Control (DLC): El campo “Control del tamaño de datos” va a indicar el tamaño en bytes de los datos que se quieren transmitir. En la especificación del protocolo se indica que el tamaño máximo del campo de datos es de 8 bytes así que necesitamos que este campo tenga un tamaño de 4 bits y su valor no puede ser superior a 8.
* Campo de Datos: Contiene la información acerca de los sensores o actuadores y como se acaba de explicar, su tamaño viene determinado por el calor del campo DLC.
* CRC: Es el código de redundancia cíclica, tiene la misma función que el código LRC de MODBUS y es para que el destinatario verifique la integridad del mensaje. Tiene una longitud de 16 bits.

Cálculo del CRC

El algoritmo que genera el CRC necesita primero un polinomio generador y al traducir ese polinomio a lenguaje binario y operando con los datos en binario del mensaje que queremos verificar obtendremos el CRC. El polinomio más comúnmente usado en las implementaciones de CAN-BUS es el conocido como CRC-15 y es el que sigue:

Que en binario tiene la forma “1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1” y en hexadecimal 0xC599El valor de CRC comienza siendo con todos sus dígitos a 1.

Pseudocódigo calculo CRC

CRC16 = 0x”FFFF”

POLYNOM = 0x”C599”

PARA cada Byte B del mensaje hacer

CRC16 = CRC16 XOR B

PARA cada bit b del Byte B hacer

SI CRC16[0] = ‘1’ ENTONCES //La posición 0 es la de menor peso

CRC16 = CRC16 XOR POLYNOM

OTRO

CRC16 = 0 >> CRC16

FIN SI

FIN PARA

FIN PARA

-Delimitador del CRC: Tendrá siempre valor recesivo ‘1’ e indica el final del CRC.

-ACK: Al igual que otros muchos protocolos en el ámbito de las telecomunicaciones, el bit de acuse de recibo sirve para indicar si se ha recibido un paquete o no. Estará en estado recesivo para el transmisor y en estado dominante para los receptores.

-Delimitador de acuse de recibo: 1 bit que si siempre en estado recesivo.

-Fin de trama: todos sus 7 bits son recesivos.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | RTR | EID | Reservado | DLC | Datos | CRC | Del  CRC | ACK | Del ACK | EOF |
| 11-29 bits | 1 bit | 1 bit | 1 bit | 4 bits | 0-8 bytes | 15 bits | CRC  1 bit | 1 bit | 1 bit | 7 bits |

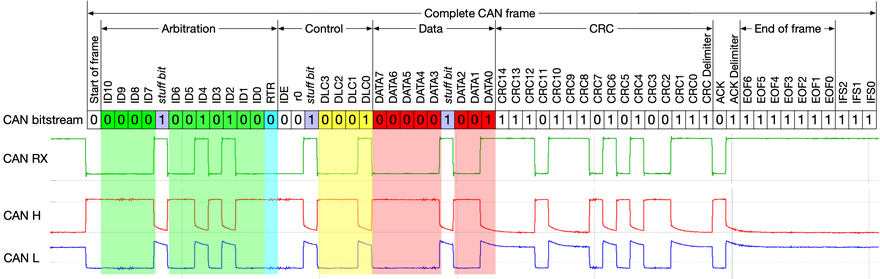
****

Ilustración 24Trama CAN de datos

### TRAMA REMOTA

Este tipo de trama se usará para realizar peticiones de información a otros nodos. En estas tramas el campo RTR está en estado recesivo, es decir con valor 1, y como se trata de una petición de datos el valor del campo DLC será 0 con lo que no existe el campo datos, inmediatamente después de DLC viene el campo CRC.

Existe otro tipo de trama en la que el campo de petición de transmisión remota (RTR) está informado con valor recesivo. Este tipo de trama se conoce como trama remota y se diferencia de una trama de datos fundamentalmente en que la primera tiene el campo RTR con valor recesivo y no tiene campo de datos, es decir, tras el campo DLC, tiene el campo CRC.

Hay que añadir además que esta trama puede ser también estándar o extendida dependiendo del valor del campo IDE.

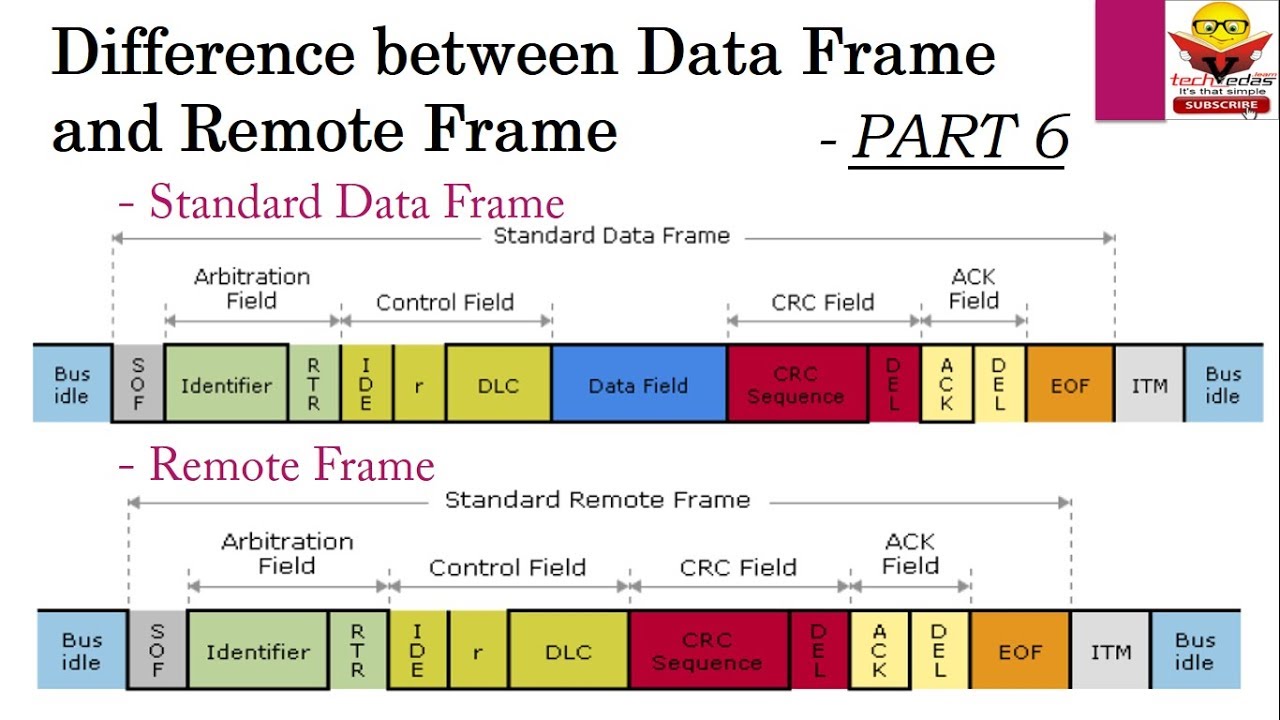


Ilustración 25 Trama CAN remota

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Identificador | RTR | Extensión Identificador | Reservado | DLC | CRC | Delimitador CRC | ACK | Del ACK | FIN TRAMA  7 bits |
| 11-19 bits | 1 bit | 1 bit | 1 bit | 4 bits | 15 bits | 1 bit | 1 bit | 1 bit | 7 bits |

### TRAMA DE SOBRECARGA

Estas tramas se utilizan para aligerar la carga de los nodos que estén saturados y que necesiten un retardo en las peticiones para procesar la información, y se consigue colocando una trama que aumenta el retardo entre envío de tramas por el bus.

Solo dispone de 2 campos, el primer campo compuesto por 6 bits en estado dominante seguidos de 8 bits en estado recesivo. Cuando el resto de nodos de la red detecta una trama de sobrecarga, responderán con flags de sobrecarga haciendo que el tráfico en la red se pare. La longitud de estos flags de sobrecarga puede ser de 6 a 12 bits.

Los receptores sabrán que se trata de una trama de sobrecarga porque en emisor no ha aplicado bit de stuff.

Además esta trama va a tener prioridad sobre las demás pueto que empieza por 0 bits dominantes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sobrecarga  6 bits | | | | | | Delimitador Sobrecarga  8 bits | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### TRAMA DE ERROR

En caso de que una trama no sea de ninguna de las anteriores estaremos ante una trama de error, que al igual que la trama de sobrecarga tiene 2 campos.

El primer campo son las banderas de error que está formado por 6 bits. Si todos estos 6 bits están en estado dominante entonces el campo error se dice que está activo pero si todos los bits de este campo están en estado recesivo entonces el campo de error está pasivo. A continuación van 8 bits recesivos que forman el campo delimitador de error.

Así si un nodo detecta un mensaje erróneo transmite una trama errónea y hará que el resto de nodos de la red transmitan tramas de error. Esto puede provocar una sobrecarga del bus así que los nodos deberán tener un contador de mensaje de error para que esto no suceda.

|  |  |
| --- | --- |
| Banderas de Error | Delimitador de banderas |
| 6 bits | 8 bits |

Tenemos dos tipos de banderas de error:

Activa: formada por 6 bits dominantes y trasmitidos por un nodo que detecta un error de red

Pasiva: sus 6 bits son recesivos y trasmitida por un nodo que detecta un error en una trama.

Para realizar una acción el nodo tiene que tener en cuenta 2 contadores: el contador de errores de trasmisión (CET) y el contador de errores de recibo (CER).

* Cuando CET o CER sea mayor que 127 o menor que 255 el nodo envía un mensaje de error pasivo por el bus.
* Cuando CET y CER sean menores que 128 se envía un mensaje de error activo.
* Cuando TEC es mayor de 255 el nodo entra en un estado de no envío, y deja de mandar mensajes por el bus.

## Seguridad del protocolo

Can bus es un protocolo de código abierto y de nivel bajo sin encriptación estándar para ninguna de las implementaciones y son muy susceptibles de sufrir un ataque “man in the middle” con lo que las fallas de seguridad han de ser solventadas en las otras capas de la pila OSI, como por ejemplo en la capa de aplicación ya que el atacante podría modificar valores y parámetros que le sirven a la ECU del vehículo para gestionar la inyección electrónica de combustible, el sistema de frenos, sensores de proximidad, etc. poniendo en peligro la vida de los ocupantes y del resto de usuarios de la vía.

Un ejemplo de esto es el grupo Volkswagen, V.A.G., el cual tiene un bus adicional para poder acceder a esta información sensible mediante la introducción de un código y sin el conocimiento de este código solo podrás monitorizar la señal del bus.

## SIMULACIÓN

El protocolo can bus se trata de un protocolo mucho más completo y mcomplejo que MODBUS, por ello nor vamos a servir de controladores canbus que se comunicaran con nuestros nodos mediante SPI, que es un protocolo serie síncrono con linea de habilitacion. La placa de Arduino va a enviar una trama de datos con unos valores y la FPGA va a ir leyendo esos valores y mostrandolos en su display de leds incorporados.

Los componentes necesarios a parte de la PFGA y el Arduino UNO son 2 controladores canbus MCP2515 que se han descrito al principio de este capítulo.

Las conexiones de mi sistema han quedado de la siguiente manera:

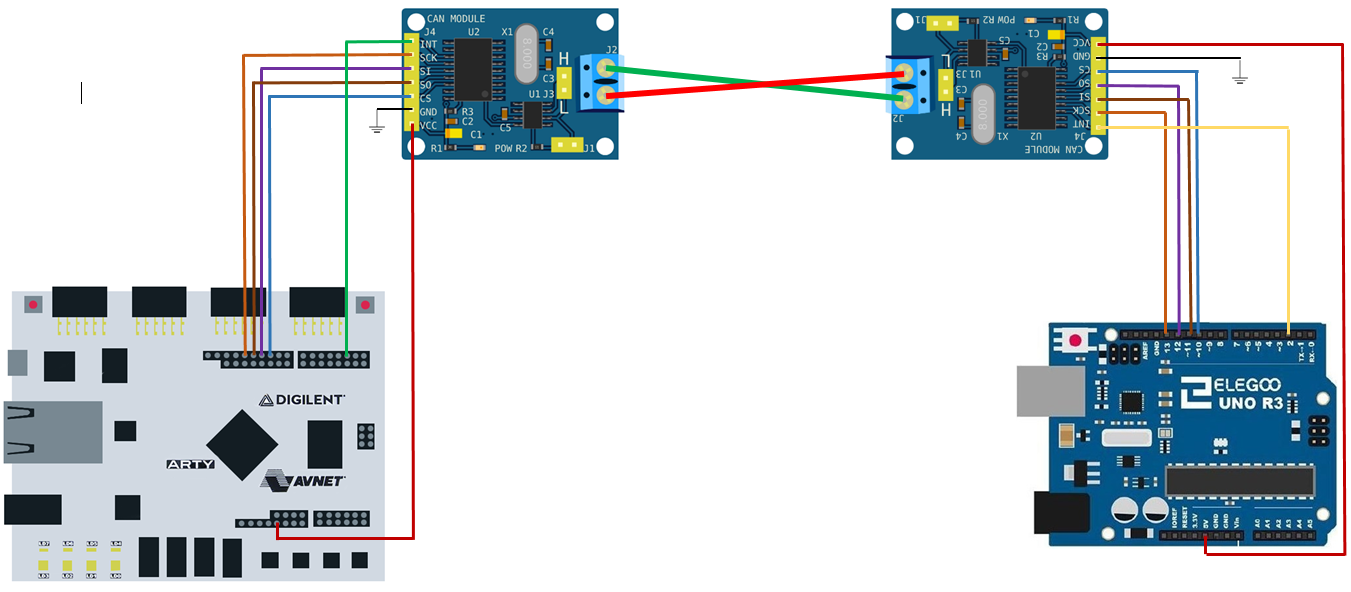


Ilustración 26 Conexiones Prototipo CANBUS

Lo primero que hay que hacer es poner en funcionamiento el puerto SPI. En Arduino es muy sencillo, hay que bajarse la librería MCP2515 e instalarla en nuestro gestor de librerias. Otra librería que vamos a necesitar es la libreria SPI para comunicarnos con el comonente.

Lo siguiente inicializar bien cada funcion y como la librería se encarga del trabajo duro nosotros unicamente tendremos que hacer una llamada a la funcion correspondiente de la librería que en mi caso es “sendMessage(&canMessage)”.

Solo nos quedaría darle los valores id, DLC y data a la estructura canMessage (En el anexo parte x).

Para la FPGA la tarea ha sido más complicada ya que no hay librerias ninguna que podamos añadir entonces se ha cogido la librería mcp2515 de arduino y se han pasado las funciones necesarias para el funcionamiento a lenguaje VHDL.

Imagenes del funcinamiento

# COMPARACION

# CONCLUSIONES

ANEXOS MOD BUS

# Bibliografía

**No hay ninguna fuente en el documento actual.**

<http://www.tolaemon.com/docs/modbus.htm>

<http://www.tolaemon.com/docs/modbus.htm>

<https://www.modbustools.com/modbus.html#Function01> instrucciones

<https://www.kvaser.com/es/lesson/session-1-can-physical-layers/> capa física can bus

<https://www.ionos.es/digitalguide/servidores/know-how/csmaca-protocolo-de-acceso-al-medio-para-redes-inalambricas/> CSMA/CA

<https://www.ionos.es/digitalguide/servidores/know-how/csmacd/> CSMA/CDO

<http://www.esd-electronics-usa.com/CAN-Remote-Frames.html%20> trama en canbus

<https://en.wikipedia.org/wiki/CAN_bus> CAN BUS WIKI

<https://github.com/sandeepmistry/arduino-CAN/blob/master/API.md> librería can bus arduino

<https://surf-vhdl.com/how-to-design-spi-controller-in-vhdl/> interfacng vhdl spi

1. [«AN228: A CAN Physical Layer Discussion»](http://ww1.microchip.com/downloads/en/AppNotes/00228a.pdf). Microchip Technology Inc. 2002. Consultado el 22 de agosto de 2015.

<https://www.cursosaula21.com/modbus-que-es-y-como-funciona/#:~:text=Modbus%20es%20un%20protocolo%20de,informaci%C3%B3n%20son%20los%20esclavos%20Modbus>.

<https://www.edaplayground.com/x/5CMQ>

<https://github.com/nandland/spi-master/tree/master/VHDL>

<https://github.com/nandland/spi-master/blob/master/VHDL/source/SPI_Master_With_Single_CS.vhd>

https://www.diagnosistips.com/can-bus/