Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Esquema de Comunicaciones Avanzado para Convertidores Modulares

Autor: Juan Jiménez Alonso

Tutores: Abraham Márquez Alcaide, Marta Laguna García

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Esquema de Comunicaciones Avanzado para Convertidores Modulares

Autor:
Juan Jiménez Alonso

Tutores:

Abraham Márquez Alcaide
Investigador posdoctoral

Marta Laguna García
Profesor Colaborador

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Esquema de Comunicaciones Avanzado para Convertidores Modulares

Autor: Juan Jiménez Alonso
Tutores: Abraham Márquez Alcaide
Marta Laguna García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mis profesores, desde el primero hasta el último.

A mis tutores, por su constante disponibilidad y por la ilusión y las ganas puestas en este trabajo. A José María Hinojo por prestarme el analizador lógico con el que se hicieron las pruebas que aportan rigor a este Trabajo de Fin de Grado. A los miembros del laboratorio de electrónica, por el constante apoyo en la elaboración de las PCBs.

A mi familia, el principal motor para que llegara hasta aquí desde pequeño, por darme motivos y vías para formarme. A mi padre, ingeniero aunque no lo diga un papel. A mi madre, inteligente, paciente y perseverante como nadie.

A mi compañera, apoyo fundamental en los días que no sale nada.

Juan Jiménez Alonso
Estudiante autor

Sevilla, 2022

Resumen

Los convertidores modulares son una solución competitiva para muchas aplicaciones como la producción de energía que está continuamente creciendo ante una demanda también creciente. Sin embargo, en determinadas condiciones de funcionamiento, principalmente en un comportamiento no balanceado, las técnicas tradicionales de control no son admisibles ya que el resultado no es bueno en aspectos de eficiencia. Es necesario aplicar por tanto, técnicas de control avanzado para lo cual es necesario tomar datos en cada intervalo de funcionamiento de un convertidor, es decir, a una frecuencia muy alta. Además estos datos deben ser transmitidos a un módulo encargado de ejecutar el control y generar una referencia que es devuelta al convertidor. El punto más sensible de este proceso son las comunicaciones, ya que no existe una solución eficiente para comunicaciones tan rápidas como las necesarias.

Estudiaremos por tanto, diferentes posibilidades de comunicaciones. Partiremos de un protocolo desarrollado por Abraham Márquez, tutor de este Trabajo de Fin de Grado, en su doctorado y lo adaptaremos a las potenciales soluciones encontradas. Nos centraremos principalmente en los protocolos de bajo nivel SPI y UART que permiten mayor flexibilidad en el mensaje y su cabecera, realizando experimentos al límite superior de velocidad permitida por los microcontroladores.

Analizaremos también el hardware necesario para las comunicaciones, partiendo del sistema ideado por A. Márquez, y desarrollaremos diseños de PCBs que hagan posibles las comunicaciones. Como medios físicos tendremos cable plano, cable RJ45 y fibra óptica, cuyos adaptadores a las señales de los microcontroladores diseñaremos, probaremos y evaluaremos. Para estas evaluaciones realizaremos experimentos que repetiremos con las mismas condiciones hasta en 7 ocasiones cada uno con el fin de obtener datos fiables. Además de comparar los diferentes enlaces y protocolos, realizaremos pruebas donde comprobemos diferentes esquemas de comunicaciones, entre dos o más dispositivos que nos permitirán analizar la respuesta del protocolo y los elementos hardware diferentes modos de funcionamiento como pueden ser comunicaciones punto a punto, comunicaciones con múltiples esclavos con pregunta y respuesta y comunicaciones con múltiples esclavos con respuesta multiplexada en el tiempo.

Se utilizará también el protocolo I2C para el control de un expansor de pines de entradas y salidas digitales, creando además una librería para facilitar su uso: leer las entradas, escribir las salidas, manejar las interrupciones, configurar los pines...

Tras las distintas pruebas veremos como aumentar la frecuencia manteniéndonos por debajo de los 10MHz supone una mejora del comportamiento importante. Sin embargo, a frecuencias altas, subidas similares o mayores en la tasa de datos no producen resultados tan notables, posiblemente porque la frecuencia limitadora en estos casos es la del microcontrolador y no la del periférico. También veremos como la UART es más fiable en primera instancia, pero en determinadas condiciones el SPI puede también tener un comportamiento óptimo.

Abstract

Modular converters are a competitive solution to many applications as power production that is increasing because of an also increasing demand. However, traditional control techniques do not work well in some operational conditions, such as unbalanced ones. Advanced control techniques are needed to be applied. Those control methods require a continuous measurement of electric data of the converter in each working interval. The data should be transmitted to a control module that executes the algorithm and generates a signal that is sent back to the converter. The communications in this proccess must be very fast and efficient. There is not a developed solution to be inmediately applied.

So we are going to study different communication possibilities. This document is based on a system developed by Dr. Abraham Márquez in his doctoral thesis, where he designed a general idea about hardware and a software protocol that will be adapted to the proposed solutions. We are going to focus on SPI and UART protocols, that allow us to configure the message and its header, in order to improve efficiency. We will do experiments in which communications frequency will be set at upper limit stablished by microcontrollers.

Futhermore, we will analize hardware needed in this communicatios and we will develop different PCB designs to achieve the required speed. Flat cable, RJ45 cable and optic fiber will be used as physic link between communicated devices. We will focus our design on the adapters of this links to the microcontrollers signals. Every experiment will be repeated 7 times at least, so that we can store reliable information. In addition to this test, we are going to make other ones in which different communications schemes will be compared, such as point to point and multislave with two modes: The first one, ask and reply and the second one, broadcast ask and time-multiplexed reply.

The I2C protocol will be used to control a digital input-output expansor and a library will be created to give easier ways to read inputs, write outputs, configure pins and handle interrupts.

After multiple test, we will see that a frequency increase at low frequencies (less than 10 MHz) result in an important reduction in communication times. However, at high frequencies the result of increasing bit rate is not so noticeable. Maybe, this is due to the microcontroller frequency, which is limiting the peripheral speed. Furthermore, we will see that UART is more reliable but in some cases, SPI is able to behave as well as UART.

Índice Abreviado

Índice

1 Introducción

La producción de energía eléctrica en España combina fuentes de energía renovables y no renovables [1]. Aunque las primeras formas de producción de trabajo mecánico a partir de una fuente de energía fueron el molino de viento y la rueda hidráulica [2], las fuentes de energía no renovables han protagonizado la generación de energía desde hace unos 200 años. A partir de la revolución industrial, los combustibles fósiles, abarcaron gran parte de diferentes industrias donde la máquina de vapor, impulsada por carbón, ganó un papel principal [3].

También surgieron el petróleo y el gas natural dentro del grupo de los combustibles fósiles. El petróleo, anteriormente usado en medicina para tratar enfermedades como lombrices, sordera o dolor de muelas, comenzó a ser usado como combustible para lámparas (queroseno) [4]. Al igual que el petróleo, el gas natural comenzó a usarse para alumbrado. Las ciudades empezaron a ganar vida noctura gracias un suministro continuo de gas[5].

El petróleo fue abarcando mercado gracias a dos hitos importantes [4]: Edwin L. Drake (Greenville, Nueva York; 29 de marzo de 1819 - Bethlehem, Pensilvania; 9 de noviembre de 1880) consigue en 1859 extraer petróleo por primera vez mediante la perforación de un pozo y Henry Ford (Dearborn, Míchigan; 30 de julio de 1863 - 7 de abril de 1947) y su compañía producen en grandes cantidades el primer coche de gasolina a partir de 1908. Muchos investigadores como Watt, Otto o Benz aportaron grandes avances en el estudio del motor de combustión interna y la termodinámica, que posibilitaron el éxito del motor de gasolina. [2]

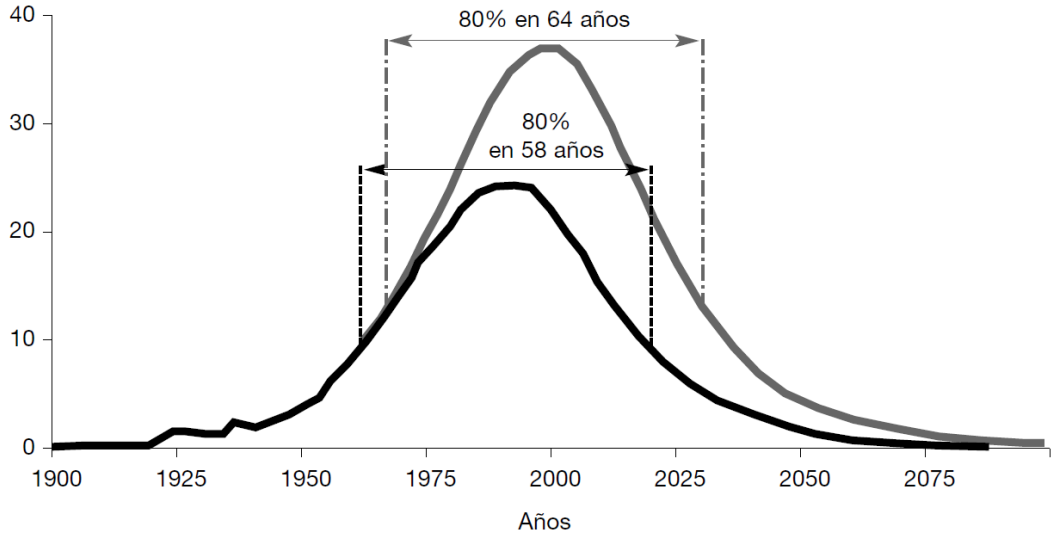


Figura 1.1 Teoría de Hubbert [6].

Parte del éxito del petróleo se explica con la teoría de Hubbert. Esta teoría predice el comportamiento de las extracciones de petróleo de un pozo, y se puede extrapolar al conjunto del mundo. La producción de petróleo tiene forma de campana, distinguiendo una zona creciente donde la extracción de petróleo es barata y se obtiene petróleo de buena calidad, y otra zona decreciente, pasado el pico o "cénit del petróleo" dónde el petróleo comienza a ser más caro y costoso de extraer.

Uno de los aspectos más llamativos de esta teoría es que afirma que el ritmo de producción del petróleo no viene determinado por la demanda o el coste, sino por la energía requerida para extraerlo. De la figura 1.1 podemos obtener varias conclusiones:

- Suponiendo que las existencias de petróleo fueran aproximadamente un 50 % superior a lo que se cree que son, tan solo durarían 6 años más.
- El ritmo al que la producción de petróleo descendederá será similar al que siguió en el ascenso, el cual fue muy elevado [7].
- Aunque la fecha del pico no se ha determinado, la mayoría de las fuentes lo sitúan en la primera veintena del siglo XXI. Grandes productores como Argelia ya han afirmado que su producción de crudo ha superado este punto [8].
- En el presente, el coste de extraer el petróleo va en ascenso y su calidad disminuyendo, hasta alcanzar el punto máximo, donde el coste de extraer un barril de petróleo igual al beneficio obtenido de producirlo.

Este hecho es uno de los motivos por el que se están tomando decisiones políticas y económicas por parte de gobiernos y grandes empresas para modificar el modelo actual de obtención de energía, como por ejemplo la petroquímica Repsol [9], que ha fijado objetivos de cero emisiones para 2050.

La utilización de combustibles fósiles para obtención de energía, transporte y otras industrias es también una de las principales causas del cambio climático, debido a la emisión de gases de efecto invernadero [10]. La preocupación social y la posición de diferentes organizaciones a favor de la necesidad de frenar el calentamiento global hacen que gran parte de las campañas de marketing empresarial estén enfocadas en el medio ambiente y la reducción de emisiones contaminantes [11].

A estos factores que propician la búsqueda de otras fuentes de energía alternativas, se le suma el crecimiento demográfico, que conlleva una creciente demanda energética.[12].

En la figura 1.2 podemos ver una evolución histórica y previsión de las fuentes de energía primaria que más se han usado y se usarán en los próximos años. Cabe destacar la aparición y expansión de la energía eólica y solar a partir de la década de 2010 y la bajada progresiva del uso del petróleo y gas natural. Si nos fijamos en la generación de electricidad en Europa podemos ver que las fuentes de energía renovables tienen un protagonismo en ascenso (figura 1.3).

El uso de fuentes de energía renovables a gran escala, para producción de electricidad principalmente, presenta varios problemas o dificultades:

- Son necesarios costosos dispositivos de conversión de energía de un tipo (eólica o fotovoltaica) a eléctrica.
- La no controlabilidad de la fuente de energía, ya que en muchos casos depende del clima determinado en una zona del planeta o el tiempo cambiante con diferentes niveles de radiación solar y velocidades del viento. Esto supone dificultades para ajustar la producción a la demanda y mantenerla.
- Conectar mecánicamente a las aspas de un molino de viento (por ejemplo) un generador eléctrico de alterna, con idea de inyectar la corriente del generador directamente en la red, no es una buena solución, ya que cambios en la velocidad del viento harían que la corriente inyectada en la red no fuera igual que la existente, provocando pérdidas de eficiencia.

Para solucionar estos problemas se utiliza la electrónica de potencia [1].

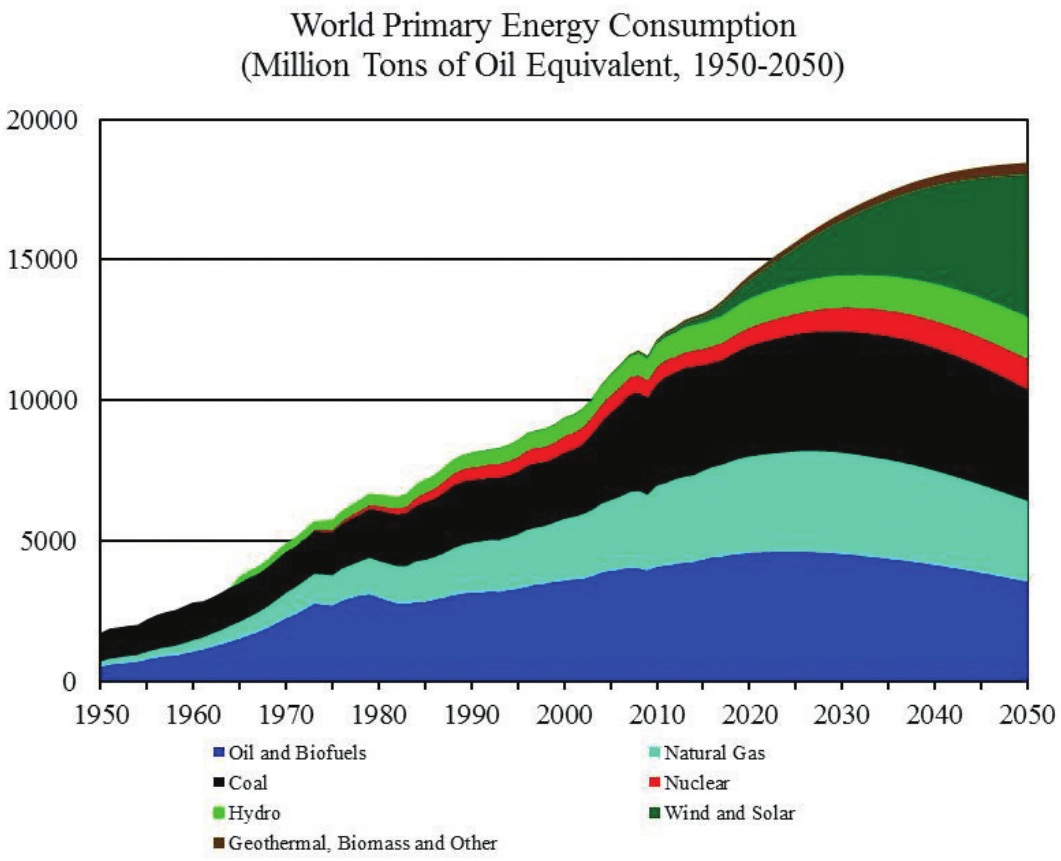


Figura 1.2 Consumo histórico de energía y proyección futura [12].

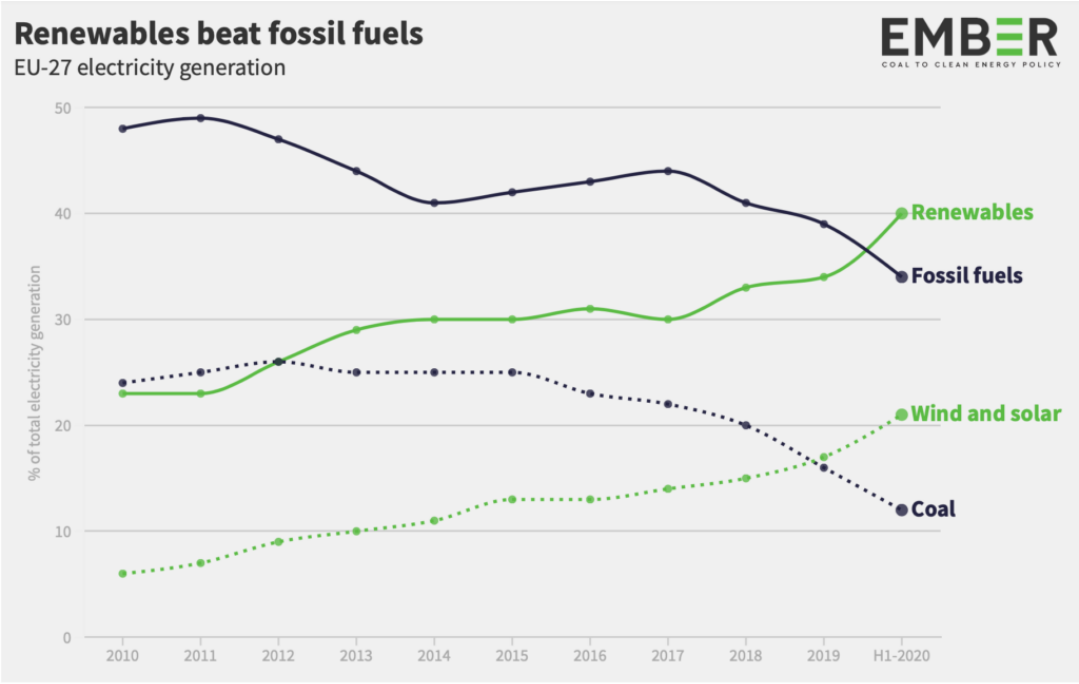


Figura 1.3 Generación de electricidad en la unión europea a partir de fuentes de energía renovables y combustibles fósiles [13].

1.1 La electrónica de potencia

Esta disciplina permite realizar transformaciones de la energía eléctrica dependiendo de la aplicación final, con una alta eficiencia, es decir, con pérdidas muy bajas. Con esta tecnología es posible obtener una corriente eléctrica estable con características similares a la de la red, a partir de la corriente obtenida de un generador eléctrico (por ejemplo).

Se basa en unos dispositivos llamados dispositivos de potencia, como por ejemplo el IGBT, que permiten conmutaciones eléctricas rápidas en los circuitos persiguiendo una baja caída de tensión en conducción y paso bajo de corriente en corte. Con estos dispositivos y más elementos como diodos, bobinas, condensadores y resistencias se diseñaron circuitos conocidos como convertidores de potencia que son capaces de realizar una conversión de tensiones y/o corrientes, de forma que la energía pueda ser utilizada de forma eficiente, según estén conectados los convertidores (a la red eléctrica o no). Se pueden clasificar en cuatro tipos: rectificadores (AC$\rightarrow$DC), inversores (DC$\rightarrow$AC), choppers o recortadores (DC$\rightarrow$DC) y cicloconvertidores (AC$\rightarrow$AC).

La electrónica de potencia tiene tanta importancia en las energías renovables porque en los circuitos de potencia existen unos controladores que adaptan la corriente dentro de un intervalo de características a la salida que deseamos. La producción eléctrica con energías renovables es muy diversa y cambiante, ya que depende en la mayoría de las ocasiones del tiempo atmosférico. Esta variabilidad hace que controlar esta energía de forma automatizada sea complicado (pero necesario porque la producción necesaria de energía es gigantesca) y la electrónica de potencia es una buena alternativa para ello.

2 Convertidores modulares

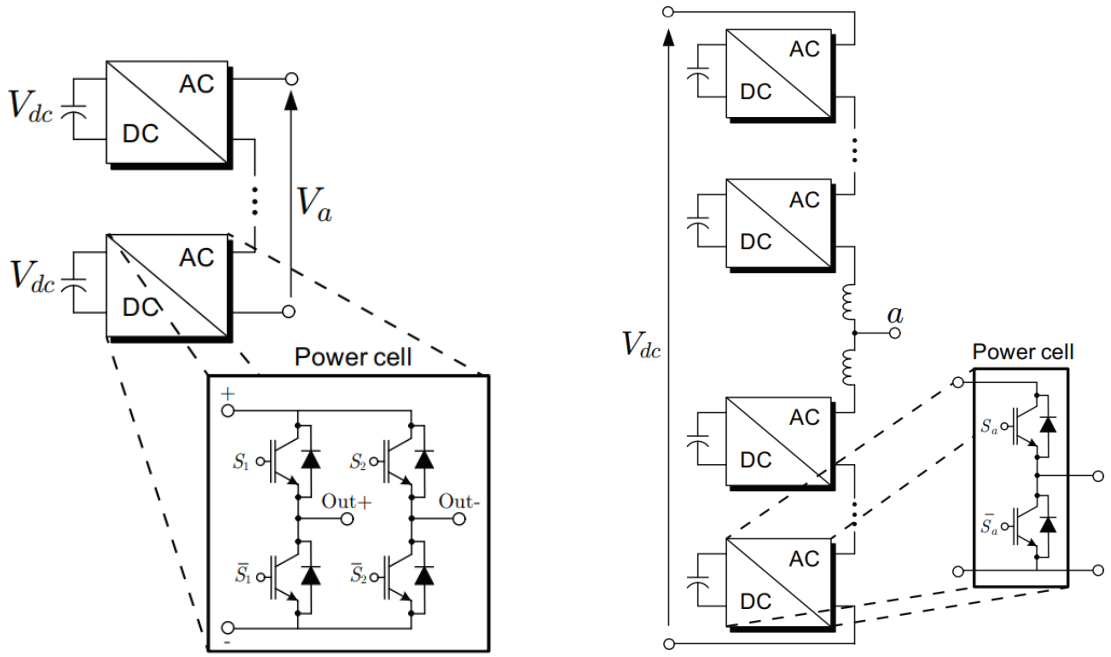
Debido a la creciente demanda de energía y los requerimientos sobre ella y su producción como la alta eficiencia, surgen los convertidores modulares basados en la conexión de múltiples convertidores de potencia en ocasiones llamados módulos de potencia o *Power Electronics Building Blocks*. Estos convertidores, además de un alto rendimiento, consiguen reducir costes de diseño, producción y mantenimiento al estar compuestos por módulos idénticos directamente reemplazables en caso de fallo y son capaces de cubrir diferentes aplicaciones según la integración que se haga con ellas. Esta constitución de los convertidores modulares también le aporta gran escalabilidad y tolerancia a fallos [14].

2.0.1 Esquemas de conexiones de los convertidores modulares

Principalmente podemos distinguir dos tendencias de diseño de los convertidores modulares: basados en módulos conectaos en serie y basados en módulos conectados en paralelo, así como configuraciones mixtas que comparten características de ambas ramas. [14]

- **Módulos conectadas en serie.** Enlazar los distintos bloques mediante conexiones series es una solución que se está proporcionando en el desarrollo de estos convertidores que consigue alcanzar altos niveles de tensión. Si un módulo funcionando en solitario puede proporcionar una tensión de V_c, el voltaje que se puede conseguir en un convertidor modular con N módulos de potencia será de NV_c. Esto supone una razonable e interesante vía para alcanzar grandes niveles de tensión como los necesarios en controladores de motores de medio - alto voltaje o sistemas de transmisión de potencia a través de altas tensiones de corriente continua, como los enlaces establecidos para la comercialización de energía eléctrica entre España y Francia. Los convertidores multinivel más conocidos, como el puente-H en cascada (figura 2.1a), son una aplicación a baja escala de un convertidor con módulos idénticos conectadas en serie. También se han desarrollado los Convertidores Modulares Multinivel (figura 2.1b), basados en esta misma idea pero con una mayor cantidad de módulos conectados que el puente-H en cascada[15].

- **Módulos conectados en paralelo.** Por otro lado, la segunda alternativa es conectar los módulos de potencia en paralelo. Realizar las conexiones de esta forma permite una distribución igualitaria de la corriente a través de los diferentes módulos del sistema de potencia. Además es posible satisfacer requerimientos de altas corrientes, incluso por encima de los límites de corriente que establecen los propios dispositivos de potencia, ya que implementar convertidores tradicionales con dispositivos de potencia de alta corriente es muy costoso. Este tipo de conexiones se pueden encontrar en plantas industriales e instalaciones domésticas fotovoltáicas, donde diferentes paneles independientes son conectados a diferentes convertidores



(a) Convertidor Puente H en Cascada.

(b) Convertidor Modular Multinivel.

Figura 2.1 Convertidores basados en módulos conectados en serie [15].

elevadores de la tensión de corriente continua y tras ello, se enlazan con conexiones paralelas a la entrada de un inversor para obtener corriente alterna.

2.0.2 Control de convertidores modulares

Para poder utilizar los convertidores modulares aprovechando todo su potencial es necesario aplicar técnicas de control y de modulación más avanzadas que las utilizadas en convertidores "tradicionales". Existen aplicaciones donde el comportamiento de los diferentes módulos no es similar, sino que tenemos un comportamiento no balanceado, por ejemplo en una central eléctrica de paneles fotovoltáicos extendida en una gran superficie, donde la radiación solar incidente es variable según el estado del cielo, nubes, etc. Además, algunos métodos de control de temperatura provocan un comportamiento no balanceado en los convertidores modulares. Estos comportamientos no balanceados controlados con las técnicas tradicionales pueden producir armónicos de frecuencia que resultan en una disminución de la vida útil en los dispositivos de potencia y por tanto, un aumento de los costes de mantenimiento. Se pierden además otros beneficios obtenidos del funcionamiento balanceado como la distribución igualitaria de la potencia y la corriente y serían necesarios mayores filtros de frecuencia a la salida, aumentando el coste y el peso de los sistemas[14].

Técnicas de control avanzado pueden aplicarse para mejorar el comportamiento de los convertidores modulares como la VAPS-PWM (variable-angle, phase-shifted PWM) [16, 17]. Para poder aplicar estas técnicas de control avanzado es necesario conocer en cada intervalo de funcionamiento las corrientes y tensiones que aparecen en los distintos convertidores, para lo cual es necesario una recolección de datos periódica a muy alta frecuencia. Estos datos deben ser enviados a una CPU que los analice y ejecute el control para devolver una referencia antes del siguiente intervalo de funcionamiento. Es decir, es necesaria la inclusión de un sistema de recogida de datos, cálculo eficiente de algoritmos de control complejos y aplicación de señales de control que funcione a una frecuencia mayor que el dispositivo de potencia[14].

El punto donde más retrasos, pérdidas de datos y otros problemas se producen es en las comunicaciones. Otro motivo para requerir comunicaciones de alta velocidad es dejar un margen de cómputo

y cálculo. Para realizar una estimación de la frecuencia de comunicación necesaria vamos a poner un ejemplo mediante varios supuestos:

- En primer lugar vamos a suponer que mandamos 16 bits de datos en cada mensaje.
- Es necesario mandar dos mensajes, uno hacia el módulo de control con los datos recogidos y otro de vuelta con la referencia a aplicar en el control.
- Suponemos también que cada mensaje lleva 8 bits de cabecera.
- Como dimensión del convertidor vamos a estimar unos 20 módulos de potencia conectados.
- Los módulos de potencia trabajan a una frecuencia de 1 kHz o superior, lo que implica que ambos mensajes y el procesamiento del módulo de control debe hacerse en un tiempo inferior a 1 ms.

Tenemos entonces la siguiente expresión, de la que resulta una frecuencia mínima de funcionamiento de 960 kHz. Normalmente tendremos cabeceras más largas y mensajes mayores, por lo que podemos estimar la frecuencia necesaria al menos en 1 MHz.

$$\frac{(16b_{datos} + 8b_{cabecera}) * 2 * 20}{f_{comunicaciones}} < 10^{-3} \tag{2.1}$$

2.1 Soluciones comerciales a las comunicaciones de alta frecuencia

Entre los diferentes protocolos que podrían ser utilizados como solución distinguimos protocolos de bajo nivel como SPI y UART o buses con un protocolo de más envergadura definido. Empezaremos comentando algunas posibilidades dentro de los buses y luego hablaremos sobre los protocolos de bajo nivel.

2.1.1 Bus CAN

Las características principales y más interesantes para nuestra aplicación del bus CAN son las siguientes:

- Una señal diferencial full-duplex, lo que aporta alta inmunidad a las interferencias electromagnéticas.
- Posibilidad de incluir múltiples dispositivos, tanto maestros como esclavos.
- Desconexión automática de nodos defectuosos.
- Es necesaria una sincronización en una fase previa de arbitraje para que todos los módulos ajusten su reloj.
- Corrección de colisiones mediante el uso de bit recesivo y dominante.
- Tasa máxima de transmisión: 1 Mbit por segundo.
- Para una cantidad de datos entre 0 y 64 bits es necesario enviar, entre bits de cabecera, verificación, finales de trama, etc. un total de 44 bits mínimo como se puede observar en la figura 2.2

Analizando el requisito de frecuencia, aunque podría estar dentro de nuestro límite, no es viable en la práctica confiar en que todo funcione a las velocidades ideales sin retrasos, ya que, en caso de haberlos, ya sea por tiempo de computo o cualquier otra causa, el bus CAN no sería una opción viable. Por otro lado, no es eficiente tener que mandar 44 bits adicionales en cada mensaje, siendo los mensajes como máximo de 64 bits. En la suposición anterior donde mandábamos 24 bits necesitaríamos mandar 68 bits en total para poder hacerlo con el bus CAN, casi el triple de lo necesario.

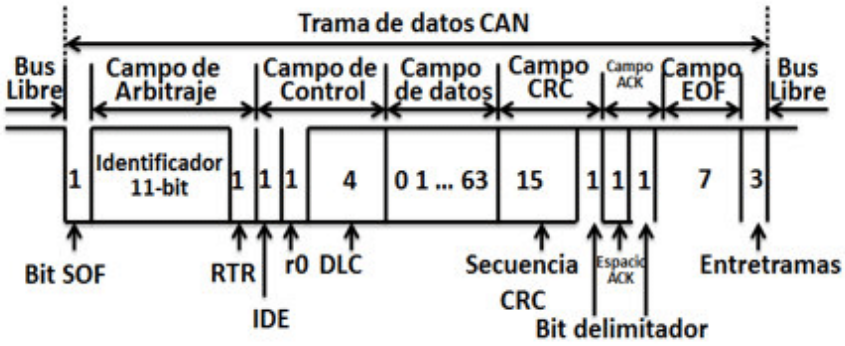


Figura 2.2 Mensaje enviado por el bus CAN. Fuente: *https://www.researchgate.net/figure/Figura-7-Formato-de-trama-CAN-20A-16_fig1_313666204*.

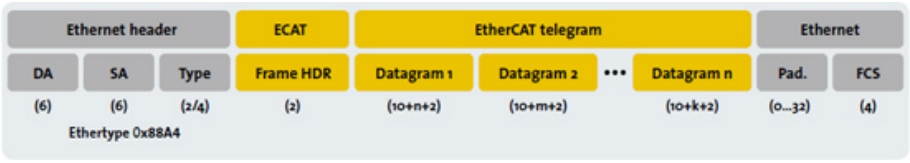


Figura 2.3 Mensaje enviado por EtherCat. Fuente: *https://www.ethercat.org/es/technology.html*.

2.1.2 Bus EtherCat

El bus EtherCat (Ethernet for Control of Automation Technology) es un protocolo industrial con alta precisión en la medición temporal de los eventos ocurridos en el bus, es decir, es un protocolo pensado para sistemas de tiempo real, como el que estamos diseñando.

- Se basa en el paso del mensaje por todos los nodos, los cuales van leyendo progresivamente el mensaje e insertando sus datos conforme pasa.
- Velocidades de transmisión de 100 Mbps.
- Aunque está estandarizado por el IEEE[1], hay una gran proporción de dispositivos del mercado que no incorporan periféricos específicos para controlarlo.
- El mensaje se incrusta en una trama de Ethernet, lo que implica mayor cantidad de bits de cabecera.
- Tiene tantos pines (o lineas) como el protocolo Ethernet.

2.1.3 Protocolos de capas inferiores

Los protocolos de bajo nivel, que no definen el cuerpo del mensaje, como SPI o UART son más versátiles ya que la cabecera es fácilmente modificable según los requisitos del sistema. Por ejemplo, para las pruebas que se van a hacer en este documento, se ha utilizado una cabecera de 4 campos de 8 bits cada uno. Cada uno de estos 4 campos tiene un significado que veremos cuando llegue el momento, pero todos son útiles. Por ejemplo, a la hora de definir una dirección del módulo receptor del mensaje, sabemos que con 8 bits tendremos suficientes, sin embargo, protocolos que definen el mensaje como CAN o EtherCat ya determinan esta longitud.

1. **UART:** El protocolo UART, Universal Asynchronous Receiver Transmitter, es un protocolo o conjunto de normas que rigen el intercambio de datos serie entre dos dispositivos. Es un

[1] IEEE: El Instituto de Ingenieros Eléctricos y Electrónicos es una asociación mundial de ingenieros dedicada a la normalización y el desarrollo en áreas técnicas

protocolo asíncrono, lo que significa que no manda una señal de reloj para sincronizar los dispositivos, sino que ambos deben estar programados a la misma velocidad. Tiene dos lineas, RX y TX, receptor y transmisor respectivamente. Aunque hay frecuencias estándares, se pueden fijar otras siempre que ambos extremos estén a la misma, por lo que es necesario hacer un estudio de las velocidades máximas de este protocolo en los microcontroladores a utilizar.

2. **SPI:** Por otro lado, el protocolo SPI, Serial Peripheral Interface es un protocolo síncrono a diferencia de la UART, es decir, hay un dispositivo que pone en una linea una señal de reloj y el resto utilizan esta señal para sincronizar sus lecturas y escrituras. Este dispositivo es conocido como maestro o máster y es encargado de iniciar todas las comunicaciones. Sus lineas o pines son:
   a) SIMO (Slave In Master Out): Línea de transmisión del maestro y recepción del resto de dispositivos, conocidos como esclavos.
   b) MISO (Master In Slave Out): Línea de transmisión de los esclavos y recepción del maestro.
   c) CLK: Señal de reloj establecida por el maestro.
   d) CS (Chip Select): Línea a través de la cual el maestro selecciona el dispositivo al que va dirigido el mensaje. Normalmente tiene lógica invertida y aunque los esclavos solo tienen una para saber si son seleccionados o no, el maestro debe tener una línea CS por cada módulo conectado en la red (figura 4.1).

   Al igual que la UART, la velocidad de funcionamiento depende del dispositivo, aunque suelen superar las frecuencias de la UART.
3. **I2C:** El protocolo I2C, I^2C, o Inter-Integrated Circuit, es un bus serie de datos destinado a las comunicaciones entre microcontroladores y sus periféricos. Es un sistema de múltiples dispositivos, síncrono al igual que SPI y con velocidades máximas entorno a los 3-5 Mbps en modos de alta velocidad. Los modos más extendidos, estándar y rápido, tienen tasas de envío de 100 kbps y 400kbps respectivamente. Tiene dos señales, una de reloj (SCL) y otra de datos (SDA) y se selecciona entre los diferentes dispositivos conectados en el bus mediante una dirección de 7 o 10 bits que se manda antes de cada mensaje, de forma que todos los elementos conectados a un bus deben tener direcciones diferentes.

2.2 Qué se propone en este TFG como solución

En el Apéndice B se ha realizado un breve estudio sobre las velocidades máximas de los protocolos SPI y UART, donde vemos que se consiguen velocidades por encima de los 10 Mpbs. Por tanto, en este documento se expondrán acciones realizadas con el fin de lograr comunicaciones rápidas y eficaces por SPI y UART orientadas hacia su aplicación en convertidores modulares de frecuencia. Esta aplicación conlleva la presencia de un módulo de control maestro que realiza solicitudes de comunicaciones y uno o varios módulos de bajo nivel.

Se planteará una solución completa tanto a nivel de software como hardware, partiendo de un protocolo definido por Abraham Márquez en su tesis doctoral [16] que analizaremos como primer punto del documento. A nivel de hardware, se partirá desde la idea de crear una PCB donde conectar una LaunchPad y se encargue de recoger los conectores y periféricos necesarios para comunicaciones, configuración, gestión de la potencia, etc. Se planteará el uso de diferentes enlaces físicos, como Fibra Óptica, cable RJ45 y cable plano en diferentes escenarios de comunicaciones: punto a punto y múltiples esclavos con RS485.

Al ser una solución en desarrollo, es conveniente que sea lo más versátil posible, pero sin que esto suponga incluir grandes complicaciones y usando la menor cantidad de señales posibles con el objetivo de crear un sistema “simple” y robusto. Dentro de la versatilidad buscada, un

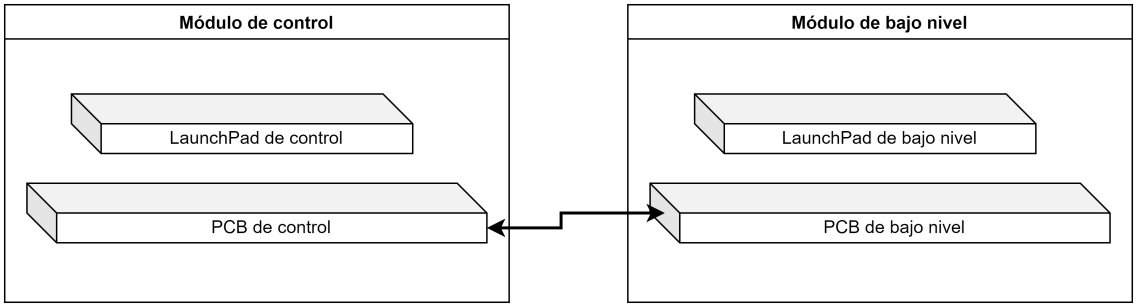


Figura 2.4 Concepto básico sobre los dispositivos que vamos a comunicar.

punto fundamental es la utilización de protocolos estándar ya que, de esta forma facilitaremos el conexionado con módulos externos en caso de ser necesario. Además, no tiene sentido programar a bajo nivel las señales que se van a utilizar, pues hay soluciones ya implementadas y optimizadas en periféricos concretos que funcionarán mejor que cualquier solución que implementemos mediante GPIO.

2.2.1 Contacto rápido con los diferentes enlaces físicos y el estándar RS485

Acabamos de comentar diferentes enlaces físicos y un protocolo o norma de las transmisiones serie (RS485). Antes de utilizarlos vamos a ver los aspectos más importantes que se deben tener en cuenta en el diseño del sistema.

- **Fibra óptica:** Es un material obtenido del vidrio utilizado para transmitir señales luminosas que viajan mediante reflexión por el cable gracias a la utilización de un recubrimiento con índice de refracción bajo, el cual evita reducciones en la intensidad de la señal. Al utilizar luz como portadora de la información, los datos no son sensibles al ruido electromagnético y también se evita la necesidad de incluir aisladores galvánicos que proporcionen aislamiento eléctrico. Solo permite comunicaciones punto a punto, ya que los cables de fibra no pueden unirse fácilmente como los cables tradicionales.
- **Cable plano:** Es un grupo de cables dispuesto de forma paralela con conectores estandarizados en los extremos. Permite la inclusión de múltiples dispositivos en el cable gracias a los conectores que pueden crimparse (pincharse) en cualquier zona del cable y recibir las señales que se transmitan o escribir en él. Obviamente solo deben usarse varios esclavos con un protocolo que permita organizarlos para que los datos escritos en la línea no se pierdan. Aunque tradicionalmente se han utilizado este tipo de cables para conexiones en paralelo, en nuestro sistema se usará para conexiones serie.
- **Cable RJ45:** Este cable y su conector homónimo han sido muy extendidos como medio de enlace para el protocolo Ethernet. Una de sus principales ventajas es que se compone de cable de pares trenzados que, unidos a señales diferenciales, hacen que la inmunidad al ruido sea muy alta incluso para grandes distancias. Para conseguir una mayor inmunidad al ruido es beneficioso (incluso necesario) conectar ambos polos de una señal diferencial en el mismo par.

Por otro lado el estándar RS485 permite utilizar la UART con más de dos dispositivos en un mismo bus ya que proporciona a los dispositivos la capacidad de forzar una alta impedancia[2] en uno de los pines, RX o TX. El estado de alta impedancia es necesario para que dos o más dispositivos puedan escribir en una línea (de forma no simultánea), ya que el dispositivo que no está escribiendo

[2] La alta impedancia es un estado de un pin en el que este se comporta como si no estuviera conectado, como si se pusiera una resistencia de valor infinito en la conexión.

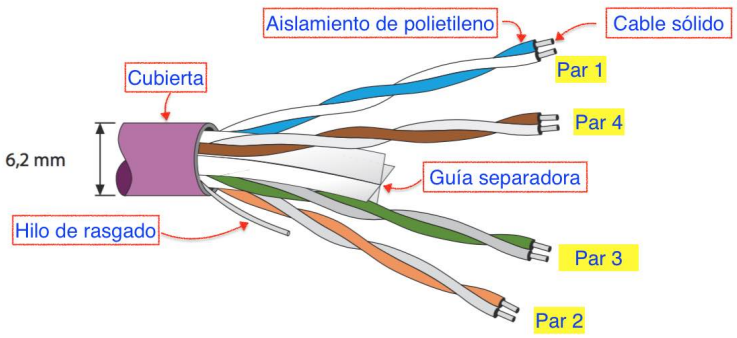
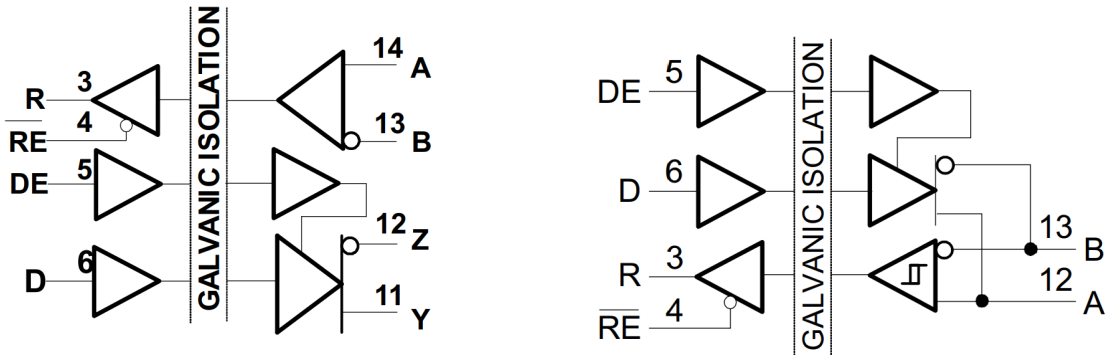


Figura 2.5 Cable de pares. Fuente: *https://elcajondelelectronico.com/cable-de-pares/*.

debería "desconectarse" momentáneamente de la línea de escritura, lo cual realiza forzando este estado de alta impedancia. Existen variantes full-duplex y half-duplex, como podemos ver en la figura 2.6. Este protocolo enviará sus bits sobre cable plano para asegurar una conexión cómoda entre los diferentes dispositivos. Utilizaremos dos GPIO para poder adaptar la señal de UART a RS485 con el dispositivo ISO3086 de Texas Instruments [18].



(a) Full Duplex.

(b) Half Duplex.

Figura 2.6 Estándar RS485 implementado por dispositivos ISO308x de Texas Instruments [18].

3 Punto de partida

A nivel de software no se parte de cero en este proyecto, sino que se establece como punto de partida un protocolo definido por Abraham Márquez, el tutor de este TFG, en su doctorado [16]. Este protocolo se ha estudiado, asimilado y modificado para funcionar acorde a la estructura del sistema, respetando siempre la estructura original en capas que se comenta a continuación. Por otro lado, se ha participado también puntualmente en el desarrollo de la PCB de control, aunque a nivel de hardware nos hemos centrado en los elementos necesarios para que esta se comunique a través de los diferentes medios.

3.1 Enfoque general

El protocolo se apoya en unas variables predefinidas de tipo mensaje el cual se describe en la figura 3.1. Contiene una cabecera de 4 octetos con los siguientes campos:

- ID: Identificación del módulo destinatario del mensaje.
- Length: Longitud del mensaje incluyendo la cabecera.
- Action: Tipo de acción que debe realizar el módulo destinatario: devolver datos, aplicar una salida...
- CRC8: Verificación de mensaje para evitar errores.

Los datos se distribuyen en grupos de 4 octetos los cuales se asumen de partida como flotantes, aunque esto no impide que se manden otros datos en el sistema final. Tenemos además dos buffers de entrada que se van turnando: si mientras se procesa el mensaje guardado en un buffer llega otro, este se guardará en el siguiente buffer. El procesamiento es mucho más rápido que la lectura pues la frecuencia del microcontrolador es mayor que la del periférico como comprobaremos al final de este capítulo, así que con estos dos buffers es suficiente para evitar pérdidas de datos.

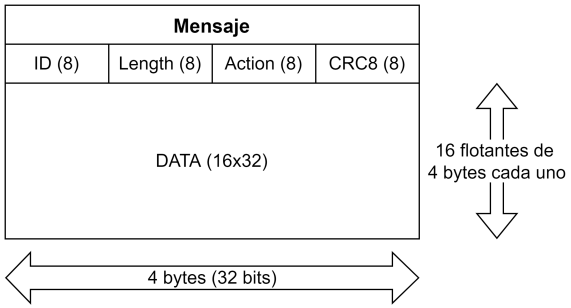


Figura 3.1 Campos de la variable tipo mensaje definida en el protocolo original.

El funcionamiento global se basa en varias capas que escriben y leen en variables globales, ya sean buffers o variables de tipo mensaje (tendremos en todo momento un solo mensaje de entrada y un solo mensaje de salida) y activan flags cuando lo hacen. También habrá capas que avisen a otras directamente a través de llamadas a funciones típicas del lenguaje de programación C[1].

La lectura está dividida en dos fases: una primera que comienza a partir de una interrupción en la que se almacenan los datos entrantes en un buffer, y una segunda donde se procesa este buffer. La escritura se hace completa una vez se inicia. En la figura 3.2 se muestra un esquema del protocolo completo, tanto la base como las modificaciones realizadas. Tanto el diseño del protocolo, como los mensajes que envía, como el funcionamiento fue elaborado por Abraham Márquez [16] y se han expuesto ligeramente en esta introducción para facilitar una mejor comprensión de las tareas que han sido realizadas.

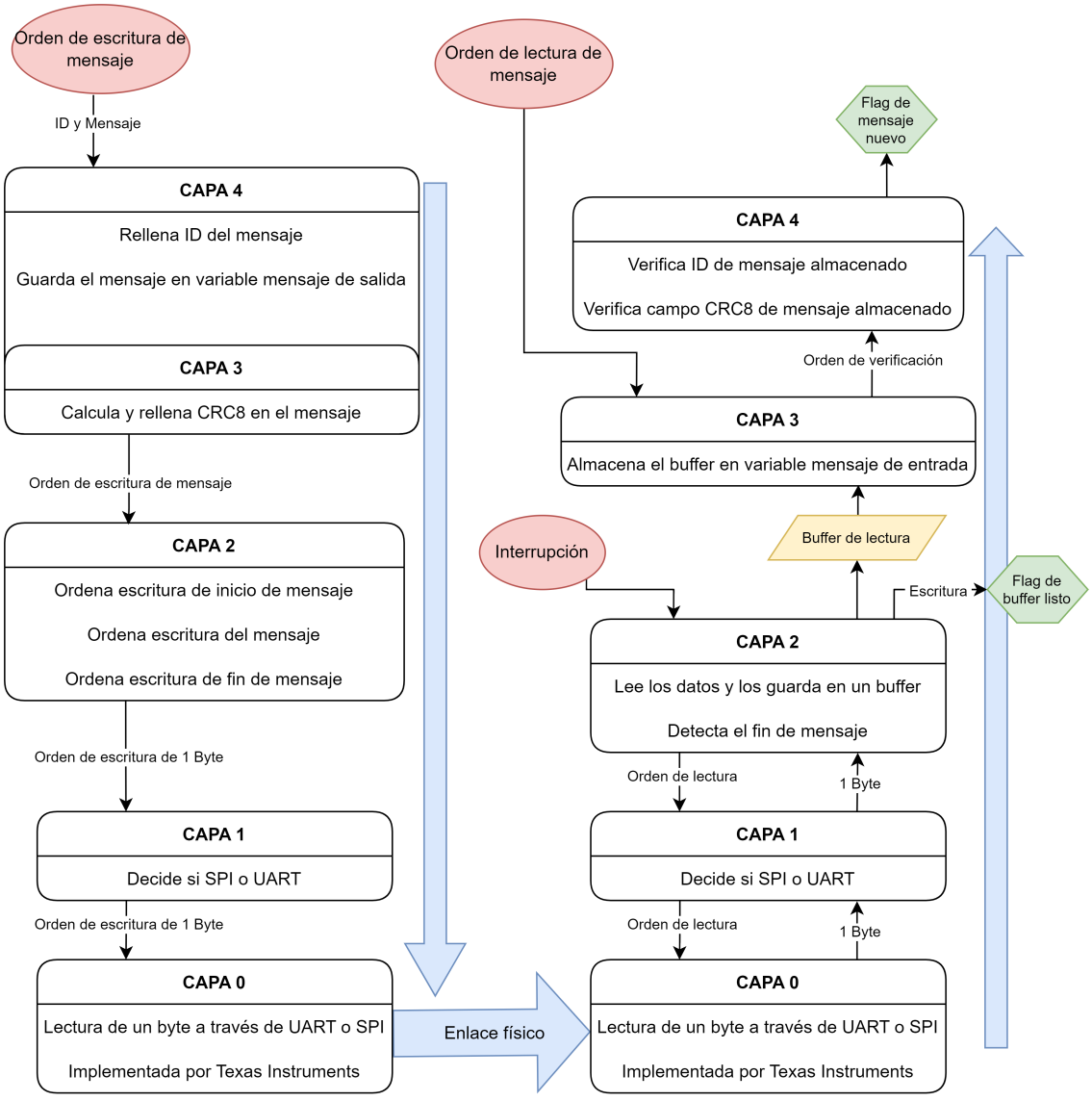


Figura 3.2 Estructura del protocolo de comunicaciones.

[1] El lenguaje C es el lenguaje de programación en el que se desarrolla el software de este trabajo

3.2 Modificaciones realizadas en el protocolo

Con una idea del protocolo de partida, se van a exponer ahora los cambios realizados que han tenido cuatro finalidades principalmente:

1. Adaptar el protocolo base a las funciones específicas de las librerías usadas y otros aspectos de la PCB específica.
2. Orientar las comunicaciones hacia esquema punto a punto. Se permitirán varios módulos de bajo nivel pero el módulo de control solo tendrá un puerto para comunicarse con todos ellos.
3. Realizar las pruebas que se exponen en este capítulo y el siguiente.
4. Mejoras leves del funcionamiento.

3.2.1 Capas 0 y 1

El protocolo estaba implementado para funcionar solo con UART. Hemos añadido la flexibilidad de poder decidir mediante una macro si las capas 0 y 1 envían y reciben los datos con UART o con SPI. Estas macros permiten también hacer una compilación condicional para configurar únicamente aquellos periféricos que vamos a utilizar. Veremos por qué esto no será necesario en el sistema final, aunque para realizar las pruebas nos ha sido realmente útil.

Comunicación full-duplex asíncrona con SPI

Para una comunicación rápida y eficaz, es necesario que, tanto el maestro como el esclavo, puedan mandar y recibir información en cualquier instante de tiempo, sin que esta se pierda en el camino. En la mayoría de los protocolos síncronos, el maestro puede escribir cuando desee, siempre que el esclavo esté esperando un mensaje o esté configurado para interrumpir su actividad y recibir los datos del maestro cuando este envía. Esto es posible porque el maestro proporciona la señal de reloj en la comunicación.

Si en un instante un esclavo quisiera mandar un mensaje al maestro, no podría hacerlo porque no hay señal de reloj en el bus (siempre proporcionada por el maestro). En el protocolo SPI, siempre es el maestro quien inicia la comunicación, de forma que, si solo quiere escuchar al esclavo, mandará a este datos inútiles para establecer el enlace entre ambos. Se plantean entonces dos alternativas:

- Suponer que solo existen comunicaciones del tipo pregunta - respuesta, donde el maestro solicita información o proporciona indicaciones y el esclavo devuelve la información requerida o datos inútiles para confirmar la recepción de la información (ACK[2]). Si forzamos nuestro protocolo a mantener este tipo de comunicaciones, estaríamos obligando al maestro a permanecer a la espera de la respuesta de un esclavo. Si el esclavo devuelve un ACK inmediatamente a la recepción no sería problemático. El problema sería si, para devolver información requerida, el esclavo necesita realizar cálculos, mediciones u otra actividad que pueda retrasar la respuesta. En este caso, el maestro quedaría a la espera un tiempo excesivo, posiblemente requerido para ejecutar acciones de control u otras comunicaciones.
- Por otro lado, se podrían definir unos instantes de tiempo, donde el esclavo tenga preparada la información que el maestro le requerirá. En este instante, el maestro inicia una comunicación con datos inútiles, y el esclavo devuelve lo que desee sin demora. Este escenario tiene tres grandes problemas entre otros:
  - El sincronismo: Maestros y esclavos necesitan estar de acuerdo en el instante en que se manda la información. Esto requiere que el maestro y el esclavo sean capaces de contar intervalos de tiempo con igual exactitud, siendo necesario que tuvieran relojes similares o un reloj múltiplo de otro.

[2] El bit de ACK (acknowledgement) es un bit que se manda en las comunicaciones por parte del dispositivo que recibe un mensaje para confirmar la recepción. En contraposición se mandara un NACK para notificar una mala recepción.

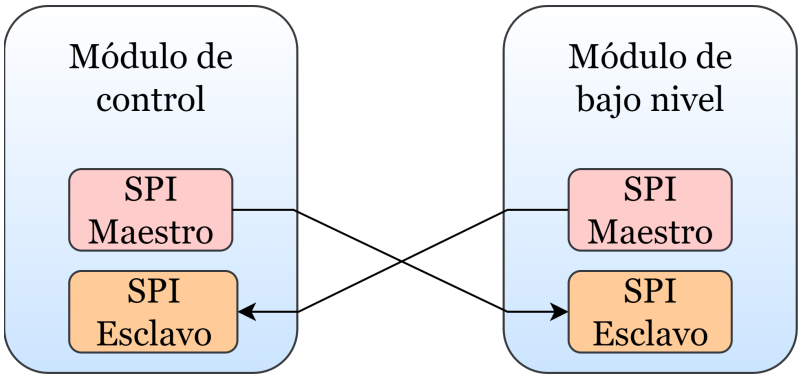


Figura 3.3 Dirección de las comunicaciones por protocolo SPI.

- El maestro no podría disponer de la información cuando quisiera. Solo podría hacerlo en los instantes que hemos mencionado, por lo que, la atención a estas comunicaciones sería una tarea prioritaria si no se quiere perder el contacto con los esclavos por varios ciclos, lo cual no es razonable en un maestro, cuya función principal suele ser ejecutar algoritmos de control.
- El problema mayor para nuestro algoritmo es el tiempo requerido para las comunicaciones. Cada vez que se quiera obtener información de un esclavo hay que mandar datos inútiles a él para establecer la comunicación, disminuyendo considerablemente la eficacia. Suponiendo que la cantidad de bit de datos que mandan el maestro y el esclavo es la misma (la cual no es una suposición alejada de la realidad ya que la mayoría de los protocolos implementados en la práctica funcionan así, con la misma configuración para maestro y esclavo), necesitaríamos mandar el doble de datos de los útiles para que la comunicación tuviera lugar.

Para no restringir la actividad de maestro o esclavo, de forma que puedan enviar información cuando deseen, se descarta realizar comunicaciones Esclavo a Maestro mediante el protocolo SPI. Sin embargo, queremos mantener las altas frecuencias que este protocolo nos ofrece, por lo que se opta por tener dos SPI. Estos SPI siempre ejecutarán comunicaciones Maestro a Esclavo, de forma que, tanto el módulo de control, como los módulos actuadores y sensores, contendrán un SPI en modo Maestro y otro en modo Esclavo. Podrán enviar en todo momento información a través del SPI Maestro y estarán preparados para recibir datos o indicaciones a través del SPI Esclavo.

Esta solución necesita mayor número de lineas, 3 por cada SPI. La linea de SOMI no se utilizará en ninguno de los dos SPI, ya que, como hemos comentado, el esclavo nunca devolverá información al maestro, por tanto puede no conectarse sin ocasionar conflictos. Sin embargo, en el Hardware diseñado se mantiene esta cuarta línea por si fuera necesario conectar un dispositivo externo a un SPI completo.

3.2.2 Capa 2

El funcionamiento está muy basado en punteros, sobre todo la lectura, ya que al tener dos buffers de entrada, es útil almacenar la dirección del primer elemento del buffer para conocer que buffer hay que procesar, en qué buffer hay que escribir los nuevos datos... Además de eso, los movimientos a través de los mensajes hacen uso de punteros. Por ejemplo en la escritura, tendremos un puntero señalando al inicio del mensaje y en la Capa 2 diremos a las capas inferiores que manden el dato apuntado por el puntero y que luego se incremente este puntero. Lo mismo ocurrirá para la lectura en el momento de guardar el buffer en una variable de tipo mensaje.

Ahora recordemos que siempre mandamos los datos en bytes, es decir, en grupos de 8 bits. Para que funcione bien la idea del párrafo anterior (mandar el dato de una posición de memoria y

aumentar el puntero), en esa dirección de memoria debe haber solo 8 bits. Dicho de otro modo, el ancho de palabra de la memoria debe ser de 8 bits para poder escribir y leer con el protocolo actual. Hemos remarcado tanto esta idea porque los microcontroladores que utilizamos tienen un ancho de memoria de 16 bits, por lo que hay que modificar esta capa.

Aunque una primera idea podría ser analizar la longitud del mensaje para conocer que tamaño de la memoria está implicado y enviar todos los datos que hayan en ella, eso no respetaría la separación entre capas que debe seguir un buen protocolo de comunicaciones, ya que estaríamos analizando el mensaje en capas bajas que no deben preocuparse por el contenido, solo realizar una lectura o escritura de los datos que le manden. Lejos de optar por este tipo de solución, se ha optado por una más sencilla, incluir un contador en cada función que recorría un mensaje a partir del puntero y solo aumentar el puntero cada dos aumentos del contador (cuando es par por ejemplo). De esta forma se pasa dos veces por la lectura o escritura sin aumentar el buffer, y realizando desplazamientos de bits es posible mandar el dato correctamente.

La figura 3.4 ejemplifica una escritura siguiendo el esquema que acabamos de comentar. Ha sido necesario seguir el mismo patrón para la lectura y para otras funciones del protocolo que se encargan de buscar el inicio y el final de un mensaje entrante.

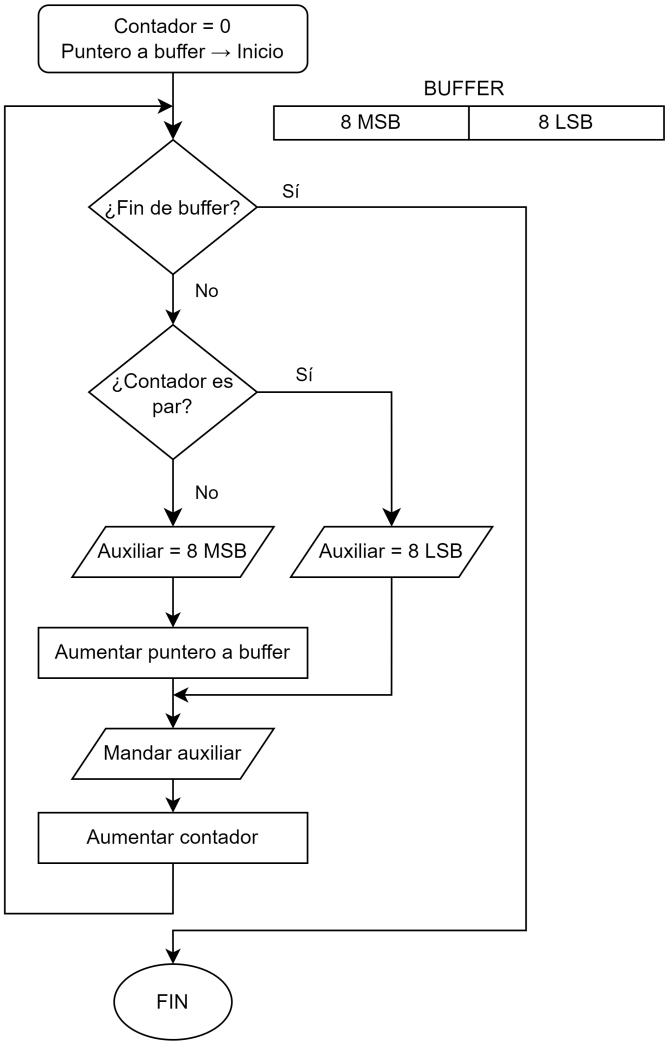


Figura 3.4 Ejemplo de escritura con ancho de memoria de 16 bits..

3.2.3 Capa 3

La rutina de interrupción de lectura planteada no contemplaba el caso en que el sistema leyera más rápido de lo que llegan los mensajes. Es decir, cada vez que llegaba una interrupción se comenzaba a escribir al principio del buffer, lo cual funcionaba bien a altas velocidades ya que, con analizar si había datos que leer antes de salir de la interrupción era suficiente, porque al ser tan alta la velocidad de envío siempre había datos esperando para ser procesados.

El problema ha aparecido al bajar la velocidad con el fin de realizar unas pruebas que expondremos más adelante. Con el funcionamiento programado, llegaba la interrupción con un byte, se guardaba al principio del buffer y se salía. Al llegar el siguiente byte, se sobreescribía el dato escrito anteriormente en el buffer (solo se pasa al otro buffer si se detecta fin de mensaje). Para que esto no ocurra, se guarda como variable global la posición del buffer por la que se van guardando los datos y en cada interrupción se comienza a escribir por ella. Si se detecta un fin de mensaje, esta posición se resetea al inicio del otro buffer de lectura.

3.2.4 Capa 4

A la hora de adquirir un mensaje, es necesario verificar que la ID escrita en la cabecera se corresponde con la identificación propia del módulo, para así comprobar que el mensaje va destinado al propio módulo y es necesaria una respuesta. Además, se detecta también una ID de broadcast, a través de la cual el módulo de control informa de que el mensaje va dirigido a todos los módulos de bajo nivel. Aprovechando que se va a analizar la cabecera en esta capa, extraemos de la capa 3 la verificación de CRC8 y lo pasamos a esta capa.

Si tenemos una capa de verificación de la ID en la lectura, parece razonable añadir una capa de escritura del campo ID en la otra rama del protocolo, por lo que se añade también una cuarta capa en la escritura encargada de fijar la ID y el CRC8.

3.3 Requerimientos iniciales de la PCB de control

Los requerimientos del sistema que se alejan de las comunicaciones prioritarias han sido definidos por el tutor del TFG Abraham Márquez y a partir de estos se ha trabajado para conseguir cumplir la mayor cantidad de ellos y de la mejor forma posible, ya que, no es viable pensar en comunicar varias PCBs con determinados protocolos si no se estudia la posibilidad real de incluirlo en un sistema donde las comunicaciones son tan importantes como muchos otros elementos.

- Comunicaciones prioritarias de alta velocidad entre el módulo de control y los módulos adyacentes. Para ello necesitamos periféricos de alta velocidad a través de los cuales se realizarán estas comunicaciones como ya se ha comentado.
- Comunicaciones no prioritarias que pueden funcionar a menor velocidad. Serán llevadas a cabo por el módulo de control hacia dispositivos de depuración del programa o servidores que almacenen los datos recogidos o algunas muestras de ellos. Pueden utilizar cualquier protocolo cotidiano, pues cuanto más estándar sean los puertos de comunicaciones más sencillo será trabajar con ellos, al poder colocar en el enlace mayor variedad de dispositivos. Se plantean principalmente como opciones el protocolo UART, el protocolo SPI y el protocolo I2C.
- Salidas de señales de PWM (Pulse Width Modulation), utilizadas por los módulos de bajo nivel para actuar sobre el sistema modular.
- Entradas analógicas con convertidores analógico a digital (ADC), utilizadas por los módulos de bajo nivel para obtener información del sistema modular.

- Entradas y salidas digitales versátiles (GPIO) que permitan crear buses de diversa funcionalidad como señales de habilitación, señales informativas de error, señales de reloj y de reset...

3.3.1 Uso del protocolo I2C

Como tenemos más requerimientos que pines accesibles, el protocolo I2C se utilizará para controlar uno o varios expansores de pines de entrada y salida (GPIO). Es decir, este protocolo no se utilizará para comunicaciones prioritarias entre el módulo de control y los módulos de bajo nivel, sino que se destinará a otras comunicaciones de baja prioridad. En concreto se utilizará el PCA9538 [19], que permite manejar hasta 8 pines de entradas y salidas como un dispositivo esclavo del protocolo SPI. Es posible variar su dirección mediante hardware, de forma que permite conectar hasta 4 PCA9538 en el mismo bus I2C, además de otros dispositivos con direcciones diferentes. Esta opción se da gracias a que la dirección de 7 bits de estos módulos viene determinada por 5 bits prefijados y 2 bits variables que permiten modificar la dirección. De esta forma obtenemos hasta 32 pines de entrada o salida independientes a partir de 2 pines de la PCB (SDA y SCL del protocolo I2C), pudiendo añadir más esclavos. Para utilizar el dispositivo con comodidad se ha elaborado una librería que permite configurar fácilmente el dispositivo, escribir sus salidas, leer sus entradas e identificar cambios mediante su salida de interrupción. Esta librería ha sido documentada con la herramienta Doxygen[3].

[3] Doxygen es un software de código abierto utilizado para generar documentación de código a partir de los comentarios que se colocan en este con un determinado formato

4 Diseño de Hardware

Para conseguir un sistema versátil, práctico y con requisitos de funcionamientos tan importantes como la velocidad de transmisión, es importante analizar la rama del Hardware. Hemos definido distintos diseños de PCBs intercambiables y con funcionalidad muy variada, haciendo estudios de los dispositivos que contienen (a nivel de potencia consumida, disponibilidad, tensiones de funcionamiento necesarias...).

4.1 Compatibilidad

Uno de los aspectos más importantes de nuestro sistema es la versatilidad entre módulo central y módulos de bajo nivel, de forma que se pretende que cualquier módulo de bajo nivel pueda ejercer como módulo central, o al menos que tenga las herramientas para ello. Incluso existe la posibilidad de que, en versiones futuras, sea uno de los módulos de bajo nivel el que realice tareas de control. Esto implica que la PCB de control debe tener las conexiones necesarias para funcionar también como PCB de bajo nivel, es decir, enlaces con las PCB de los actuadores y los sensores (que se usarán si se funciona como módulo de bajo nivel), además de las PCBs de comunicaciones (que serán usadas por el módulo de control). Debido a ello, en este documento hablaremos de PCB de control aunque pueda ser utilizada para otras tareas que no sean el control, como el manejo directo del convertidor modular. En el apartado 4.2.1 analizaremos por qué es bueno separar los periféricos de comunicaciones de la PCB de control, razonamiento que es extrapolable a las PCBs de sensores y actuadores.

4.1.1 Compatibilidad entre PCB de control y PCB de bajo nivel

Como se ha comentado en la introducción, tendremos una única PCB llamada PCB de control que podrá realizar tanto las funciones de control como las funciones de bajo nivel. Con este fin contendrá tanto dispositivos de comunicaciones prioritarias y no prioritarias (SPI, UART e I2C), como periféricos que proporcionan señales de actuación (PWM) y periféricos de recepción de información (ADC), además de múltiples entradas y salidas de propósito general (GPIO).

4.1.2 Compatibilidad entre LaunchPads

Por otro lado, tenemos dos tipos de LaunchPads que queremos conectar a la PCB de control: la LaunchXL-F28379D y la LaunchXL-F28377S . Ambas contienen microcontroladores de la familia C2000 Delfino, que según fabricante son soluciones con mayor capacidad de procesamiento, memoria y periféricos de baja potencia, además de tener capacidad de realizar controles más eficientes. Además, han sido seleccionadas por el tutor del TFG ya que su periférico PWM es muy

versátil y permite variar en tiempo real el valor y la fase, lo cual es ideal para el control avanzado de sistemas modulares.

Para conseguir colocar los periféricos en la PCB de control de forma que sean perfectamente funcionales independientemente de la LaunchPad que se coloque encima, en este punto del proyecto se trabajó sobre una tabla de gran tamaño que recogía las funciones interesantes para el sistema que cada pin físico podía tener en alguna de las dos LaunchPads. Con pin físico nos referimos a los terminales que salen hacia fuera, en los que se puede conectar un dispositivo. No todos estos pines tienen alguna función (algunos no están conectados al microcontrolador) y no todos los periféricos del microcontrolador tienen salida hacia un pin físico. Obviamente, la mayoría de pines físicos pueden realizar diferentes actividades según se configuren por software. Por ejemplo, en la LaunchPad F28377S tenemos 4 UARTs independientes, aunque solo 3 de ellas están conectadas a uno o varios pines físicos, y estos pines físicos también pueden usarse para GPIO u otras funcionalidades detalladas en la hoja de características.

Si tratamos de abarcar el máximo de elementos posibles en los 80 pines físicos que tiene cada LaunchPad tendríamos:

- 30 pines de ADC (entradas analógicas para recibir datos en el módulo de bajo nivel).
- 12 pines de PWM (salidas del módulo de bajo nivel para actuar sobre el convertidor).
- 2 SPIs (4 pines cada uno).
- 2 UARTs (2 pines cada uno).
- 2 I2Cs (2 pines cada uno).
- 10 pines de propósito general con capacidad para interrumpir a la CPU.
- 1 pin de propósito general sin capacidad para interrumpir a la CPU.

Sin embargo, el pin STE del protocolo SPI ha resultado ser conflictivo por no tener una ubicación compatible entre ambas LaunchPads. Por un primer momento parecía que no era importante, pues a priori parecía funcionar como un Chip Select, por lo que no sería necesario en comunicaciones punto a punto. Aunque las interrupciones se regían por este pin, se podía eliminar por software y provocar que las interrupciones se detectaran a partir del pin de entrada SIMO. Sin embargo, a altas frecuencias no se conseguía un buen funcionamiento al no ser las condiciones óptimas de funcionamiento del periférico.

Por otro lado, en la escritura no debería ser necesario el STE ya que, cada periférico SPI implementado en los microcontroladores solo tiene un STE, lo cual no permite comunicaciones desde un master a varios esclavos, ya que necesitaríamos tantas líneas de esclavos como número de esclavos tengamos (figura 4.1). La solución que el fabricante aporta este tipo de comunicaciones es la utilización de pines GPIO como STE, activándolos antes de comenzar una transmisión y desactivándolos al finalizar. Esto no funciona bien a grandes velocidades por un motivo fundamental: no se conoce con certeza el instante en el que se termina de mandar un dato. La escritura y la lectura se hace mediante el acceso a unos registros, los cuales son mandados por el periférico al detectar escritura en ellos o almacenan la información leída. El problema reside en que, después de meter un dato en el registro de escritura, no es posible determinar el instante en el que ese registro termina de ser copiado en el buffer. A bajas velocidades no sería necesario conocer el instante concreto, pero a tan altas velocidades hemos comprobado que colocar el pin STE en un GPIO no es una opción viable.

Finalmente se decide colocar en la PCB de control 4 pines de STE, uno para cada SPI y para cada LaunchPad, según podemos ver en la tabla 4.1. Esto nos fuerza a eliminar un I2C, lo cual no es problemático en principio al ser I2C un protocolo para buses con más de dos dispositivos, o sea, que permitiría conectar varios dispositivos al mismo periférico de I2C. Además, el pin de propósito general sin capacidad para funcionar por interrupciones se utilizará como habilitador de salida del protocolo RS485.

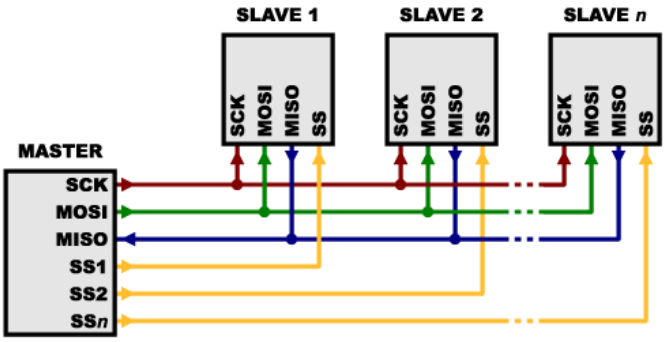


Figura 4.1 Señales en una comunicación SPI con varios esclavos. Fuente: *https://www.prometec.net/bus-spi/*.

Tabla 4.1 Pines con función de STE (Protocolo SPI).

	SPI maestro	SPI esclavo
LaunchXL-F28377S	Pin 50	Pin 8
LaunchXL-F28379D	Pin 59	Pin 19

4.1.3 Composición final de la PCB de control

- Un SPI maestro de alta velocidad.
- Un SPI esclavo de alta velocidad.
- Una UART de alta velocidad.
- Una UART de baja velocidad rutada a un conector externo y hacia un módulo conversor de UART a Ethernet, para posibles comunicaciones con un servidor remoto.
- Un I2C.
- Dos expansores de entradas y salidas a través de I2C conectados al mismo periférico.
- 30 pines de ADC.
- 12 pines de PWM.
- 2 pines para añadir a la UART de alta velocidad el protocolo RS485: un habilitador de escritura y un habilitador de lectura.
- Un switch de piano para configuración de los diferentes periféricos que dependen de la LaunchPad conectada y el protocolo elegido para las comunicaciones. Este switch de piano se conectará a uno de los expansores de pines.

4.2 Enlaces físicos

Cuando hablamos de comunicaciones rápidas o de alta frecuencia, influye también el medio en el que se propaga la señal. Para probar nuestro protocolo utilizaremos tres enlaces físicos diferentes ya comentados: cables de fibra óptica, cable plano flexible y cable RJ45. Las comunicaciones inalámbricas no se han contemplando ni suelen ser una opción en este tipo de proyectos por varios motivos entre los que destacamos: el coste de los dispositivos de emisión y recepción de señales, la alta fiabilidad que se necesita en estas comunicaciones, siendo necesario reducir al máximo las pérdidas o corrupción de los datos, y la alta velocidad de trasmisión que se requiere, características difíciles de conseguir en comunicaciones inalámbricas rápidas en entornos industriales.

4.2.1 PCBs de comunicaciones

A modo de puente entre el módulo de control y las señales que se transmiten por los enlaces físicos que se han comentado, se han diseñado unas PCBs auxiliares encargadas de tratar la señal para adecuarla al medio de comunicación correspondiente. Estos diseños han sido fabricados en el laboratorio y hemos comprobado su funcionamiento correcto en varias situaciones, para posteriormente solicitar su fabricación. Las ventajas de usar estas PCBs en lugar de instalar los módulos necesarios en la PCB de control son varias:

- En primer lugar permite un ahorro de espacio en la PCB de control, que no se traduce en una reducción de su tamaño, sino en la ocupación de ese espacio por otros dispositivos, haciendo la PCB más versátil y completa. Dado que el tamaño de la PCB de control viene fijado (al menos en una dimensión) por el rack en el que se instalará, trasladar dispositivos que no se vayan a usar en todo momento a otras PCBs que se puedan conectar cableadas permite liberar espacio para otros elementos que no tienen esta posibilidad.
- Las PCBs de comunicaciones proporcionan aislamiento galvánico, es decir, no hay contacto eléctrico entre el conector de la PCB de control y el conector del enlace físico. Así se evita que dispositivos de control críticos o sensibles puedan sufrir daños por las corrientes de los actuadores. Tener los dispositivos que crean el aislamiento en PCBs separadas permite situar visualmente el aislamiento y cerciorarnos de que existe, además de ahorrar espacio como ya hemos comentado. En ocasiones el aislamiento lo proporciona el propio adecuador de la señal como se verá más adelante. En contraposición, la fibra óptica no necesita aislamiento ya que no establece contacto eléctrico de forma intrínseca.
- La alimentación a estos dispositivos debe ser proporcionada por la PCB de control, pero para mantener el aislamiento (ambas zonas de la PCB de comunicación deben alimentarse sin romper el aislamiento galvánico) y asegurar los niveles de tensión se utilizarán transformadores DC-DC que realizan esta función. Si no tuvieramos las PCBs auxiliares, sería otro dispositivo más que deberíamos colocar en la PCB de control.
- Probablemente el beneficio más importante reside en que si un protocolo de comunicaciones no se está utilizando, su PCB puede desconectarse reduciendo el espacio que ocupa todo el dispositivo y la potencia que debe proporcionar la PCB de control.

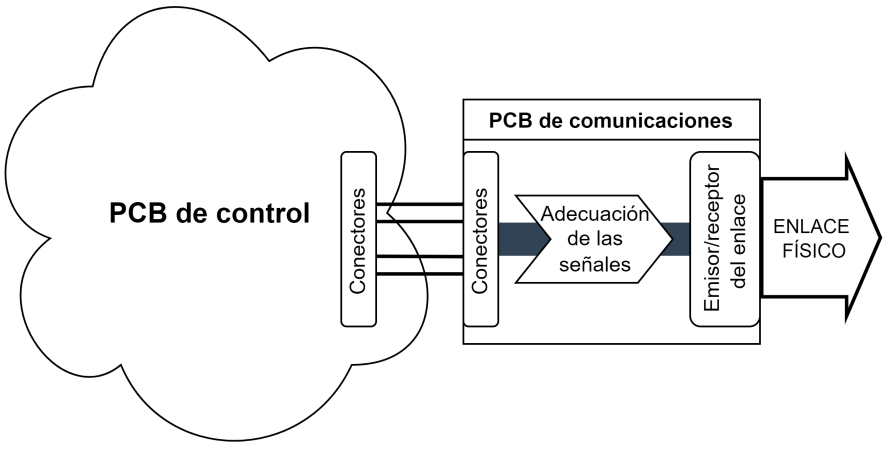


Figura 4.2 Esquema funcional de las PCBs de comunicaciones.

En concreto se han creado tres modelos de PCBs de comunicaciones, uno por cada medio (fibra óptica, cable plano o cable RJ45), que se utilizarán por pares imitando el esquema de la figura 4.3.

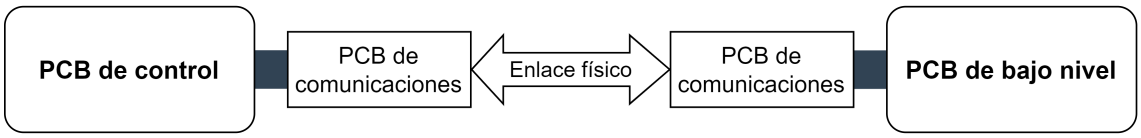


Figura 4.3 Esquema abstracto de la línea de comunicación.

1. El primer modelo, y más simple por tener menos componentes necesarios, se utilizará para mandar y recibir señales por fibra óptica del protocolo UART. Además de no tener aislamiento, se llevan a cabo transmisiones y recepciones directas de los datos, es decir, las señales TX y RX del micro se traducen a señal luminosa con un transmisor y se convierten de vuelta a señal eléctrica mediante un receptor, sin incluir protocolos físicos (adicionales a UART) o interfaces intermedias. Los transmisores y receptores de fibra óptica así como el propio cable tienen un coste elevado, por lo que reducir al máximo el número de señales que van a utilizar este medio es muy conveniente.

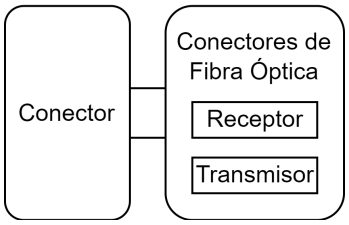


Figura 4.4 Esquemático de PCB de comunicaciones por Fibra Óptica con protocolo UART.

2. En segundo lugar, tenemos el modelo del cable plano, que también se utilizará para mandar señales del protocolo UART. En este caso hemos añadido a la pila de protocolos el estándar RS485. Las principales características que nos motivan para incluirlo son: la inmunidad al ruido electromagnético al manejar señales diferenciales y la posibilidad de conectar varios esclavos en una misma línea. Para ello necesitamos transformar nuestras dos líneas de UART en líneas diferenciales y los módulos de bajo nivel deben forzar un estado de alta impedancia a su salida para posibilitar que otros esclavos puedan escribir en la misma línea, tal y como se vió en la sección 2.2.1. Para satisfacer estos requerimientos se usa el dispositivo ISO3086 de Texas Instruments [18], el cual es un transmisor-receptor de RS485. Además, este dispositivo proporciona aislamiento galvánico entre las líneas que salen de los módulos y las líneas diferenciales, tal y como se ve en la figura 2.6.

   Todos los diseños de PCBs han sido comprobados y mejorados por el tutor de este Trabajo de Fin de Grado, Abraham Márquez, cuya experiencia en este campo es abundante y necesaria. Por ejemplo, en la PCB de RS485 hemos incluído un filtro RC que evita bajas frecuencias (paso baja) por si fuese necesario. De la misma forma, en esta PCB se han añadido resistencias de terminación al final del cada bus diferencial conectando ambas líneas.

3. Por otro lado, el tercer modelo utiliza cable RJ45 para el protocolo SPI. Este modelo a su vez se divide en tres: uno receptor, uno transmisor y otro que engloba recepción y transmisión (es el resultado de unir las dos primeras en una PCB). Hablaremos en esta sección de las PCBs de recepción o transmisión. Son las más complejas en cantidad de elementos, ya que no contamos con un único dispositivo que modifique las señales a modo diferencial y además aporte aislamiento. Se utilizarán en su lugar dispositivos que pasan una sola señal a diferencial

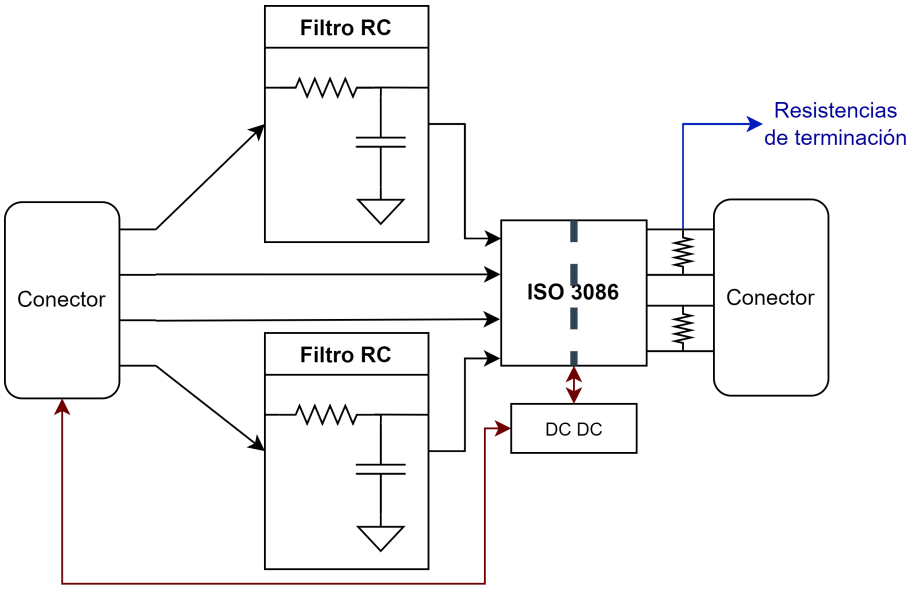


Figura 4.5 Esquemático de PCB de comunicaciones por cable plano con estándar RS485.

o viceversa (llamados "Transmisores y receptores diferenciales" en la figura 4.6), siendo necesario uno de ellos por cada señal que queramos transmitir (las 4 señales del protocolo SPI). Por otro lado, el aislador galvánico no es bidireccional, sino que tiene definidos aquellos pines que son entradas y cuales salidas. Recordemos que en el protocolo SPI tenemos 3 señales que viajan desde el maestro hasta el esclavo (MOSI, CLK y STE) y una señal que viaja desde el esclavo hacia el maestro (MISO), por lo que debemos tener un aislador de al menos 4 señales que permita 3 señales en una dirección y otra señal en dirección opuesta. También

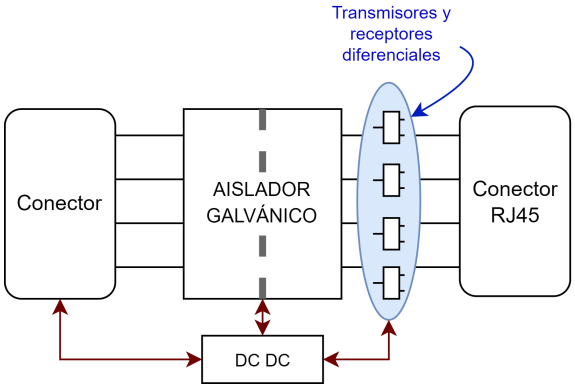


Figura 4.6 PCB de comunicaciones por RJ45 con protocolo SPI.

es importante conocer la dirección de las señales para analizar si necesitamos un transmisor o un receptor diferencial. Colocaremos un transmisor diferencial cuando queramos recibir como entrada una señal que viaja en un solo conductor y proporcionar una señal diferencial, mientras que será necesario un receptor si la entrada es diferencial y necesitamos pasarla a un solo conductor. Estos dispositivos se colocarán en los extremos de los buses diferenciales. Para comprender mejor la idea reflejada tenemos las figuras 4.7 y 4.8.

A pesar de que el número de dispositivos a incluir en esta PCB ha aumentado considerablemente, la complejidad mayor no está todavía expuesta. Como se verá en la sección 4.2.2, la misma PCB puede utilizarse tanto para entrada como para salida en protocolos UART, girando

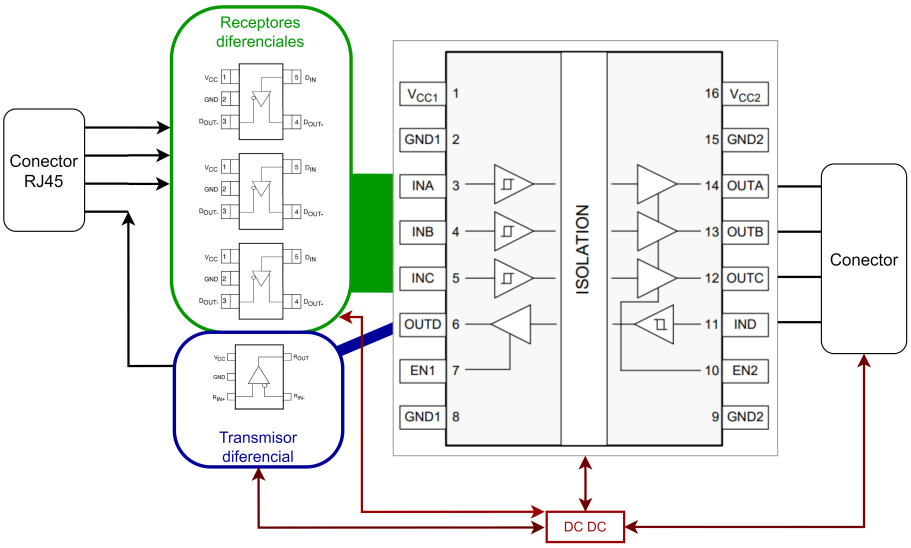


Figura 4.7 PCB de SPI receptor (esclavo).

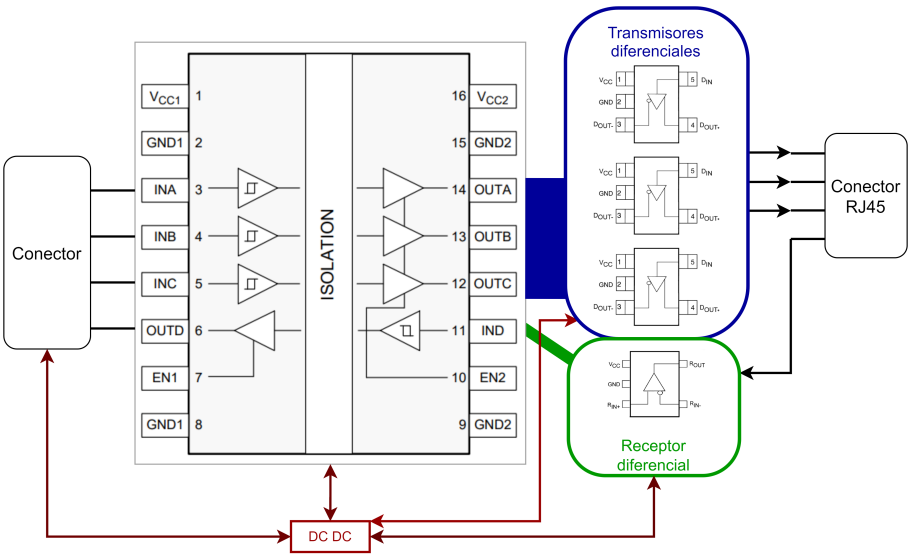


Figura 4.8 PCB de SPI transmisor (maestro).

los cables de forma que los pines conmuten, conectando la entrada con la salida. Sin embargo, en SPI esto no ocurre: recordando la sección 2.1.3, los pines de SPI cambian su dirección según el dispositivo que lo controla sea maestro o esclavo, así la señal de reloj (CLK) siempre sale del maestro o el MISO siempre sale del esclavo. Por tanto, el lado del maestro tendremos 3 señales que salen y 1 que entra, que vistas desde el otro extremo (esclavo) serán 3 señales entrantes y una saliente. Es por este motivo por el que se necesitan dos modelos diferentes de PCBs para este tipo de comunicaciones, como hemos comentado al inicio de este punto y se aclara en las figuras 4.7 y 4.8, cuyos aspectos más relevantes son:

- Cada PCB lleva transmisores o receptores diferenciales según la dirección del pin, es decir, según se utilice en el extremo del maestro o del esclavo.
- El aislador llevará una determinada orientación, según necesitemos 3 señales de entrada o 3 señales de salida. Observar que la posición de los conectores cambia de lado en las figuras ya que cambia el sentido de la comunicación.

Como nuestro sistema incluirá dos SPI con direcciones opuestas, o sea, un maestro y un esclavo, se fabricará el tercer sub-modelo (que une los dos primeros) con los conectores de SPI maestro y esclavo colindantes, y se dispondrán también de esta forma en la PCB de control, de forma que, para uso normal del protocolo como hemos previsto, se podrá utilizar esta PCB y será más cómodo por tener un conexionado un poco más simple.

Como último aspecto importante tenido en cuenta a la hora de diseñar esta PCB cabe destacar la disposición de los pares de cables que componen el cable RJ45. Los conectores de RJ45 en los que terminan los cables homónimos tienen conectados los pares como podemos ver en la figura 4.9. Para nuestra aplicación no es relevante el número de par que se coloca en cada pin, por lo que no vamos a estudiar las normas a las que se hace referencia en la imagen. Sin embargo, sí debemos tener en cuenta que el cable que utilicemos tenga el conector unido de la misma forma en ambos extremos. Para el diseño es importante esta imagen porque como ya hemos comentado, es beneficioso para señales diferenciales que ambas componentes estén contenidas en el mismo par, aspecto a tener en cuenta en la colocación de elementos en la PCB y en el rutado.

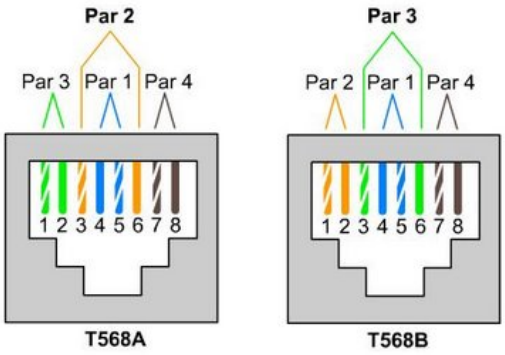


Figura 4.9 Disposición de los pares en un cable RJ45. Fuente: *https://mundotelecomunicaciones1.blogspot.com/2014/10/como-ponchar-un-cable-utp-con-un.html*.

4.2.2 Conexiones de PCBs de comunicaciones

Tanto la fibra óptica como el cable plano, permiten una conmutación sencilla de las líneas, de forma que el pin de recepción de un extremo puede conectarse con el pin de transmisión del otro y viceversa, aunque con el cable plano no es trivial ver el giro necesario del cable al ser mayor el número de señales que viajan por el bus y la cantidad de opciones diferentes de conexionado. Esto se ha tenido en cuenta en el diseño de la PCB correspondiente, de forma que, para facilitar el conexionado se han colocado las señales como indica la figura 4.10, de forma que cada señal se corresponde con una columna. Si suponemos ahora que solo utilizamos cuatro de las cinco filas del conector, que ocurrirá en la mayoría de las ocasiones, para conmutar todas entre sí bastaría con girar 180° el conector.

En el Apéndice A se detallan los aspectos más relevantes que se deben tener en cuenta a la hora de conectar las diferentes PCBs del sistema.

Por su parte el SPI no requiere una conmutación de lineas por su propia construcción, pero debemos recordar que tenemos 2 SPI en cada nodo del sistema, por lo que sí será necesario una conmutación de buses de forma que nunca se conecten entre sí dos maestros o dos esclavos. Es importante distinguir que no se están conmutando líneas del propio periférico, como sí se hace con la UART, donde la transmisión debe conectarse con una recepción. En SPI esto no ocurre, no existe una señal de recibir y otra de transmitir, sino que se etiquetan según el dispositivo sea maestro o

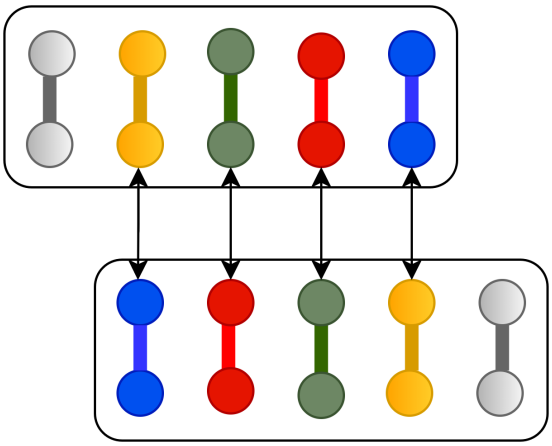


Figura 4.10 Rutado entre pines del conector de cable plano en la PCB de RS485.

esclavo, de forma que existe una línea de envío del maestro y recepción del esclavo, y otra línea de recepción del maestro y envío de los esclavos.

5 Pruebas del hardware diseñado

Los datos que se van a exponer a continuación son resultado de múltiples experimentos en los que se ha variado la velocidad de transmisión y la longitud de los datos para obtener los siguientes tiempos (figura 5.1):

1. Tiempo total: Es el tiempo que se emplea en mandar y recibir un mensaje. Se dispara el temporizador al inicio, el módulo de control manda un mensaje y recibe otro de vuelta, el cual procesa antes de parar este temporizador. Si el módulo de control esperara a recibir una respuesta antes de realizar otras tareas, este sería el tiempo total que el módulo estaría dedicado al envío de un mensaje y recepción de su respuesta.
2. Tiempo de envío: Tiempo necesario para crear el mensaje y mandarlo. Es el tiempo real que está el microcontrolador ocupado en el envío de un mensaje. Transcurrido este tiempo puede que el mensaje no esté en el bus al completo, pero sí ha sido escrito en los registros del periférico correspondiente (UART o SPI) y el microcontrolador ya se dedica a otras tareas, siendo el periférico correspondiente el encargado de terminar el envío.
3. Tiempo de recepción y procesado: Intervalo que se necesita para recibir el mensaje y procesarlo (guardarlo en una variable de tipo mensaje, analizar la cabecera para ver si el módulo es el destinatario y avisar al sistema en caso afirmativo).
4. Tiempo de recepción: Es el tiempo durante el cual se está copiando en el buffer de entrada los datos de un mensaje entrante.

Estos tiempos han sido obtenidos en cada condición de funcionamiento al menos 7 veces, lo que ha permitido controlar con exactitud las velocidades de transmisión comprobando la repetitividad de los experimentos. Además, en cada iteración se ha controlado que los mensajes que viajan en ambas direcciones sean idénticos en ambos extremos, o sea, que las transmisiones hayan sido exitosas. En este apartado estableceremos comparativas entre los distintos protocolos, pero con anterioridad a eso vamos a mostrar los conjuntos de medidas de todas las repeticiones de algunos experimentos. Las medidas se representan en milisegundos y están tomadas con un reloj de precisión igual a 0,04 microsegundos, o visto de forma inversa, con un reloj de frecuencia 25MHz. La longitud máxima de los datos prevista por el protocolo es de 16 flotantes, ocupando cada flotante 4 octetos, por lo que la longitud máxima en número de octetos es de 64. Se realizarán pruebas mandando desde 5 flotantes hasta 15.

Dos de las gráficas (figuras 5.5 y 5.6) que se muestran aparentan tener solo un experimento, sin embargo en todas hay datos de 7 experimentos, lo que ocurre es que en esos casos el temporizador cuenta exactamente el mismo número de ciclos. Esto tiene sentido ya que se corresponden con el tiempo que dura un envío y el tiempo que dura una recepción. Estas operaciones, una vez iniciadas no son interrumpidas nunca, siempre se espera a mandar o recibir un mensaje completo antes de

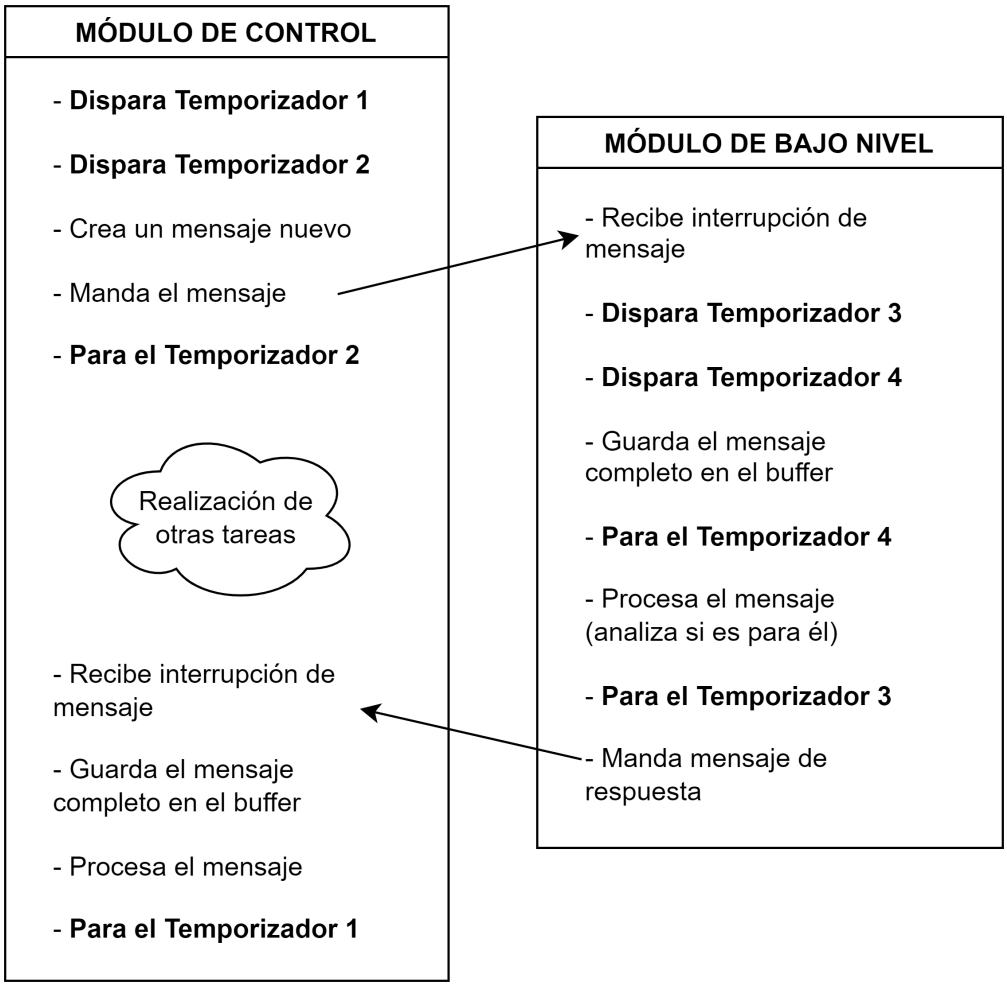


Figura 5.1 Intervalos de tiempo medidos en los experimentos.

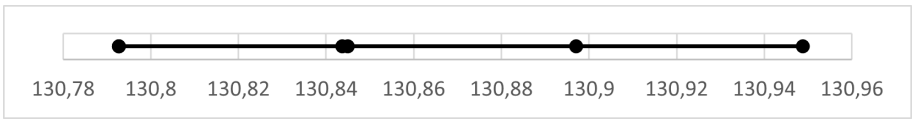


Figura 5.2 Tiempo desde que el módulo de control crea el mensaje hasta que procesa uno de respuesta. Temporizador 1. RS485. 9600 baudios. 5 bytes de datos.

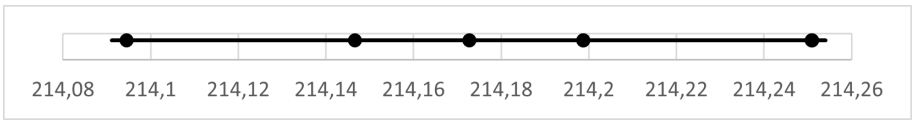


Figura 5.3 Tiempo desde que el módulo de control crea el mensaje hasta que procesa uno de respuesta. Temporizador 1. RS485. 9600 baudios. 15 bytes de datos.

pasar a otra tarea, por lo que es coherente que los microcontroladores midan siempre los mismos tiempos para cada experimento, pues el número de tareas siempre es el mismo, al realizarse en todo momento la más prioritaria.

Sin embargo, si entramos en la creación del mensaje y en su adquisición vemos leves variaciones (figuras 5.2, 5.3 y 5.4. Estas variaciones del orden de las décimas de milisegundo en el caso más desfavorable pueden deberse a que el microcontrolador no esté realizando exactamente las mismas

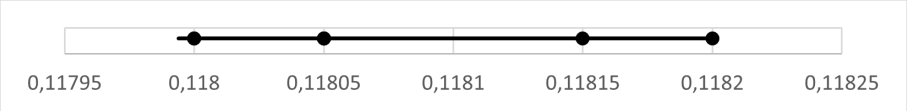


Figura 5.4 Tiempo desde que se recibe la interrupción hasta que se procesa el mensaje. Temporizador 3. RS485. 12,5 Megabaudios. 12 bytes de datos.

Figura 5.5 Tiempo desde que se comienza la recepción del mensaje hasta que el último byte entra en el buffer. Temporizador 4. RS485. 115200 baudios. 15 bytes de datos.

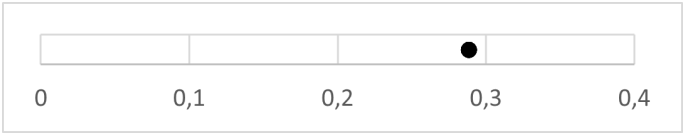


Figura 5.6 Tiempo que transcurre desde que se comienza el envío hasta que se finaliza. Temporizador 2. RS485. 2,5 Megabaudios. 10 bytes de datos.

tareas, o a estados diferentes de las FIFOs, etc. Sin embargo, estas diferencias son tan pequeñas respecto a las comparaciones que vamos a hacer a continuación que no supone ningún problema.

5.0.1 SPI

En los experimentos de SPI se utilizarán como conexiones cables de puente, ya que los transmisores diferenciales no han sido recibidos a tiempo por parte del distribuidor. En las gráficas 5.7 y 5.8 podemos visualizar una comparación entre los tiempos comentados anteriormente. Podemos apreciar que el tiempo total transcurrido desde que se crea el mensaje hasta que se procesa una respuesta es mucho mayor que los otros tres medidos ya que estarían englobados dentro del primero. Podemos destacar que mandar un mensaje de 15 flotantes (que será como los más largos que se manejarán aproximadamente) a máxima velocidad a través de SPI (50 MHz) conlleva un tiempo de 1,2 milisegundos, si se espera una respuesta. Como en realidad no se espera la respuesta, sino que se recibe por interrupción, **el microcontrolador realmente está ocupado** (sin contar la creación del mensaje) la suma del tiempo de envío con el tiempo de recepción y procesado: **menos de 0,4 milisegundos**. El tiempo total es excesivo para los requisitos de comunicaciones que habíamos establecidos, pero no se deben a las frecuencias de los protocolos, sino a la frecuencia máxima del microcontrolador como veremos más adelante.

Es destacable también en ambas gráficas el reducido tiempo de procesado de un mensaje. Es decir, el tiempo que se tarda en recibir un conjunto de datos, copiar el buffer con los datos en una variable de tipo mensaje y decidir si es un mensaje entrante está muy cercano al tiempo de recepción únicamente. Cuando el protocolo se integre en el sistema, existirá un tiempo entre la recepción del mensaje y su procesado, ya que la CPU normalmente estará ejecutando otras tareas que finalizará antes de procesar un mensaje nuevo.

Si comparamos las gráficas 5.7 y 5.8 que reflejan los tiempos de envío en función de la longitud de mensaje, podemos apreciar que, a velocidades altas (50 MHz), la cantidad de datos no influye tanto en el tiempo de envío como sí lo hace conforme se reduce la velocidad, por ejemplo a 1 MHz. Esto es coherente ya que, conforme bajamos la velocidad, los retrasos introducidos por querer mandar grandes cantidades de datos repentinamente (un mensaje de 15 flotantes por ejemplo) serán mayores. Visto desde otro punto de vista, el microcontrolador sigue funcionando a alta frecuencia y manda

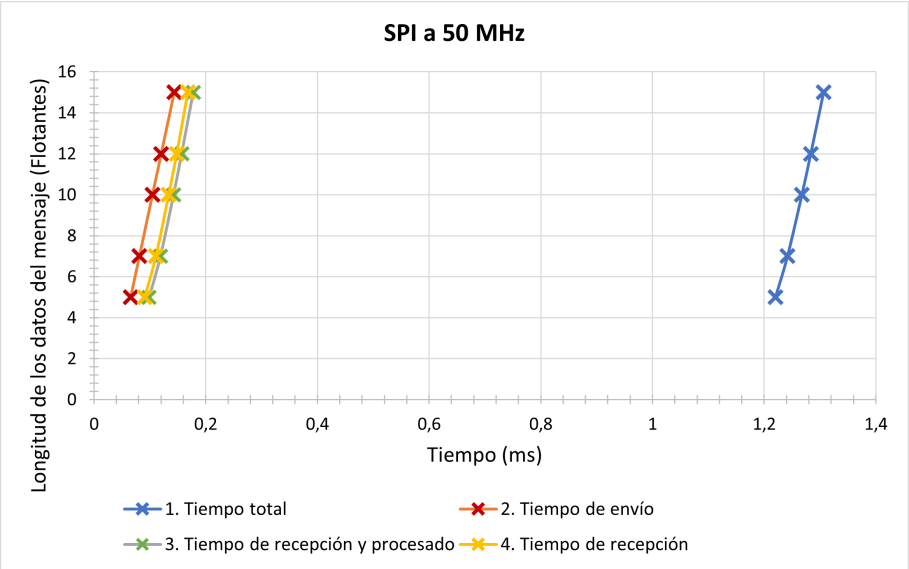


Figura 5.7 Tiempos que miden los cuatro temporizadores según la longitud del mensaje a 50MHz.

datos al periférico de salida a gran velocidad. Conforme estos datos llegan se van acumulando en la FIFO a la espera de que la salida esté desocupada, hasta un máximo de 16 octetos y el propio protocolo se encarga de frenar el envío si la FIFO está completa para evitar pérdida de datos. Lo que ocurrirá para frecuencias de envío y recepción de 1 MHz es que la velocidad de salida está restringiendo la velocidad de funcionamiento del microcontrolador, es decir, lo está frenando para que los datos no colapsen en el bus.

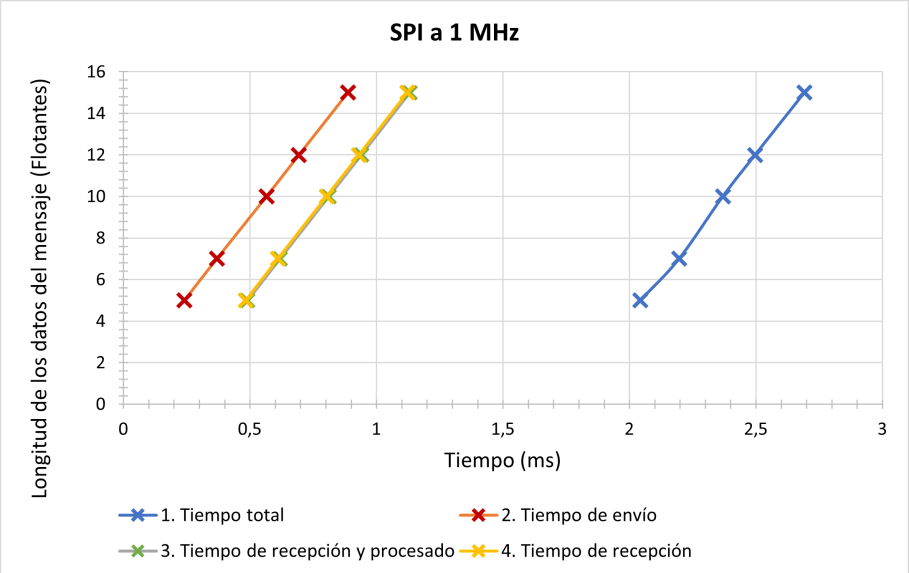


Figura 5.8 Tiempos que miden los cuatro temporizadores según la longitud del mensaje a 1MHz.

Las ideas mencionadas anteriormente podemos verlas reflejadas a la perfección en las figuras 5.9 y 5.10. Para una mejor visualización de los datos se ha representado el eje vertical en escala logarítmica. La primera gráfica hace referencia al tiempo completo desde que se inicia la creación de un mensaje para su envío hasta que se procesa su respuesta. Vemos como la longitud de los datos es mucho más relevante a tasas de envío bajas, como 0.5 MHz que a frecuencias altas superiores a 10 MHz (12,5 MHz y 50 MHz se representan en ambas gráficas).

Por otro lado, vemos como aumentar la frecuencia de 0,5 MHz a 1 MHz (aumentar al doble) supone reducir los tiempos más de 1,5 milisegundos. Sin embargo, aumentarla de 12,5MHz a 50MHz (aumentar más del triple) conlleva reducciones de tiempo menores a 0,1 milisegundos, tan pequeñas que no son perceptibles en las gráficas. En este caso está ocurriendo lo contrario al caso anterior: ahora la FIFO no llega llenarse porque el SPI pone sus datos a la salida antes de tener nuevos datos esperando para ser enviados, de forma que **la velocidad limitadora de las comunicaciones no la impone el periférico, sino el propio micro**, que da órdenes de envío más lento que la velocidad de envío de los datos del SPI.

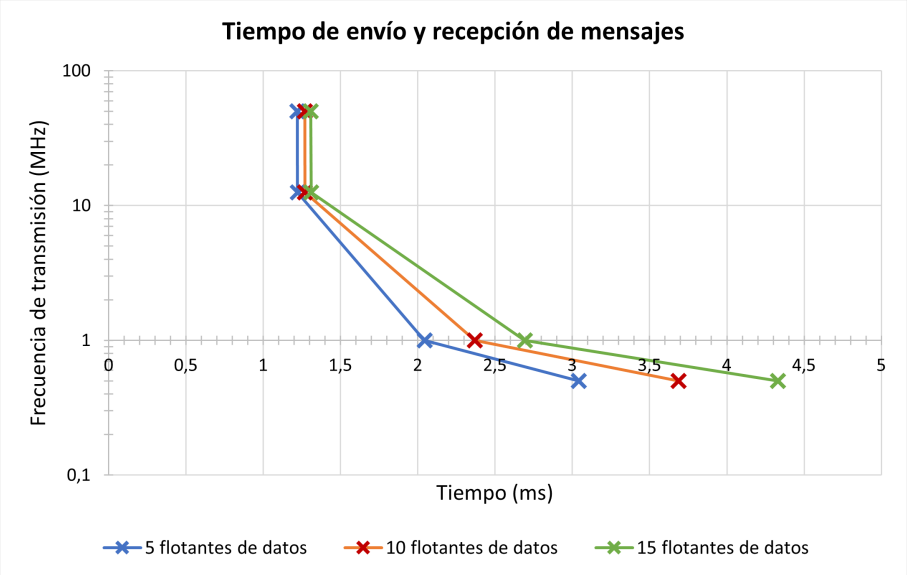


Figura 5.9 Tiempo total de envío y recepción en función de la frecuencia y la cantidad de datos.

En la recepción y el procesado de mensajes ocurre exactamente lo mismo con un efecto aún más acentuado a frecuencias bajas: un aumento en la cantidad de datos recibidos provoca un aumento notorio en el tiempo necesario para la adquisición del mensaje: aumentar en 5 unidades la cantidad de flotantes enviados supone un aumento aproximado de 0,5 milisegundos. A frecuencias altas, el número de datos recibidos no causa tanta diferencia ya que la diferencia de tiempos sería menor a 0,1 segundo.

5.0.2 RS485

El comportamiento analizado para SPI se repite para UART, haciendo uso del protocolo RS485. El tiempo total de envío y recepción a través de UART con 12,5 MHz de frecuencia es similar al tiempo total de envío y recepción del protocolo SPI a 12,5 MHz y a 50 MHz. Es decir, cambiar de UART a SPI para aumentar la velocidad no supone una mejora drástica como sí lo supondría si manejáramos velocidades menores. Esto puede deberse a dos factores principalmente:

- Como hemos comentado anteriormente, a estas frecuencias tan altas el sistema está regido por la frecuencia del microcontrolador, no por la frecuencia del periférico, al tener el periférico velocidad suficiente como para evitar que la FIFO se llene.
- Por otro lado, debemos remarcar que el enlace físico con el que se están realizando las pruebas de SPI es cable de conexiones de PCBs, también conocido como cables de puente o cables Dupont. En el futuro sistema estas comunicaciones viajarán a través de cable RJ45 según la PCB que hemos diseñado, portando señales diferenciales, mejorando así su inmuidad al ruido electromagnético.

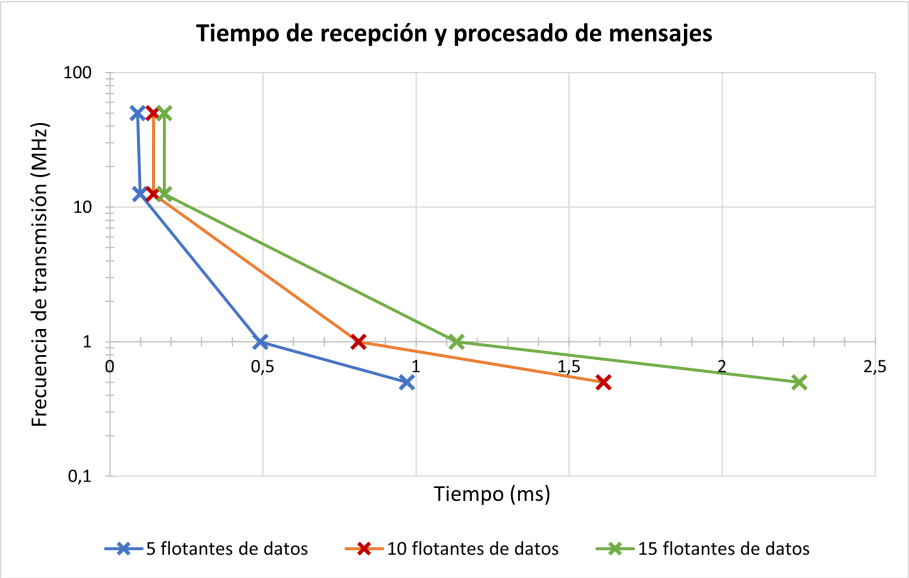


Figura 5.10 Tiempo total de recepción y procesado en función de la frecuencia y la cantidad de datos.

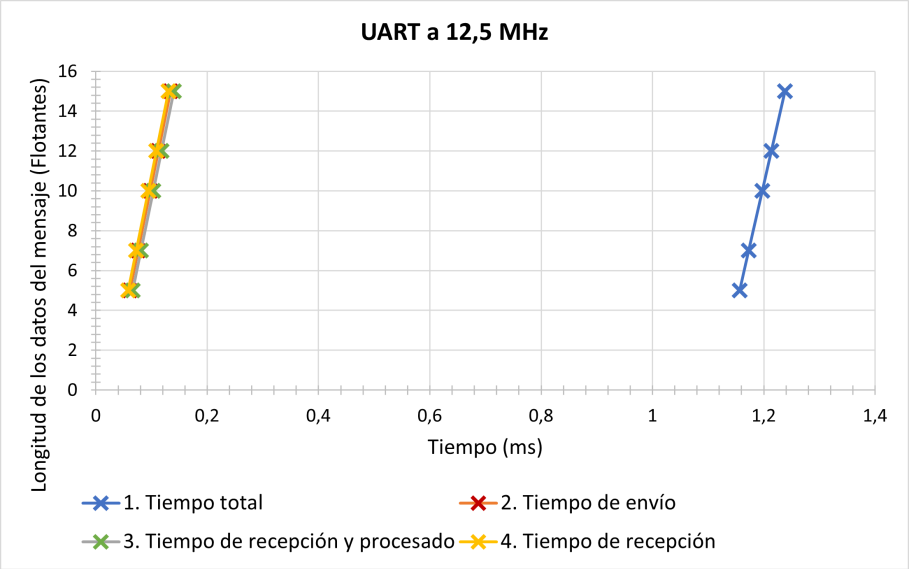


Figura 5.11 Tiempos que miden los cuatro temporizadores según la longitud del mensaje a 12.5 MHz.

Para obtener la gráfica 5.12 hemos bajado la tasa de envío hasta 115200 baudios, una frecuencia estándar en comunicaciones. De esta forma podremos comparar las velocidades de las comunicaciones existentes en sistemas "cotidianos" con las velocidades que estamos manejando en este sistema. El tiempo total para una transmisión de 15 flotantes (60 Bytes) contando desde que se crea el mensaje hasta que se procesa su respuesta es de 19 ms aproximadamente, 15 veces mayor que si lo hacemos a máxima velocidad (12,5 MHz).

El tiempo de recepción sigue siendo similar al tiempo de recepción más procesado, incluso la diferencia de tiempos es menor si lo comparamos con el resto de tiempos medidos en este experimento. Este comportamiento se debe a que el microcontrolador sigue funcionando a su frecuencia normal, por lo que el procesado se hace a la misma velocidad en todos los experimentos, es en la recepción y en el envío donde vemos ese reflejado ese coste temporal, ya que los periféricos

están programados para escribir o leer datos a velocidades muy bajas.

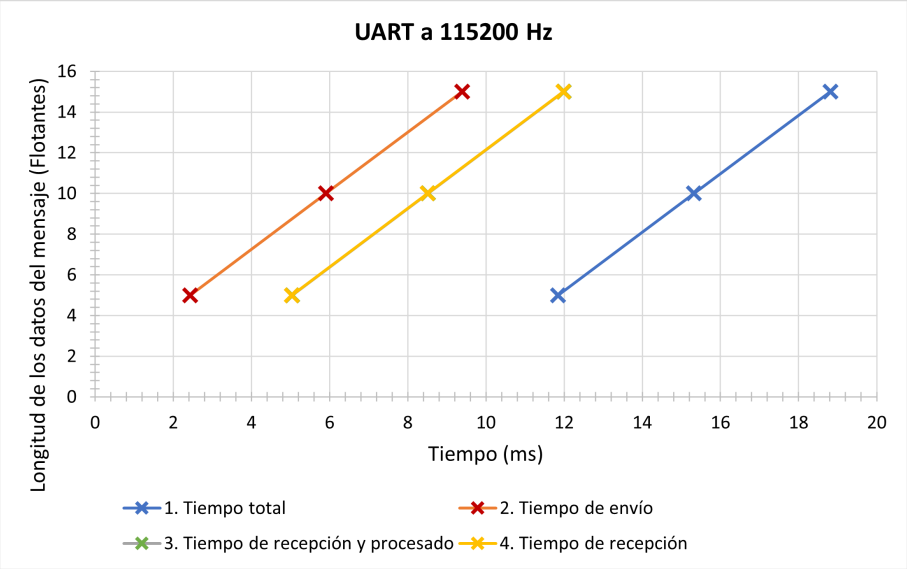


Figura 5.12 Tiempos que miden los cuatro temporizadores según la longitud del mensaje a 115200 Hz.

Para corroborar las ideas expuestas con el módulo SPI, realizamos con la UART experimentos parecidos. Apreciamos en la gráfica 5.13 cómo la cantidad de datos mandados llega a ser determinante para frecuencias bajas, pero es casi imperceptible para frecuencias altas. Es importante tener en cuenta que el eje horizontal abarca un intervalo mucho mayor en estos experimentos debido a que las iteraciones realizadas a 115200 baudios implican unos tiempor mucho mayores.

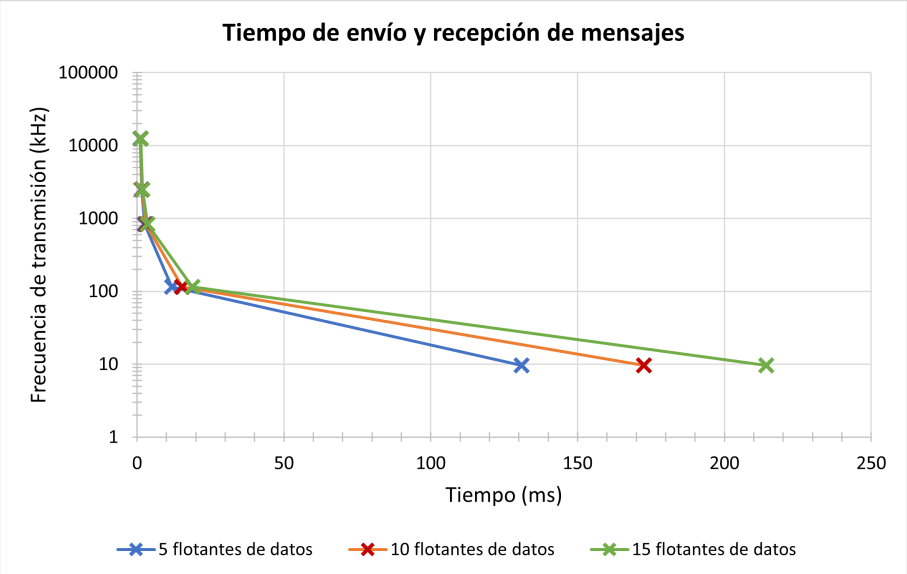


Figura 5.13 Tiempo total de envío y recepción en función de la frecuencia y la cantidad de datos.

Para poder visualizar mejor los datos obtenidos a altas frecuencias, repetimos el experimento solo con las frecuencias de 12’5 MHz, 2’5 MHz y 0’83 MHz, tal y como se muestra en la figura 5.14. Apreciamos ahora que la cantidad de datos en el tiempo a velocidades altas, aunque no es tan relevante como si lo es a velocidades bajas, sigue siendo importante. En esta ocasión no

apreciamos un límite superior en el beneficio temporal que obtenemos al aumentar la frecuencia como parecíamos obtener en el SPI, a pesar de que el beneficio se sigue reduciendo al aumentar la velocidad.

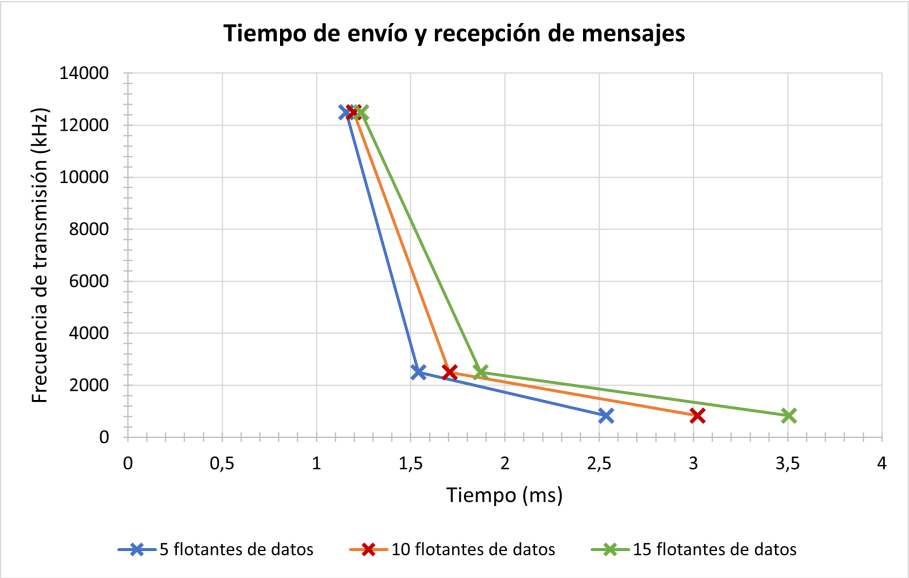


Figura 5.14 Tiempo total de recepción y procesado en función de la frecuencia y la cantidad de datos.

5.0.3 Fibra Óptica

Como los tiempos que hemos analizado en los experimentos anteriores los fija el microcontrolador o el protocolo de comunicaciones, las gráficas que se obtendrían de analizar el protocolo UART sobre Fibra Óptica serían similares a las que hemos obtenido para RS485. En este apartado por tanto vamos a analizar el propio enlace físico. Para ello hacemos tres experimentos con los diferentes enlaces y protocolos, mandando y recibiendo mensajes a 12,5 MHz (velocidad máxima de la UART pero no del SPI, que puede alcanzar hasta 50 MHz).

Los resultados obtenidos en la figura 5.15 reflejan que la Fibra Óptica es el enlace más rápido a pesar de que sus beneficios suelen verse en largas distancias (las distancias en estos experimentos se encuentran en el intervalo de 10 a 20 centímetros). El cable plano se comporta de forma parecida a la Fibra Óptica y el cable de puente con SPI provoca más retrasos en las comunicaciones. Estos pueden ser debidos tanto al cable como a la constitución del periférico, ya que la implementación de la UART por parte de Texas Instruments en los microcontroladores utilizados es más simple que la implementación del SPI, o sea, el SPI necesita modificar más registros y líneas antes de enviar un mensaje que la UART.

5.0.4 Fiabilidad del protocolo

Como último punto de estas pruebas vamos a hacer referencia a los errores obtenidos. En todas las pruebas de UART, tanto a través de fibra óptica como a través de cable plano con RS485 los datos se han enviado de forma correcta. Solo debemos notificar en dos ocasiones fallos inesperados de configuración (quizás provocados por la continua carga de programas diferentes) que han sido solucionados con una re-subida del fichero al microcontrolador. Todos los mensajes que han sido mandados, han llegado al otro extremo con todos sus campos idénticos a como se habían escrito en la creación del mensaje.

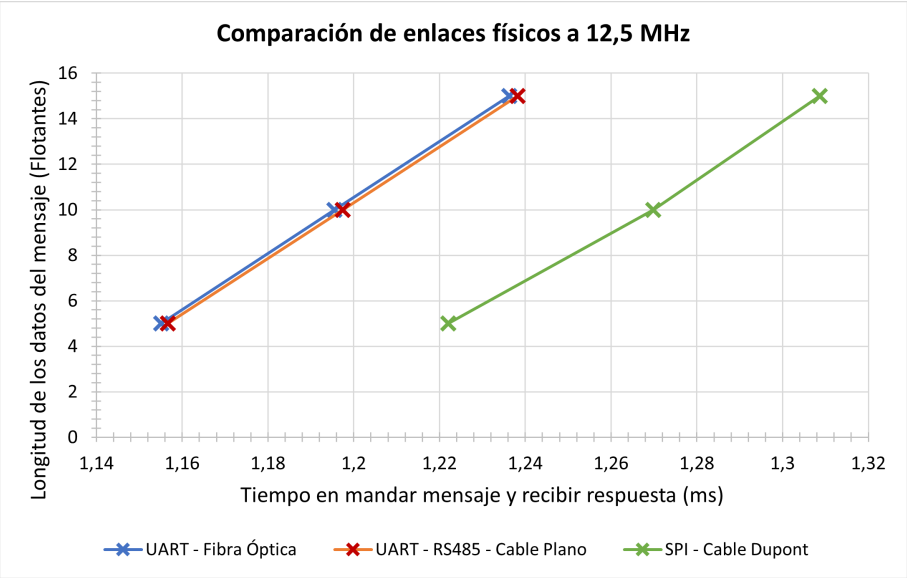


Figura 5.15 Comparativa de los tiempos totales con los diferentes enlaces.

En cuanto al protocolo SPI hemos encontrado diferencias según la PCB que usábamos. Colocando en ambos extremos de la línea dos LaunchPad F28377S resultaba un número de errores que no era despreciable, aproximadamente uno o dos bits erróneos cada tres mensajes completos. El mismo resultado obtenemos al comunicar una LaunchPad F28377S con una LaunchPad F28379D. Sin embargo, al colocar en ambos extremos dos LaunchPad F28379D no se han registrado fallos, habiendo realizado 4 veces más experimentos con esta configuración que en los otros dos casos. Por tanto, si se necesitan frecuencias cercanas a los 50MHz que permite el SPI es recomendable utilizar la LaunchPad F28379D.

6 Pruebas en distintos escenarios de comunicaciones

En este capítulo vamos a llevar a cabo un segundo conjunto de pruebas donde no nos vamos a centrar en el protocolo o el enlace, sino en el tipo de comunicación. Uno de los puntos más importantes de este protocolo es que las comunicaciones siempre estarán formadas por un solo transmisor y un solo receptor en una línea. Esto no implica la existencia solo de comunicaciones punto a punto, sino que también es posible llevar a cabo comunicaciones con múltiples esclavos conectados al mismo bus. Utilizaremos para ello comunicaciones por cable plano con protocolo RS485, que permiten fácilmente la inclusión de varios dispositivos. Además, con los conectores IDC[1] se pueden conectar al cable varios módulos de bajo nivel con relativa facilidad, ya que el código software y el conexionado es el mismo.

Los distintos experimentos que hagamos no pueden ser imitados con la estructura de protocolos SPI que se implementará en el sistema: dos SPI, uno maestro para mandar datos y otro esclavo para recibirlos. Como ya comentamos anteriormente, el mensaje siempre sale de un SPI maestro hacia un SPI esclavo, proporcionando además el maestro dos lineas adicionales, el reloj y el Chip Select o STE. Si se quiere permitir la escritura de dos módulos de bajo nivel, necesitaríamos conectar en una misma línea los dos CLK salientes de los módulos de bajo nivel, que entraría en el módulo de control, e igual ocurriría con el STE y la línea de datos MOSI. En cuanto a esta última no tenemos ningún problema porque el propio periférico implementado por Texas Instruments permite forzar una alta impedancia en este pin cuando no se utiliza. El problema reside en que los pines CLK y STE no tienen esta posibilidad.

Resumiendo el párrafo anterior, el SPI implementado por Texas Instruments (ni la mayoría de los que se encuentran en las placas del mercado) están pensados para sincronizar la escritura de varios maestros. Muchos de ellos sí están preparados para que un solo maestro se comunique con varios esclavos y que estos respondan por el mismo canal SPI (a través del SOMI), y los SPI que vamos a utilizar también incluyen esta posibilidad. Sin embargo nuestro sistema no va a hacer uso de las líneas SOMI como ya comentamos en la sección 3.2.1, por lo tanto, aunque podríamos realizar pruebas parecidas a las que vamos a realizar con la UART a través de RS485, carece de sentido hacerlas utilizando las líneas de SOMI ya que no van a reflejar la realidad y las conclusiones que podamos sacar podrán ser erróneas.

Por otro lado, el correcto funcionamiento del protocolo SPI ya se comprobó en el capítulo anterior y la única diferencia práctica entre SPI y UART (además del número de lineas) es la velocidad de transmisión, la cual no vamos a poder apreciar en los cronogramas que obtengamos porque se ha tenido que reducir la velocidad por motivos de resolución del aparato de medida.

[1] Un conector I2C es un conector especial utilizado para las conexiones mediante cable plano que restringen los posibles modos de comunicaciones entre macho y hembra a uno solo.

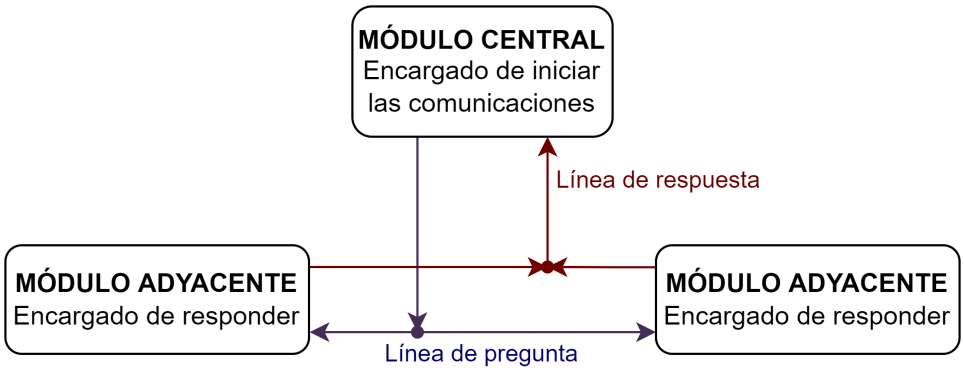


Figura 6.1 Esquema del conexionado realizado para las pruebas.

Los experimentos realizados han sido monitorizados con un analizador lógico[2], cuya escala de representación no es lo suficientemente reducida como para apreciar cambios de estado en las lineas a altas frecuencias. Es decir, el eje horizontal del cronograma tiene una longitud del orden de las décimas de segundo, mientras que, para poder apreciar las comunicaciones de nuestras señales a 50 MHz (frecuencia del SPI), hubiéramos necesitado longitudes por debajo de las unidades de milisegundo. Para que las señales de RS485 sean apreciables hemos reducido la frecuencia máxima de 12,5 Megabaudios [3] a 9600 baudios.

6.1 Experimentos realizados

Las pruebas que se han llevado a cabo implican a un módulo de control o módulo maestro, y varios módulos de bajo nivel o módulos esclavos. Aunque en el protocolo RS485 no existe la idea de módulo maestro y módulo esclavo al ser un protocolo asíncrono, cualquier referencia a maestro y esclavo estará referida al módulo que inicia la comunicación y al módulo que contesta, respectivamente.

Tendremos para ello un módulo que inicia la comunicación y dos módulos que responden al primero, aunque lo harán de dos formas diferentes según el experimento. Ambos módulos de respuesta estarán escuchando siempre en el bus con una alta impedancia en su salida para posibilitar la escritura del otro módulo de respuesta. Solo habilitarán su salida si proceden a escribir en la línea de respuesta.

6.1.1 Pregunta y respuesta

En primer lugar programaremos los módulos de respuesta para que esperen una solicitud del módulo central. Procesarán el mensaje y si el campo *ID* de la cabecera es igual a su propia identificación, responderán inmediatamente. Es decir, el módulo central escribirá un mensaje y rellenará el campo *ID* con el número del módulo adyacente al que va dirigido. Todos los módulos de respuesta leerán el mensaje, pero solo aquel que su ID se corresponda con la del mensaje activará la salida y responderá.

En concreto, vamos a realizar comunicaciones secuenciales entre el módulo central y los módulos adyacentes, de forma que, cuando el módulo central reciba la respuesta del primero, se comunicará con el segundo. No esperar la respuesta del primer módulo adyacente antes de solicitar una respuesta del segundo podría ocasionar que ambos escribieran simultáneamente en la línea de respuesta, lo

[2] Un analizador lógico es un dispositivo sensor que se conecta a una linea digital y es capaz de reconocer los valores y flancos de las señales, lo que facilita el debug de estos sistemas.

[3] El baudio es una medida de velocidad de transmisión usada en comunicaciones y representa el número de bits que se envían por segundo.

cual no es nada deseable ya que se perderían los datos. La secuencia seguida se detalla en la figura 6.2

1. El módulo central manda el mensaje al primer módulo de respuesta.
2. Se manda la respuesta por parte del módulo adyacente.
3. El módulo central manda el mensaje al segundo módulo.
4. Se genera y se manda la segunda respuesta.

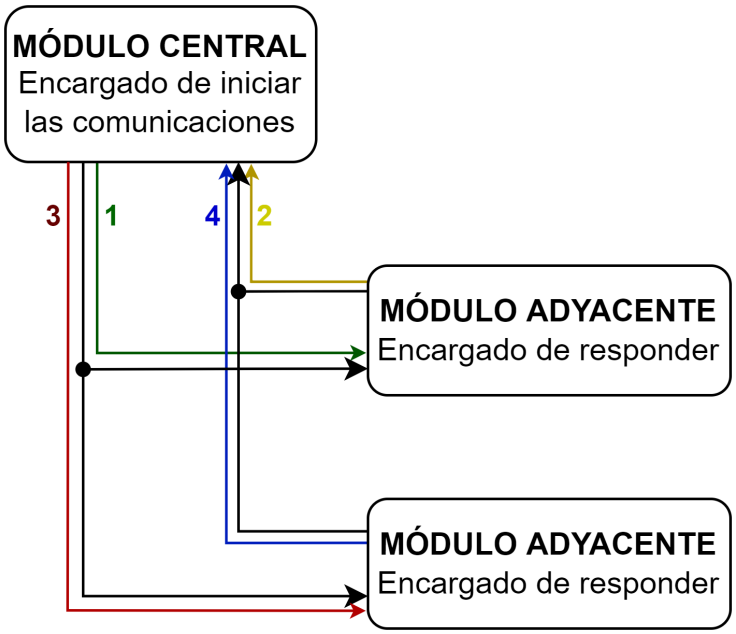


Figura 6.2 Secuencias de mensajes en el experimento de pregunta y respuesta.

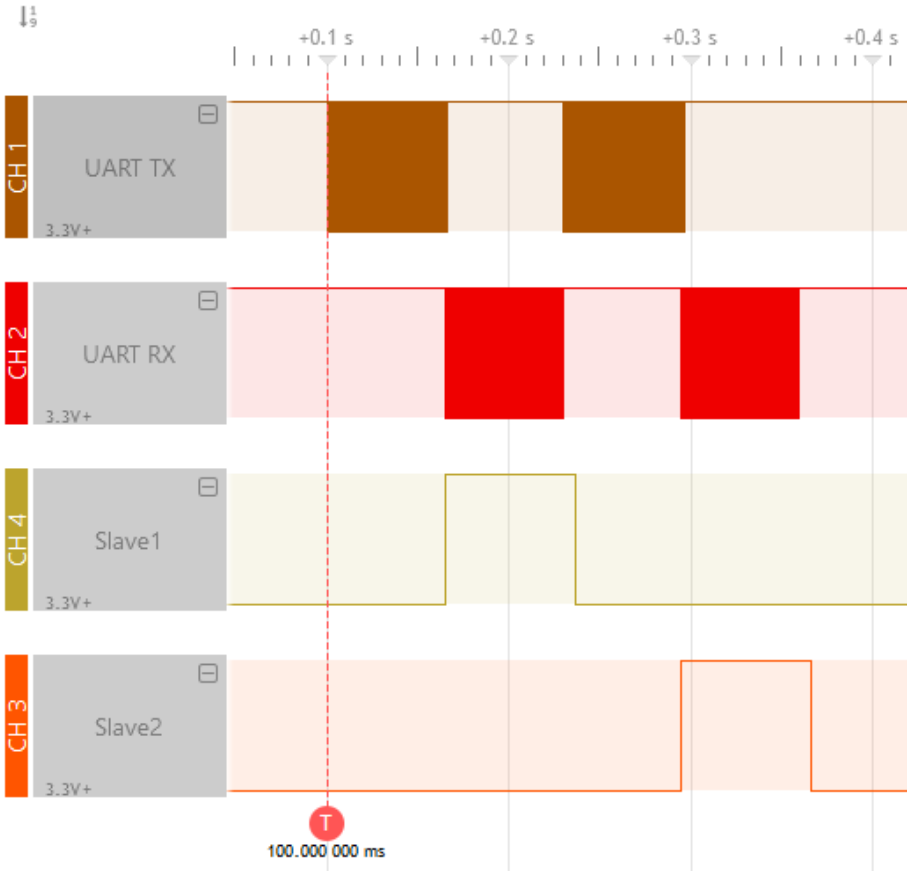


Figura 6.3 Experimento de pregunta y respuesta..

En la figura 6.3 podemos ver rectángulos que se corresponden con acumulaciones de flancos de subida y de bajada. En la primera línea tenemos la salida de transmisión del módulo central, de forma que cada rectángulo marrón es en realidad una pregunta hacia los módulos adyacentes. En el segundo canal vemos la línea de respuesta, donde escriben ambos módulos adyacentes. Los canales 3 y 4 son las señales de habilitación de escritura de los módulos adyacentes, activadas por ellos mismos. La utilidad de estas señales es informarnos de cuándo está libre la línea de respuesta, y en caso de no estarlo, saber que módulo de respuesta la está ocupando. Podemos destacar los siguientes aspectos:

- Los mensajes en ambos sentidos son perfectamente secuenciales, cuando uno de los módulos recibe un mensaje escribe inmediatamente el siguiente, de forma que el tiempo de adquisición del mensaje entrante es inapreciable.
- El primer rectángulo marrón es una solicitud al primer módulo adyacente, llamado Slave1 en la imagen. Cuando el mensaje termina, este módulo responde, habilitando su salida (línea CH4 Slave1) y escribiendo su respuesta en el bus.
- De forma inmediata a la recepción de la primera respuesta, el módulo central escribe la solicitud para el módulo Slave2, que realiza el mismo comportamiento que el primero.
- Según el protocolo que hemos programado, el mensaje saliente del módulo central se va adquiriendo casi simultáneamente en los buffers de entrada de los módulos adyacentes y se analiza cuando el mensaje termina.
- Los tiempos que rigen el comportamiento del sistema son los tiempos de escritura de los mensajes, debido a la longitud considerable del mensaje y a las velocidades reducidas del protocolo en estas pruebas.
- El tiempo de análisis del mensaje y comienzo de escritura de la respuesta es despreciable frente al tiempo de envío.
- Las habilitación de escritura de los módulos adyacentes se activa en el instante previo a la escritura y se alarga un tiempo posterior al fin de esta. Esto se debe al funcionamiento de la escritura en los periféricos SPI y UART de Texas Instruments, el cual vamos a recordar. Para ordenar la escritura se debe escribir en un registro el dato, la CPU detecta esta escritura y copia el dato en la línea de salida del periférico correspondiente. No es posible detectar cuándo todos los bits de este registro han sido mandados, por lo que se introduce un temporizador que permite esperar un tiempo prudencial antes de deshabilitar la salida. Este tiempo se ha estimado mediante varias pruebas, pero cuando todo el sistema esté funcionando al completo posiblemente haya que prolongarlo ya que los ciclos de instrucciones serán más largos. El único requisito que debe cumplir este tiempo es que permita mantener activa la salida hasta que el último bit de la respuesta haya sido enviado.
- Es importante destacar que ninguno de los módulos se encuentra esperando un mensaje de ningún otro. Todos ellos escriben en el bus “cuando desean” sin saber si hay alguien escuchando. El punto importante para que esto funcione es que las entradas de nuestro protocolo funcionan por interrupción. De forma que, cuando algún módulo manda un mensaje todos los módulos conectados en la línea de escucha paran su actividad, acumulan ese mensaje en un buffer y continúan su actividad normal, procesando el buffer en el instante que tengan programado. Así se evitan por un lado tiempos de espera que pueden utilizarse para otras funciones, y no se pierden datos porque aunque no se esté siempre disponible para procesar el mensaje, este se adquiere y se procesa más tarde.

Podemos ver mediante esta prueba que el protocolo funciona también para comunicaciones punto a punto y que los campos de la cabecera son correctamente interpretados por todos los módulos.

1. El módulo central manda el mensaje a los dos módulos de respuesta.
2. Se manda la respuesta por parte del primer módulo adyacente.
3. Se genera y se manda la segunda respuesta.

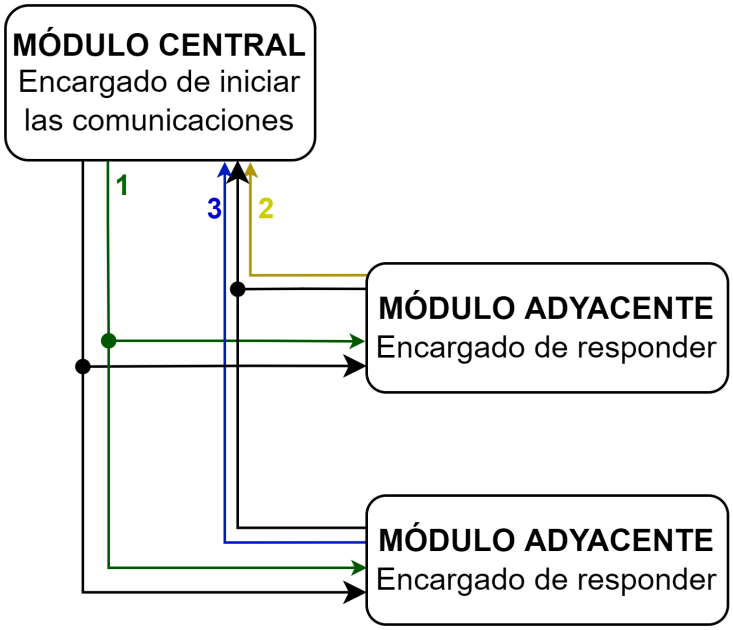


Figura 6.4 Secuencias de mensajes en el experimento de multiplexación en el tiempo.

Una variante de este tipo de comunicaciones veremos en el sistema final. Cuando el módulo central necesite reclamar datos que no se obtienen en el ciclo normal, mandará una pregunta y el módulo de bajo nivel responderá. La única diferencia será que el módulo de bajo nivel no estará programado para hacerlo inmediatamente, sino que podría realizar cálculos o mediciones necesarias previas al envío de la información solicitada.

6.1.2 Respuesta multiplexada en el tiempo

En la segunda prueba a realizar se pretende que todos los módulos adyacentes respondan a una misma petición del módulo central. Recordemos que todos los módulos adyacentes están normalmente escuchando en la línea de envío del módulo central, por lo que, aunque el mensaje no lleve a un módulo como destinatario, este módulo lo procesará igualmente para poder conocer si es el destinatario o no.

Para que no haya corrupción o pérdida de datos, las respuestas se proporcionarán multiplexadas en el tiempo, o sea, cada uno de los módulos adyacentes tiene asignado un intervalo de tiempo en el que debe responder sin dilación, pues puede entorpecer la escritura del siguiente módulo o incluso dañar sus datos.

Este comportamiento se ha conseguido dotando a los módulos de respuesta de un temporizador que empieza a contar una vez finalizada la recepción y el procesamiento de un mensaje, y cuyo periodo depende de la identificación del propio módulo. La identificación del mensaje será una identificación fija que todos los módulos reconocerán como ID de broadcast (difusión) (0xFF en estas pruebas). Cuando se reconoce esta ID, se activa el temporizador, el cual avisa de su finalización mediante interrupción y con ello el módulo de respuesta activa su salida y envía el mensaje.

El hecho de que el periodo del temporizador dependa de la ID propia del módulo es lo que provoca la multiplexación el tiempo, a la vez que una respuesta ordenada. El módulo con ID igual a 1 no esperará, el módulo con ID igual a 2 esperará un tiempo precalculado multiplicado por 1, el tercer módulo esperará ese mismo tiempo precalculado multiplicado por 2, etc. Por otro lado, sería conveniente también prolongar este tiempo en función de la longitud de la respuesta, sin embargo no se ha implementado pues este dato no se conocerá hasta fases más avanzadas del proyecto.

Una alternativa podría ser suponer siempre el caso más desfavorable, es decir, esperar siempre el tiempo correspondiente al mensaje más largo que se vaya a mandar, lo cual reduciría la probabilidad de colisiones pero ralentizaría el sistema. Para esta prueba la longitud de todos los mensajes, tanto de pregunta como de respuesta, se fija a 5 bytes, aunque si se quisiera repetir la prueba en futuras versiones del protocolo, el tiempo a esperar debería seguir la expresión 6.1. De la misma forma que en la prueba anterior, el Tiempo Base dependerá del tiempo de ciclo de los microprocesadores, por lo que se deberá analizar con todo el sistema funcionando.

$$TiempoTotal = (ID - 1) * TiempoBase * CantidadDatos \tag{6.1}$$

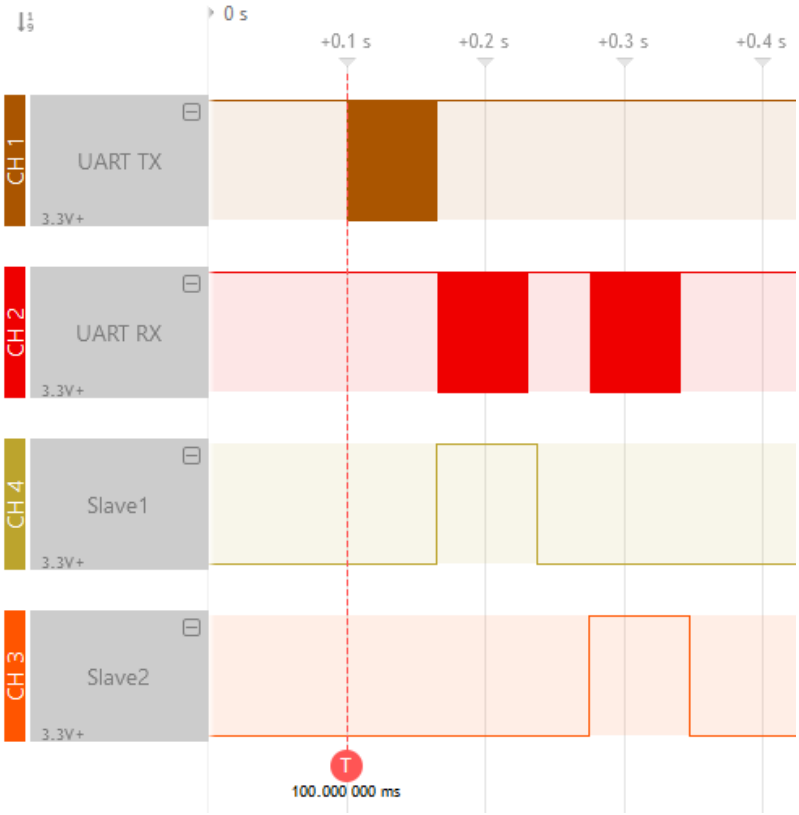


Figura 6.5 Respuesta multiplexada en el tiempo con módulos de respuesta 1 y 2.

De la figura 6.5 podemos hacer incapié en los siguientes puntos:

- El primer módulo adyacente responde inmediatamente después a la recepción del mensaje, pues entiende que es el primer módulo en responder ya que su identificación es 1 y por tanto no debe esperar.
- Entre ambas respuestas hay un tiempo de espera de aproximadamente el 70 % del tiempo de escritura del mensaje que reduce la posibilidad de colisiones entre ambos mensajes. Este tiempo puede reducirse al máximo, ya que en ninguno de los experimentos realizados el tiempo ha variado más de un 1 % de la media de los datos obtenidos (pueden apreciarse repeticiones de experimentos en las gráficas 5.2 - 5.5). En las pruebas se ha alargado en exceso para poder apreciarlo con facilidad y poder destacarlo.
- Al igual que en la prueba anterior, es necesario esperar un tiempo prudencial antes de deshabilitar la escritura de un módulo adyacente. Este tiempo debe ser al menos suficiente para que el propio módulo termine de escribir, pero en esta prueba es importante también el tiempo máximo que puede estar habilitada la escritura. Dado que en esta ocasión un módulo

escribe a continuación de otro, es necesario cerciorarse de que se deshabilita la escritura del primer módulo antes de proceder a escribir con el segundo. En todos los casos, el tiempo extra que se habilita la escritura (para asegurar que se manda el mensaje completo) debe ser menor que el tiempo de margen introducido entre dos respuestas consecutivas. Este concepto se refleja en la figura 6.6, donde deberemos garantizar que A siempre sea menor que B.

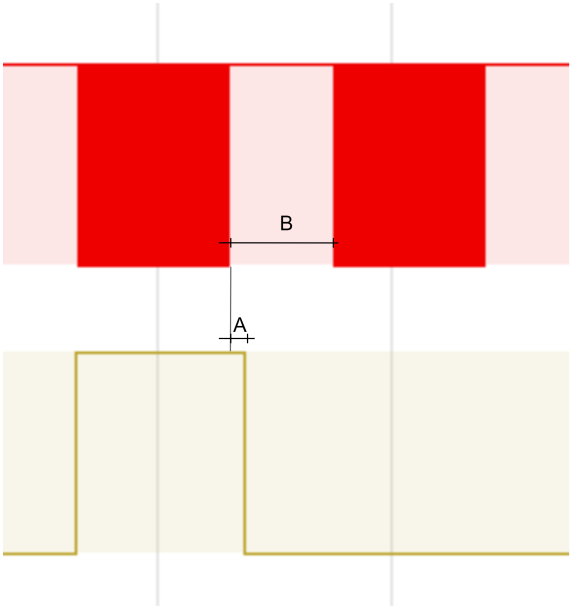


Figura 6.6 Restricción temporal de los márgenes de seguridad: A<B.

En la figura 6.7 vemos un comportamiento similar al anterior, pero en este caso hemos modificado la identificación de los módulos adyacentes a 2 y 3, con el fin de comprobar que estas pruebas son repetitivas con un número mayor de módulos conectados.

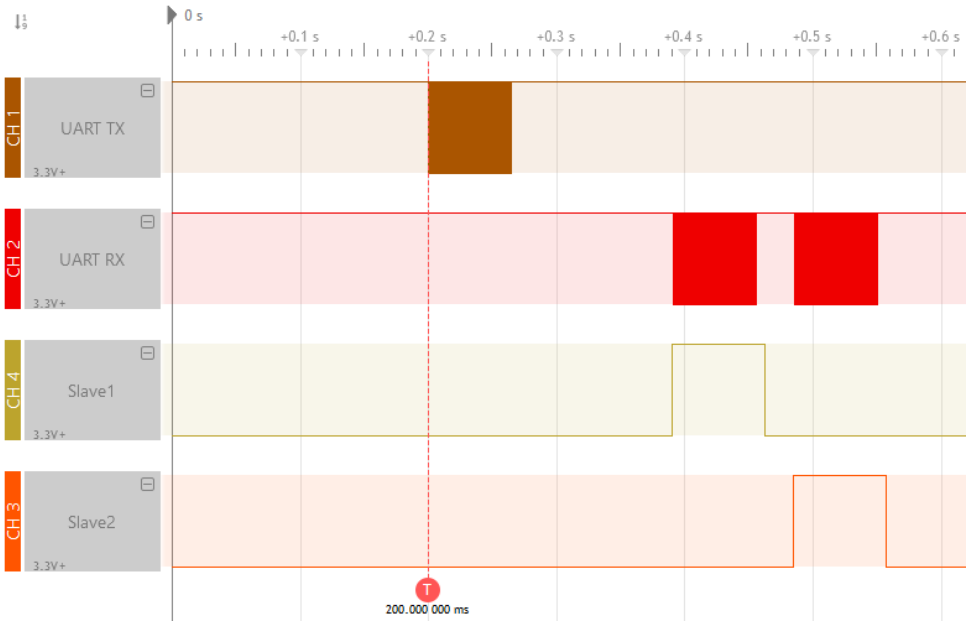


Figura 6.7 Respuesta multiplexada en el tiempo con módulos de respuesta 2 y 3.

7 Líneas futuras y conclusiones finales

Además de las modificaciones del protocolo llevadas a cabo, muchas otras acciones han sido llevadas a cabo para asegurar un buen funcionamiento, como el estudio experimental del comportamiento de las FIFOs de entrada y salida, estudio profundo de los registros de los protocolos, diferentes experimentos con distintos temporizadores para asegurar trabajar a la velocidad máxima, experimentos con los protocolos tanto con funciones de DriverLib de Texas Instruments como a partir de la escritura y lectura de registros, etc. Sin embargo, analizando los resultados surgen ideas de posibles mejoras:

- En el sistema final no seleccionaremos el protocolo UART o SPI a través de una macro, sino que el código se cargará completo en los microcontroladores y seleccionaremos una alternativa u otra en función de una entrada a la que conectaremos un interruptor manual.
- De la misma forma, aunque se ha intentado conseguir una compatibilidad total entre ambas LaunchPads no se ha conseguido plenamente. Los periféricos de UARTs se encuentran en los mismos pines, pero no utilizan exactamente el mismo periférico, es decir, en un determinado pin tendremos la línea RX de la UART A en una Launchpad y la UART B en otra (cada microcontrolador tiene 4 UARTs), lo cual es importante a la hora de configurar el software. Es necesario por tanto conocer que placa está conectada y se realizará también con un interruptor en una entrada digital, en lugar de con una macro como hasta ahora.
- El pin STE del protocolo SPI está colocado en diferentes lugares en ambas placas, por lo que añadiremos un conmutador que conecte a cada SPI (recordemos que teníamos un SPI maestro y otro esclavo en cada módulo) y haga conexión entre la línea de STE y el pin de la LaunchPad donde este se encuentra.
- Por otra rama completamente diferente planteamos esta mejora. Si se diera el caso de que la FIFO de entrada del módulo que recibe tuviera más datos que la FIFO de salida del módulo que envía, el módulo que envía no dejaría de hacerlo hasta que su FIFO se llenase y podría provocar que la FIFO de recepción se desbordara. Normalmente esto no ocurrirá, pero se podría dejar una posición de la FIFO de margen parando de mandar cuando la FIFO de salida esté llena a falta de un mensaje,
- Además, se ha detectado un problema que puede darse si se transmite a bajas velocidades (lo hemos comprobado en las pruebas). Como el microcontrolador funcionará más rápido que el periférico de envío, los datos se acumularán en la FIFO de salida. Actualmente, cuando la FIFO está llena se para la ejecución esperando a que haya hueco suficiente en la FIFO. Para evitar estas esperas podría utilizarse la interrupción de la FIFO de salida, la cual puede avisar cuando esté por debajo de un cierto umbral. Podríamos establecer un funcionamiento como el siguiente:

1. Crear mensaje.
2. Mientras haya hueco en la FIFO, mandar los datos en grupos de 8 bits (como se realiza ahora).
3. Si la FIFO se llena, activar interrupción que avise, por ejemplo, cuando tiene la mitad del espacio disponible.
4. Realizar otras tareas.
5. Cuando la interrupción salte, borrar flag y volver al paso 2.
6. Salir del bucle cuando se complete el envío del mensaje.

La mayoría de estas mejoras surgen de las pruebas que hemos realizado a baja velocidad. El protocolo actual funciona de forma correcta y cumpliendo los requerimientos establecidos. Podemos resaltar que podemos realizar comunicaciones a 50MHz, en las cuales hemos enviado hasta 120 bits en 1,2 ms aproximadamente, incluyendo en este tiempo los tiempos de procesamiento, de espera... Recordemos que el tiempo máximo para realizar estas operaciones era de 1 ms. Por otro lado, con las comunicaciones a través de UART cuya velocidad máxima es de 12,5 MHz, hemos conseguido tiempos de 1,3 ms aproximadamente.

Como ya se ha comentado, observamos que a tan altas frecuencias cambiar de UART a SPI aumentando la frecuencia más del triple (de 12,5 a 50 MHz), supone una leve mejora aunque no tan grande como sucede a frecuencias bajas. Esto puede deberse a que, como los microcontroladores funcionan a 200MHz como máximo, con un periférico que modifica su salida a 50MHz, la CPU no tiene tiempo para efectuar las modificaciones, ya que habrá acciones de envío y lectura que no sean atómicas, es decir, que ocupen más de una instrucción. Por ejemplo, antes de enviar se comprueba el estado de la FIFO de salida, lo que provocará al menos un ciclo más necesario en cada escritura.

En cuanto a los diseños de las PCBs, hemos podido comprobar el correcto funcionamiento de los dos modelos que funcionan con UART y se dejan planteados los experimentos necesarios para comprobar el funcionamiento de la PCB de SPI que no ha sido verificada por falta de dispositivos (receptores diferenciales en concreto). Esto no nos ha impedido realizar pruebas con SPI, aunque se han utilizado cables de puente, lo que posiblemente haya provocado algún fallo detectado y añada tiempos de retraso en las comunicaciones. En las comparaciones de los 3 enlaces utilizados, la fibra óptica es más rápida que el cable plano y mucho más rápida que el cable de puente. Sin embargo, aunque su comportamiento sea mejor, los beneficios obtenidos sobre el cable plano no son suficientemente significativos como para afrontar el coste económico de utilizar fibra óptica en todos los enlaces.

Finalmente, tratando el problema de los fallos, se ha conseguido un protocolo robusto ya que no se han detectado errores de bits en mensajes enviados por fibra óptica o cable plano, es decir, por UART. En cuanto al protocolo SPI, la LaunchXL F28379D consigue un comportamiento perfecto si se coloca a ambos extremos de la línea. Las combinaciones de LaunchXL F28379D y LaunchXL F28377S, así como un sistema con dos LaunchXL F28377S en ambos extremos, no aportan tan buenos resultados obteniendo una cantidad media de fallos de medio bit por mensaje aproximadamente.

Estos fallos se observan principalmente en el mensaje que escribe la LaunchPad 377S. Esta placa de desarrollo permite habilitar o deshabilitar la salida de un módulo SPI, ya sea maestro o esclavo mediante un bit llamado TALK. Según información de distintos foros, entre ellos el foro oficial de Texas Instruments “E2E design support”, se debe deshabilitar la escritura del módulo maestro mediante el bit TALK si no se está escribiendo, para evitar el envío de restos antiguos de mensaje o residuos de la FIFO. En trabajos futuros, si se quieren conseguir comunicaciones a 50MHz por SPI con la LaunchXL F28377S se deberá empezar por analizar en detalle el comportamiento del bit TALK, aunque se debe tener en cuenta que la LaunchXL F28377S es una placa cuya fabricación ha sido interrumpida definitivamente, con intención deer sustituida por la LaunchXL F28379D.

Apéndice A
Hardware construido

Para facilitar el conexionado de las PCBs de comunicaciones entre sí (con los enlaces físicos) se enuncian en primer lugar los requisitos más importantes, no solo para conseguir un buen funcionamiento con las características que se detallan en este documento, sino también para evitar cortocircuitos y otros fallos que pueden ser fatales para algunos componentes críticos.

Más tarde, recopilaremos el Hardware construido junto con sus dispositivos más relevantes.

A.1 Guía de conexionado de los elementos del sistema

El conexionado de las fibras ópticas es sencillo: son dos únicas líneas las cuales deben cruzarse de forma que cada línea se conecta al transmisor en un extremo y al receptor en la otra. Los transmisores de fibra óptica instalados son grises y los receptores negros, por lo que no debería haber ningún problema en este caso. El conexionado de los cables de RJ45 también es cruzado, observando que, si se colocan los puertos RJ45 enfrentados no es necesario cruzar los cables porque el giro ya se ha realizado con la PCB. Se recomienda crimpar el conector del módulo de control por un lado del cable y el conector de los módulos de bajo nivel por el otro lado, todos por el mismo. En la figura A.1 podemos ver hacia arriba los conectores donde pincharíamos los módulos de bajo nivel y a la derecha crimpado hacia abajo el conector del módulo central. De esta forma pueden conectarse todas las PCBs de cable plano "mirando" hacia el mismo lado.

Todas las conexiones realizables entre las distintas PCBs están preparadas para poder ser conectadas mediante cable plano con conectores IDC o sin cable, mediante unas placas intermedias que conmutan los pines para permitir esta doble opción de conexionado. El aspecto más importante a tener en consideración es que todos los conectores tienen una fila conectada a tierra (si el conector es de 5x2, tendrá 5 pines conectados a tierra). Además, al estar los conectores colocados siempre al borde de las PCBs podemos distinguir una fila interna a la PCB y otra externa. En la PCB de control

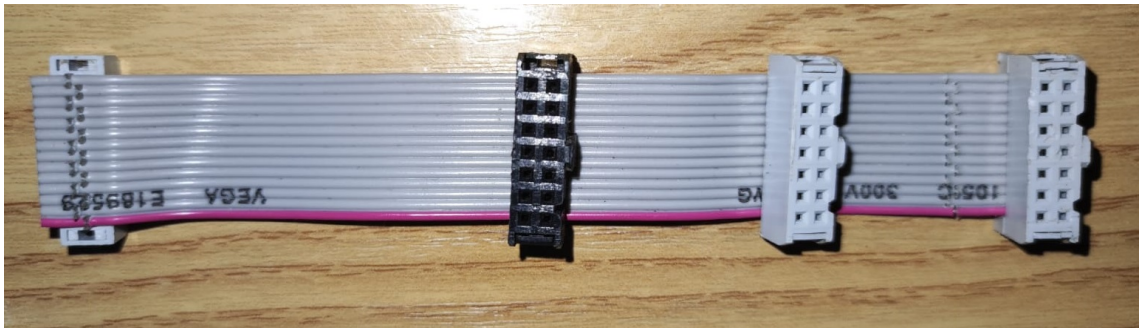


Figura A.1 Modo recomendado de crimpar los conectores al cable plano.

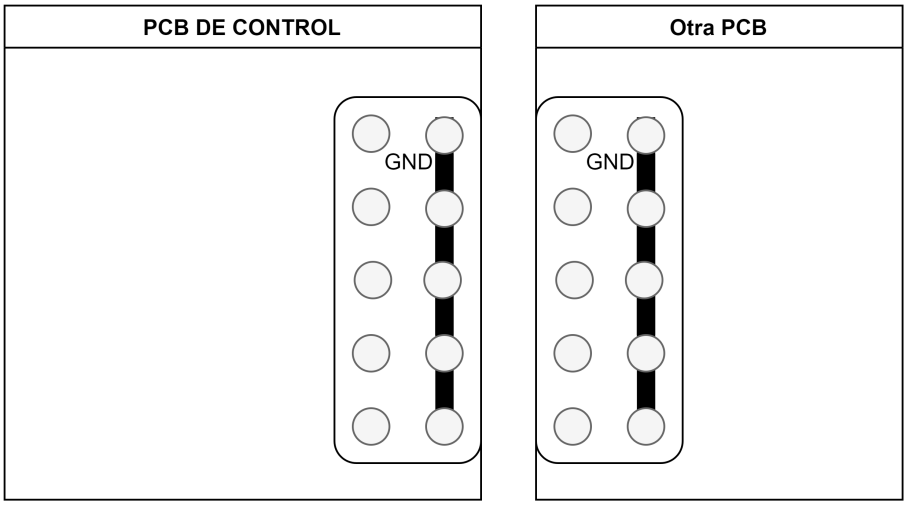


Figura A.2 Disposición de la tierra en la mayoría de las PCBs. Puede ocurrir que uno de los extremos sea alimentación, normalmente el superior si existiese, pero las tierras siempre irán colocadas así.

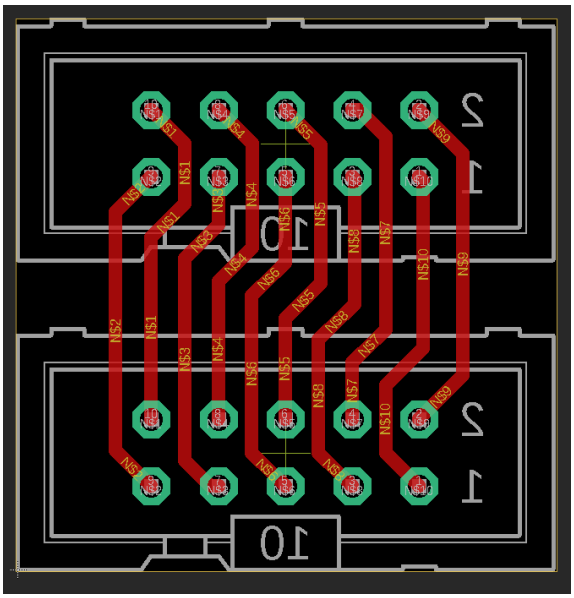


Figura A.3 Rutas de la PCB auxiliar de conexiones de 5x2.

la fila de GND es la linea externa en todos sus conectores, mientras que en las demás PCBS, la línea de GND es la interna en todos los conectores, de forma que se puede hacer una correspondencia pin a pin entre ambos conectores tal y como se muestra en la figura A.2.

Las placas intermedias solo tienen rutas que realizan una inversión del conector, como se puede apreciar en la siguiente captura del diseño realizado para el conector de 5x2. Se han diseñado en total 3 PCBs con esta estructura, una por cada dimensión de conector que vamos a usar (5x2, 8x2 y 20x2).

El conexionado mediante cable entre la PCB de control y la PCB de comunicaciones o PCB de bajo nivel está restringido a una única posición por el uso de conectores IDC, cuyo macho tiene un recubrimiento de plástico con un alojamiento para una pestaña del conector hembra, de forma que solo existe una forma posible de realizar la conexión.

A.2 Resumen del Hardware diseñado

Vamos a comentar a continuación un resumen de las PCBs diseñadas y sus componentes principales.

1. Enlace de Fibra Óptica para UART.
   - Conector para enlace con la PCB de control.
   - Un transmisor de fibra óptica.
   - Un receptor de fibra óptica.
2. Enlace de cable plano para UART a través de RS485.
   - Conector para enlace con la PCB de control.
   - Conector hembra IDC para conector con el cable plano.
   - Dispositivo ISO3086 por el que obtenemos RS485 a partir de UART y dos pines de GPIO.
   - Módulo de alimentación DC-DC con aislamiento.
3. Enlace de cable RJ45 para protocolo SPI. Se generarán 3 modelos: uno para conectarlo a un SPI en modo maestro, otro para conectarlo a un SPI en modo esclavo y otro que recoja ambos modos. Los componentes que listamos a continuación son para los dos primeros casos, en el último modelo que engloba ambos modos la cantidad de componentes será el doble.
   - Conector para enlace con la PCB de control.
   - Aislador galvánico con 3 canales en un sentido y un cuarto canal en sentido inverso.
   - Cuatro transmisores y receptores diferenciales. Se necesitarán 3 transmisores y un receptor o un transmisor y tres receptores, según el modelo.
   - Conector RJ45.
   - Módulo de alimentación DC-DC
4. PCBs de enlace entre las diferentes PCBs de comunicaciones u otras, que permiten conectar sin cable respetando las conexiones pin a pin. Se han creado tres diferentes para cada tamaño de conector.
   - 10 pines (5x2)
   - 16 pines (8x2)
   - 40 pines (20x2)

Apéndice B
Análisis de las velocidades máximas en protocolos UART y SPI

Aunque las comunicaciones prioritarias deben ser lo más rápidas posibles, debemos analizar si alcanzan las velocidades necesarias para controlar los convertidores modulares. En primer lugar nos centraremos en el funcionamiento teórico de los dispositivos, según la hoja de características del fabricante y en el documento principal se recogerán pruebas realizadas para comprobar experimentalmente las conclusiones obtenidas.

Los periféricos UART, llamados por Texas Instruments SCI (Serial Communicaitons Interface), SPI (Serial Peripheral Interface) y McBSP (Multichannel Buffered Serial Port) están conectados a un reloj de velocidad inferior a la de la CPU: LSPCLK, Low Speed Peripheral Clock. Este reloj se crea mediante un divisor de frecuencia del reloj del sistema (SYSCLK). Por defecto, se divide la frecuencia principal entre 4, pero para buscar altas velocidades puede modificarse el divisor a 1 (se utilizarían los periféricos nombrados a máxima velocidad). Para ello, puede utilizarse la siguiente función de la DriverLib después de configurar el reloj:

Código B.1 Aumento al máximo de la frecuencia del reloj que controla la UART y el SPI.

```
SysCtl_setLowSpeedClock(SYSCTL_LSPCLK_PRESCALE_1);
```

Dado que la placa trabaja a 200 MHz como máximo, con la sentencia anterior podemos fijar el reloj base de los protocolos a esta frecuencia. Esto no significa que los periféricos funcionen a esta velocidad, ya que cada uno de ellos impone otras restricciones. El SCI o UART tiene como frecuencia máxima teórica la del reloj LSPCLK dividido por 16. Con un reloj de 200MHz, este periférico podrá funcionar teóricamente a 12.5MHz como máximo. Si nos planteamos bajar la velocidad de transmisión, existen valores discretos de velocidades que se pueden seleccionar. En concreto, 65535 valores diferentes que vienen determinados por dos registros de 8 bits que se unen para formar un número de 16 bits. Este número es llamado BRR y actúa como una especie de divisor del reloj LSPCLK. Es decir, las mayores velocidades se conseguirán con BRR pequeñas. La relación entre BRR y la velocidad es la siguiente:

$$\begin{aligned} SCIAsynchronousBaud &= LSPCLK/((BRR+1)*8) \\ BRR &= LSPCLK/(SCIAsynchronousBaud*8)-1 \\ 1 &\leq BRR < 65536 \end{aligned} \tag{B.1}$$

Por su parte, el protocolo SPI tiene una frecuencia máxima igual a la del reloj LSPCLK dividido por 4, aunque se deben tomar precauciones para no superar la frecuencia admitida por los GPIO

donde se realicen las conexiones. Como en el caso anterior, existe un divisor encargado de seleccionar la velocidad llamado BRR (SPIBRR en este caso):

$$\begin{aligned} SPIBaudRate &= LSPCLK/(SPIBRR+1) \\ 3 &\leq SPIBRR < 128 \end{aligned} \tag{B.2}$$

De esta forma, las frecuencias más altas seleccionables para la UART son 12.5 MHz, 8.33 MHz y 6.25 MHz y para el SPI 50 MHz, 40 MHz y 33.3 MHz.

Un aspecto importante de la implementación de ambos protocolos por parte de Texas Instruments es que la UART tiene registros independientes en la salida y la entrada mientras que el SPI comparte el registro de más bajo nivel entre la salida y la entrada de forma que, cuando llegan bits se cargan por la parte izquierda y luego se copian en otro registro y cuando se escribe se copia el dato en el primer registro y van saliendo por la derecha conforme se mandan. En nuestro sistema no será problemático al utilizar dos SPI diferentes en los que ninguno utiliza la línea de recepción.

Apéndice C
Librería del expansor de pines I2C generada por Doxygen

PCA9538 library

Generated by Doxygen 1.9.4

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 PCA Struct Reference

`#include <pca9538.h>`

Data Fields

- Uint16 Inreg
- Uint16 Outreg
- Uint16 Polreg
- Uint16 Confreg

3.1.1 Field Documentation

3.1.1.1 Confreg

`Uint16 Confreg`

Copy of configuration device register. This register indicates the direction of the GPIO pins. This register is synchronized at the start of running.

3.1.1.2 Inreg

`Uint16 Inreg`

Copy of inputs device register. The register in device saves the value of inputs in real time. This register is updated in every reading.

3.1.1.3 Outreg

```
Uint16 Outreg
```

Copy of outputs device register. This register is updated in every writing.

3.1.1.4 Polreg

```
Uint16 Polreg
```

Copy of polarity device register.

The documentation for this struct was generated from the following file:

- pca9538.h

Chapter 4

File Documentation

4.1 pca9538.c File Reference

```
#include "pca9538.h"
#include "F2837xS_device.h"
#include "F2837xS_Examples.h"
```

Macros

- #define INT 21

Functions

- void PCA_Config (void)

  PCA_Config Configure I2C for PCA9538.
- void PCA_InOut (Uint16 bits)

  PCA_InOut Set the direction of GPIO pins (input or output).
- void PCA_WriteByte (Uint16 bits)

  PCA_WriteByte Write the value of outputs GPIO bits.
- void PCA_WriteBit (Uint16 bit, Uint16 value)

  PCA_WriteBit Write the value of one output GPIO bit. You can write more than one bit at the same function call if the value is the same, i.e. you can specify more than one pin but only one value and all specified pins will be writed with value.
- Uint16 PCA_ReadByte (void)

  PCA_ReadByte Read the value of inputs GPIO bits. If an output pin is read, the value of this bit is undefined.
- Uint16 PCA_ReadBit (Uint16 bit)

  PCA_ReadBit Read the value of one input GPIO bit.
- void PCA_WriteData (Uint16 Command, Uint16 Data)

  PCA_WriteData Write data in a register specified by Command.
- Uint16 PCA_ReadData (Uint16 Command)

  PCA_ReadData Read data in a register specified by Command.
- void PCA_intPin_config (void)

  Helpful function to configure interrupt pin of the device.
- Uint16 ChangedIn (void)

  ChangedIn Informs about pins that caused an interrupt.

Variables

- PCA IOexp

4.1.1 Macro Definition Documentation

4.1.1.1 INT

```
#define INT 21
```

4.1.2 Function Documentation

4.1.2.1 ChangedIn()

```
Uint16 ChangedIn (
            void  )
```

ChangedIn Informs about pins that caused an interrupt.

Returns

Bit-masks of pins that have changed.

4.1.2.2 PCA_Config()

```
void PCA_Config (
            void  )
```

PCA_Config Configure I2C for PCA9538.

4.1.2.3 PCA_InOut()

```
void PCA_InOut (
            Uint16 bits )
```

PCA_InOut Set the direction of GPIO pins (input or output).

Parameters

bits	16-bits argument that indicates the direction of pines: 1 as input and 0 as output. 8 MSBs are ignored.

4.1.2.4 PCA_intPin_config()

```
void PCA_intPin_config (
            void  )
```

Helpful function to configure interrupt pin of the device.

4.1.2.5 PCA_ReadBit()

```
Uint16 PCA_ReadBit (
            Uint16 bit )
```

PCA_ReadBit Read the value of one input GPIO bit.

Parameters

bit	Bit-mask that indicates a pin to read.

Returns

Value of the pin. High voltage is indicated as 1. Low voltage is indicated as 0.

4.1.2.6 PCA_ReadByte()

```
Uint16 PCA_ReadByte (
            void  )
```

PCA_ReadByte Read the value of inputs GPIO bits. If an output pin is read, the value of this bit is undefined.

Returns

16-bit number that indicates the value of every bit. 8 MSBs do not have useful information.

4.1.2.7 PCA_ReadData()

```
Uint16 PCA_ReadData (
            Uint16 Command )
```

PCA_ReadData Read data in a register specified by Command.

Parameters

Command	Specifies the device register to read.
Data	Data to read in the register.

4.1.2.8 PCA_WriteBit()

```
void PCA_WriteBit (
            Uint16 bit,
            Uint16 value )
```

PCA_WriteBit Write the value of one output GPIO bit. You can write more than one bit at the same function call if the value is the same, i.e. you can specify more than one pin but only one value and all specified pins will be writed with value.

Parameters

bit	Bitmask that indicates a pin to write. If this pin is an input, nothing occurs. You can indicate more than one bit this way: BIT1 | BIT5 | BIT6
value	Value to write in pin (0 or 1).

4.1.2.9 PCA_WriteByte()

```
void PCA_WriteByte (
            Uint16 bits )
```

PCA_WriteByte Write the value of outputs GPIO bits.

Parameters

bits	16-bit argument that indicates the value to write in pins. 8 MSBs are ignored. Bits that correspond to input pins (Configure Register) are ignored too.

4.1.2.10 PCA_WriteData()

```
void PCA_WriteData (
            Uint16 Command,
            Uint16 Data )
```

PCA_WriteData Write data in a register specified by Command.

Parameters

Command	Specifies the device register to write in.
Data	Data to write in the register.

4.1.3 Variable Documentation

4.1.3.1 IOexp

```
PCA IOexp  [extern]
```

4.2 pca9538.h File Reference

C2000 Library for PCA9538 GPIO expander. It is useful to configure the device, read inputs, write outputs and handle interrupts.

```
#include "F2837xS_device.h"
#include "F2837xS_Examples.h"
```

Data Structures

- struct PCA

Macros

- #define HighSpeed
- #define ADDR 0x0070
- #define INREG 0x00
- #define OUTREG 0x01
- #define POL 0x02
- #define CONFREG 0x03
- #define ALL 0xFF
- #define NONE 0x0000
- #define BIT0 0x0001
- #define BIT1 0x0002
- #define BIT2 0x0004
- #define BIT3 0x0008
- #define BIT4 0x0010
- #define BIT5 0x0020
- #define BIT6 0x0040
- #define BIT7 0x0080

Functions

- void PCA_Config (void)

  PCA_Config Configure I2C for PCA9538.
- void PCA_InOut (Uint16 bits)

  PCA_InOut Set the direction of GPIO pins (input or output).
- void PCA_WriteByte (Uint16 bits)

  PCA_WriteByte Write the value of outputs GPIO bits.
- void PCA_WriteBit (Uint16 bit, Uint16 value)

  PCA_WriteBit Write the value of one output GPIO bit. You can write more than one bit at the same function call if the value is the same, i.e. you can specify more than one pin but only one value and all specified pins will be writed with value.
- void PCA_WriteData (Uint16 Command, Uint16 Data)

  PCA_WriteData Write data in a register specified by Command.
- Uint16 PCA_ReadByte (void)

  PCA_ReadByte Read the value of inputs GPIO bits. If an output pin is read, the value of this bit is undefined.
- Uint16 PCA_ReadBit (Uint16)

  PCA_ReadBit Read the value of one input GPIO bit.
- Uint16 PCA_ReadData (Uint16)

  PCA_ReadData Read data in a register specified by Command.
- Uint16 ChangedIn (void)

  ChangedIn Informs about pins that caused an interrupt.
- void PCA_intPin_config (void)

  Helpful function to configure interrupt pin of the device.

4.2.1 Detailed Description

C2000 Library for PCA9538 GPIO expander. It is useful to configure the device, read inputs, write outputs and handle interrupts.

Version

1.0

Date

17/06/2022

Author

Juan Jimenez

```
PCA IOexp={0,0,0,0};
interrupt void pca_int(void);
int main(){
    PCA_Config();
    EALLOW; // This is needed to write to EALLOW protected registers
    PieVectTable.XINT1_INT = &pca_int; //Save interruption
    EDIS; // This is needed to disable write to EALLOW protected registers
    PCA_intPin_config();
    PCA_InOut(ALL);
    while(1);
}
interrupt void pca_int(void){
Uint16 pins;
pins=ChangedIn();
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

4.2.2 Macro Definition Documentation

4.2.2.1 ADDR

```
#define ADDR 0x0070
```

Device I2C Slave address. 2 LSBs are established by hardware. In this case, they are connected to ground.

4.2.2.2 ALL

```
#define ALL 0xFF
```

Bit-mask helpful macro to write a TRUE value in all bits of a register.

4.2.2.3 BIT0

```
#define BIT0 0x0001
```

Bit-mask helpful macro to write a TRUE value in LSB bit of a register. Binary value: 00000001

4.2.2.4 BIT1

```
#define BIT1 0x0002
```

Bit-mask helpful macro to write a TRUE value in indicated bit of a register. Binary value: 00000010

4.2.2.5 BIT2

```
#define BIT2 0x0004
```

Bit-mask helpful macro to write a TRUE value in indicated bit of a register. Binary value: 00000100

4.2.2.6 BIT3

```
#define BIT3 0x0008
```

Bit-mask helpful macro to write a TRUE value in indicated bit of a register. Binary value: 00001000

4.2.2.7 BIT4

```
#define BIT4 0x0010
```

Bit-mask helpful macro to write a TRUE value in indicated bit of a register. Binary value: 00010000

4.2.2.8 BIT5

```
#define BIT5 0x0020
```

Bit-mask helpful macro to write a TRUE value in indicated bit of a register. Binary value: 00100000

4.2.2.9 BIT6

```
#define BIT6 0x0040
```

Bit-mask helpful macro to write a TRUE value in indicated bit of a register. Binary value: 01000000

4.2.2.10 BIT7

```
#define BIT7 0x0080
```

Bit-mask helpful macro to write a TRUE value in MSB bit of a register. Binary value: 10000000

4.2.2.11 CONFREG

```
#define CONFREG 0x03
```

Command to specify the reading or writing in device Configuration Register.

4.2.2.12 HighSpeed

```
#define HighSpeed
```

Use the device at high speed mode: 400kHz

4.2.2.13 INREG

```
#define INREG 0x00
```

Command to specify the reading or writing in device Inputs Register.

4.2.2.14 NONE

```
#define NONE 0x0000
```

Bit-mask helpful macro to write a FALSE value in all bits of a register.

4.2.2.15 OUTREG

```
#define OUTREG 0x01
```

Command to specify the reading or writing in device Outputs Register.

4.2.2.16 POL

```
#define POL 0x02
```

Command to specify the reading or writing in device Inputs Polarity Register.

4.2.3 Function Documentation

4.2.3.1 ChangedIn()

```
Uint16 ChangedIn (
            void  )
```

ChangedIn Informs about pins that caused an interrupt.

Returns

Bit-masks of pins that have changed.

4.2.3.2 PCA_Config()

```
void PCA_Config (
            void  )
```

PCA_Config Configure I2C for PCA9538.

4.2.3.3 PCA_InOut()

```
void PCA_InOut (
            Uint16 bits )
```

PCA_InOut Set the direction of GPIO pins (input or output).

Parameters

bits	16-bits argument that indicates the direction of pines: 1 as input and 0 as output. 8 MSBs are ignored.

4.2.3.4 PCA_intPin_config()

```
void PCA_intPin_config (
            void  )
```

Helpful function to configure interrupt pin of the device.

4.2.3.5 PCA_ReadBit()

```
Uint16 PCA_ReadBit (
            Uint16 bit )
```

PCA_ReadBit Read the value of one input GPIO bit.

Parameters

bit	Bit-mask that indicates a pin to read.

Returns

Value of the pin. High voltage is indicated as 1. Low voltage is indicated as 0.

4.2.3.6 PCA_ReadByte()

```
Uint16 PCA_ReadByte (
            void  )
```

PCA_ReadByte Read the value of inputs GPIO bits. If an output pin is read, the value of this bit is undefined.

Returns

16-bit number that indicates the value of every bit. 8 MSBs do not have useful information.

4.2.3.7 PCA_ReadData()

```
Uint16 PCA_ReadData (
            Uint16 Command )
```

PCA_ReadData Read data in a register specified by Command.

Parameters

Command	Specifies the device register to read.
Data	Data to read in the register.

4.2.3.8 PCA_WriteBit()

```
void PCA_WriteBit (
            Uint16 bit,
            Uint16 value )
```

PCA_WriteBit Write the value of one output GPIO bit. You can write more than one bit at the same function call if the value is the same, i.e. you can specify more than one pin but only one value and all specified pins will be writed with value.

Parameters

bit	Bitmask that indicates a pin to write. If this pin is an input, nothing occurs. You can indicate more than one bit this way: BIT1 | BIT5 | BIT6
value	Value to write in pin (0 or 1).

4.2.3.9 PCA_WriteByte()

```
void PCA_WriteByte (
            Uint16 bits )
```

PCA_WriteByte Write the value of outputs GPIO bits.

Parameters

bits	16-bit argument that indicates the value to write in pins. 8 MSBs are ignored. Bits that correspond to input pins (Configure Register) are ignored too.

4.2.3.10 PCA_WriteData()

```
void PCA_WriteData (
            Uint16 Command,
            Uint16 Data )
```

PCA_WriteData Write data in a register specified by Command.

Parameters

Command	Specifies the device register to write in.
Data	Data to write in the register.

4.3 pca9538.h

Go to the documentation of this file.

```
1 /*
2  * pca9538.h
3  *
4  *  Created on: 12 jun. 2022
5  *      Author: Juan
6  */
34 #ifndef PCA9538_H_
35 #define PCA9538_H_
36
37 #include "F2837xS_device.h"        // F2837xS Headerfile Include File
38 #include "F2837xS_Examples.h"      // F2837xS Examples Include File
39
40 #define HighSpeed
41 #define ADDR 0x0070
43 #define INREG 0x00
44 #define OUTREG 0x01
45 #define POL 0x02
46 #define CONFREG 0x03
49 #define ALL 0xFF
50 #define NONE 0x0000
51 #define BIT0 0x0001
52 #define BIT1 0x0002
53 #define BIT2 0x0004
54 #define BIT3 0x0008
55 #define BIT4 0x0010
56 #define BIT5 0x0020
57 #define BIT6 0x0040
58 #define BIT7 0x0080
60 typedef struct{
61     Uint16 Inreg;
62     Uint16 Outreg;
63     Uint16 Polreg;
64     Uint16 Confreg;
65 }PCA;
66
70 void PCA_Config(void);
71
77 void PCA_InOut(Uint16 bits);
78
83 void PCA_WriteByte(Uint16 bits);
84
90 void PCA_WriteBit(Uint16 bit, Uint16 value);
91
97 void PCA_WriteData(Uint16 Command, Uint16 Data);
98
103 Uint16 PCA_ReadByte(void);
104
110 Uint16 PCA_ReadBit(Uint16);
111
117 Uint16 PCA_ReadData(Uint16);
118
123 Uint16 ChangedIn(void);
124
128 void PCA_intPin_config(void);
129
130
131
132 #endif /* PCA9538_H_ */
```

Index

Índice de Figuras

Bibliografía

[1] J. David, L. Parada, and O. M. H. Gómez, "Electrónica de potencia: Aplicación en fuentes de energía renovables," *Rev. Invest. Univ. Quindío.(Col.)*, vol. 25, pp. 154–158, 2014.

[2] J. E. Gaviria Ríos, J. H. Mora Guzmán, and J. Ramiro Agudelo, "Historia de los motores de combustión interna." [Online]. Available: https://revistas.udea.edu.co/index.php/ingenieria/article/view/326361/20783635

[3] J. C. Schallenberg Rodríguez, *Energías renovables y eficiencia energética*. Instituto Tecnológico de Canarias, 2008.

[4] "¿Cómo se convirtió el petróleo en el combustible de la modernidad?" [Online]. Available: https://www.lavanguardia.com/historiayvida/historia-contemporanea/20171211/47313522972/como-se-convirtio-el-petroleo-en-el-combustible-de-la-modernidad.html

[5] J. Carles, A. Manubens, F. Xavier, and B. Salom, "El gas de hulla en la europa latina hasta mediados del siglo xx, historia, tecnología e innovación," 2021.

[6] J. Feal Vázquez, "El mundo actual del petróleo," 2006.

[7] "World oil production and peaking outlook." [Online]. Available: https://web.archive.org/web/20080625012113/http://peakoil.nl/wp-content/uploads/2006/09/asponl_2005_report.pdf

[8] "Algeria: Opec has reached its production limit." [Online]. Available: https://www.greencarcongress.com/2005/03/algeria_opec_ha.html

[9] "Proyecto gemi-reads." [Online]. Available: https://www.repsol.com/es/sostenibilidad/medio-ambiente/gemi-reads/index.cshtml

[10] J. Luis and U. Fernández, "El cambio climático: Sus causas y efectos medioambientales."

[11] A. M. Camarasa Belmonte, "Algunas reflexiones sobre la percepción del cambio climático en una muestra de población adulta de nivel cultural medio." [Online]. Available: https://roderic.uv.es/handle/10550/39752

[12] S. Montecinos and D. Carvajal, "Energías renovables: Escenario actual y perspectivas futuras," 2018. [Online]. Available: https://www.researchgate.net/publication/343922050

[13] "Las energías renovables superan por primera vez a la generación fósil en la unión europea." [Online]. Available: https://www.pv-magazine.es/2020/07/23/las-energias-renovables-superan-por-primera-vez-a-la-generacion-fosil-en-la-ue/

[14] V. G. Monopoli, A. Marquez, J. I. Leon, M. Liserre, G. Buticchi, L. G. Franquelo, and S. Vazquez, "Applications and modulation methods for modular converters enabling unequal cell power sharing: Carrier variable-angle phase-displacement modulation methods," *IEEE Industrial Electronics Magazine*, vol. 16, no. 1, pp. 19–30, 2022.

[15] J. I. Leon, S. Vazquez, and L. G. Franquelo, "Multilevel converters: Control and modulation techniques for their operation and industrial applications," *Proceedings of the IEEE*, vol. 105, no. 11, pp. 2066–2081, 2017.

[16] A. Márquez, "Variable-angle modulation techniques for modular power converters," Ph.D. dissertation, Seville University, 2018.

[17] A. Márquez Alcaide, J. I. Leon, R. Portillo, J. Yin, W. Luo, S. Vazquez, S. Kouro, and L. G. Franquelo, "Variable-angle ps-pwm technique for multilevel cascaded h-bridge converters with large number of power cells," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 8, pp. 6773–6783, 2021.

[18] *ISO308x Isolated 5-V Full- and Half-Duplex RS-485 Transceivers datasheet (Rev. I)*, 2017th ed., Texas Instruments, 2008.

[19] T. Instruments, *PCA9538 Remote 8-Bit I2C AND SMBus Low-power I/O Expander with Interrupt Output, Reset, and Configuration Registers datasheet (Rev. G)*, march, 2021 ed., 2006.