Disseny i Implementació d'un Sistema de Comunicació per a Controladors de Motocicletes

Pol Martinez Salom

Resumen– Se ha propuesto como objetivo para éste trabajo idear e integrar un sistema de comunicación basado en Ethernet para un microcontrolador, siendo éste utilizado en un cuadro de instrumentación para motocicletas, y además comparar y evaluar su funcionamiento en contraste a otros métodos de comunicación relevantes en el mundo de la automoción, siendo el principal sujeto el bus CAN. El proyecto ha permitido llevar a cabo el desarrollo de controladores hardware además de una serie de librerías software a fin de tener la posibilidad de utilizarse de forma conjunta con aplicaciones de alto nivel en motocicletas.

Palabras clave– Automoción, Bus, Comunicación, Driver, Ethernet, Firmware, Microcontrolador

Abstract– The proposed objective for this project has been to design and implement an Ethernet based communication system for an existing microcontroller, which has its main use in motorcycle dashboards. The communication system performance and overall functionality has also been evaluated and compared with other relevant communication methods, with the most prominent being the CAN bus. The project has also allowed to develop hardware controllers as well as software libraries for Ethernet, in an effort to possibly allow the system to be used in higher level applications in motorcycles.

Keywords– Automotion, Bus, Communication, Driver, Ethernet, Firmware, Microcontroller

1 Introducción y motivación

La informática en el mundo automovilístico tiende a evolucionar lentamente, priorizando seguridad y familiaridad antes que innovación. Además, en el área específica de las motocicletas, donde se trabaja con una limitación de espacio y precio, los avances tecnológicos tienden a ir varios años por detrás respecto a vehículos como los coches, por lo que nos encontramos con unas prestaciones ligeramente más limitadas.

En términos de comunicación, el conocido bus CAN (Controller Area Network)[1] es el encargado de la comunicación entre componentes y controladores en una motocicleta, y asegura un funcionamiento a tiempo real, con alta resiliencia a fallos y a colisiones.

• E-mail de contacto: 1563964@uab.cat
• Mención realizada: Enginyeria de Computadors
• Trabajo tutorizado por: Dolores Rexachs
• Curso 2022/23

Sin embargo, su velocidad no escala adecuadamente para sistemas donde necesitamos un ancho de banda mucho mayor (video, sonido, reprogramación), limitando su uso a sistemas de control a tiempo real y diagnosis.

Una alternativa que está surgiendo en el sector es el Automotive Ethernet, que utiliza capas de transporte físico diseñadas para su implementación en el mundo automovilístico. Ésta alternativa provee el sistema con un ancho de banda de hasta 100 mbps, permitiendo la implementación de varios sistemas de alta demanda, incluso la integración con sistemas y servicios basados en nube (clima, hora, actualizaciones automáticas).

El proyecto tiene como uno de sus fines poder ser integrado en un entorno actualmente en desarrollo, contando con un hardware ya especificado además de un ecosistema de software y capa de aplicación ya existente. Va a ser esencial entonces, desarrollar éste proyecto en verso al microprocesador Renesas RH850, destinado a fines automovilísticos.

2 BASE TEÓRICA

Debido a que el proyecto utiliza diversos sistemas ya existentes que interactuan entre sí, es importante establecer una base de conocimiento teórico para ver los hechos más relevantes de cada uno de los protocolos y sistemas que se van a utilizar.

2.1. Ethernet

Ethernet es un conjunto de especificaciones de comunicación correspondientes tanto a la capa de transporte físico como a la de enlace de datos[2], enfocado a la comunicación en red, es decir, de distintos nodos utilizando el mismo bus físico para interactuar entre sí. Ésta funcionalidad concreta no se ha explorado en éste trabajo, pues depende de la aplicación que se le quiera dar al sistema.

La comunicación entre distintos *endpoints* se realiza mediante tramas Ethernet, que contienen información sobre la dirección del transmisor y destinatario así como información como el tamaño o tipo de la trama.

Durante el trabajo no se ha tenido en cuenta el tipo de trama, sólo el destinatario y la forma general de la cabecera, mientras que los detalles se han reservado para una aplicación de más alto nivel, ya que no entra dentro del alcance del proyecto.

2.2. CAN Bus

El bus CAN, o Control Area Network es un protocolo de bus enfocado a mensajes entre distintas ECUs(Electronic Control Unit) dentro de un vehículo, basándose en un único bus compartido, y por lo tanto tiene una estructura similar a la de Ethernet. El protocolo fue creado con el fin de utilizarse en sistemas de tiempo real, por lo que posee un sistema de prioridades de transmisión que facilita el uso compartido del bus.

2.3. BroadR-Reach

BroadR-Reach (BRR) establece el protocolo de capa física para redes Ethernet, destinado principalmente a aplicaciones automotive dónde la tolerancia a fallos es especialmente relevante. Fue definido por la compañía Broadcom, y adoptado por OPEN Alliance cómo estándar para comunicación ethernet en vehículos. Más adelante fué estandarizado por IEEE 802.3 con el nombre 100BASE-T1[3].

Por encima de ésta capa física se transmite Ethernet estándar, lo que permite una rápida implementación de otros conocidos protocolos cómo IP y TCP/IP.

2.4. Ethernet Physical Layer

Dentro del modelo de capas OSI, BroadR-Reach representa la capa de transferencia física entre dos puntos. En ambos destinos, la señal es procesada por un dispositivo conocido como *Physical Layer Transciever* (PHY). Éste dispositivo se encarga principalmente de la conversión entre señales digitales y la señal específica del medio donde esté conectado, sirviendo, por ejemplo, de intermediario entre una capa de transporte Ethernet estándar y un bus BroadR-Reach.

Por otro lado, en un dispositivo potencialmente distinto, se encuentra un módulo MAC de Ethernet. El módulo MAC se encarga de iniciar transferencias de datos de forma independiente del medio físico dónde se vaya a realizar, a través del PHY.

Éstos dos módulos se unen entre ellos por una *Media Independent Interface* (MII). Ésta interfície estandarizada entre módulos MAC y PHY actuar transparentemente el uno del otro. Cuenta con diversas señales tanto de lectura y escritura, detección y propagación de errores, y un bus de control conocido como Serial MII (SMII).

2.5. RH850

El RH850 de Renesas[4] es una familia de microcontroladores de 32-bits con fines automotivos. Los microcontroladores ofrecen una potencia relativamente alta además de una diversidad de módulos hardware, entre los que se encuentran comunicadores de CAN y Ethernet, pero no el PHY. El modelo concreto que se va a ver utilizado durante el desarrollo del proyecto es el RH850/D1M1A [5], dedicado a clústeres de instrumentos y equipado con un procesador gráfico.

2.6. Direct Memory Access

El acceso directo a memoria es una herramienta controlada por hardware presente en muchos microprocesadores con el fin de orquestar transmisiones de memoria entre distintos módulos hardware y la memoria principal (RAM). Su uso principal está en aplicaciones de comunicación, especialmente en esas con un gran ancho de banda y en microprocesadores que no se pueden permitir dedicar todo su tiempo de ejecución en comunicaciones. En el caso de éste proyecto, el procesador RH850 se va a utilizar en un sistema orientado a una producción real, por lo que se va a tener en cuenta que el sistema Ethernet debe funcionar con el mínimo coste de ejecución posible para soportar las aplicaciones de alto nivel que se puedan utilizar como capa superior. Ha sido importante pues una buena configuración del DMA del microcontrolador, como se verá más adelante.

3 OBJETIVOS

El objetivo principal del trabajo es:

- Diseño e implementación de un stack ethernet para un microcontrolador de fines automovilísticos, utilizando como capa de transporte física el protocolo BroadR-Reach (100BASE-T1), destinado a aplicaciones automovilísticas, con alta resiliencia a fallos y teniendo en cuenta la arquitectura del software para una posible aplicación posterior a otros proyectos.

Se tienen como objetivos secundarios:

- Análisis de funcionalidad del protocolo de comunicación, así como sus prestaciones y rendimiento esperado/obtenido.
- Comparativa de prestaciones en verso a protocolos que cumplen un propósito similar

- Implementación en un proyecto ya existente, incorporando funcionalidad adicional de ethernet
- Mantener el sistema altamente modular tanto a nivel de software como de hardware, permitiendo intercambiar las partes de forma sencilla.
- Conseguir que el sistema no tenga dependencia de un sólo dispositivo PHY, sino que éste pueda ser cambiado en producción y reconocido a nivel software, con un repertorio de librerías adaptándose dinámicamente a la oferta de componentes del momento

Como dispositivo *Physical Layer Transciever* (PHY) utilizaremos el NXP TJA1101B [6], que mediante una interfaz *Media Independent Interface* (MII) provee la capa BroadR-Reach que utilizaremos (Fig. [1]). Ésta además, cuenta con un seguido de pruebas y funcionalidades de loopback que facilitará el testeo de las distintas fases. La implementación de ésta funcionalidad construirá una plataforma de desarrollo para posibles proyectos de más complejidad (Streaming de video desde un dispositivo móvil, tratamiento de grandes cantidades de datos etc.) , además de avanzar la tecnología disponible en el mercado de las motocicletas y acercarse un poco más al de los coches.

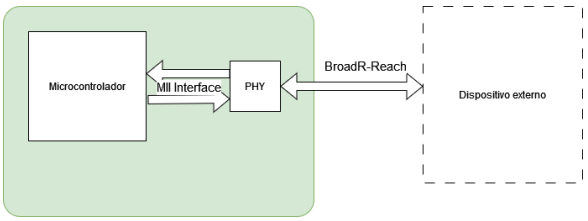

Fig. 1: Diagrama de bloques del sistema

4 METODOLOGÍA

La implementación del protocolo en el microcontrolador ha requerido de un extenso desarrollo software, pues deberá convivir con el resto de firmware y capa de aplicación usados en proyectos de la empresa. Eso significa que se ha debido seguir en todo momento una arquitectura del software correcta, modular y altamente resiliente a fallos. Además, los drivers de bajo nivel han sido pensados para ajustarse a cambios posibles a nivel hardware, como puede ser un cambio en el módulo PHY usado para impulsar el BroadR-Reach.

La placa y el montaje de los distintos componentes vienen montados por la empresa, proviniendo de un proyecto ya existente con la finalidad de poder rápidamente probar la incorporación del módulo Ethernet.

Por lo tanto se empezará el proyecto con una base sólida y probada a nivel hardware que permitirá que el trabajo se centre en la implementación de éste nuevo módulo y minimice los problemas y errores externos que se salen de los límites del trabajo. Además, se dispondrá de una capa de aplicación de un proyecto real, por lo que se podrá ver cómo éste trabajo encaja en un ámbito profesional.

El diagrama resultante del sistema a implementar se puede apreciar en Fig.2.

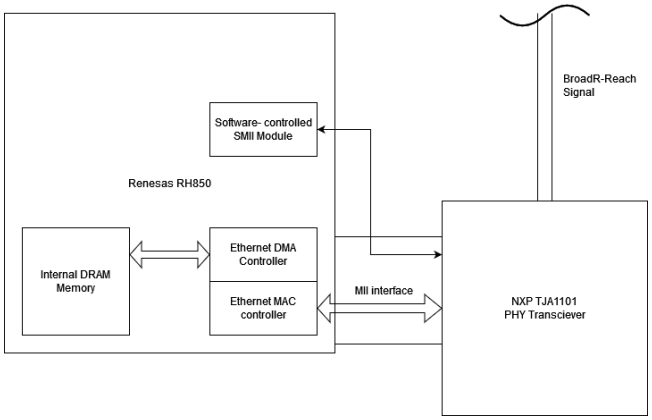

Fig. 2: Diagrama detallado de las partes importantes del sistema

4.1. Planificación inicial

El trabajo realizado ha constado de tres fases bien definidas:

- Crear los drivers de bajo nivel para el módulo Ethernet MII incluido en el microcontrolador RH850, que se encargan de enviar y recibir información desde la memoria del micro hasta el módulo hardware mediante Acceso Directo a Memoria (DMA).
- Comunicar el microcontrolador con el dispositivo PHY mediante un pequeño controlador SMI además del protocolo MII saliente del módulo del microcontrolador.
- Realizar y configurar la comunicación por la capa física, interconectando dispositivos. Ésta es la fase más propensa a errores de transmisión, por lo que se ha implementado un sistema robusto de tratamiento de errores

Además, en caso de que sea posible en el margen temporal, se podría añadir una cuarta fase

- Encontrar un caso de uso e implementarlo en la capa de aplicación ya existente.

El desarrollo ha sido pues progresivo, y ha requerido el meticulosos análisis de la documentación proveída por Renesas e NXP. El proyecto se ha realizado siguiendo el orden descrito de forma que el trabajo se hace desde los componentes más cercanos al microcontrolador hacia a fuera para garantizar una buena eficiencia y metodología a la hora de realizar los tests progresivos (Ver Anexo A.2).

5 DISEÑO Y DESARROLLO

En la siguiente sección se explica el diseño y desarrollo llevado a cabo para cada módulo participante detallado en la figura 2 y 3. Los componentes a tener en cuenta son los siguientes:

- Ensamblado y preparación del hardware
- Librería Genérica SMI
- Librería SMII específica para TJA1101B

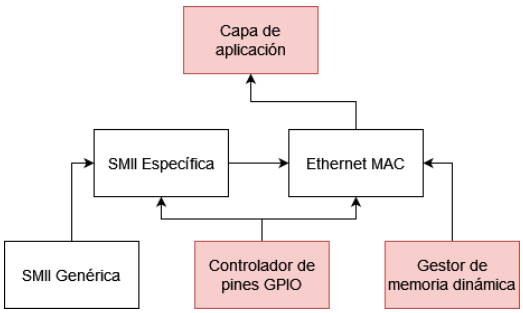

Fig. 3: Jerarquía del software diseñado e implementado. Las flechas representan las inclusiones de las distintas bibliotecas. En rojo, las librerías que no entran en el ámbito de éste proyecto y por lo tanto no han sido implementadas

- Librería Ethernet MAC

El funcionamiento del sistema es dictado en gran medida por el recorrido de la señal Ethernet desde su creación en el microcontrolador hasta que llega a su destino al otro lado del enlace BroadR-Reach. Cada etapa de la transmisión ha sido asignada a una o varias librerías de software encargadas de una única función, a fin de conseguir una capa software modular y sencilla de usar de cara a la aplicación de más alto nivel. Por ejemplo, Si se quiere implementar un servicio de streaming a través de Ethernet, o incluso un servidor web, no se debe requerir de ninguna funcionalidad de bajo nivel adicional, pudiendo usar las librerías desarrolladas en éste trabajo para desarrollar esa aplicación. Finalmente la modularidad permitiría el reuso de éstas librerías para poder ser usadas en microcontroladores distintos, requiriendo solamente de cambios específicos del hardware.

Durante las subsecciones siguientes se explican las partes en las que se ha dividido el sistema y las librerías encargadas de controlar cada una de esas partes. Finalmente se ha desarrollado una librería Ethernet que genera el nexo de unión entre todas las funcionalidades, y que debe servir como interfície para el software de alto nivel desarrollado. Las figuras referentes a la API creada se pueden encontrar en el anexo A.3.

5.1. Ensamblado y preparación del hardware

A la hora de hacer el montaje del hardware necesario, se ha tomado como base la placa y circuitos de un proyecto comercial en desarrollo, con el objetivo de que se tenga una base sólida y estable para el desarrollo del proyecto, así como para la posterior implementación del sistema ethernet con un software de aplicación compatible con el hardware.

El microcontrolador central es el RH850 de Renesas, preparado con un módulo MAC, además de capacidad gráfica y alta velocidad de procesamiento para aplicar el software de aplicación por encima. El dispositivo PHY ha sido el TJA1101B de NXP. éste será montado dentro de la placa del cuadro de instrumentos mediante la interfaz MII con el RH850. Se ha montado un oscilador externo de 25MHZ encargado de generar el clock de las señales Rx y Tx de la interfaz MII, lo que genera la transmisión de 100 Mbps de BroadR-Reach. La conexión BRR sale de la placa por un conector estándar, permitiendo conectar con cualquier dispositivo que soporte el protocolo.

5.2. Librería Genérica SMII

Debido a la necesidad de realizar una comunicación entre el microcontrolador y el dispositivo PHY, se ha desarrollado una librería genérica capaz de utilizar el protocolo SMII (Integrado en la interfaz MII) de cara a proveer llamadas de lectura y escritura a registros del dispositivo PHY. Ésta librería en concreto tiene un propósito genérico, y sólo implementa la comunicación estándar con las primitivas read y write de 16 bytes cada una para comunicarse.

Gracias a la estandarización del protocolo SMII, ha sido posible separar la comunicación mediante éste bus para todos los dispositivos PHY comunicados mediante la interfície MII.

El protocolo SMII no tiene implementación hardware en el microcontrolador usado, por lo que se llevará a cabo a partir de los pines DATA y CLOCK controlados mediante software. Es importante remarcar que la velocidad de transmisión, pues, va a depender de la velocidad de procesado del microcontrolador. Además, debido a la implementación software, toda comunicación va a tener lugar de forma síncrona al programa, lo cual por un lado facilita el flujo de control del programa pero también vuelve el flujo más susceptible a potenciales eventos no controlables (interrupciones externas, bugs de software etc).

Pese a que la implementación del protocolo no resulta especialmente compleja, los tiempos de las señales no tienen porque coincidir con los timings ajustados del dispositivo PHY al otro lado de la conexión. Ésto ha causado que para el desarrollo de la librería se hayan tenido que introducir delays en el driving de puertos para alcanzar los tiempos especificados por el fabricante, en este caso para un clock con periodo de 400ns y señales estables durante 50ns.

Éstos timings son específicos de cada dispositivo, con algunos incluso soportando la velocidad nativa del microprocesador sin necesidad de delays o retrasos entre cambios en las señales. Aun así, se ha optado por unos timings genéricos que garanticen el funcionamiento correcto en la mayor parte de dispositivos, incluso si así no se consigue la mayor velocidad en algunos dispositivos, ya que en la mayor parte de casos, la comunicación SMII es escasa más allá de la inicialización del dispositivo en sí.

5.3. Librería Específica SMII TJA1101B

Des del punto de vista de ingeniería del software uno de los objetivos principales a afrontar durante la realización del trabajo es el de mantener un sistema modular que sea capaz de adaptarse a cambios de hardware. Siendo el objetivo desarrollar una librería Ethernet genérica para la arquitectura, sirviendo para en un futuro poder hacer una integración en proyectos destinados a producción. Uno de los puntos interesantes pues, es no limitar la etapa de producción a un dispositivo PHY concreto, sino disponer de un repertorio de librerías específicas para cada hardware, que después serán enlazadas dinámicamente a partir del hardware que se pueda identificar por SMII.

Es entonces esencial hacer una separación de la capa SMII única de cada hardware, en el caso de éste trabajo se trata de el NXP TJA1101B. El módulo actual por lo tanto, utiliza la librería genérica SMII a modo de driver para lanzar peticiones de lectura y escritura, cada una de ellas

siguiendo los ajustes de cada dispositivo concreto, que gracias a la estandarización del protocolo por parte de ISO [3] contiene una serie de funcionalidades comunes que es pueden asumir que existen, sobretodo referentes a estados internos, errores y a funcionalidades genéricas, permitiendo al módulo principal hacer llamadas a métodos propios de la librería que se adaptan a cada dispositivo específico. Además, cada método de la librería incorpora un sistema de detección y corrección de errores genérico permitiendo a la librería principal diagnosticar varios problemas en cada una de las etapas de conexión.

El software de la librería concretamente desarrollada para el TJA1101B contiene varias primitivas de configuración del dispositivo. Entre ellas destaca el monitoraje del enlace, o *link* proporcionado por el dispositivo PHY y otra conexión al otro lado del cable BRR. Sin éste link, no tiene sentido la transmisión de datos, ya que las señales mandadas no son recibidas por nadie. La librería además incluye la opción de mandar mensajes de loopback externos al microprocesador, es decir, transmitiéndose a través de los sistemas proporcionados por el sistema PHY sin necesidad de tener un link establecido, además de la posibilidad de transmitir éstos mensajes a través de BroadR-Reach hacia una interfaz externa. Fig. [4] y [5].

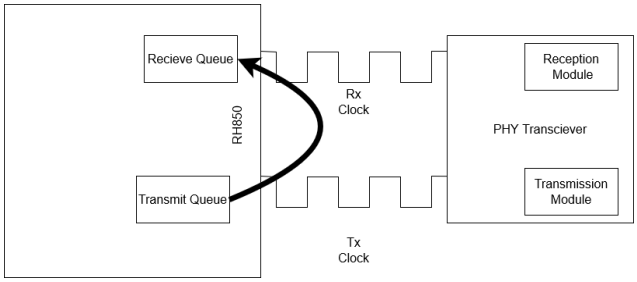

Fig. 4: Loopback interno

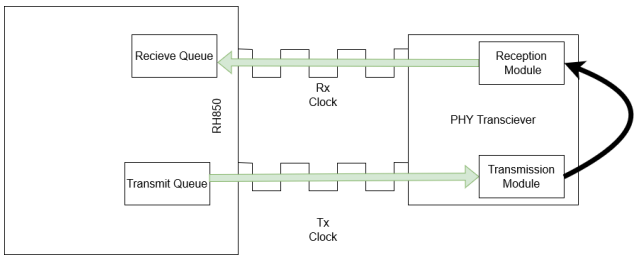

Fig. 5: Loopback externo

En la Fig. [4] Se aprecia el flujo de datos en un modo loopback interno, donde las tramas Ethernet son enrutadas internamente en el microcontrolador, pasando directamente de la salida hasta la entrada. Éste modo utiliza las señales de reloj pertinentes a la interfaz MII para controlar la entrada y salida de datos, por lo que el dispositivo PHY debe estar operativo para utilizarse. Éste funcionamiento estaba listo e implementado en el informe de progreso I.

En la Fig. [5] Podemos ver el flujo de datos en un modo loopback externo, es decir, transparente al microcontrolador.

En éste caso todos los datos provenientes de la salida del microcontrolador son transmitidos con éxito a través de la interfície MII, siendo el dispositivo PHY el encargado de reproducir éstas mismas señales entrantes en su salida para establecer un modo loopback. Éste modo comprueba el funcionamiento correcto del PHY Transciever y de la interfície MII una vez se ha podido comprobar el funcionamiento interno del microcontrolador.

5.4. Ethernet MAC

El firmware controlador de la funcionalidad MAC del microcontrolador ha sido inicialmente, creado con el fin de proveer funcionalidad de loopback básica contra el dispositivo PHY. Su objetivo principal es el de arbitrar y controlar el acceso a memoria asíncrono a los datos y metadatos usados para las transmisiones hacia el PHY, además de llamar a la librería SMII genérica para iniciar conexiones.

El diagrama del funcionamiento básico de la libreria se puede apreciar en la Fig.[6].

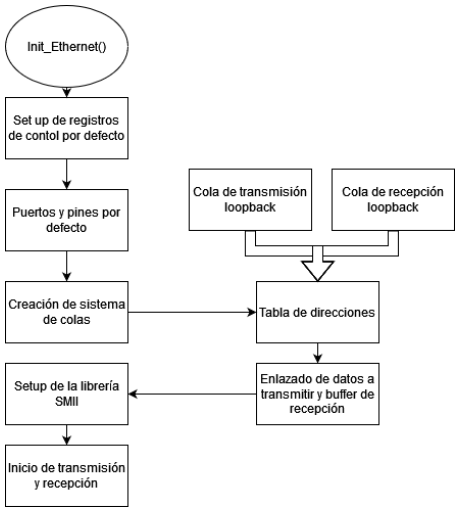

Fig. 6: Funcionamiento general de la librería MAC

Ésta librería, además, sirve como captación de errores propagados en cualquiera de las otras etapas, y permite diagnosticar y tratar de solucionar el problema, posiblemente alertando al usuario y al programador, por lo que el conexionado con otros módulos, especialmente PHY, deben ser robustos.

Para garantizar un buen funcionamiento de la totalidad de partes que conforman el sistema, la librería se ha dotado de una gestión de errores que distingue entre errores recuperables y no recuperables, además de mantener una pila de los errores ocurridos y su función relacionada. De ésta forma, en tiempo de ejecución se puede observar mediante un debugger conectado al microcontrolador los distintos errores ocurridos. Los errores recuperables se consideran aquellos que ocurren en tiempo de ejecución y no han sido causados por un mal uso de la librería, por ejemplo errores de transmisión causados por un enlace BRR imperfecto o un problema al otro lado de la línea. Éstos errores pueden ser reportados o no dependiendo de variables de preprocesamiento a la hora de compilar el programa. Éso resulta útil a la hora de evaluar tests cómo los que se han realizado en éste trabajo, o para poder comprobar el correcto funcionamiento de la librería. El reporte de éstos errores deberá estar desactivado de cara a el uso normal de la librería, ya que ya existen interrupciones hardware preparadas para lidiar con ellos.

Por otro lado, los errores no recuperables son todos aquellos que se producen por un mal uso de la librería por parte del usuario, y son reportados por el mismo sistema comentado anteriormente de forma que en el debugger se puedan apreciar los errores ocurridos. Se consideran errores no recuperables por ejemplo las llamadas a funciones con parámetros incorrectos o en momentos inoportunos, como por ejemplo iniciar una transmisión mientras el sistema está en modo configuración. Además, si el sistema en algún momento entra en algún estado indefinido por algún problema inesperado (por ejemplo overrun de interrupciones). Éstos errores, de forma parecida a los recuperables, pueden ser desactivados de la misma forma que los otros, pasando a servir como valores de retorno de cara a la propagación de errores. También se pueden tratar como errores recuperables, en cuyo caso sí que son reportados pero la ejecución no será interrumpida.

A la hora de formalizar el uso que se le quiere dar a la biblioteca, se ha diseñado una API de funcionamiento para la librería, dando unas pautas concisas sobre el uso de ésta. Es necesario clarificar que ha habido diversas iteraciones sobre el uso y el funcionamiento interno de la librería, debido sobretodo a la necesidad de establecer un balance sobre la abstracción del hardware que provee la biblioteca y las funcionalidades de ésta. Ha sido fundamental entonces establecer un sistema base robusto que permita ambas, sin ser especialmente complejo pero sin sacrificar funcionalidad.

Se ha llegado a la conclusión que el alcance y funcionalidad de la librería se debe limitar a:

- Inicializar los registros y el módulo hardware a una configuración simple, con la opción de modificar algunos de los valores mediante llamadas a funciones en tiempo de ejecución. Es importante entonces proveer una robusta herramienta de gestión de errores, indicando en la pila de errores si hay configuraciones conflictivas entre sí
- Arrancar el dispositivo PHY y enlazar las funciones de forma dinámica en tiempo de ejecución, usando la pila de errores también para indicar al usuario si ha habido algún problema de comunicación entre los módulos, El módulo PHY será también inicializado a una configuración básica.
- Proveer un gestor de colas de recepción y transmisión para el DMA del módulo ethernet. El hardware debe interactuar de forma asíncrona con éstas colas y la librería debe asegurarse de orquestar el acceso a memoria tanto del usuario como del hardware. El usuario por lo tanto define la estructura de las colas de firma indirecta (abstracción de la complejidad), teniendo una serie de primitivas que cubran la mayoría de casos de uso así como una primitiva personalizada que no provee abstracción pero permite sacar todo el potencial al hardware.

  Además, la memoria alojada para las colas tiene que ser de tipo dinámica, ya que el protocolo Ethernet no establece un tamaño fijo de tramas, y la naturaleza asíncrona del hardware necesita buffers de tamaño no conocido hasta la ejecución.
- Funciones para definir rutinas de tratamiento de interrupciones en base al sistema de colas, permitiendo al usuario establecer rutinas personalizadas sobre las colas mencionadas anteriormente. Las interrupciones soportadas a nivel hardware son por cada paquete leído, al escribir cierto elemento de la cola y al completar una cola entera. La librería se encarga de inicializar las interrupciones y enlazarlas con las rutinas propuestas por el usuario, además de recopilar información útil incluso si el usuario no va a utilizar la rutina (por ejemplo mantener un registro de si una cola ha terminado de procesarse por hardware).
- Funciones para interrogar el estado de la librería, por ejemplo, el estado del dispositivo PHY, estado de las colas, dirección MAC, errores etc.
- Herramientas auxiliares para facilitar el desarrollo con la librería, como pueden ser estructuras de headers ethernet de distintos tipos listas para usar en las colas y enums para establecer parámetros concretos en lugar de valores numéricos al trabajar con la librería.

La interfaz MII es controlada por la lógica de los controladores SMII mencionaros anteriormente además de la asignación de pines de entrada y salida que harán la función de MII.

El problema más complejo a la hora de realizar ésta tarea ha sido conseguir verificar el funcionamiento, pues ésta no tiene la capacidad de comprobar su propio funcionamiento sin la ayuda de otras librerías. Por ejemplo, no es capaz de realizar una funcionalidad de loopback de sus colas de transmisión hacia las de recepción sin tener un clock entrante por los pines de la interfaz MII, requiriendo así del funcionamiento y configuración del módulo PHY.

5.5. Implementación total del sistema

Como ya se ha comentado en las subsecciones anteriores, el ecosistema de librerías creadas para lograr la implementación de funcionalidad ethernet ha sido completado en su totalidad según los objetivos propuestos. De cara a poder considerar que el sistema está terminado, se han tenido que realizar pruebas contra un dispositivo exterior al sistema, que quede al otro lado de la conexión BroadR-Reach y que permita evaluar su funcionamiento correcto.

De cara a elegir qué dispositivo conectar para realizar las pruebas, se ha considerado el uso de una interfície que pueda ser operada mediante un ordenador de sobremesa, utilizando un software capaz de reconocer y analizar tramas, así como de generarlas. Además, el sistema debe permitir un uso a tiempo real para la posterior avaluación de rendimiento así como un posible uso futuro para desarrollar sistemas automobilísticos que utilicen el protocolo Ethernet montado durante éste proyecto.

Se ha decidido utilizar las herramientas proporcionadas por Vector, incluyendo el software CANoe y la interfaz de redes VN6520. Ésta combinación de hardware y software ya se estaba utilizando para medir y simular buses CAN en proyectos en desarrollo por la empresa, por lo que su opcional uso de redes ethernet ha permitido una fácil integración con el resto de sistemas. Además, el doble uso de CAN y Ethernet permitirá hacer comparativas entre ambos protocolos e incluso simularlos a la vez interactuando entre ellos.

5.6. Tests y pruebas de feedback

Durante el desarrollo de la librería se ha buscado mantener un perfil ligero en cuanto a utilización de recursos de CPU y de interrupciones, a fin de permitir una fácil implementación de una posterior capa de aplicación. Es importante pues cuantificar y valorar de forma objetiva el funcionamiento correcto del sistema en general. Para éste segundo informe de progreso se han propuesto las siguientes pruebas de rendimiento:

- Paquete simple
- Paquete extenso
- Batería de paquetes simples
- Batería de paquetes compuestos

Además, cada una de las pruebas se realiza en ambos modos de loopback mencionados anteriormente. A partir de éstas pruebas se espera poder medir el ancho de banda efectivo, la latencia de transmisión y recepción y el overhead del tratamiento de tramas.

5.6.1. Funcionamiento de los tests

En las pruebas realizadas se busca medir con precisión el tiempo de transmisión de cada paquete. Para ello vamos a utilizar un temporizador hardware capaz de medir el tiempo con una precisión de un microsegundo. Además, se va a evitar el uso de interrupciones para su funcionamiento, a fin de evitar el *overrun* de interrupciones con las generadas por el módulo Ethernet y así garantizar tiempos precisos.

Para medir el tiempo, tomaremos la referencia del temporizador justo al señalizar al módulo ethernet un inicio de transmisión, y lo pararemos en la rutina de tratamiento de interrupción cuando ocurra la llegada de un paquete(Fig.[7]).

De ésta manera, se mide no sólo el tiempo de transmisión, sino también el overhead de procesado de la trama por parte del hardware. Ésto nos permitirá calcular dónde se encuentra el cuello de botella de la transmisión, y qué tipo de transmisión resulta la más eficiente dado el sistema.

Las tramas transmitidas contendrán en todos los casos un header Ethernet II estándar de 14 bytes, y un *payload* con un tamaño adaptable al tamaño total del paquete deseado. Ambas partes serán procesadas por separado, a fin de simular una aplicación real y aplicar presión al sistema DMA para observar un overhead realista

5.6.2. Modos de funcionamiento

Como ya se ha comentado anteriormente, los modos de loopback interno y externo (Fig. [4] y [5]) serán los candidatos a ser probados durante éste segundo informe.

En el modo de loopback interno se espera poder apreciar el mejor caso posible para el microprocesador utilizado, sin comunicación con ningún dispositivo exterior. El overhead observado será el producido por los componentes internos del módulo Ethernet.

En el modo de loopback externo se podrá apreciar el overhead añadido de la transmisión por el PHY transciever.

5.6.3. Latencia

La latencia se mide como el tiempo que tarda el sistema en modo loopback desde que se señaliza un inicio de transmisión hasta que se recibe la trama por completo, coincidiendo con el tiempo medido con el temporizador hardware descrito anteriormente.

5.6.4. Ancho de banda

Podemos medir el ancho de banda de cada transmisión dividiendo el tamaño de la trama transmitida por la latencia de la transmisión. Hay que recordar que el estándar BroadR-Reach limita el ancho de banda máximo a 100Mbps, por lo que los valores medidos podrán acercarse, pero nunca llegar, a ese valor.

Adicionalmente se puede calcular el ancho de banda efectivo para una transmisión, utilizando la misma formula pero midiendo el tamaño de la trama sin contar el header, ya que no se puede considerar como información útil para la mayoría de aplicaciones. Veremos como varía el ancho de banda efectivo dependiendo del tamaño total y la cantidad de paquetes mandados.

5.6.5. Overhead

El overhead de una transmisión se puede obtener considerando el ancho de banda máximo. Éste nos proporciona una velocidad de transmisión ideal. Por ejemplo, una transmisión de 60 bytes a 100Mbps debería tardar $(60 * 8)b/100Mbps = 4{,}8us$. Todo tiempo adicional se puede medir como un overhead de la comunicación (Fig. [8]).

Éste overhead resulta más difícil de medir en transmisiones partidas en varias tramas, ya que existe un cierto paralelismo de procesamiento cuando varias tramas se encolan simultáneamente, como se puede apreciar en la Fig.[9]. En este caso, el overhead medido no coincide con el la suma de cada una de las tramas por separado, representando solamente los tiempos que se pueden medir.

5.6.6. Tests no realizados

Durante el planteamiento de los tests y pruebas a realizar, se ha propuesto una prueba de loopback remota, donde los datos se transmiten mediante BRR a un destinatario al otro lado del cable, en éste caso el VN6520 de Vector. Sin embargo, la naturaleza de éste test es altamente no determinista, puesto que muchos factores afectan o pueden afectar a los resultados. Principalmente el software y hardware contra el que hacemos los tests, además de factores cómo la longitud del cable y su conexionado. Debido a ésto los resultados obtenidos de éstas pruebas han sido poco fiables, y no representan ninguna situación real concreta de la que se puedan sacar resultados provechosos, siendo la prueba de loopback externa una medida más realista a la hora de comprobar el funcionamiento del sistema.

5.7. Resultados

En la tabla 1 se pueden ver las definiciones de cada test realizado, donde Test 1 y 2 prueban el Paquete Simple, 3 y 4 Paquete Extenso, 5 y 6 Batería de paquetes simples y por último 7 y 8 Batería de paquetes extensos.

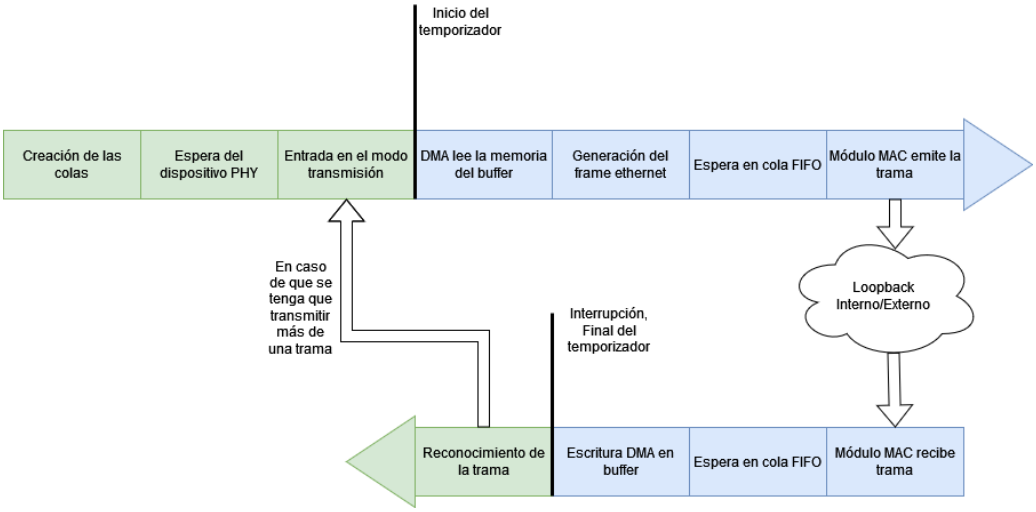

Fig. 7: Diagrama de tiempo de cada medida realizada

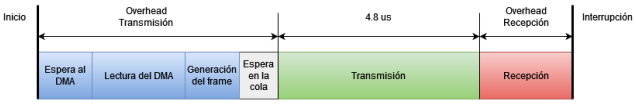

Fig. 8: Visualización del overhead en una transmisión

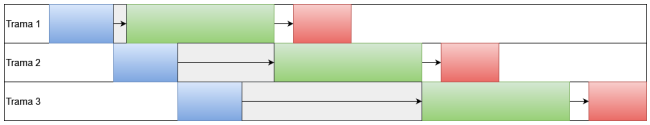

Fig. 9: Visualización del overhead en una transmisión múltiple

TABLA 1: COMPOSICIÓN DE CADA TEST Y TIEMPO MEDIDO

Test	Tipo	Tramas	Bytes	Total	Tiempo(us)
1	In.	1	60	60	15
2	Ex.	1	60	60	16
3	In	1	1514	1514	153
4	Ex	1	1514	1514	154
5	In	25	60	1500	176
6	Ex	25	60	1500	177
7	In	25	1514	37850	3109
8	Ex	25	1514	37850	3111

TABLA 2: DATOS DERIVADOS DE CADA TEST

Test	Tiempo(us)	Baudrate(Mb/s)	Overhead(us)
1	15	32	10.2
2	16	30	11.2
3	153	79.16	31.9
4	154	78.64	32.9
5	176	68.18	56
6	177	67.79	57
7	3109	97.39	81
8	3111	97.33	83

A continuación se muestran las métricas derivadas en la tabla 2

Además, como se ha explicado en la sección 5.6.4, se puede calcular un ancho de banda efectivo teniendo en cuenta que los bytes correspondientes a los headers no se consideran datos útiles. Para cada trama enviada podemos considerar que 14 de sus bytes (tamaño de header EthernetII) no son útiles. Considerando ésto podemos elaborar la tabla 3.

6 DISCUSIÓN

Durante el desarrollo del proyecto se ha podido apreciar el funcionamiento de un sistema Ethernet básico, aplicado pensando en un posible uso en un proyecto real en producción. Se ha desarrollado una librería pensada para tener el menor overhead posible en cuanto a requerimientos de tiempo de procesamiento y para poder aprovechar el máximo posible la velocidad que nos permite el protocolo Ethernet.

A partir de ahí se puede realizar una comparación con el bus CAN, actualmente en uso en la mayoría de sistemas informáticos en vehículos. El principal atractivo de éste bus en el campo de la automoción es su fuerte apuesta por el tiempo real. CAN requiere a todos los nodos conectados identificarse con un número, de forma similar a la dirección MAC de Ethernet, pero con la diferencia que el bus garantiza una asignación de prioridades según el número identificador, permitiendo que se resuelvan las colisiones haciendo que se resuelvan las colisiones haciendo permanecer la trama de mayor prioridad en el bus. De ésta forma, el bus CAN se asegura de tener un sistema en tiempo real eficiente, lo que no se puede garantizar con una conexión Ethernet. Sin embargo, durante el proyecto se ha explorado la comunicación punto a punto, donde las colisiones se minimizan o incluso se anulan en el caso de una transmisión unidireccional. Además, como se puede apreciar en la tabla 2, obtenemos unos tiempos de transmisión y procesado simple de datos con latencias que pueden ser menores de 10us si contamos la unidireccionalidad, totalmente viable para sistemas a tiempo real. Además, el sistema de arbitración de CAN no permite físicamente buses de velocidad superior a 1Mbit/s de velocidad[7], aunque en la mayoría de casos los

TABLA 3: ANCHO DE BANDA ÚTIL POR CADA TEST

Test	Tamaño útil	Baudrate Útil(Mb/s)
1	15	24.53
2	16	23
3	153	78.43
4	154	77.92
5	176	52.27
6	177	51.98
7	3109	96.49
8	3111	96.43

buses estándar utilizan el baudrate de 125 Kbit/s.

Además, las tramas de CAN tienen un tamaño fijo de 112 bits contando delimitadores de inicio de trama, con una información útil de 64 bits como máximo, necesitando protocolos de más alto nivel como ISO-TP para encadenar transmisiones de datos de mayor tamaño, siendo en general un protocolo ineficiente para la transmisión de grandes cantidades de datos. En el sistema Ethernet implementado, se ha llegado a un baudrate máximo de 96.49 Mbit/s útiles, lo que representa una eficiencia mucho mayor a la que se puede obtener mediant e CAN. Además, el soporte hardware del DMA permite que el overhead software de las comunicaciones sea mínimo.

En general, podemos ver la eficiencia del sistema Ethernet implementado cuando realizamos grandes transferencias de datos en una sola dirección, obteniendo con los mayores paquetes posibles velocidades cerca de las óptimas. Ésta propiedad permite casos de uso como transmisión de imágenes en tiempo real, descarga de firmware y reprogramación, y sobretodo la implementación de protocolos de más alto nivel que quedan fuera del alcance de éste trabajo (IP, ARP, TCP, etc). En general, las transmisiones de información cortas y cíclicas en broadcast són más rápidas que su equivalente en CAN (podemos verlo en los tests 1 y 2), sin embargo no cuentan con las medidas de arbitración de bus que provée CAN.

La importáncia de las comunicaciones globales entre controladores de vehículos residen en la fluidez de las comunicaciones y la garantía del tiempo real y no en el ancho de banda, por lo que el bus CAN sigue siendo la forma más adecuada de realizar las transmisiones, ya que en Ethernet las posibles colisiones, aunque raras, entorpecen las comunicaciones. El análisis de como podría funcionar un sistema Ethernet como el montado en un vehículo real para comunicación entre unidades de control queda fuera del alcance del proyecto.

Hace falta entonces considerar que un protocolo como Ethernet en un vehículo no sustituye al uso del bus CAN, sino que lo complementa extendiendo la funcionalidad de las distintas partes del vehículo mediante velocidades de transmisión más rápidas y protocolos de comunicación de alto nivel.

6.1. Conclusiones

Durante el desarrollo del trabajo se ha llevado a cabo de forma exitosa la implementación de un sistema Ethernet mediante el conexionado BroadR-Reach, tanto a nivel de software como de hardware, se ha detallado la implementación de éste y se ha explorado su funcionamiento en verso a el principal protocolo de comunicación utilizado en el mundo automovilístico (CAN). Además, se han desarrollado tests para verificar no solamente su funcionamiento, pero también sus puntos más fuertes. Además, se ha podido desarrollar la librería firmware de forma modular y cumpliendo todos los objetivos propuestos para éste trabajo.

Queda como posible trabajo posterior la implementación de protocolos correspondientes a otras capas OSI por encima de la capa Ethernet, como puede ser TCP e IP.

REFERENCIAS

[1] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[2] P. Saxena, "Osi reference model–a seven layered architecture of osi model," *International Journal of Research (IJR)*, vol. 1, no. 10, pp. 1145–1156, 2014.

[3] "Ieee standard for ethernet," *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pp. 1–5600, 2018.

[4] Renesas, "Rh850 main page," https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rh850-automotive-mcus, 2021, [Online; accessed 09-June-2023].

[5] Renesas, "Rh850 reference," https://www.renesas.com/us/en/document/mah/rh850d1ld1m-group-users-manual-hardware?r=1054191, 2023, last accessed 21 March 2023.

[6] NXP, "Tja1101b datasheet," https://www.nxp.com/docs/en/data-sheet/TJA1101B.pdf, 2023, last accessed 21 March 2023.

[7] S. Corrigan, "Controller area network physical layer requirements," *Application Report slla270*, 2008.

ANEXOS

A.1. Acrónimos

- BRR - *BroadR-Reach*
- CAN - *Control Area Network*
- PHY - *Physical Layer Transciever*
- MAC - *Media Access Control*
- MII - *Media Independent Interface*
- SMII - *Serial MII*
- DMA - *Direct Memory Access*

A.2. Desarrollo temporal del trabajo

En ésta sección se aprecia la organización temporal que se ha seguido en el trabajo, desde el punto de vista de inicio del proyecto así como todas las revalorizaciones hechas en cada informe de progreso entregado. Fig.[10]

Se ha estimado inicialmente un trabajo de dos a tres semanas para cada fase del proyecto (unas 40 - 60 horas por fase), siendo las fases las explicadas en la sección 4.1.

A.3. Viabilidad

El principal motivo por el cual se ha propuesto éste trabajo es que la posibilidad de implementación de la funcionalidad de Ethernet se encuentra disponible dentro del microcontrolador que como empresa ya se está usando en varios proyectos activos, es decir, no requerimos de presupuesto adicional más allá del montaje del hardware y del módulo PHY externo.

A.4. Documentación del Software

Durante el desarrollo del proyecto se ha generado una API para interactuar con el software creado. En las siguientes figuras Fig. 13 y 14 Se pueden apreciar en formato abreviado.

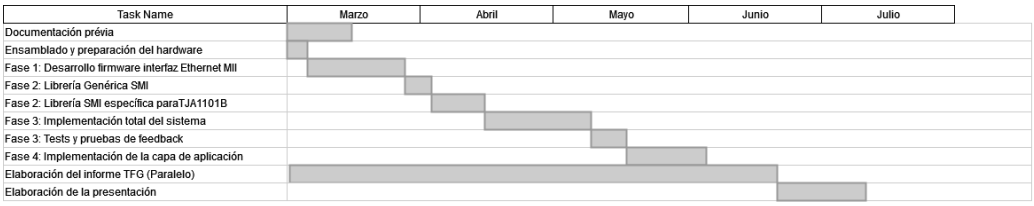

Fig. 10: Diagrama de planificación inicial

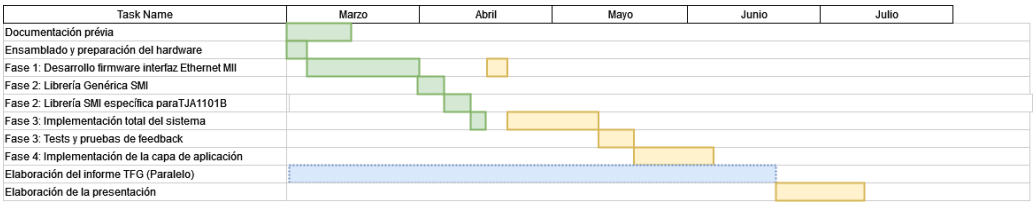

Fig. 11: Diagrama completado a fecha de informe de progreso I. En verde, trabajo completado, en azul, trabajo en progreso, en amarillo, trabajo restante y replanificado.

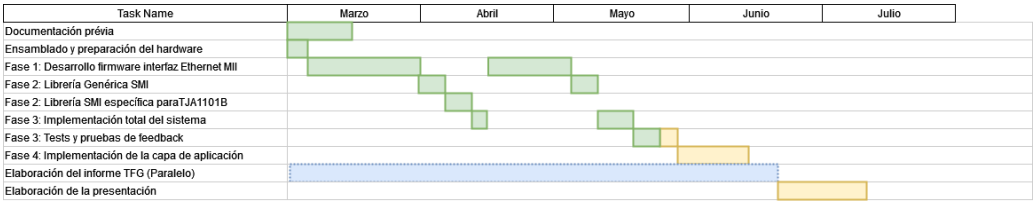

Fig. 12: Diagrama completado a fecha de informe de progreso II. En verde, trabajo completado, en azul, trabajo en progreso, en amarillo, trabajo restante y replanificado.

Fig. 13: Enumeraciones de C creadas para la API.

Functions

void	**vfn_ETHM_init** (void) Initializes the hardware module, software values and begins PHY startup.
e_ETHM_mode_t	**e_ETHM_getCurrentMode** (void) Returns the library state.
void	**vfn_ETHM_process** (void) Internal process function, should be called every 1 to 5 ms.
e_ETHM_phyState_t	**e_ETHM_getPhyState** (void) Gets the state of the PHY device. if **ETHM_UNKNOWN**, the initialization might have failed or the PHY device has no support for polling.
e_ETHM_Error_t	**e_ETHM_setupRecieveQueue** (**e_ETHM_recieveQueueSlot_t** _e_slot, uint8_t _u08_maxDescriptorAmount) Creates a new reception queue and assigns it to a slot.
e_ETHM_Error_t	**e_ETHM_setupTransmitQueue** (**e_ETHM_transmitQueueSlot_t** _e_slot, uint8_t _u08_maxDescriptorAmount) Creates a new transmission queue and assigns it to a slot.
e_ETHM_Error_t	**e_ETHM_addRecieveDescriptor** (**e_ETHM_recieveQueueSlot_t** _e_slot, **e_ETHM_rcvDescriptorType_t** _e_type, void *_ptr, uint16_t _num, **e_ETHM_descriptorInterrupt_t** _interrupt) Adds a reception descriptor to a newly created queue.
e_ETHM_Error_t	**e_ETHM_addTransmitDescriptor** (**e_ETHM_transmitQueueSlot_t** _e_slot, **e_ETHM_tsmDescriptorType_t** _e_type, void *_ptr, uint16_t _num, **e_ETHM_descriptorInterrupt_t** _interrupt) Adds a transmission descriptor to a newly created queue.
e_ETHM_Error_t	**e_ETHM_restartReceptionQueue** (**e_ETHM_recieveQueueSlot_t** _e_slot) Re initializes the queue and restores all the descriptors as they were first created.
e_ETHM_Error_t	**e_ETHM_restartTransmissionQueue** (**e_ETHM_transmitQueueSlot_t** _e_slot) Re initializes the queue and restores all the descriptors as they were first created.
uint8_t	**u08_ETHM_getReceptionFrameNum** (**e_ETHM_recieveQueueSlot_t** _e_slot) gets the total amount of complete frames stored in the queue.
r_bool_t	**bln_ETHM_isTransmissionFinished** (**e_ETHM_transmitQueueSlot_t** _e_slot)
r_bool_t	**bln_ETHM_isReceptionFinished** (**e_ETHM_transmitQueueSlot_t** _e_slot)
e_ETHM_Error_t	**e_ETHM_beginTransmission** (**e_ETHM_transmitQueueSlot_t** _e_slot) begin Transmission of a certain queue. The queue must have been initialized fist.
e_ETHM_Error_t	**e_ETHM_getReceptionFrame** (**e_ETHM_recieveQueueSlot_t** _e_slot) get a struct containing the first frame available and moves the queue one position.
uint64_t	**u64_ETHM_getMacAddress** (void)
e_ETHM_Error_t	**e_ETHM_setReceptionInterrupt** (**e_ETHM_recieveQueueSlot_t** _e_slot, void(*_fn_frameRecieved)(**s_ETHM_frameDefinition_t**)) Setup an interrupt callback for every frame recieved. Leave at **NULL** in order to deactivate it.
e_ETHM_Error_t	**e_ETHM_bindCallback** (**e_ETHM_descriptorInterrupt_t** _e_interruptSlot, void(*_func)(**s_ETHM_frameDefinition_t**)) Links a callback to an interruption id used in **e_ETHM_addRecieveDescriptor** and **e_ETHM_addTransmitDescriptor**. When a descriptor with this interrupt id is processed this callback will be called.

Fig. 14: Funciones de C creadas para la API