Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

Desarrollo del control de un 4WD AUTOSAR basado en Toolchain abierto

Autor: Antonio Pérez Laguarda
Tutores: Eduardo Hidalgo Fort y Fernando Muñoz Chavero

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

Sevilla, 2021

Trabajo Final de Máster
Ingeniería de Telecomunicación

Desarrollo del control de un 4WD AUTOSAR basado en Toolchain abierto

Autor:
Antonio Pérez Laguarda

Tutores:
Eduardo Hidalgo Fort
Investigador Postdoctoral
Fernndo Muñoz Chavero
Catedrático de Universidad

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Final de Máster: Desarrollo del control de un 4WD AUTOSAR basado en Toolchain abierto

Autor: Antonio Pérez Laguarda

Tutores: Eduardo Hidalgo Fort y Fernando Muñoz Chavero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A Ana

A mis maestros

Agradecimientos

Antes de cerrar esta etapa de mi vida, me gustaría agradecer en primer lugar a mis padres y mi hermano, por estar conmigo durante estos añosapoyarme en los momentos difíciles. A Ana por acompañarme y animarme en todo momento.

A los amigos que he tenido la suerte de conocer durante mi etapa en la Escuela, que han hecho que todos estos años hayan sido mucho más llevaderos.

También agradecer al proyecto 4WD llevado acabo entre el Grupo de Ingeniería Electrónica (GIE) y Évolution Synergétique Automotive SL (EVO) por el cual he podido entrar al mundo laboral.

Finalmente, dar las gracias a mi compañero Marcelo, a Fernando Muñoz por brindarme esta oportunidad, a mi tutor Eduardo Hidalgo por su dedicación y ayuda durante todo el proyecto y a Alberto Aragón por su predisposición y seguimiento a lo largo del mismo, ya que sin ellos no hubiera podido llevarse a cabo.

Resumen

Este Trabajo Final de Máster tiene como objetivo el desarrollo de un vehículo (plataforma hardware) cuya arquitectura esté basada en AUTOSAR. La memoria de este trabajo comienza desarrollando el contexto en el cual surge este proyecto y los objetivos planteados en el mismo. El siguiente capítulo hace un análisis del estándar AUTOSAR donde se explica el motivo por el cual es importante su utilización y cada una de las capas de su arquitectura. En el tercer capítulo se definen los requisitos necesarios del sistema organizándolos en diferentes grupos. Los dos siguientes capítulos serán el Diseño y Desarrollo del Sistema, dividiéndose ambos en dos apartados, uno relativo a la parte hardware y otro a la parte software. A continuación, se expondrán los resultados obtenidos durante el desarrollo del proyecto y, finalmente, se recogen las conclusiones obtenidas y las posibles líneas de trabajo futuras.

Abstract

The objective of this Master's thesis is the development of a vehicle (hardware platform) whose architecture is based on AUTOSAR. The report of this work begins by developing the context in which this project arises, and the objectives set out in it. The second chapter analyses the AUTOSAR standard, explaining why it is important to use it and each of the layers of its architecture. The next chapter defines the necessary system requirements and organises them into different groups. The next two chapters are the Design and Development of the System, both divided into two sections, one relating to the hardware part and the other to the software part. Next, the results obtained during the development of the project will be presented and, finally, the conclusions obtained, and the possible lines of future work will be included.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Este proyecto se centrará en el desarrollo de un sistema basado en AUTOSAR sobre una plataforma hardware que simulará a un vehículo real.

A continuación, se introducirán los antecedentes del proyecto, donde se explicará la situación actual en relación con los vehículos y su desarrollo para, seguidamente, explicar la motivación del proyecto y cómo surge el mismo. En tercer lugar, se expondrán los objetivos con los que cuenta el trabajo y, finalmente, cómo ha sido el desarrollo del proyecto para alcanzar esos objetivos marcados.

1.1. Antecedentes

Un vehículo está compuesto por distintas ECUs (Unidad de Control Electrónico), que básicamente consisten en un sistema embebido en la electrónica del vehículo que controla uno o más sistemas o subsistemas eléctricos en un vehículo.

Como puede observarse en la Figura 1-1. Crecimiento de la complejidad de las ECUs, existe un claro incremento en el número de ECUs existentes en los vehículos y su complejidad, por lo que surge la necesidad de definir un proceso estandarizado de desarrollo de software en el sector. Es principalmente con este propósito con el que se funda la asociación AUTOSAR en el año 2003, crear una arquitectura de software estandarizada para ser usada en el desarrollo de ECUs.

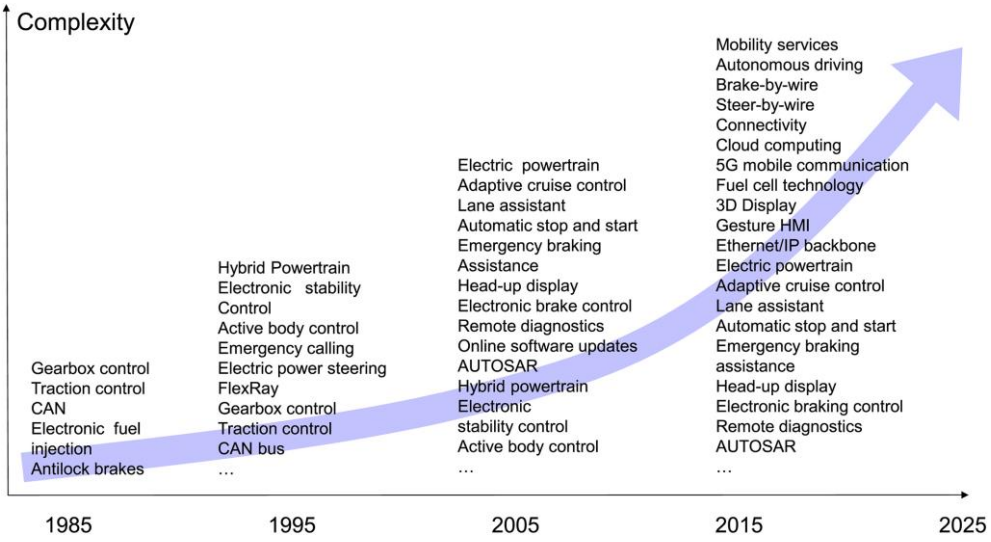

Figura 1-1. Crecimiento de la complejidad de las ECUs

Hoy en día, AUTOSAR está completamente asentado y es mundialmente utilizado. El estándar especifica como debe organizarse el software y como se establece la comunicación entre distintos elementos dentro de él.

1.2. Motivación

La principal motivación de AUTOSAR es reducir los costes de desarrollo y mantenimiento mediante la reutilización APIs estandarizadas en distintos proyectos. Hasta la creación de AUTOSAR, el software embebido de automoción se escribía de acuerdo con los requisitos que dictaba el hardware específico en el que se ejecutaba ese software. Por lo que cualquier cambio en el hardware provocaba un cambio del software. Este

aspecto se traduce en una pérdida de dinero y de tiempo. Para mitigar estos problemas, AUTOSAR divide en diferentes capas la arquitectura de software, para hacer posible que el software desarrollado para las ECUs pueda ser independiente del hardware en el que será implementado.

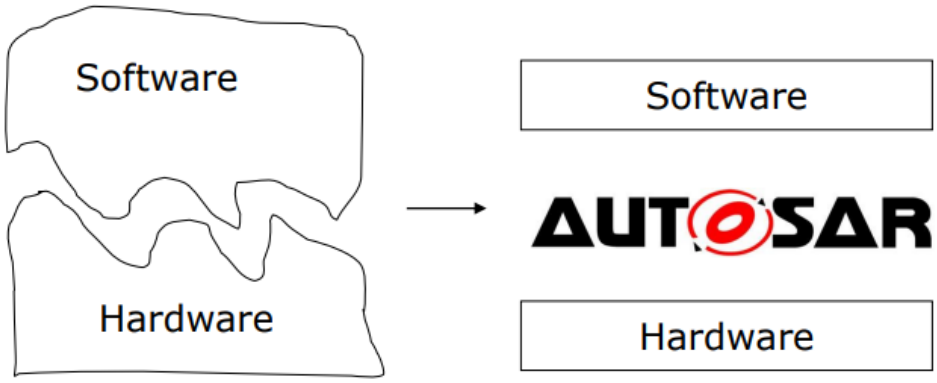

Figura 1-2. Integración de SW y HW con AUTOSAR

Dentro de este marco, surge la posibilidad de colaboración entre el Grupo de Ingeniería Electrónica (GIE) de la Escuela Técnica Superior de Ingenieros de Sevilla con la empresa Évolution Synergétique Engineering Automotive S.L. (EVO).

En el marco de esa colaboración, se propone la realización de un proyecto en el que se desarrollará una plataforma hardware (vehículo) y software basada en una arquitectura AUTOSAR.

1.3. Objetivos

En este apartado se enumerarán los objetivos marcados para el proyecto.

- **Estudio y análisis de AUTOSAR**. Al tratarse de un estándar muy extenso y complejo, se realizará un estudio y análisis previo donde se pretenderá dar una visión general del mismo.
- **Análisis del mercado**. Habrá que realizar un estudio del mercado y de las distintas herramientas software y hardware disponibles.
- **Desarrollo de plataforma hardware**. Se implementará una plataforma hardware a escala que simulará un vehículo.
- **Desarrollo software**. Se desarrollará todo lo necesario a nivel software para cumplir con el propósito del proyecto.
- **Calidad**. Se realizará un desarrollo basado en simulaciones y sobre una plataforma de testeo.

1.4. Metodología

En primer lugar, hay que señalar que el proyecto llevado a cabo en colaboración con Évo ha sido desarrollado por dos personas: Marcelo Simbaña Romero y el autor de este tfm.

Para alcanzar los objetivos comentados anteriormente, el proyecto fue dividido en dos partes que se asignarían a cada uno de nosotros. La primera de ellas trata de la parte superior de la arquitectura de AUTOSAR, concretamente de la Capa de Aplicación y RTE, de la que se encargaría el compañero Marcelo Simbaña. Y la segunda de ellas, el nivel más bajo dentro de la arquitectura de AUTOSAR, el Basic Software (BSW), de la que me encargaría yo, además del diseño y desarrollo HW de la plataforma. Estos módulos dentro de AUTOSAR serán explicados en detalle en el siguiente capítulo.

A pesar de que el trabajo desarrollado en la Capa de Aplicación y RTE fue desarrollado por Marcelo Simbaña, algunas partes serán explicadas, con un grado de profundidad menos a la parte de BSW, para poder tener una visión completa del sistema y una correcta comprensión del proyecto.

Durante el transcurso del proyecto ha habido una comunicación directa con EVO, para llevar un control sobre los hitos alcanzados y los problemas que han ido surgiendo.

Por otro lado, al realizarse el proyecto en colaboración con una empresa, se han utilizado herramientas de gestión y producción reales en el marco industrial como BitBucket, Jira, Confluence…

Inicialmente, al proyecto se le asignó una duración de un año, aunque finalmente concluyó a los nueve meses al producirse la incorporación de ambos desarrolladores a la empresa.

2 INTRODUCCIÓN A AUTOSAR

AUTOSAR AUTOmotive Open System ARchitecture) es una asociación de desarrollo a nivel mundial formada por entidades interesadas en la industria del automóvil y fundada en 2003. Persigue crear y establecer una arquitectura de software abierta y estandarizada para las unidades de control electrónico (Electronic Control Units - ECU) para la automoción.

2.1. Historia

AUTOSAR es una alianza de OEMs (Original Equipment Manufacturer – Fabricante de Equipo Original), TIER 1 (proveedores de automóviles de primer nivel), fabricantes de semiconductores, proveedores de software, proveedores de herramientas, etc. Teniendo en cuenta las diferentes arquitecturas E/E (Electricidad/Electrónica) del sector de automoción en los mercados actuales y futuros, la asociación estableció una norma industrial abierta de facto para una arquitectura de software de automoción. Servirá como infraestructura básica para la gestión de funciones tanto de las futuras aplicaciones como de los módulos de software estándar.

La asociación fue fundada en julio de 2003 por BMW, Bosch, Continental, DaimlerChrysler, Siemens VDO y Volkswagen para desarrollar y establecer un estándar industrial abierto para la arquitectura E/E del automóvil. En noviembre de 2003, Ford Motor Company se unió como socio principal, y en diciembre se unieron Peugeot Citroën Automobiles SA y Toyota Motor Corporation. En noviembre de 2004, General Motors también se convirtió en socio principal.

Los miembros de AUTOSAR se reparten en las siguientes categorías: socios principales, socios premium, socios de desarrollo y socios asociados. En la actualidad, AUTOSAR cuenta con 9 socios principales, 56 socios premium, 56 socios de desarrollo y 142 socios asociados.

2.2. Objetivos

Sus objetivos incluyen:

- Escalabilidad a diferentes variantes de vehículos y plataformas.
- Transferibilidad del software.
- Tener en consideración los requisitos de disponibilidad y seguridad.
- Ser un marco de colaboración entre varios socios.
- Uso sostenible de recursos naturales.
- Mantenimiento a lo largo de todo el "Ciclo de vida del vehículo".
- Abstracción entre software y hardware para obtener un desarrollo más flexible.
- Mejora de la calidad del software.
- Reducir costes a través de la reutilización de funciones, métodos de desarrollo, herramientas y software.

2.3. Arquitectura

La arquitectura de AUTOSAR, como se puede apreciar en la siguiente figura, distingue tres capas software: Application, Runtime Environment (RTE) y Basic Software (BSW).

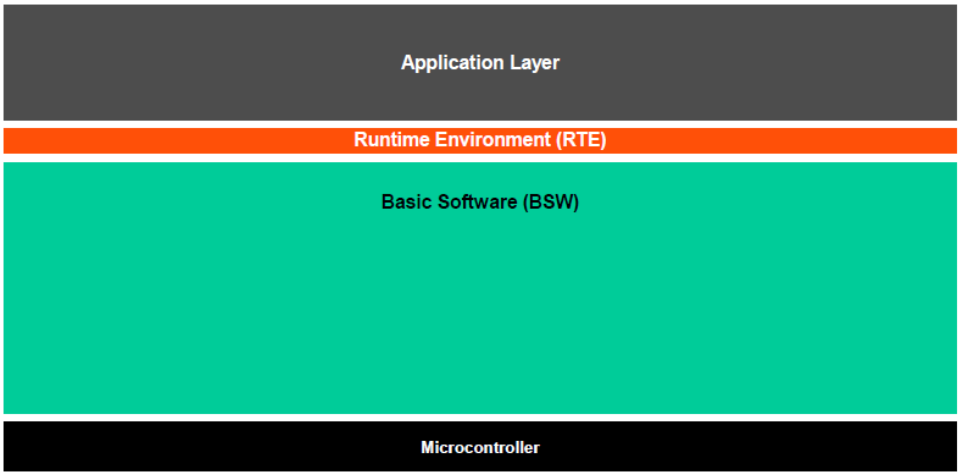

Figura 2-1. Arquitectura AUTOSAR

2.3.1. Basic Software

La capa de BSW puede definirse como un módulo de software estandarizado que ofrece varios servicios necesarios para el correcto funcionamiento de la parte funcional de las capas superiores.

Esta capa se divide a su vez en cuatro capas o módulos (Figura 2-2. Arquitectura AUTOSAR. Basic Software (I)): Services Layer, ECU Abstraction Layer (ECUAL), Microcontroller Abstraction Layer (MCAL) y Complex Drivers. Estos, a su vez, se dividen en distintos grupos funcionales (Figura 2-3. Arquitectura AUTOSAR. Basic Software (II)).

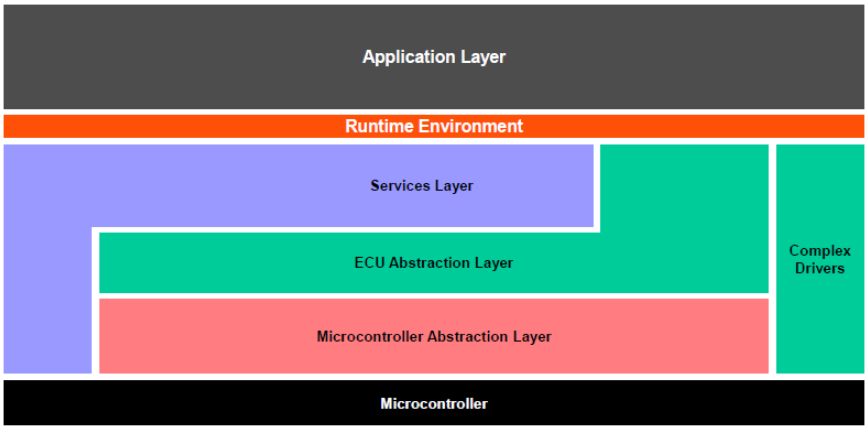

Figura 2-2. Arquitectura AUTOSAR. Basic Software (I)

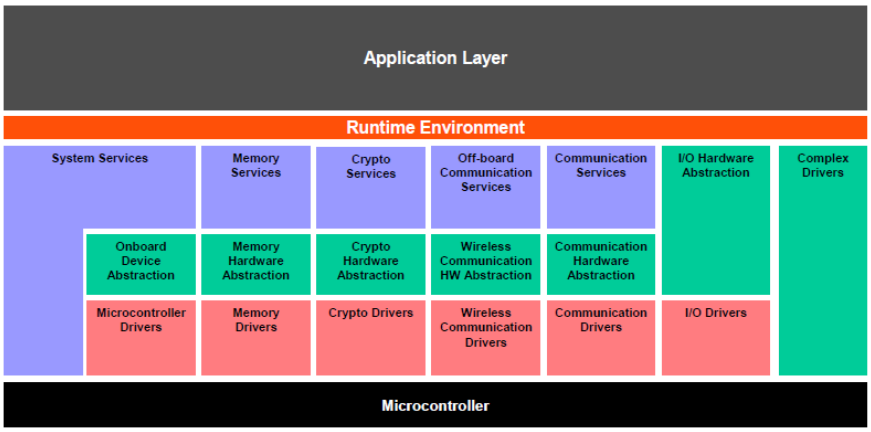

Figura 2-3. Arquitectura AUTOSAR. Basic Software (II)

2.3.1.1. Microcontroller Abstraction Layer

La MCAL es la capa de nivel más bajo dentro del BSW. Ésta contiene drivers internos, que son módulos software con acceso directo al microcontrolador y a los periféricos internos.

Su tarea principal es hacer a las capas superiores independientes del microcontrolador, lo que supone un aspecto muy importante en la arquitectura AUTOSAR, ya que el desarrollo del resto del BSW, el RTE y la capa de aplicación no están sujetos al fabricante del microcontrolador. Esta abstracción se traduce en una reducción de costes y tiempo de desarrollo.

Su implementación es dependiente del microcontrolador. Es decir, el fabricante proporciona esta capa para trabajar sobre ella. Normalmente, esta capa es configurada a través de herramientas de autogeneración de código.

Su interfaz superior es estándar e independiente al microcontrolador.

2.3.1.2. ECU Abstraction Layer

La ECUAL contiene drivers para dispositivos externos. Ofrece una API para acceder a periféricos independientemente de su localización (interna o externa al microcontrolador) y su conexión al microcontrolador (puertos, tipo de interfaz…)

La tarea de la ECUAL es hacer a las capas superiores independientes del layout hardware de la ECU.

Su implementación es independiente del microcontrolador y dependiente del layout hardware de la ECU.

Su interfaz superior es independiente tanto del microcontrolador como del hardware de la ECU.

2.3.1.3. Complex Drivers

La capa de Complex Drivers conecta directamente el hardware con el RTE. El motivo por el cual esta capa es necesaria es la integración de funcionalidades de propósito especial que se encuentran fuera del alcance del estándar de AUTOSAR o con unas restricciones de tiempo muy elevadas.

Su implementación puede ser dependiente de la aplicación, del hardware de la ECU y del microcontrolador, al igual que su interfaz superior.

2.3.1.4. Capa de Servicios

Es la capa más alta dentro del BSW y también es importante para la capa de Aplicación.

Mientras que el acceso a las señales digitales de entrada y salida están cubiertas por la ECUAL, la capa de servicios ofrece:

- Funcionalidad de sistema operativo
- Comunicación de la red del vehículo y servicios de administración

- Servicios de memoria (NVRAM Management)
- Servicios de diagnóstico (UDS, Error Memory y Fault Detection)
- ECU State Management, mode management
- Control del flujo de programa lógico y temporal (Wdog manager)

Su tarea es ofrecer servicios básicos para aplicaciones, RTE y módulos del BSW.

Su implementación es mayoritariamente independiente del microcontrolador y del hardware de la ECU. La interfaz superior es independiente del microcontrolador y del hardware de la ECU.

2.3.2. Runtime Environment

El RTE tiene como tarea principal hacer a los SWCs (Software Components) de la Capa de Aplicación independientes de la ECU. El término de SWC se definirá en detalle en el siguiente capítulo.

Esta capa permite la comunicación entre los componentes de la capa superior, además de permitir la comunicación entre estos componentes con servicios del BSW.

Su implementación es completamente dependiente de la ECU y la aplicación, y por ello es generada individualmente para cada ECU.

2.3.3. Capa de Aplicación

La Capa de Aplicación se encuentra en el nivel más alto de la arquitectura software definida por AUTOSAR. El estándar define que la implementación de esta capa se realizará a través de componentes a diferencia de los niveles inferiores que se implementaban por capas.

Dentro de esta capa es necesario definir algunos conceptos clave: software components, puertos, interfaces y runnables.

Un SW-C constituye la forma más simple de una aplicación con cierta funcionalidad. Estos están conectados con la ayuda de puertos que facilitan la comunicación entre los SW-Cs, así como con el BSW. Dentro de los SW-Cs, hay ciertas entidades llamadas Runnables, que son básicamente los procedimientos que contienen su implementación real. AUTOSAR también define dos tipos de interfaces: sender/receiver y client/server.

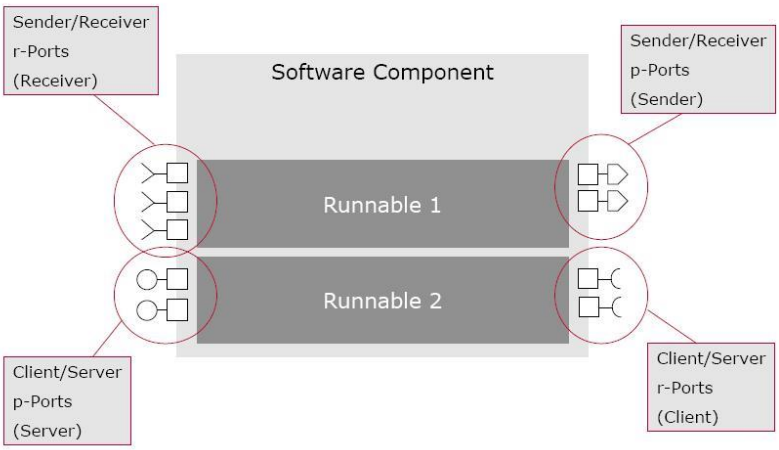

Figura 2-4. Software Component

Una vez obtenida una visión general de AUTOSAR, en el siguiente capítulo se detallarán cuáles son los requisitos de este TFM, separándolos en tres grupos: requisitos de toolchain, requisitos de arquitectura y requisitos funcionales/no funcionales.

3 REQUISITOS DEL SISTEMA

Como se ha comentado anteriormente, el propósito de este capítulo es definir los requisitos que precisa el sistema para alcanzar los objetivos especificados en el punto 1.3. Estos requisitos serán clasificados dentro de tres grupos.

En primer lugar, se definirán los requisitos de toolchain. Se explicarán los elementos necesarios a nivel hardware y software y se justificará la elección final. En segundo lugar, se expondrá qué arquitectura es necesaria para el sistema. Por último, se enumerarán tanto los requisitos funcionales como los no funcionales.

Cabe señalar que los requisitos relativos tanto a la capa de aplicación como al RTE no serán explicados en detalle, simplemente serán mencionados, ya que están fuera del alcance del proyecto, como se expuso en el capítulo de Introducción.

3.1 Requisitos de toolchain

Dentro de los requisitos de toolchain, como se ha comentado antes, tanto los elementos hardware como los elementos software requeridos serán explicados.

3.1.1 Requisitos hardware

Para cumplir los objetivos fijados para el proyecto a nivel hardware, serán necesarios distintos elementos:

- Plataforma y motores: en primer lugar, será necesaria una plataforma hardware con cuatro motores DC para emular al vehículo.
- Placa de evaluación: como núcleo del sistema se dispondrá de una placa de evaluación donde se desarrollará el proyecto.
- Conexión inalámbrica: para la conexión inalámbrica, se necesitarán tanto un módulo que integrado en el sistema como un dispositivo externo para poder comunicarse con él.

En la búsqueda de plataformas hardware para el sistema se barajaron distintas opciones. Las plataformas para proyectos desarrollados en Arduino o Raspberry Pi eran las más abundantes, algunas de ellas incluían distintos dispositivos para utilizar en el sistema, como puede observarse en alguna de las siguientes imágenes.

Figura 3-1. Plataforma hardware (I)

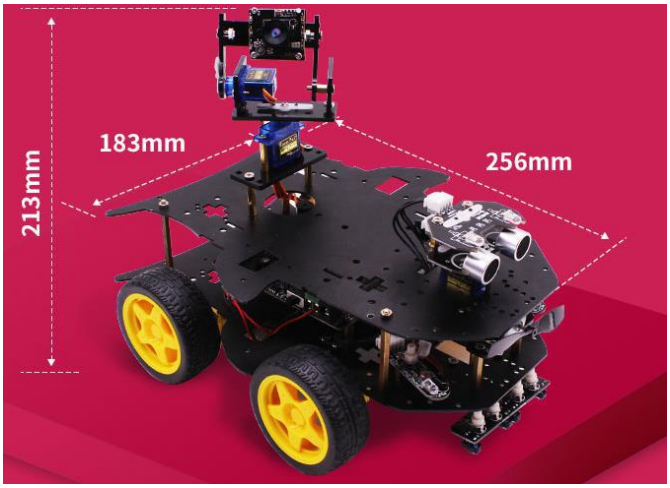

Figura 3-2. Plataforma hardware (II)

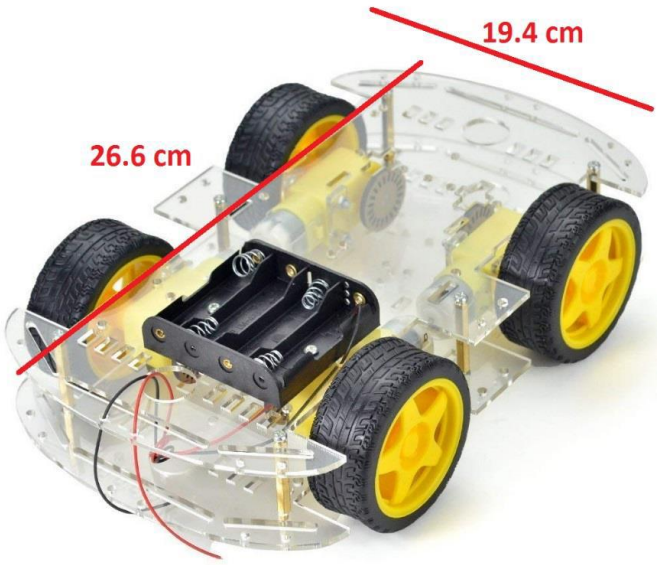

Figura 3-3. Plataforma hardware (III)

A pesar de que su precio es bastante económico (entre 10€ y 30€), se descartaron este tipo de plataformas, principalmente debido a su tamaño y robustez, ya que no sería posible ubicar todos los dispositivos necesarios en estas plataformas, tanto por espacio como por peso, ya que los motores no tendrían potencia suficiente para funcionar correctamente.

Es por esto por lo que se busca una plataforma de tamaño superior, más robusta, y con motores con mayor potencia, seleccionándose finalmente la siguiente.

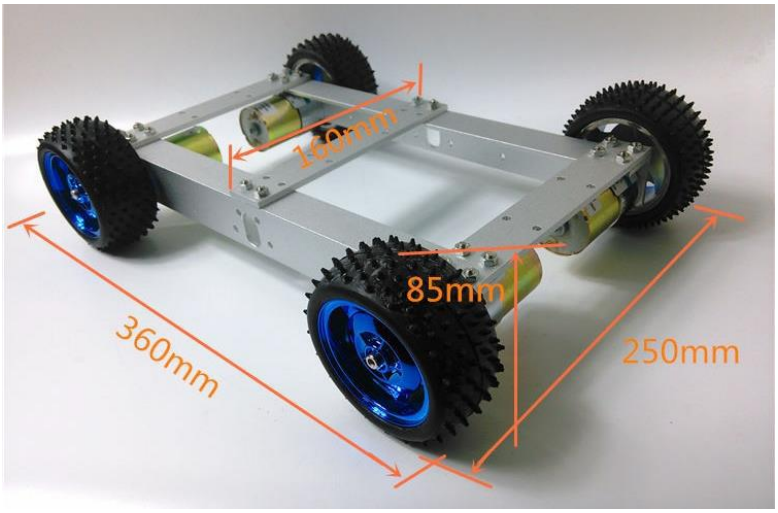

Figura 3-4. Plataforma hardware seleccionada

Aunque su precio es más elevado, unos 100€, su tamaño y los motores DC de 12V que incluye hace que sea la plataforma elegida.

Otro motivo por el cual la plataforma anterior fue la elegida es que los motores DC incluyen un encoder por cada motor que, durante el desarrollo del proyecto, permitirá el análisis del comportamiento de los motores.

Para poder establecer una comunicación inalámbrica entre el vehículo y un controlador externo será necesaria la utilización de un módulo o interfaz inalámbrica. Para el proyecto se usará bluetooth como protocolo de comunicación.

El módulo que se utilizará será el HC-05 que, mediante el puerto serie, permitirá la conexión entre la placa de evaluación con un controlador externo.

Para enviar la información a nuestro sistema, se utilizará un teléfono móvil Android. En el cual se instalará una aplicación de diseño propio que se explicará en detalle capítulos siguientes.

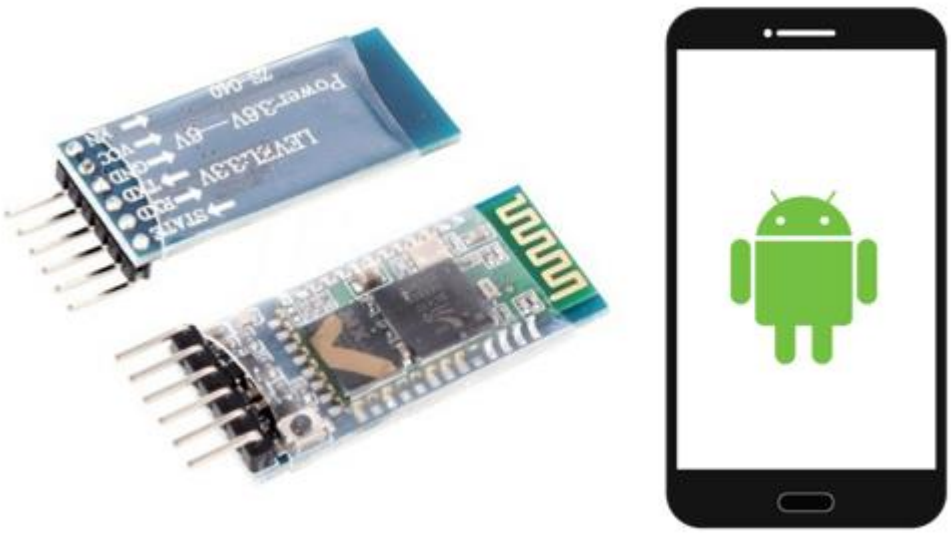

Figura 3-5. Elementos para conexión inalámbrica

Por último, a nivel hardware, se necesita una placa de evaluación que, como se comentó anteriormente, será el núcleo del sistema. Una vez iniciado el análisis de mercado de las placas existentes para poder seleccionar una, se valoraron distintos aspectos. Algunos de ellos se listan a continuación:

- Automotive: se premiará que la placa de evaluación escogida esté diseñada y pensada para su uso en proyectos automotive.
- Soporte: interesa que la placa cuente con un buen soporte por parte del fabricante.
- Interfaces: el número y tipo de interfaces que posee la placa también es un aspecto de importancia para, en un futuro, poder conectarle distintos elementos externos (programadores, CAN, etc.)
- Precio: ya que el sector de automoción cuenta con un gran rango de precios, se buscará un equilibrio entre precio y prestaciones.

En base a los puntos anteriores, se seleccionaron dos fabricantes para elegir una placa de evaluación que hará de núcleo del sistema: Renesas e Infineon. Ambos ofrecen una amplia gama de placas dirigidas al sector de automoción.

A continuación, se hará una descripción de las placas de evaluación disponibles junto con los programadores disponibles en cada uno de los fabricantes para, posteriormente, definir cuál opción es la más apropiada.

Tanto Renesas como Infineon disponen de distintas familias de microcontroladores enfocadas al sector de automoción. En las siguientes tablas se listan tanto los microcontroladores disponibles dentro de ambos fabricantes, así como los programadores necesarios para su utilización:

Tabla 3-1. Microcontroladores ofrecidos por Renesas

Microcontroladores ofrecidos por Renesas			
Familia	**Referencia**	**Precio**	**Herramientas de desarrollo (IDE)**
RL78/F13	QB-R5F10BMG-TB	19,78 $ (Digi-Key) 28,91 € (Rutronik24) 19.75 € (Mouser)	El fabricante ofrece herramientas de desarrollo para los microcontroladores, pero no ofrecen soporte en Europa.
RL78/F14	QB-R5F10PPJ-TB	37.39 € (Avnet) 28,88 € (Rutronik24)	
	YRDKRL78F14	191,87 € (Mouser)	
	Y-BLDC-SK-RL78F14	782.29 $ (Avnet)	
RL78/F15	Y-QB-R5F113TL-TB	39,38 € (rutronik24)	
RH850/D1x RH850/E2x RH850/F1x RH850/P1x	Y-RH850-X1X-MB-T1-V1 (Versión estándar)	656,25 € (Rutronik24)	
	Y-RH850-X1X-MB-T2-VX (Versión “network”)	€1.060 (Avnet)	
	Es necesario una placa extra llamada PiggyBack para poder realizar la depuración.	-	

Tabla 3-2. Programadores ofrecidos por Renesas

Progrmadores ofrecidos por Renesas			
Programador	**Referencia**	**Características**	**Precio**
E2 Emulator	RTE0T00020KCE00000R	Alta velocidad de descarga (hasta el doble que la versión E1) Procesamiento de comunicación CAN Soporte para disparadores internos y externos.	475.99€ (Avnet)
E2 Emulator Lite	RTE0T0002LKCE00000R	Características similares al E1 a un precio más asequible.	56.58 € (Digi-Key)
E1 Emulator	R0E000010KCE00	Funcionalidad básica de depuración.	159.38 $ (Digi-Key)
E20 Emulator	R0E000200KCT00	Equivalente a E1 con más soporte.	891,30 € (Digi-Key)

Tabla 3-3. Microcontroladores ofrecidos por Infineon

Microcontroladores ofrecidos por Infineon			
Familia	**Referencia**	**Precio**	**Herramientas de desarrollo (IDE)**
Starter Kit TC233	KIT_AURIX_TC233LP_TRB	350€ (Infineon)	Existen entornos de desarrollo compatible con las placas ofrecidas y sin requerir una licencia para su uso.
AURIX Motor Control Application Kit with TFT Display	KIT_AURIX_TC234_MOTORCTR	449€ (Infineon)	
AURIX Application Kit TC234TFT	KIT_AURIX_TC234_TFT	149€ (Infineon)	
AURIX Application Kit TC237 TFT	KIT_AURIX_TC237_TFT	149€ (Infineon)	
Starter Kit TC234	KIT_AURIX_TC234LP_TRB	350€ (Infineon)	

Tabla 3-4. Programadores ofrecidos por Infineon

Programadores ofrecidos por Infineon			
Programador	**Referencia**	**Características**	**Precio**
MiniWiggler V3	KIT_MINIWIGGLER_3_USB		99€ (Infineon)

Tras realizar un análisis de ambos fabricantes y el hardware que ofrecen, se puede apreciar como Renesas ofrece un soporte escaso en Europa, por lo que se descarta su utilización en el proyecto.

Infineon sin embargo ofrecía más soporte y un precio dentro de lo estimado, por lo que se seleccionó este fabricante. Dentro de las posibilidades existentes, el Starter Kit TC233 fue el elegido.

Figura 3-6. Starter Kit TC233 (Infineon)

Una vez comenzado el proyecto con la placa de evaluación comentada, surgieron distintas complicaciones durante la integración, por lo que se decidió cambiar de placa de evaluación por una de NXP, en concreto la S32K144. El principal problema encontrado durante el desarrollo fue la generación de código del sistema operativo y de la MCAL, ya que la herramienta ofrecida para el desarrollo del OS y de la MCAL (Tresos Studio de Elektrobit) constaba de poco soporte por lo que surgieron problemas en el desarrollo de ambos que, con la documentación, tiempo y conocimiento que había entonces, hizo que se planteara el uso de otro microcontrolador/SW.

Esta placa de evaluación no fue considerada inicialmente al tratarse de un fabricante emergente en el sector que no se encontraba en la cartera de suppliers conocidos al inicio del proyecto.

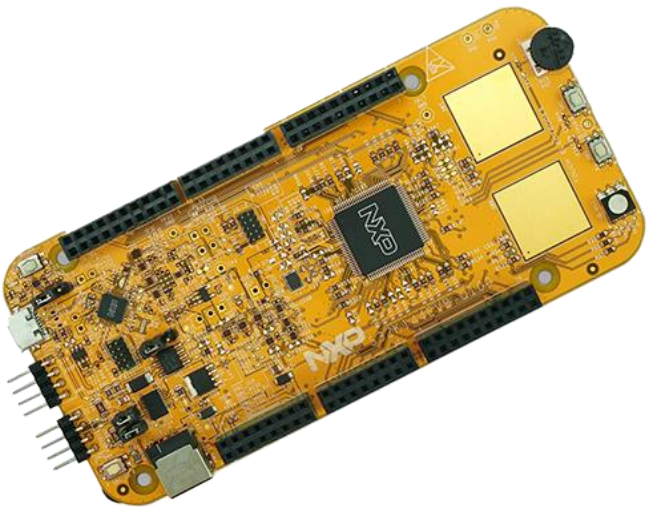

Figura 3-7. S32K144 Evaluation Board (NXP)

Esta placa de evaluación, además de tener un coste inferior a las anteriormente descritas (89$), cuenta también con multitud de soporte, documentación y guías de desarrollo para facilitar su uso, así como un gran abanico de herramientas software compatibles.

Las características de la placa de evaluación S32K144 se muestran en detalle en la siguiente tabla:

Tabla 3-5. Características S32K144

MCU	32-bit Arm Cortex-M4F S32K14 MCU
SBC	UJA1169TK: Mini high speed CAN system basis chip
Transceivers	TJA1027: LIN 2.2A/SAE J2602
Adaptadores	OpenSDA
Coste	~70€
Compatibilidad	Arduino UNO pinout compatible
Interfaces	CAN, LIN, UART/SCI
Potencia	MicroUSB/12V
Componentes	2 x touchpad 1 x potenciómetro 1 x RGB LED 2 x botones

Tras definir los requisitos hardware necesarios para el proyecto, en el siguiente apartado se procederá a definir los requisitos software.

3.1.2 Requisitos Software

En los inicios del proyecto, una de las primeras tareas fue la realización de un estudio sobre las distintas herramientas software existentes para, posteriormente, realizar una selección acorde a las necesidades del proyecto.

Al comenzar con la búsqueda de las herramientas software disponibles que ofrecen servicios compatibles con AUTOSAR, llama la atención las pocas alternativas existentes, donde destaca un grupo reducido de proveedores, que ofrecen unas herramientas a un coste altísimo, en torno a unos 100.000€ podría costar adquirir un toolchain SW completo. Los proveedores más importantes dentro del sector son:

- Mentor
- Vector
- dSpace
- ElektroBit
- KPIT
- ETAS

Figura 3-8. Principales proveedores del sector

Tras conocer los principales proveedores, se seleccionaron las herramientas que serían necesarias para satisfacer la arquitectura AUTOSAR. En la siguiente tabla se recogen las herramientas disponibles para cada una de las capas de AUTOSAR.

Tabla 3-6. Herramientas disponibles por módulo de AUTOSAR

Módulo de AUTOSAR	**Herramientas disponibles**
Capa de aplicación	AUTOSAR Blockset (Mathworks) System Desk (dSpace) DaVinci Developer (Vector) ASCET-DEVELOPER (ETAS)
RTE	RTA-RTE (ETAS) DaVinci Configurator (Vector)
BSW	ISOLAR-B, RTA-BSW, RTA-OS (ETAS) EB Tresos Studio, EB Tresos AutoCore, EB Tresos AutoCore OS (ElektroBit) DaVinci Configurator (Vector)

Inicialmente, se contactó con algunos de estos proveedores solicitando presupuesto para poder hacer uso de las herramientas expuestas anteriormente. A pesar del carácter de investigación que tenía el proyecto, el presupuesto que ofrecen estos proveedores sigue siendo muy cercano a los 100.000€ comentados anteriormente por lo que se escapa del presupuesto establecido para el proyecto.

3.1.2.1 Selección de toolchain

Tras confirmar los altos costes que supondría trabajar con las herramientas anteriores, surge la necesidad de buscar alternativas más económicas o incluso gratuitas. Debido a esto, contar con herramientas que cumplan AUTOSAR en su totalidad se hace prácticamente imposible, por lo que se buscarán herramientas que ofrezcan soluciones que, sin ser consideradas full-AUTOSAR, sean compatibles con el estándar.

Aunque la capa de aplicación y RTE no son parte de este proyecto, para tener una vista global del toolchain, se comentará brevemente las soluciones disponibles para ambas.

Una de las opciones propuestas dentro de la Capa de Aplicación se trata de una extensión de Matlab llamada AUTOSAR Blockset, la cual es accesible al desarrollarse el proyecto dentro de la Universidad de Sevilla, ya

que ésta ofrece a sus estudiantes acceso a multitud de herramientas software, entre las que se encuentra Matlab. Por lo que el software utilizado para el desarrollo de la Capa de Aplicación será AUTOSAR Blockset.

Para el RTE, se desarrollará una herramienta propia basada en Python que, tras realizar un análisis de un fichero .csv, generará el código necesario para el RTE.

Dentro del BSW, tras el análisis de las herramientas disponibles a bajo coste o coste cero, se hará uso de dos herramientas para el desarrollo del Sistema Operativo y del BSW.

Tras el análisis de todas las opciones disponibles, Erika Enterprise será el software utilizado para el Sistema Operativo. Es un sistema operativo en tiempo real para sistemas embebidos el cual cumple con el estándar OSEK/VDX, del cual surge AUTOSAR. Este software ofrece además un entorno de desarrollo basado en Eclipse sobre el que se desarrollará el proyecto completo.

Para desarrollar el BSW, con el objetivo de conseguir unos costes mínimos, la alternativa escogida no es full-AUTOSAR. En lugar de hacer uso de una MCAL, se utilizará la HAL que ofrece el fabricante de la placa de evaluación que se usará en el proyecto, cuya utilizacíon es análoga y gratuíta.

Para finalizar, se muestra en la siguiente tabla la selección final del software.

Tabla 3-7. Selección final del software

Módulo de AUTOSAR	**Herramientas disponibles**
Capa de aplicación	AUTOSAR Blockset (Mathworks)
RTE	Herramienta python de desarrollo propio
BSW	Erika Enterprise (OS) HAL ofrecida por el fabricante de la placa de evaluación (BSW)

Figura 3-9. Selección final del software

La configuración y uso de las herramientas seleccionadas se explicará en detalle en capítulos posteriores.

3.2 Requisitos de arquitectura

En cuanto a la arquitectura del sistema, el principal requisito es que sea una arquitectura compatible con AUTOSAR o, en caso de que no sea posible una arquitectura full-AUTOSAR, que la totalidad o algunos de sus módulos sean parcialmente compatibles con el estándar.

Como se comentó en el primer capítulo del proyecto, dentro de la arquitectura AUTOSAR, el alcance del proyecto se centra en el nivel más bajo, por lo que se obviarán los requisitos necesarios tanto en el RTE como en la capa de aplicación.

Dentro del BSW, serán necesarios los siguientes módulos:

- Sistema operativo: para el proyecto será necesario tener un sistema operativo en tiempo real (RTOS) que sea capaz de gestionar las distintas tareas e interrupciones necesarias, scheduling…
- I/O Hardware Abstraction Layer: el propósito de tener esta capa de abstracción será hacer transparente el hardware localizado al más bajo nivel de los componentes localizados en la capa de aplicación.
- Comunicación (UART): se necesitará también contar con comunicación la placa y el exterior, tanto para depuración como para el envío de órdenes al vehículo.
- GPIO: debido a los elementos externos que serán necesarios integrar en el sistema, también habrá que configurar las entradas y salidas del sistema
- PWM: para el control de los motores y la actuación sobre ellos, se precisará de un módulo PWM (Pulse Width Modulator).

3.3 Requisitos funcionales y no funcionales

3.3.1 Requisitos funcionales

Los requisitos funcionales que deberá cumplir el sistema se agruparán dentro de otro subgrupo que se denominará requisitos dinámicos. Donde todos los requisitos existentes en ese punto estarán relacionados con la dinámica de movimiento que deberá tener el vehículo.

3.3.1.1 Requisitos dinámicos

- El sistema debe ser capaz de circular hacia delante.
- El sistema debe ser capaz de circular hacia atrás.
- El sistema debe ser capaz de circular hacia la derecha.
- El sistema debe ser capaz de circular hacia la izquierda.
- El sistema debe ser capaz de controlar la velocidad.

3.3.2 Requisitos no funcionales

Los requisitos no funcionales serán clasificados en cuatro bloques. El primero de ellos serán los requisitos de interfaz, donde irán los requisitos relacionados con las interfaces de comunicación con las que contará el sistema. El siguiente será los requisitos de recursos, es decir, los requisitos relacionados con la alimentación y consumo del sistema. Seguidamente, se enumerarán los requisitos relacionados con la calidad del proyecto. Y, por último, los requisitos físicos, como aspectos relacionados con la plataforma hardware, motores, etc.

3.3.2.1 Requisitos de interfaz

- El sistema se comunicará de manera inalámbrica con un dispositivo de control.
- El sistema debe ser controlable a través de un dispositivo externo.
- El sistema debe contar con una interfaz de depuración

3.3.2.2 Requisitos de recursos

- El sistema debe estar alimentado por baterías.

3.3.2.3 Requisitos de calidad

- Uso de una plataforma de testeo (HIL).

- Desarrollo basado en simulaciones.
- Uso de herramientas de generación de código para funciones de alto nivel.

3.3.2.4 Requisitos físicos

- El sistema debe contar con una unidad principal.
- El sistema debe contar con cuatro motores DC.
- El sistema debe tener un tamaño adecuado para soportar distintos dispositivos como sensores, baterías, placa de evaluación.

Una vez finalizada la descripción de los distintos requisitos existentes para el proyecto, en el siguiente capítulo se explicará cómo se ha realizado el diseño del sistema, tanto software, como hardware

4 DISEÑO DEL SISTEMA

En el presente capítulo se describirá el diseño del sistema, siendo éste dividido en diseño hardware y diseño software.

En el diseño hardware se expondrá, en base a un diagrama que contendrá los elementos hardware presentes en el sistema, en qué consisten cada uno de estos elementos y cómo se conectan entre ellos

Por otra parte, en el diseño software, al igual que en el hardware, serán explicados los elementos software necesarios y su forma de conexión, situados dentro de la arquitectura AUTOSAR.

4.1 Diseño hardware

En el diseño hardware, el núcleo será la placa de evaluación S32K144 de NXP, la cual facilitará la interactuación entre el resto de los elementos del sistema.

Los elementos hardware presentes en el sistema son los siguientes:

- Placa de evaluación S32K144 (NXP)
- Batería
- Motores DC
- Encoders
- Controladores para los motores
- Controlador bluetooth
- Móvil

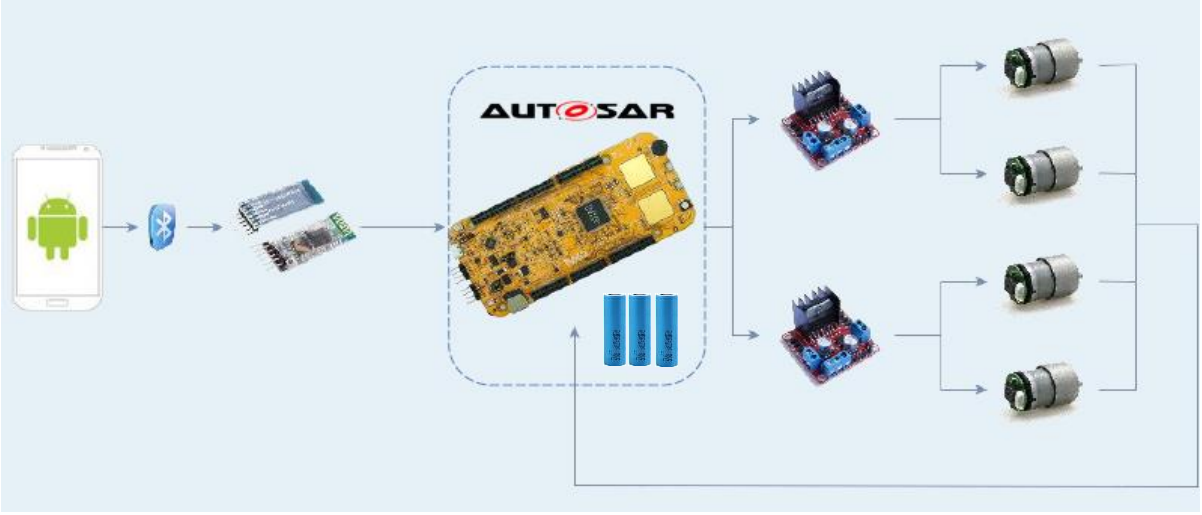

Figura 4-1. Arquitectura hardware del sistema

A continuación, se ofrecerá una visión general del sistema, sus conexiones y los elementos necesarios, explicándose con más detalle en el siguiente capítulo cada uno de ellos.

4.1.1 Placa de evaluación S32K144 – Batería

La placa de evaluación soporta distintas opciones de alimentación: micro USB o 12V. A lo largo del proyecto, las formas de alimentar la placa han ido evolucionando por necesidades de este. En un primer momento, para las primeras pruebas, se alimentaba la placa mediante un micro USB, que al poco tiempo fue sustituido por una batería de plomo de 12V de tensión nominal y 7Ah, cuyo principal inconveniente era su peso, 2.65kg.

Figura 4-2. Batería de plomo de 12V

Debido al peso, los motores no eran capaces de alcanzar su funcionamiento óptimo, por lo que se busco una alternativa que redujese este peso considerablemente, seleccionándose finalmente tres baterías de litio de 3.7V y 2.6Ah que, conectadas en serie consiguen la tensión necesaria para alimentar la placa. El peso de cada una de estas baterías es de 48g, siendo el total 144g.

Figura 4-3. Batería de litio de 3.7V

A continuación, se muestra una tabla comparativa de ambas baterías.

Tabla 4-1. Tabla comparativa de las baterías

	Batería de plomo	**Batería de litio**
Tensión nominal	12V	3.7V
Capacidad	7Ah	2.6Ah
Peso	2.65kg	48g
Tamaño	151 x 65 x 97.5	69 x 19 ± (2-1 mm)

4.1.2 Placa de evaluación S32K144 – Controladores – Motores DC – Encoders

Para el control completo de los cuatro motores DC será necesario en primer lugar un controlador para los motores, en el proyecto se ha usado el modelo L298N. Consiste en un puente H que permite controlar la velocidad y la dirección de dos motores de corriente continua, por lo que se hará uso de dos controladores L298N.

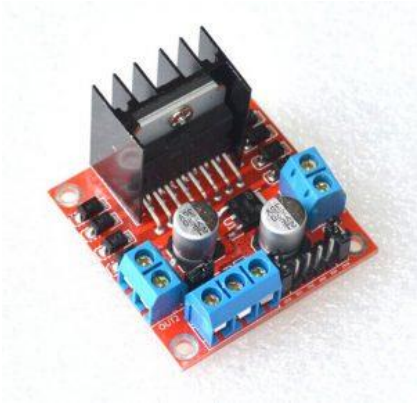

Figura 4-4. Controlador L298N

Con el propósito de recibir información de los motores, cada uno de ellos trae incorporado un encoder de cuadratura con el cual se podrá calcular las rpm de cada motor y, tras analizar esta información, compensar la potencia entregada a algún motor si fuera necesario.

Figura 4-5. Encoder de cuadratura

4.1.3 Placa de evaluación S32K144 – Controlador Bluetooth – Móvil

En los inicios del proyecto, se hacía uso de un depurador USB como el de la figura a través del cual se enviaban órdenes al sistema y se recibía información de depuración.

Esta opción fue sustituida por un dispositivo bluetooth conectado a la placa de evaluación que se comunicaría de manera inalámbrica con un dispositivo móvil.

Figura 4-6. Elementos hardware para envío y recepción de datos

4.2 Diseño software

El diseño software ha sido realizado sobre la arquitectura de capas de AUTOSAR, ubicando los componentes que se requieren en la capa indicada para ellos, pudiendo ser ubicados en MCAL, BSW, RTE o APP.

A continuación, se realizará una breve descripción por capas de los componentes software utilizados, cuya ubicación y conexión se aprecian en el siguiente diagrama.

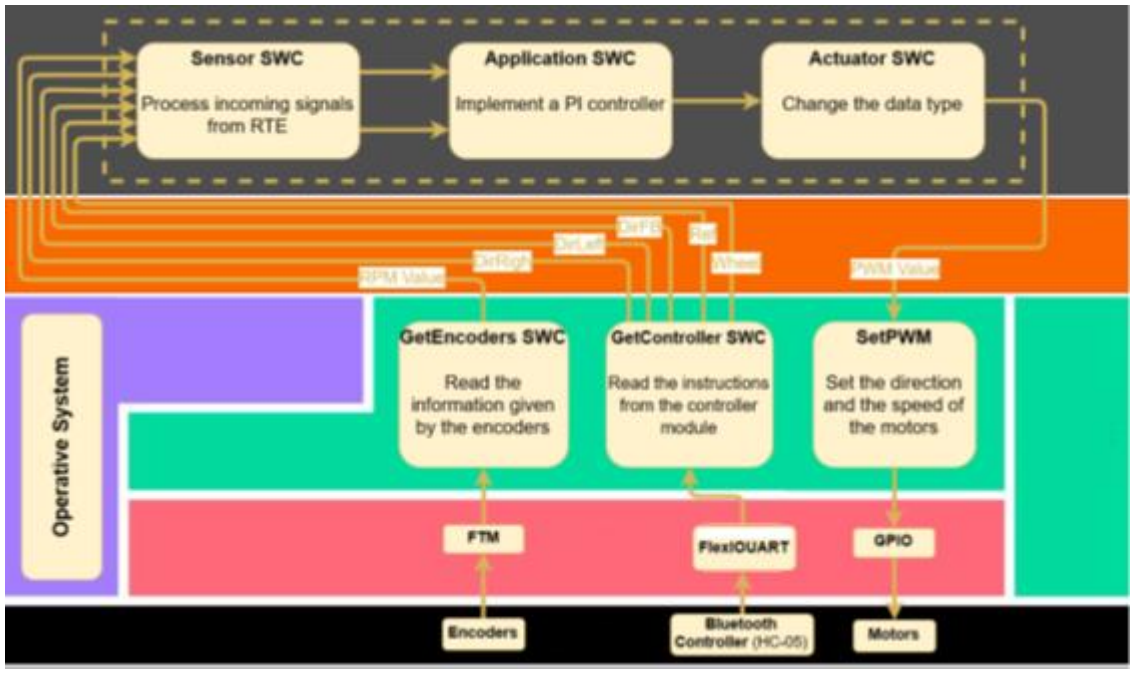

Figura 4-7. Arquitectura software del sistema

4.2.1 Basic Software

4.2.1.1 MCAL

Como MCAL del sistema será utilizada, como ya se ha comentado, la HAL ofrecida por NXP. Para el proyecto se utilizarán algunos de los módulos que ofrece: FTM, FlexIO UART, y GPIO.

A continuación, se realizará una breve descripción explicando la funcionalidad ofrecida por cada uno de ellos, por qué es necesario y será utilizado en el proyecto y la relación que tienen con la capa superior (ECU

Abstraction Layer).

Algunas de las características o funcionalidades ofrecidas por el módulo FTM (FlexTimer Module) son tanto la generación de señales PWM como el procesamiento de estas. Dentro del proyecto, será utilizado para realizar el procesado de las señales recibidas por parte de los cuatro encoders, permitiendo conocer parámetros de interés como pueden ser el periodo o el duty cycle de estas señales. Para alcanzar el objetivo del procesado de las señales, será necesaria la implementación de un módulo de desarrollo propio en la ECU Abstraction Layer llamado GetEncoders.

Además, el módulo FTM ofrece también una funcionalidad de contador que será utilizada por el sistema operativo con el principal objetivo de fijar las periodicidades del mismo.

El módulo FlexIO del que dispone la HAL es altamente configurable y permiter emular un amplio rango de protocolos de comunicación como UART, I2C, SPI, o LIN, entre otros. Concretamente se usará el protocolo de comunicación UART con el objetivo de enviar información necesaria para depuración y del procesamiento de información recibida por dispositivo bluetooth. Al igual que ocurre con el módulo anterior, también será necesario desarrollar un módulo en la ECU Abstraction Layer que se denomina GetController.

Para actuar sobre las entradas y salidas del sistema será necesario un módulo GPIO (General Purpose Input/Output), el cual será el encargado de enviar las señales necesarias para que los motores DC funcionen como se espera. Este módulo también precisará del desarrollo del módulo SetPWM dentro de la ECU Abstraction Layer.

Por ultimo, en la siguiente tabla se sintetiza la información relativa a los módulos necesarios dentro de la HAL:

Tabla 4-2. Resumen módulos HAL

Módulo	Funcionalidad	Módulo ECU Abstraction Layer
FTM	Lectura de los encoders, actuación sobre los motores, periodicidad de tareas.	GetEndoders, OS
FlexIO UART	Depuración y comunicación con el dispositivo bluetooth	GetController
GPIO	Actuación sobre los motores	SetPWM

4.2.1.2 ECU Abstraction Layer

Dentro de la ECU Abstraction Layer, se contará con tres SWCs o módulos de desarrollo propio para propósitos que se explicarán a continuación.

El primero de los módulos será **GetEncoders**, cuyo principal propósito es realizar la lectura recibida por cada uno de los encoders para, posteriormente, analizarla y obtener un valor en rpm que será enviado a la capa de aplicación.

Su diagrama de flujo se muestra en la Figura 4-8. Diagrama de flujo de GetEncoders, donde en primer lugar deberá arrancarse el módulo FTM necesario para realizar una captura de los flancos de subida y bajada de los dos pulsos de cada uno de los encoders. Una vez capturados, pueden obtenerse los valores deseados de frecuencia y rpm de cada uno de ellos.

Figura 4-8. Diagrama de flujo de GetEncoders

Otro módulo que será necesario será **GetController**, el cual procesará las instrucciones enviadas por el dispositivo móvil y enviará a la capa de aplicación tanto el sentido de giro como la velocidad.

En la Figura 4-9. Diagrama de flujo de GetController se define un diagrma de flujo simplificado del funcionamiento de este. En primer lugar, se obtiene un buffer por parte del controlador con 6 valores, la mitad de ellos hacen referencia al eje x y la otra mitad al eje y. Este buffer debe ser procesado para poder sacar información útil para el sistema como el sentido (adelante/atrás o izquierda/derecha) y la velocidad.

Figura 4-9. Diagrama de flujo de GetController

Por último, el tercer módulo es **SetPWM** y su tarea es fijar la dirección y velocidad de cada uno de los motores en función de la información recibida por la capa de aplicación. La actuación sobre los motores se realizará a través del GPIO.

De la capa de aplicación se recibirá una señal por cada uno de los motores que, tras ser procesada por el módulo, se obtendrán tanto el sentido de giro de cada uno de ellos como su velocidad que serán enviadas al módulo GPIO.

Figura 4-10. Diagrama de flujo de SetPWM

Además de estos tres módulos, también se encuentra dentro de esta capa el OS, que será el encargado de gestionar las tareas e interrupciones necesarias en el sistema, entre otras.

El sistema operativo elegido es Erika3 de Evidence, que consiste en un RTOS de automoción que cumple con las necesidades del proyecto.

4.2.2 RTE

Aunque no forma parte del alcance del proyecto, se explicará a continuación brevemente cómo se ha diseñado el RTE. En el siguiente capítulo donde se desarrollarán cada uno de los elementos del sistema con más detalle será omitido.

El RTE será generado haciendo uso de una herramienta Python de desarrollo propio que, mediante el análisis de una hoja de Excel que incluye todas las señales necesarias, generará los ficheros necesarios. En la siguiente figura se puede apreciar cómo se realiza esa generación en base a la hoja de Excel.

Signal					SWC	App_SWC	BSW	IoHwAb
Data Access Mode	Port Name	Data Element	Runnable Name	Type of data	Inport	Outport	Inport	Outport
Explicit Read	RPMIoHw	RPMFRValue	Runnable_App_SWC	sint16	Yes	No	No	Yes
Explicit Read	RPMIoHw	RPMFLValue	Runnable_App_SWC	sint16	Yes	No	No	Yes
Explicit Read	RPMIoHw	RPMBRValue	Runnable_App_SWC	sint16	Yes	No	No	Yes
Explicit Read	RPMIoHw	RPMBLValue	Runnable_App_SWC	sint16	Yes	No	No	Yes
Explicit Read	RefIoHw	RefValue	Runnable_App_SWC	uint8	Yes	No	No	Yes
Explicit Read	DirFBIoHw	DirFBValue	Runnable_App_SWC	uint8	Yes	No	No	Yes
Explicit Send	PWMIoHw	PWMFRValue	Runnable_App_SWC	uint8	No	Yes	Yes	No
Explicit Send	PWMIoHw	PWMFLValue	Runnable_App_SWC	uint8	No	Yes	Yes	No
Explicit Send	PWMIoHw	PWMBRValue	Runnable_App_SWC	uint8	No	Yes	Yes	No
Explicit Send	PWMIoHw	PWMBLValue	Runnable_App_SWC	uint8	No	Yes	Yes	No
Explicit Read	DirRightIoHw	DirRightValue	Runnable_App_SWC	uint8	Yes	No	No	Yes
Explicit Read	DirLeftIoHw	DirLeftValue	Runnable_App_SWC	uint8	Yes	No	No	Yes
Explicit Read	WheelCoefIoHw	WheelCoefValue	Runnable_App_SWC	uint8	Yes	No	No	Yes

```
18  //RTE API Implementation Function Definitions
19  Std_ReturnType Rte_Read_RPMIoHw_RPMFRValue(sint16 *data){
20    (*data) = Rte_BufferValue_RPMFRValue;
21  return(RTE_E_OK);
22  }
23
24  Std_ReturnType Rte_Read_RPMIoHw_RPMFLValue(sint16 *data){
25    (*data) = Rte_BufferValue_RPMFLValue;
26  return(RTE_E_OK);
27  }
```

Figura 4-11. Generación de ficheros del RTE

4.2.3 APP

Al igual que ocurre con el RTE, la capa de aplicación tampoco entra dentro del alcance del proyecto, pero, de igual manera que en el anterior apartado, se explicará brevemente para tener una visión completa del sistema.

Dentro de esta capa se implementarán 3 SWCs: Sensor SWC, Application SWC y Actuator SWC.

El Sensor SWC tiene como objetivo la conversión del valor en rpm a valores de referencia y dirección. Los valores de referencia tomarán valores entre cero (parado) y 100 (máxima velocidad).

El segundo de ellos, Application SWC estará compuesto por un controlador PI

Y, por último, Actuator SWC simplemente convertirá el tipo de dato a un entero de ocho bits.

En la siguiente figura se puede ver el funcionamiento de la capa de aplicación sin entrar en profundidad.

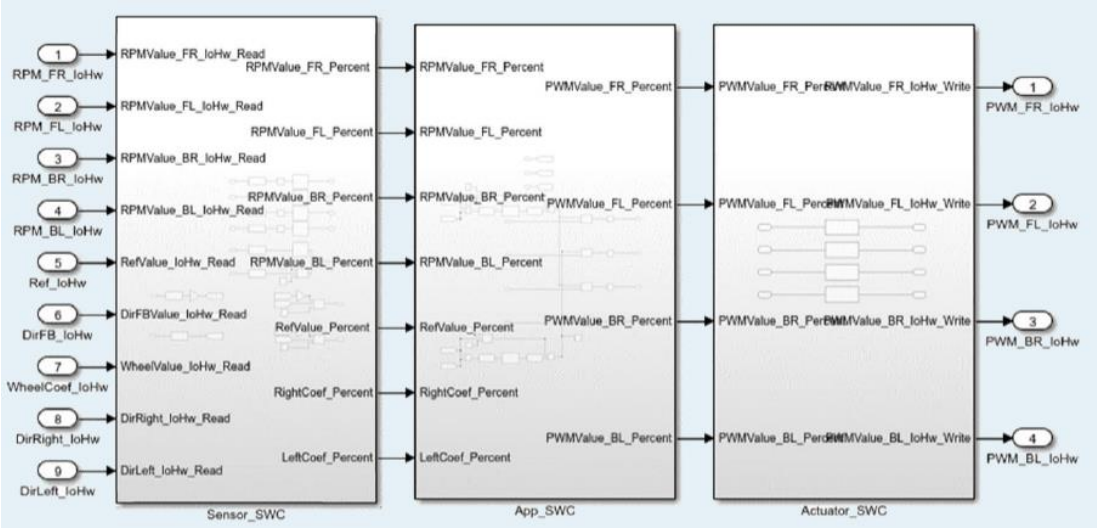

Figura 4-12. Diagrama de la Capa de Aplicación

4.2.4 Aplicación móvil

Por último, será necesario el diseño de una aplicación móvil que permita la conexión con el módulo bluetooth conectado a la placa de evaluación y el envío de información para poder controlar el sistema. Se pretende conseguir una interfaz sencilla que, mediante un joystick, envíe la información al sistema.

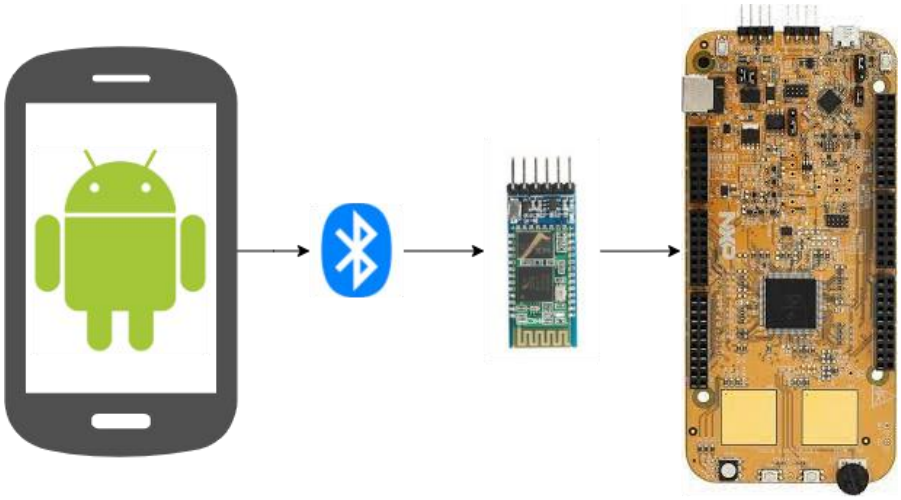

Figura 4-13. Funcionamiento conexión bluetooth

Todos los elementos citados anteriormente serán analizados en profundidad en el siguiente capítulo.

5 DESARROLLO DEL SISTEMA

En este capítulo se explicarán en detalle todos los elementos hardware y cómo se ha desarrollado el software, detallando las herramientas utilizadas y cómo ha sido configurado.

5.1 Desarrollo hardware

5.1.1 Módulo Bluetooth

Como se ha comentado en capítuos anteriores, se ha integrado un módulo Bluetooth (HC-05) como controlador y se ha diseñado una aplicación Android. El desarrollo de la aplicación se explicará en el apartado Desarrollo Software.

El HC-05 es un módulo Bluetooth diseñado para la comunicación inalámbrica. Las especificaciones del HC-05 Bluetooth son las siguientes

- Funcionamiento de baja potencia, de 3 a 5V
- Viene con antena integrada
- Interfaz UART con velocidad de transmisión programable
- Las velocidades de transmisión soportadas son 9600, 19200, 38400, 57600, 115200, 230400 y 460800.
- Se conecta automáticamente al último dispositivo encendido por defecto.
- Frecuencia: 2,45 GHz
- Puede utilizarse en una configuración maestra o esclava.

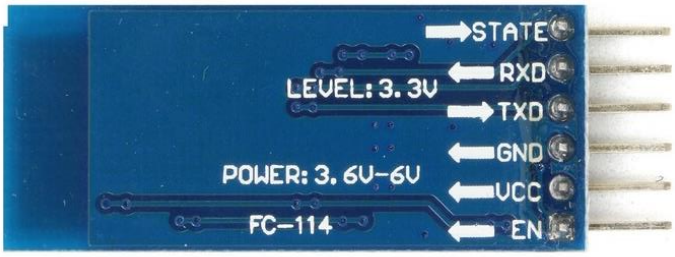

Figura 5-1. Pinout HC-05

Cada uno de los pines del módulo se explican en la siguiente tabla.

Tabla 5-1. Pines módulo HC-05

Pin	Descripción
Key/EN	Se utiliza para poner el módulo Bluetooth en modo de comandos AT. Si el pin Key/EN está en alto, el módulo funcionará en modo de comandos. De lo contrario, por defecto está en modo de datos. La tasa de baudios por defecto del HC-05 en modo comando es de 38400bps y 9600bps en modo datos
VCC	5V o 3.3V
GND	Tierra
TXD	Los datos recibidos de forma inalámbrica por el módulo Bluetooth se transmiten en serie en este pin
RXD	Los datos recibidos serán transmitidos de forma inalámbrica a través de este pin
State	Este pin mostrará si el módulo está conectado o no

5.1.2 Controladores para motores

Para el control de los cuatro motores DC, como ya se comentó en capítulos anteriores, se ha decidido usar el controlador L298N, que permite trabajar con tensiones de hasta 35V y controlar el sentido de giro de los motores.

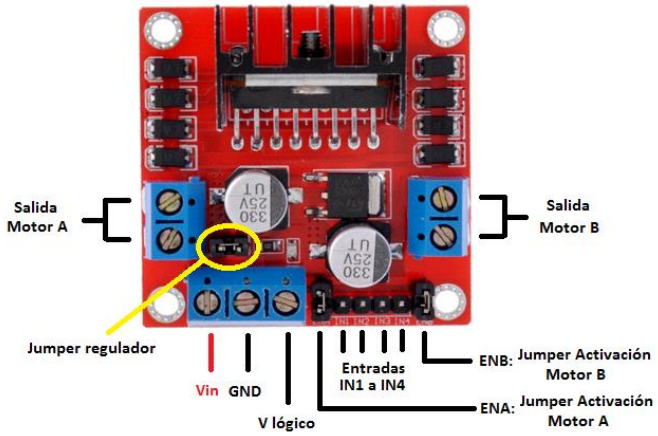

Figura 5-2. Controlador L298N

La alimentación del módulo varía en función del jumper regulador. Si se encuentra activo, el módulo permite una alimentación de entre 6V a 12V (Vin). El regulador se encuentra activo y el pin V lógico tendrá un voltaje de 5V.

Cuando el jumper se encuentra inactivo, el módulo permite una alimentación de entre 12V a 35V (Vin). Como el regulador no está funcionando, habrá que conectar el pin de +5V a una tensión de 5V para alimentar la parte lógica del L298N.

La opción utilizada para el proyecto es la primera de ellas, teniendo conectado el jumper y alimentando el módulo con una tensión de 12V.

Como se explicó en el apartado de diseño, serán necesarios dos controladores para manejar los cuatro motores.

En cuanto al control de giro y velocidad de los motores, los pines implicados son ENA, IN1 e IN2 para controlar el motor A y ENB, IN3 e IN4 para controlar el motor B.

Los pines ENA y ENB desactivan la salida, por lo que se puede conectar permanentemente mediante el uso de un jumper, o conectar una señal PWM para controlar la velocidad de giro, siendo esta segunda opción la utilizada en el proyecto.

La siguiente tabla muestra las configuraciones para definir el sentido de giro de los motores.

Tabla 5-2. Configuraciones para asignar sentido de giro a los motores

	Adelante	**Atrás**	**Parado**
IN1/IN3	HIGH	LOW	LOW
IN2/IN4	LOW	HIGH	LOW

5.1.3 Encoders

Como ya se ha comentado, cada uno de los motores contará con un encoder de cuadratura que permitirá conocer tanto velocidad de giro de cada uno de los motores como el sentido de giro. Su funcionamiento se basa en la generación de dos señales cuadradas con un desfase de 90º, de ahí su nombre, al ocupar un cuadrante.

Como puede observarse en la Figura 5-4. Funcionamiento encoder, el sentido de giro podrá determinarse en función de qué señal esté por delante. En el ejemplo, si la señal o canal A está por delante del canal B, el motor girará en sentido horario y, por el contrario, si B está por delante de A, el sentido será antihorario.

Para el cálculo de la velocidad o rpm de cada uno de los motores, a diferencia del cálculo del sentido de giro, solo será necesario monitorizar una de las dos señales generadas.

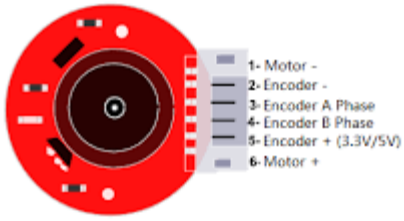

Figura 5-3. Pinout encoder

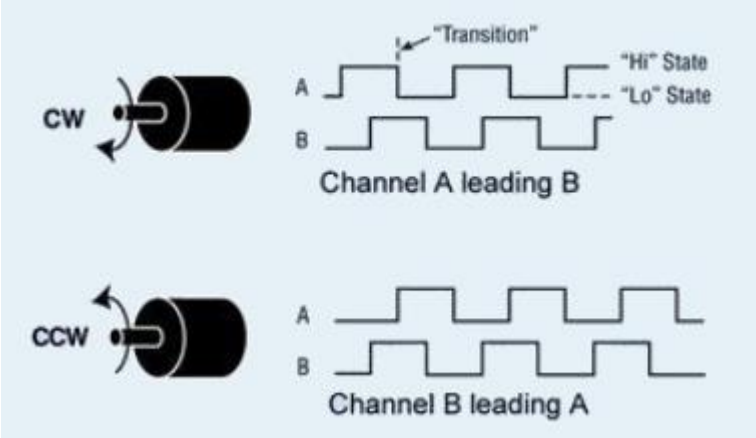

Figura 5-4. Funcionamiento encoder

5.2 Desarrollo software

Como se ha comentado en el capítulo anterior, dentro del desarrollo software, no se entrará en detalle en las tareas de desarrollo de aplicación y RTE, siendo el BSW, y el desarrollo de la aplicación para el dispositivo móvil los puntos a desarrollar.

5.2.1 Basic Software

Para un mejor entendimiento de las principales actividades y desarrollos llevados a cabo dentro del Basic Software, éstos serán divididos en los tres siguientes grupos: MCAL, ECU Abstraction Layer y OS.

5.2.1.1 MCAL

Para la configuración de los módulos necesarios para la MCAL, el entorno de desarrollo utilizado será S32 Design Studio, ofrecida por NXP.

Este entorno de desarrollo integrado está basado en Eclipse y enfocado a los microcontroladores fabricados por NXP. Alguno de los aspectos importantes a destacar de este entorno son los siguientes:

- Software integrado para la configuración de la HAL.
- Multitud de soporte y comunidad activa.
- Soporte para distintos depuradores.
- Soporte de RTOS.

En resumen, el S32 ha sido el entorno de desarrollo seleccionado ya que permite la configuración, implementación y depuración sin necesidad de utilizar distintas herramientas.

S32 Design Studio ofrece distintas herramientas o software integrados en el entorno. Para el presente proyecto, la más interesante es el Processor Expert, ya que ofrece la posibilidad de configurar la HAL del micro (pines, reloj, periféricos…).

Processor Expert ofrece distintas ventanas al usuario:

- **Component inspector**. Permite configurar los componentes del proyecto.
- **Component library**. Muestra todos los componentes soportados.
- **Configuration Registers**. Muestra un resumen de los ajustes de inicialización de los periféricos.
- **Memory Map**. Muestra el espacio de direcciones y asignación de memoria externa e interna.
- **Components**. Muestra un componente embebido que puede ser utilizado en Processor Expert.
- **Initialization sequence**. Permite personalizar la secuencia de inicialización de los componentes.
- **Processor**. Muestra el procesador utilizado.

En la siguiente figura se muestra la ventana Component Library, donde se pueden ver los módulos que pueden ser configurados:

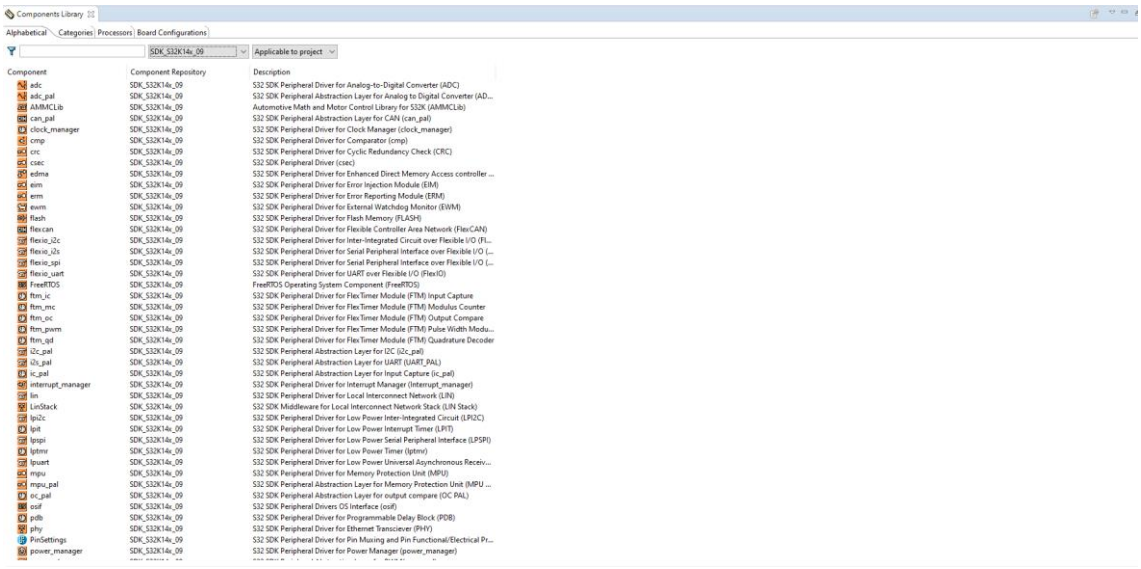

Figura 5-5.Components library

Para los objetivos fijados para el proyecto, será necesaria la configuración de los siguientes módulos: Pin Settings, Clock Manager, FlexIO UART, FTM IC, FTM PWM y FTM MC.

5.2.1.1.1 Pin Settings

El componente Pin Settings ofrece la posibilidad de configurar los pines según su categoría.

Este componente consta de cuatro propiedades:

- **Routing**. Configuración del rutado de los pines.

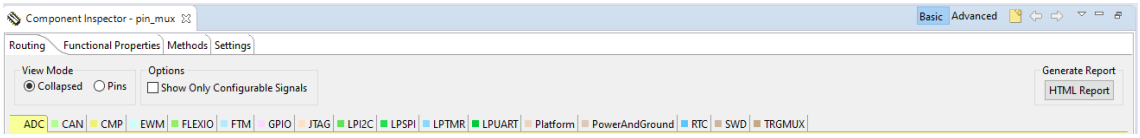

Figura 5-6. Pin Settings – Routing

- **Functional properties**. Configuración de las propiedades de cada uno de los pines. Por ejemplo, permite definir, entre otras funcionalidades, si el pin está asociado a una interrupción o si es pull up o pull down.

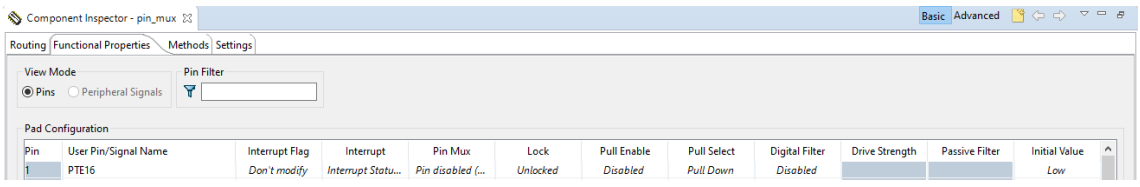

Figura 5-7. Pin Settings – Functional properties

- **Methods**. Muestra los métodos de cada uno de los componentes e indica si es código autogenerado o no.

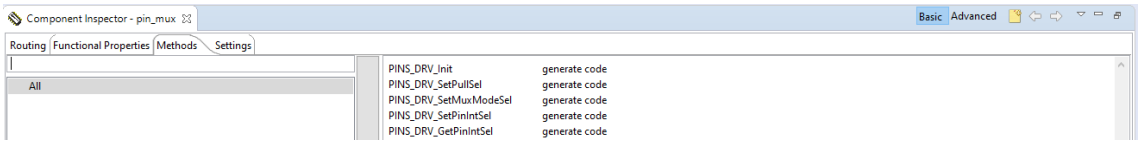

Figura 5-8. Pin Settings – Methods

- **Settings**. Muestra los componentes compartidos con el componente Pin Settings.

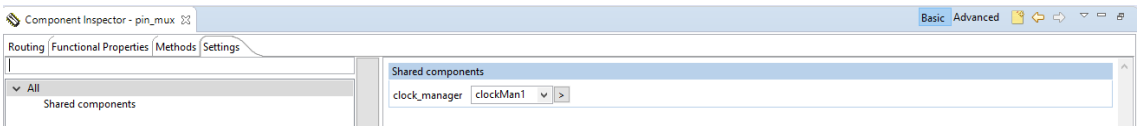

Figura 5-9. Pin Settings – Settings

Los pines GPIO a configurar son los siguientes:

Tabla 5-3. Pines GPIO

Pin	**Funcionalidad**
PTA11 (Out)	Señal para fijar el sentido de giro del motor trasero derecho. Conectada al L298N (IN1)
PTA12 (Out)	Señal para fijar el sentido de giro del motor trasero derecho. Conectada al L298N (IN1)
PTC10 (Out)	Señal para fijar el sentido de giro del motor trasero izquierdo. Conectada al L298N (IN4)
PTC11 (Out)	Señal para fijar el sentido de giro del motor trasero izquierdo. Conectada al L298N (IN3)
PTA14 (Out)	Señal para fijar el sentido de giro del motor delantero derecho. Conectada al L298N (IN3)
PTE7 (Out)	Señal para fijar el sentido de giro del motor delantero derecho. Conectada al L298N (IN4)
PTE0 (Out)	Señal para fijar el sentido de giro del motor delantero izquierdo. Conectada al L298N (IN2)
PTE9 (Out)	Señal para fijar el sentido de giro del motor delantero izquierdo. Conectada al L298N (IN1)

El resto de pines necesarios para los modúlos FTM y FlexIO UART se definirán en los siguientes puntos.

5.2.1.1.2 FlexIO UART

El controlador FlexIO UART proporciona servicios para la comunicación UART utilizando el módulo FlexIO disponible en el procesador S32K144.

Algunas de las características más significativas son:

- Modo de interrupción, DMA o polling
- Proporciona funciones de transmisión y recepción bloqueantes y no bloqueantes
- Tasa de baudios y número de bits configurables
- Sólo un bit de parada
- No admite bit de paridad

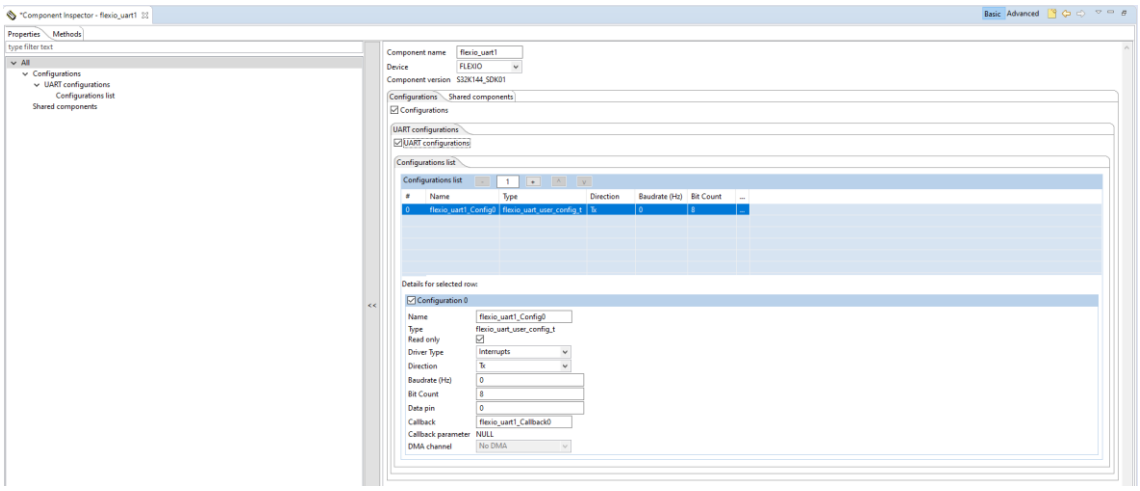

Figura 5-10. FleXIO UART (Component Inspector)

Como se puede apreciar en la figura anterior, las propiedades que pueden configurarse en el módulo FlexIO UART son:

- **Name**. Nombre de la configuración.
- **Type**. Tipo de la estructura de configuración.
- **Read only**. Especifica como se genera la configuración. Las estructuras de solo lectura se declaran con const.
- **Driver type**. Tipo de controlador. Puede ser basado en interrupciones, polling o DMA.
- **Direction**. Transmisión o recepción.
- **Baudrate**. Tasa de baudios en Hz.
- **Bit count**. Número de bits por palabra.
- **Data pin**. Pin a utilizar para la transmisión o recepción de datos.
- **Callback**. Función callbarck para reporter eventos.
- **Callback parameter**. Parámetro para la función callback master. Si es utilizado debe ser inicializado en tiempo de ejecución.
- **DMA channel**. Número de canal DMA.

Como ya se ha comentado, serán necesarios dos canales, uno para la depuración, y otro para la comunicación con el dispositivo bluetooth. Para la depuración será de transmisión y para la comunicación bluetooth de recepción.

En la siguiente tabla se recogen las configuraciones para cada uno de los módulos FlexIO UART.

Tabla 5-4. Configuración FlexIO UART

	FlexIO UART Rx (Bluetooth)	**FlexIO UART Tx (Debug)**
Name	flexio_uart_RX_Config0	flexio_uart_TX_Config0
Type	flexio_uart_user_config_t	flexio_uart_user_config_t
Read only	Si	Si
Driver type	Interrupción	Interrupción
Direction	Rx	Tx
Baudrate (Hz)	115200	115200
Bit count	8	8
Data pin	3 (PTA1)	2 (PTA0)
Callback	-	-
Callback parameter	-	-
DMA channel	-	-

5.2.1.1.3 FTM PWM

En el módulo FTM PWM, se realiza la configuración de cada uno de los canales con PWM. En el caso del proyecto, serán necesarios cuatro canales, uno por cada motor.

Existen diferentes posibilidades a la hora de configurar este módulo, en este caso, solo aplica la configuración independiente del PWM, vista que se puede apreciar en la figura siguiente.

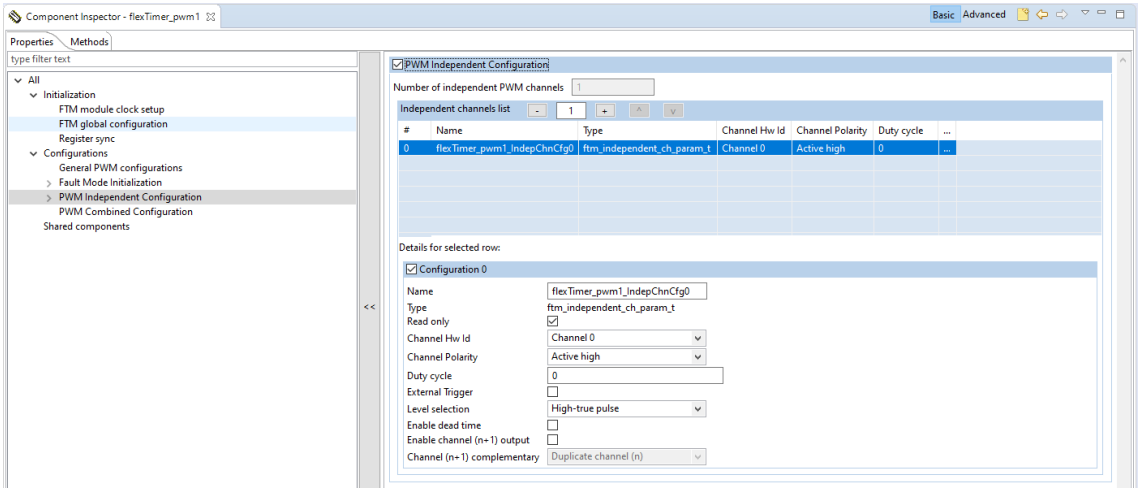

Figura 5-11. FTM PWM (Component inspector)

A continuación, se definen brevemente los parámetros de interés.

- **Name**. Nombre de la configuración.
- **Type**. Tipo de estructura de la configuración.
- **Read only**. Especifica como se genera la configuración. Las estructuras de solo lectura se declaran

con const.

- **Channel Hw Id**. Modo del canal (From FTM channel 0 to FTM channel 7)
- **Channel polarity**. Modo de salida del PWM. Puede ser polaridad activa a nivel alto o activa a nivel bajo.
- **Duty cyle**. Ancho del pulso PWM. Debe ser un valor comprendido entre 0 y 0x8000. 0 = Señal inactiva (duty del 0%). 0x8000 = señal activa (duty del 100%)
- **External trigger**. Activa o desactiva la generación de un trigger del canal.

Serán necesarios, por tanto, cuatro canales que tendrán la misma configuración que se muestra en la tabla siguiente.

Tabla 5-5. Configuración FTM PWM

	Configuración FTM PWM
Name	flexTimer_pwm1_IndepChnCfg0
Type	ftm_independet_ch_param_t
Read only	Si
Channel Hw Id	[0, 1, 2, 3]
Channel polarity	Activa a nivel bajo
Duty cycle	0x1000
External trigger	No

5.2.1.1.4 FTM IC

El módulo FTM Input Capture será utilizado para detectar los flancos de subida y bajada de los pulsos generados por los encoders. Al tener un encoder por cada motor, y cada encoder genera dos pulsos, se necesitarán 8 canales.

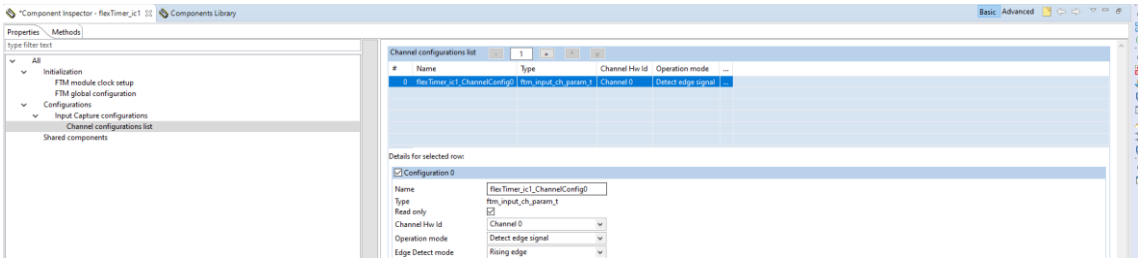

Figura 5-12. FTM IC (Component inspector)

Los parámetros que habría que configurar son los siguientes:

- **Name**. Nombre de la configuración.
- **Type**. Tipo de estructura de la configuración.
- **Read only**. Especifica como se genera la configuración. Las estructuras de solo lectura se declaran con const.
- **Channel Hw Id**. Modo del canal (From FTM channel 0 to FTM channel 7)
- **Operation mode**. Modo de operación. Puede ser detección de flanco o medida de la señal.

- **Edge detect mode**. Modo de detección de flanco. Puede ser flanco de subida, flanco de bajada o flanco de subida o bajada.

La configuración de los ocho canales necesarios se muestra en la siguiente tabla.

Tabla 5-6. Configuración FTM IC

	Configuración FTM IC
Name	flexTimer_ic_ChannelConfig0
Type	ftm_input_ch_param_t
Read only	Si
Channel Hw Id	[0, 1, 2, 3, 4, 5, 6, 7]
Operation mode	Detección de flanco
Edge detect mode	Flanco de subida

5.2.1.1.5 FTM MC

Con el componente FlexTimer Modulus Counter se permitirá la creación de contadores para controlar las periodicidades requeridas por el sistema operativo. La vista ofrecida por el IDE para su configuración es mostrada a continuación.

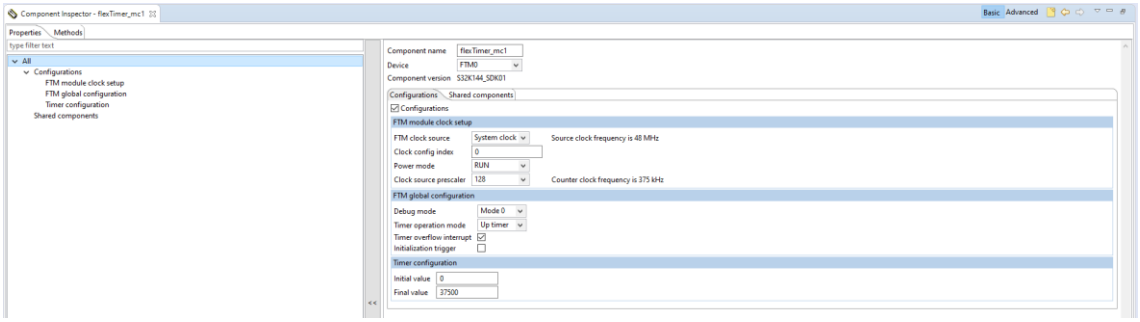

Figura 5-13. FTM MC (Component inspector)

Concretamente serán necesarios dos contadores, el primero de ellos será utilizado para la activación del flujo de tareas del sistema operativo (flexTimer_mc1), y el segundo para detectar si los motores están o no en movimiento (flexTimer_mc2).

Los parámetros que deben ser configurados se explican brevemente en la siguiente lista:

- **Component name**. Nombre del componente.
- **Device**. Módulo FTM.
- **FTM clock source**. Selección de la fuente de reloj FTM. Pueden ser seleccionados reloj del sistema, reloj de frecuencia fija y reloj externo.
- **Clock config index**. Este índice seleccionará la configuración de reloj que se utilizará cuando se use FTM.
- **Power mode**. Si se selecciona el reloj del sistema como fuente de reloj, este parámetro debe configurarse para obtener la frecuencia que depende del modo de alimentación (RUN, VLPR, HSRUN). El controlador no es responsable de cambiar el modo de potencia, por lo que la aplicación debe hacerlo.

- **Clock source prescaler**. Preescalador de fuente de reloj FTM. Selecciona uno de los 8 factores de división para la fuente de reloj seleccionada (1, 2, 4, 8, 16, 32, 64, 128).
- **Debug mode**. Selecciona el comportamiento de FTM en modo de depuración. Puede ser modo 0, modo 1, modo 2 y modo 3.
- **Timer operation mode**. Modo de funcionamiento del FlexTimer. Puede ser up timer o up-down timer.
- **Timer overflow interrupt**. Activa o desactiva la interrupción por desbordamiento del temporizador.
- **Initialization timer**. Habilita o deshabilita la generación del disparo de inicialización.
- **Initial value**. Valor inicial del contador.
- **Final value**. Valor final del contador

Las dos configuraciones necesarias se encuentran en la siguiente tabla:

Tabla 5-7. Configuración FTM MC

	Configuración FTM MC	
Component name	flexTimer_mc1	flexTimer_mc2
Device	FTM2	FTM3
FTM clock source	Reloj del sistema	Reloj del sistema
Clock config index	0	0
Power mode	RUN	RUN
Clock source prescaler	128	128
Debug mode	Modo 0	Modo 0
Timer operation mode	Up	Up
Timer overflow interrupt	Si	Si
Initialization trigger	No	No
Initial value	0	0
Final value	18750	3375

5.2.1.1.6 Clock Manager

Por último, el módulo Clock Manager, cuya configuración se ha utilizado la ofrecida por la herramienta, al estar asociada al microcontrolador del proyecto. Únicamente deben activarse los puertos y módulos utilizados (FTM, FlexIO…).

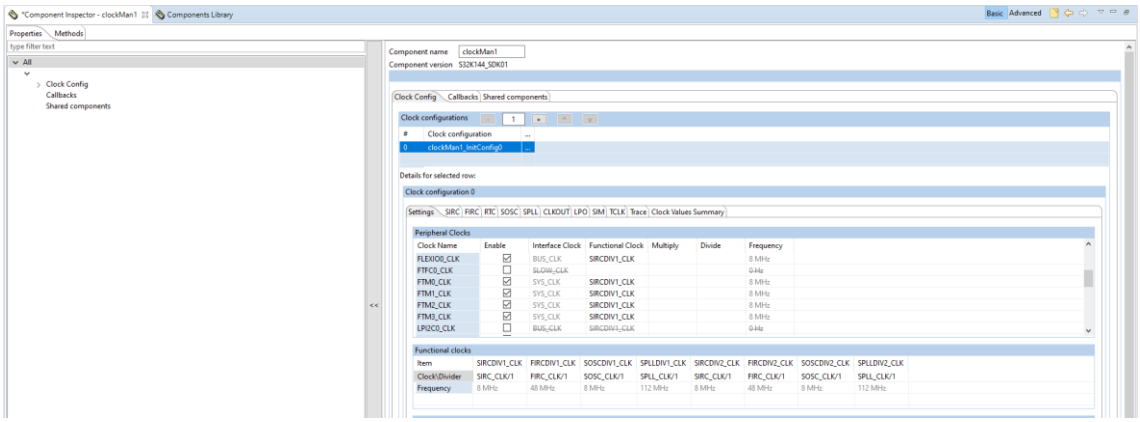

Figura 5-14. Clock Manager (Component inspector)

A continuación, a modo de resumen, se muestran todos los pines utilizados y su objetivo.

Tabla 5-8. Pines utilizados y funcionalidad

PTD0 (OUT)	Señal PWM al motor FR (Conectada al L298N)
PTD1 (OUT)	Señal PWM al motor BR (Conectada al L298N)
PTD15 (OUT)	Señal PWM al motor BL (Conectada al L298N)
PTD16 (OUT)	Señal PWM al motor FL (Conectada al L298N)
PTA0 (OUT)	FlexIO UART TX (Usada para debug)
PTA1 (IN)	FlexIO UART RX (Conectada al módulo BT)
PTA11 (OUT)	Señal para fijar el sentido de giro del motor BR (Conectada al L298N - IN1)
PTA12 (OUT)	Señal para fijar el sentido de giro del motor BR (Conectada al L298N - IN2)
PTA13 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder FL
PTA14 (OUT)	Señal para fijar el sentido de giro del motor FR (Conectada al L298N - IN3)
PTA17 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder FL
PTB0 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder FR
PTB1 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder FR
PTC10 (OUT)	Señal para fijar el sentido de giro del motor BL (Conectada al L298N - IN4)
PTC11 (OUT)	Señal para fijar el sentido de giro del motor BL (Conectada al L298N - IN3)
PTD9 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder BL
PTD11 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder BL
PTE0 (OUT)	Señal para fijar el sentido de giro del motor FL (Conectada al L298N - IN2)
PTE6 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder BR
PTE7 (OUT)	Señal para fijar el sentido de giro del motor FR (Conectada al L298N - IN4)
PTE9 (OUT)	Señal para fijar el sentido de giro del motor FL (Conectada al L298N - IN1)
PTE16 (IN)	Configurado para detectar flancos de subida/bajada de uno de los pulsos del encoder BR

Tras desarrollar todos los elementos necesarios para la MCAL, se procede a explicar la ECU Abstraction Layer.

5.2.1.2 ECU Abstraction Layer

A continuación, se describen los tres SW-Cs implementados en la ECU Abstraction Layer, los cuales ya han sido explicados de manera general en el apartado 4.2.1.2, donde se descrito su propósito y se ha explicado su funcionamiento con un diagrama de flujo a alto nivel.

5.2.1.2.1 GetController

En este SW-C de abstracción de la ECU la información dada por el controlador es leída y almacenada en los buffers RTE. Esta información cambia con respecto al sprint anterior, ahora es una cadena de 6 caracteres, donde los tres primeros se refieren al eje x, y los otros tres al eje y.

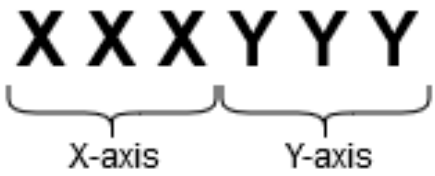

Figura 5-15. Información recibida del controlador

El rango de valores de cada eje oscila entre 0 y 200 con un paso de una unidad. La siguiente figura muestra algunos de los casos extremos.

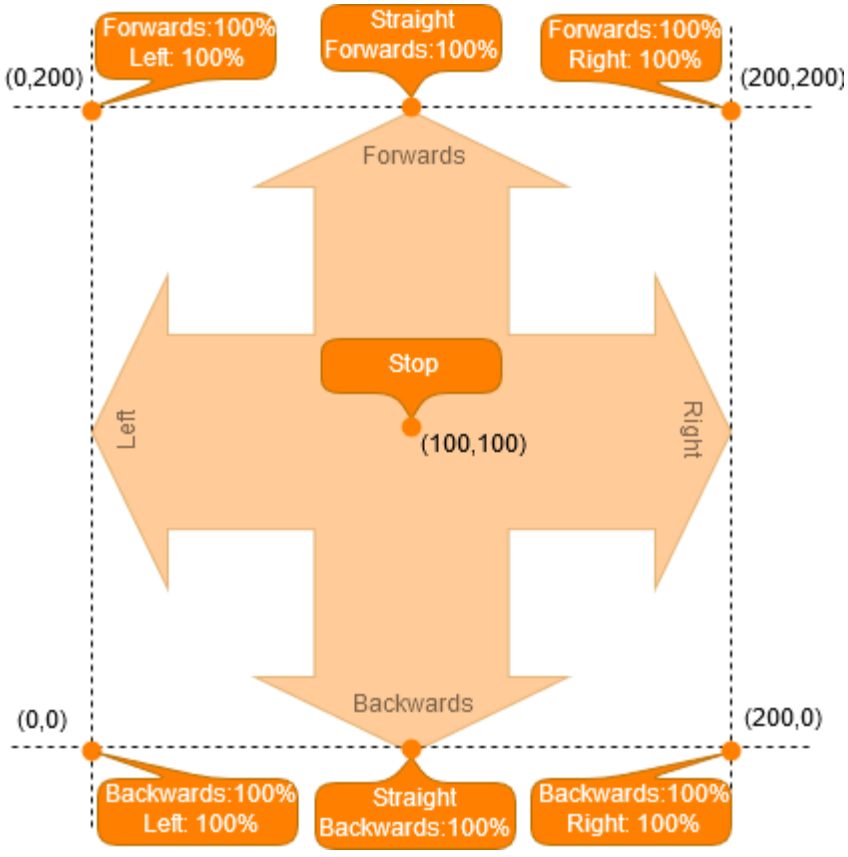

Figura 5-16. Ejemplo de funcionamiento de GetController

Los archivos contenidos en el SW-C son:

- IoHwAb_GetController_SWC.c: definición del void Runnable_GetController_SWC(void) que recibe la información del controlador y envía la dirección y la velocidad al RTE.
- IoHwAb_GetController_SWC.h: fichero de cabecera de IoHwAb_GetController_SWC.c, donde se encuentran todas las definiciones, includes y prototipos de funciones.
- Rte_GetController_SWC.h: fichero de cabecera del RTE, donde se encuentran todas las definiciones, includes y prototipos de funciones.

Las interfaces usadas con el RTE son las siguientes:

- void Get_DirValue(uint8 data): esta interfaz almacena el valor de dirección (dir_sended) leído en un buffer RTE.
- void Get_RefValue(uint8 data): esta interfaz almacena el valor de referencia de la velocidad (speed_sended) leído en un buffer RTE.

5.2.1.2.2 GetEncoders

El objetivo de la lectura de los codificadores consiste en obtener tanto la velocidad (frecuencia/rpm) como el sentido de giro de los motores.

El proceso seguido es el siguiente, se calcula el tiempo entre dos flancos de uno de los pulsos del encoder (FTM MC) y, como resultado, se calcula la frecuencia. Para el cálculo del sentido de giro, cuando se produce un flanco de subida en uno de los encoders, se mide el nivel en el otro y, según esté en nivel alto o bajo, se determina el sentido de giro.

Los componentes HAL/MCAL utilizados para la obtención de la velocidad y el sentido son:

- Pins Driver: se utiliza para configurar 4 pines GPIO como entrada con una interrupción de flanco de subida y para medir el nivel de uno de los pulsos del encoder. Las funciones utilizadas para ello son:
  - void PINS_DRV_SetPinIntSel(PORT_Type * const base, uint32_t pin, port_interrupt_config_t intConfig)
  - pins_channel_type_t PINS_DRV_ReadPins(const GPIO_Type * const base)
- FTM Modulus Counter Driver: usado para obtener la frecuencia. Las funciones usadas por este son:
  - uint32_t FTM_DRV_CounterRead(uint32_t instance)
  - status_t FTM_DRV_CounterStop(uint32_t instance)
  - status_t FTM_DRV_CounterStart(uint32_t instance)

Los archivos del SW-C son los siguientes:

- IoHwAb_GetEncoders_SWC.c: definición del void Runnable_GetEncoders_SWC(void) que lee la información dada por los codificadores y la envía al RTE.
- IoHwAb_GetEncoders_SWC.h: archivo de cabecera para IoHwAb_GetEncoders_SWC.c, donde se encuentran todos los defines, includes y prototipos de funciones.
- Rte_GetEncoders_SWC.h: Fichero de cabecera de RTE donde se encuentran los prototipos de los runnables e interfaces.

Las interfaces RTE utilizadas son las siguientes y el comportamiento de las cuatro es el mismo, almacenar el valor de las rpm. El signo indica la dirección, "-" significa hacia atrás y "+" hacia adelante.

- void Get_RPMFLValue(sint16 data)
- void Get_RPMFRValue(sint16 data)
- void Get_RPMBLValue(sint16 data)
- void Get_RPMBRValue(sint16 data)

5.2.1.2.3 SetPWM

El propósito de este SW-C es realizar la lectura del PWM almacenado en los buffers del RTE y, en función de esos valores, la dirección y el valor del PWM se envían a los motores. Las interfaces RTE son las siguientes:

- void Set_PWMFRValue(uint8 *data);
- void Set_PWMFLValue(uint8 *data);
- void Set_PWMBRValue(uint8 *data);
- void Set_PWMBLValue(uint8 *data);

El rango del PWM oscila entre -100 y 100, pero el duty cycle oscila entre 0 y 32768. La conversión necesaria se realiza en este SW-C.

Para enviar esta información a los motores (dirección y duty cycle), se utilizan cuatro funciones definidas en el

fichero IoHwAb.c:

- void Left_Forward_Engine(uint8_t dir, uint16_t dutyCycle)
- void Left_Back_Engine(uint8_t dir, uint16_t dutyCycle)
- void Right_Forward_Engine(uint8_t dir, uint16_t dutyCycle)
- void Right_Back_Engine(uint8_t dir, uint16_t dutyCycle)

5.2.1.3 OS

En Erika3 (el IDE seleccionado para el proyecto) todos los elementos del RTOS, como tareas, alarmas, recursos, son estáticos y están predefinidos en el momento de compilar la aplicación. Para especificar qué objetos existen en una aplicación concreta, Erika3 utiliza el lenguaje OIL, que es un sencillo lenguaje de descripción de texto. El lenguaje OIL forma parte del estándar OSEK/VDX.

OSEK/VDX ofrece un entorno uniforme que permite una utilización eficaz de los recursos para el software de aplicación de las unidades de control de automóviles. El sistema operativo OSEK es un sistema operativo de un solo procesador destinado a las unidades de control integradas distribuidas.

Todos los objetos del sistema OSEK se describen mediante elementos definidos en el fichero .oil: CPU, OS, APPMODE, ISR, RESOURCE, TASK, COUNTER, EVENT, ALARM...

Antes de entrar en profundidad en cada uno de los objetos utilizados, en la siguiente figura se puede apreciar el de tareas del sistema y los runnables ejecutados en cada una de ellas.

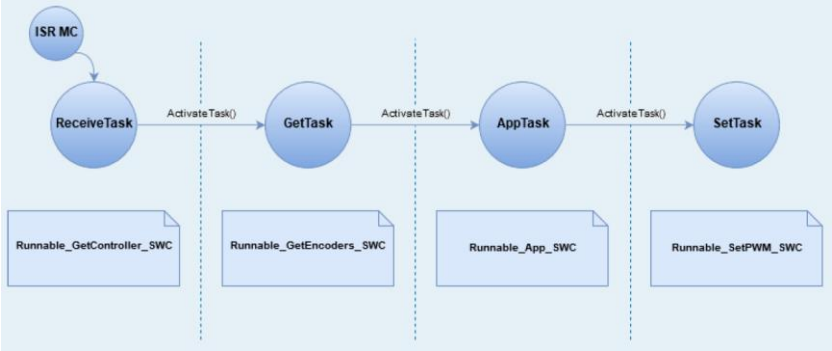

Figura 5-17. Flujo de tareas del OS

Se puede apreciar como se trata de un flujo secuencial, la primera de las tareas se activa a través de una interrupción que es activada cuando el contador del módulo FlexTimer Modulus Counter alcanza su valor máximo, y, tras la ejecución del runnable correspondiente a esa tarea, hace una llamada a la función ActivateTask() pasando como parámetro de entrada la tarea a iniciar. Este procedimiento se repite de manera secuencial hasta llegar a la tarea SetTask donde finaliza el flujo de tareas. Una vez alcanzado de nuevo el valor máximo del contador la tarea inicial es activada y se repite de nuevo el proceso.

A continuación, se describe detalladamente cuáles de los objetos mencionados anteriormente son necesarios para el proyecto y cómo están configurados.

- Dentro de la definición del sistema OSEK, el objeto OS define las propiedades del sistema. Alguno de los parámetros más relevantes son:
    - EE_OPT: ofrece distintas opciones de compilación. A nivel de código se traduce en #defines.
    - CPU_DATA y MODEL: procesador usado.
    - IDLEHOOK: indica si existe una tarea en segundo plano y su nombre.
    - MCU_DATA, MODEL y BOARD_DATA: microcontrolador, modelo y nombre de la placa utilizada.
    - STARTUPHOOK, ERRORHOOK, SHUTDOWNHOOK y PRETASKHOOK, POSTTASKHOOK: indica las rutinas incluidas en el kernel.

Tabla 5-9. Sección OS

OS

OS myOs {
EE_OPT = "OS_EE_APPL_BUILD_DEBUG";
EE_OPT = "OS_EE_BUILD_DEBUG";
CPU_DATA = CORTEX_M {
MODEL = M4;
MULTI_STACK = TRUE;
SYS_STACK_SIZE = 16384;
IDLEHOOK = TRUE {
HOOKNAME = "idle_hook";
};
EXECUTE_FROM_RAM = FALSE;
TRACER = OFF;
};
MCU_DATA = S32K1XX {
MODEL = S32K144;
};
BOARD_DATA = S32K144EVB_Q100;
LIB = S32_SDK {
BOARD = S32K144EVB_Q100;
VERSION = "0.8.6 EAR";
STAND_ALONE = TRUE;
};
STATUS = EXTENDED;
STARTUPHOOK = FALSE;
ERRORHOOK = TRUE;
SHUTDOWNHOOK = FALSE;
PRETASKHOOK = FALSE;
POSTTASKHOOK = FALSE;
USEGETSERVICEID = TRUE;
USEPARAMETERACCESS = TRUE;
USERESSCHEDULER = TRUE;
USEORTI = FALSE;
KERNEL_TYPE = OSEK {
CLASS = ECC2;
RQ = MQ;
};
};

- En el objeto APPDATA se listan todos los ficheros fuente necesarios para la compilación.

Tabla 5-10. Sección APPDATA

APPDATA

```
APPDATA myApp {
APP_SRC = "src/App/App_SWC/App_SWC.c";
APP_SRC = "src/App/App_SWC/App_SWC_data.c";
APP_SRC = "src/App/GetEncoders_SWC/IoHwAb_GetEncoders_SWC.c";
APP_SRC = "src/App/GetController_SWC/IoHwAb_GetController_SWC.c";
APP_SRC = "src/App/SetPWM_SWC/IoHwAb_SetPWM_SWC.c";
APP_SRC = "src/App/OsTask.c";
APP_SRC = "src/Rte/Rte.c";
APP_SRC = "src/Bsw/conf/pin_mux.c";
APP_SRC = "src/Bsw/conf/flexTimer_pwm1.c";
APP_SRC = "src/Bsw/conf/flexTimer_mc1.c";
APP_SRC = "src/Bsw/conf/flexTimer_mc2.c";
APP_SRC = "src/Bsw/conf/flexio_uart_TX.c";
APP_SRC = "src/Bsw/conf/flexio_uart_RX.c";
APP_SRC = "src/Bsw/conf/clockMan1.c";
APP_SRC = "src/Bsw/IOHwAb.c";
APP_SRC = "src/Bsw/conf/Cpu.c";
APP_SRC = "src/Bsw/conf/dmaController1.c";
};
```

- El objeto TASK permite la configuración de las tareas del sistema, asignándoles prioridad (se considera prioridad mínima el valor cero), tipo de scheduling, si tiene algún recurso asociado…

Tabla 5-11. Sección TASK

TASK	**Descripción**

TASK GetTask {
PRIORITY = 5;
STACK = PRIVATE {
SIZE = 4096;
EXTENDED = TRUE;
};
EVENT = EdgeEvent;
EVENT = NoEdgeEvent;
RESOURCE = ResFLEXIOMC;};
	Realiza una llamada al runnable definido en el fichero IoHwAb_GetEncoders_SWC.c: **Runnable_GetEncoders_SWC**. Los parámetros EVENT y RESOURCE indica qué eventos o recursos están asocidos a esta tarea.

TASK AppTask { PRIORITY = 5; };	Realiza una llamada al runnable definido en el fichero App_SWC.c: **Runnable_App_SWC**.
TASK SetTask { PRIORITY = 5; };	Realiza una llamada al runnable definido en el fichero IoHwAb_SetPWM_SWC.c: **Runnable_SetPWM_SWC**.
TASK DebugTask { PRIORITY = 1; SCHEDULE = NON; RESOURCE = ResFLEXIOMC; };	La tarea DebugTask es utilizada para mandar información leída por los encoders a través de la FlexIO UART.
TASK ReceiveTask { PRIORITY = 5; RESOURCE = ResFLEXIOMC; };	Realiza una llamada al runnable definido en el fichero IoHwAb_GetController_SWC.c: **Runnable_GetController_SWC**.

- RESOURCE se utiliza para coordinar los accesos concurrentes de varias tareas con diferentes prioridades a los recursos compartidos.

Tabla 5-12. Sección RESOURCE

RESOURCE	**Descripción**

RESOURCE ResEdge {
RESOURCEPROPERTY = INTERNAL;
};
	Se utiliza para asegurar la completa ejecución de las interrupciones que lo contienen.

RESOURCE ResFLEXIOMC {
RESOURCEPROPERTY = INTERNAL;
};
	Se utiliza para independizar las interrupciones asociadas a los dos contadores definidos.

- El objeto ISR se utiliza para especificar las interrupciones del sistema.

Tabla 5-13. Sección ISR

ISR	**Descripción**

ISR FLEXIOMC2_ISR2 {
PRIORITY = 2;
CATEGORY = 2;
SOURCE = "FTM3_OVF_RELOAD";
RESOURCE = ResEdge;
RESOURCE = ResFLEXIOMC;
};
	Esta interrupción se ejecuta cuando el módulo FTM MC 2 alcanza su valor máximo. Esto significa que los motores están parados.

ISR FLEXIOMC_ISR2 {
PRIORITY = 2;
CATEGORY = 2;
SOURCE = "FTM2_OVF_RELOAD";
RESOURCE = ResEdge;
RESOURCE = ResFLEXIOMC;
};
	Interrupción que se ejecuta cuando el módulo FTM MC 1 alcanza su valor máximo y se utiliza para activar la primera de las tareas del del proyecto.

ISR EdgeDetectionFR_ISR { CATEGORY = 2; SOURCE = "PORTB"; PRIORITY = 2; RESOURCE = ResEdge; };	Estas cuatro interrupciones tienen como objetivo la detección de un flanco de subida de un encoder.
ISR EdgeDetectionBL_ISR { CATEGORY = 2; SOURCE = "PORTD"; PRIORITY = 2; RESOURCE = ResEdge; };	
ISR EdgeDetectionBR_ISR { CATEGORY = 2; SOURCE = "PORTE"; PRIORITY = 2; RESOURCE = ResEdge; };	
ISR EdgeDetectionFL_ISR { CATEGORY = 2; SOURCE = "PORTA"; PRIORITY = 2; RESOURCE = ResEdge; };	

5.2.2 Aplicación móvil

El módulo HC-05 ha sido modificado para trabajar con una tasa de baudios de 115200 baudios utilizando comandos AT. Para ello, es necesario conectar el pin "Key" a nivel alto. Para enviar estos comandos, el módulo HC-05 se ha conectado al PC mediante un convertidor de serie a USB. Además, se ha cambiado el nombre del módulo.

Los comandos AT utilizados han sido los siguientes

- AT+NAME=EVO4WD: Establece el nombre del dispositivo como "EVO4WD"
- AT+UART=115200,1,0: Establece la velocidad de transmisión del dispositivo, el bit de parada y el bit de paridad.

Para diseñar la aplicación se utilizó la herramienta "MIT App Inventor", que cuenta con un entorno de programación visual que permite construir aplicaciones totalmente funcionales para smartphones y tabletas. Su uso es muy intuitivo y permite crear la aplicación mediante el uso de bloques. El diseño de la interfaz gráfica también es bastante sencillo. La aplicación tiene el siguiente aspecto.

Figura 5-18. Interfaz gráfica de la aplicación móvil

La aplicación es bastante fácil de usar, a través de un joystick se envían las coordenadas X e Y al módulo por bluetooth. Estas coordenadas van desde la posición 0 hasta la posición 200, siendo (100,100) la posición de parada, como se muestra en la figura anterior. La aplicación está configurada para enviar datos al módulo cada 100 milisegundos.

En las siguientes figuras se puede apreciar la implementación de la aplicación en el entorno de desarrollo ofrecido por la herramienta.

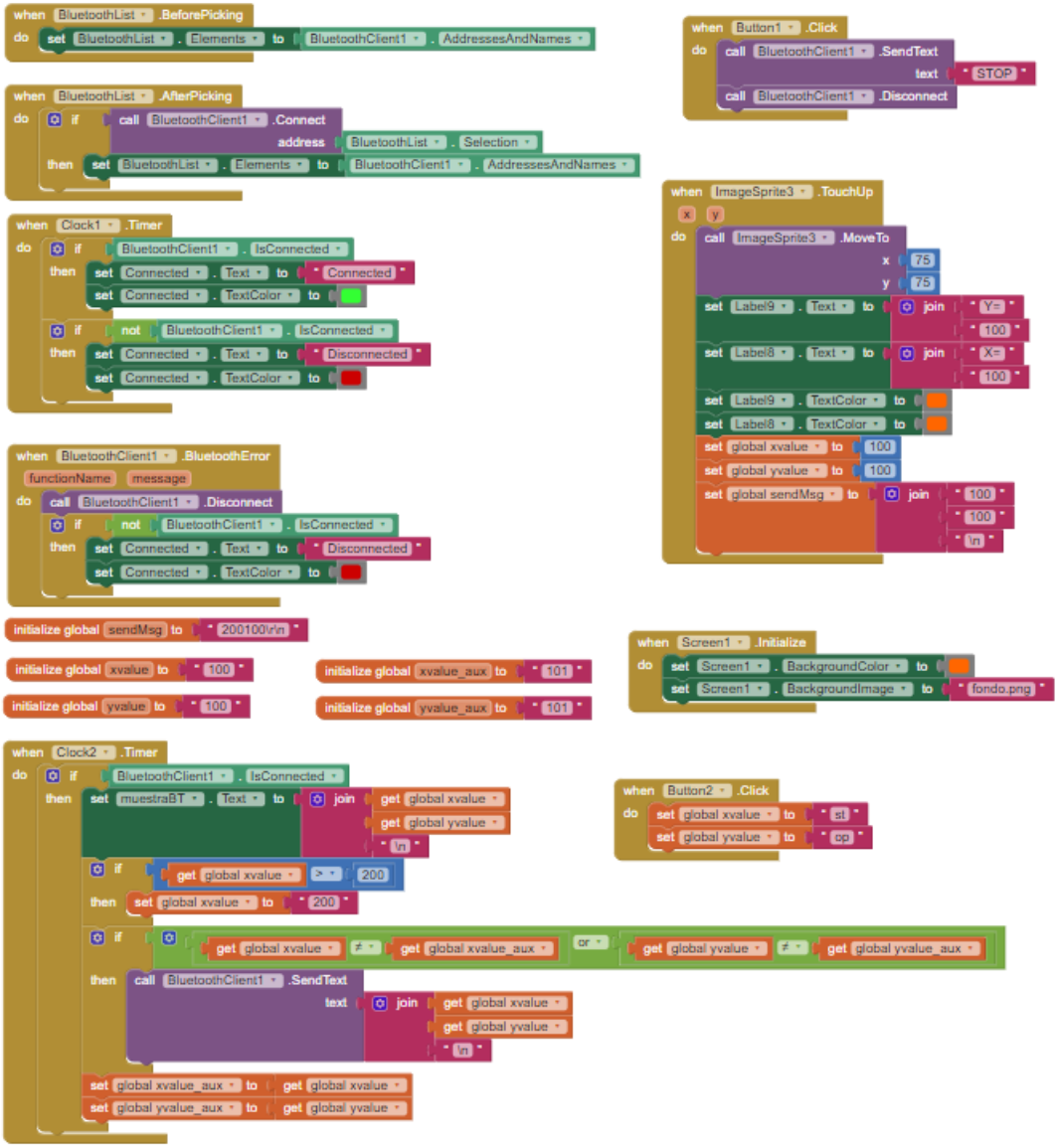

Figura 5-19. Implementación de la aplicación en entorno de desarrollo (I)

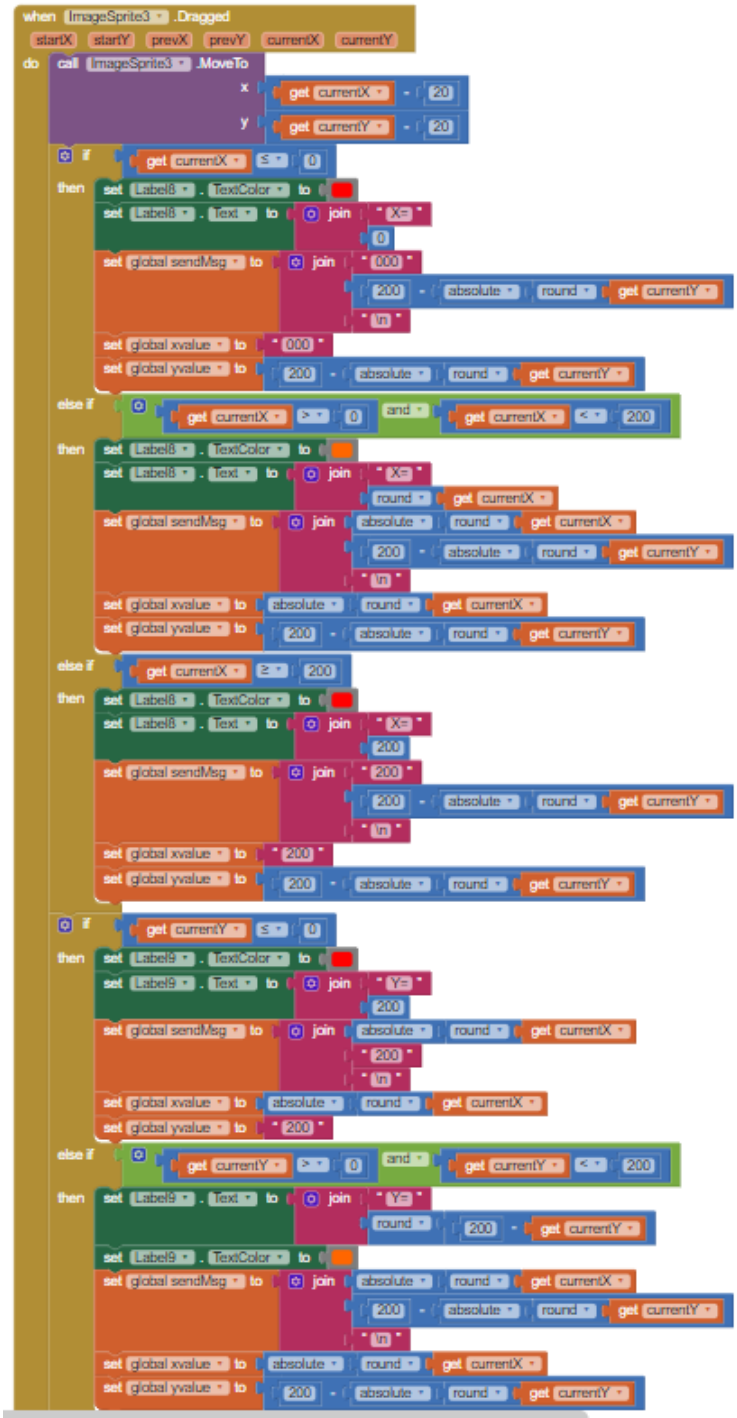

Figura 5-20. Implementación de la aplicación en entorno de desarrollo (II)

6 RESULTADOS

En este capítulo se expondrán tanto las pruebas que se han ido realizando a lo largo del desarrollo del proyecto como los resultados obtenidos. Para finalizar el capítulo se mostrará una tabla resumen donde se listarán los objetivos y requisitos iniciales y si han sido alcanzados o no.

Durante la realización del proyecto se han ido realizando numerosas pruebas para verificar que el comportamiento es el esperado y, en el caso que no fuera así, realizar modificaciones o ajustes para conseguir el objetivo esperado.

- Integración del software. Al haber desarrollado software en distintas capas de la arquitectura AUTOSAR y utilizando herramientas diferentes, una de las tareas que se tenían que realizar durante el desarrollo del proyecto era la integración de los distintos módulos software como el OS, MCAL, ECU Abstraction Layer, RTE APP.
- Integración del hardware. De manera similar a la integración software, todos los elementos hardware también debían ser integrados correctamente.
- Calibración de motores. Los motores, a pesar de ser el mismo modelo, ofrecían un comportamiento ligeramente distinto cada uno de ellos. Por lo que tuvo que realizarse una calibración y análisis del comportamiento de cada uno de ellos.
- Depuración vía UART. Uno de los aspectos más importantes a la hora del desarrollo ha sido contar con una salida UART en la que podían obtenerse parámetros de interés para el sistema y en tiempo real.
- Movimiento del vehículo. Se han realizado distintas pruebas con el vehículo ya montado para evaluar el movimiento de este. Con estas pruebas, por ejemplo, se comprobó que era necesario el cambio de batería de plomo a batería de litio debido al excesivo peso de la primera.
- Schedulling del OS. En cuanto al flujo de tareas del sistema operativo, también se han realizado diferentes tests asignando periodicidades distintas a cada una de las tareas, cambiando su orden de ejecución y linkando cada una de ellas a recursos o interrupciones.

Tras la realización de todas las tareas desarrolladas durante este proyecto, se han alcanzado los siguientes objetivos iniciales:

- Estudio y análisis de AUTOSAR.
- Análisis del mercado.
- Coste reducido.
- Desarrollo de plataforma hardware.
- Desarrollo software.
- Integración de los elementos del sistema. (AUTOSAR)
- Calidad.

En las siguientes tablas se sintetizan todos los requisitos iniciales y si han sido cumplidos o no.

Tabla 6-1. Requisitos funcionales cumplidos

Requisitos funcionales	**Cumplido**
El sistema debe ser capaz de circular hacia delante	Si
El sistema debe ser capaz de circular hacia atrás	Si
El sistema debe ser capaz de circular hacia la derecha	Si
El sistema debe ser capaz de circular hacia la izquierda	Si
El sistema debe ser capaz de controlar la velocidad	Si

Tabla 6-2. Requisitos no funcionales cumplidos

Requisitos no funcionales	**Cumplido**
El sistema se comunicará de manera inalámbrica con un dispositivo de control	Si
El sistema debe ser controlable a través de un dispositivo externo	Si
El sistema debe contar con una interfaz de depuración	Si
El sistema debe estar alimentado por baterías	Si
Uso de una plataforma de testeo (HIL)	Si
Desarrollo basado en simulaciones	Si
Uso de herramientas de generación de código para funciones de alto nivel	Si
El sistema debe contar con cuatro motores DC	Si
El sistema debe tener un tamaño adecuado para soportar distintos dispositivos como sensores, baterías, placa de evaluación	Si

7 CONCLUSIONES Y LÍNEAS FUTURAS

En este último capítulo se realizará un análisis de las conclusiones y posibles líneas de trabajo futuras que han surgido durante la realización del proyecto. Se detallarán aspectos tales como el aprendizaje, experiencias, dificultades que han surgido durante el transcurso del proyecto y qué se podría mejorar o modificar para conseguir con ello un enriquecimiento del proyecto.

7.1 Conclusiones

El objetivo principal de este proyecto era el desarrollo de una plataforma hardware con una arquitectura software basada en AUTOSAR y puede decirse que ha sido alcanzado en todos los aspectos. En primer lugar, se ha conseguido una arquitectura AUTOSAR sin coste, además de un toolchain completo que permite la configuración y desarrollo de esta arquitectura en todas sus capas, desde el Basic Software hasta la Capa de Aplicación.

La tarea de conseguir una arquitectura AUTOSAR a bajo coste no fue nada fácil ya que el mercado en el que se encuentra el proyecto es un mercado cerrado y limitado, donde las herramientas o software necesario era ofrecido por pocas empresas y a un coste muy elevado.

También han surgido dificultades respecto a la utilización del software seleccionado en el inicio del proyecto ya que algunos de ellos no ofrecían demasiado soporte, como es el caso de Erika, donde surgieron algunos problemas durante el desarrollo cuya solución no fue inmediata debido a esta falta de soporte. Aún así, se ha conseguido superar estos impedimentos surgidos en el desarrollo.

Finalmente, se puede decir que a pesar de las complicaciones que hayan podido surgir durante el proyecto, se han alcanzado los requisitos fijados al inicio a todos los niveles.

A nivel personal, la realización de este proyecto ha sido tremendamente positiva ya que me ha dado la oportunidad de adquirir nuevos conocimientos y afianzar algunos que ya había adquirido durante mis estudios.

Además, el hecho de tratarse de un proyecto para una empresa me ha permitido desarrollarme en un marco industrial, con herramientas de producción reales, integración continua…

Por último, este proyecto también me ha permitido comenzar mi camino en el mundo laboral, inicialmente para el GIE participando en una colaboración con EVO en la que se desarrolló el presente proyecto y, una vez finalizado, incorporándome como empleado a EVO donde tengo la suerte de seguir trabajando.

7.2 Líneas futuras

Si bien con el desarrollo del presente proyecto, como ya se ha comentado en el capítulo anterior, se han cumplido los principales objetivos marcados inicialmente, existen diversos puntos que pueden aumentar en gran medida la calidad y el alcance del mismo.

A continuación, se listarán algunos de los más destacados:

- **Modificar el flujo de tareas**. El OS está definido para que sus tareas se ejecuten siguiendo un flujo secuencial cada cierto tiempo. Una posible mejora sería que la ejecución de cada una de esas tareas se realizara de manera independiente con distintas periodicidades.
- **Incluir un depurador J-Link**. La inclusión de un depurador J-Link haría mucho más eficiente el trabajo y daría mucha más fluidez al desarrollo permitiendo una depuración más completa, sin depender de la UART para depuración.

- **Obtener un giro eficiente del vehículo**. Actualmente, debido a que ninguno de los motores es exactamente igual y el peso del vehículo, entre otros, el giro del vehículo no es todo lo eficiente que podría ser. Por lo que se podría realizar un estudio y una calibración más exhaustiva a cada uno de los motores para mejorar este aspecto.
- **Reemplazar la HAL**. Como ya se ha comentado, por temas de coste, la solución está basada en la HAL ofrecida por NXP. La sustitución de esta por una MCAL sería una posible línea de trabajo futura.
- **Documentar el código**. Debido principalmente a que el proyecto terminó antes de lo previsto, la documentación sobre el código no se realizó utilizando herramientas de generación de documentación como Doxygen, tal y como se pretendía inicialmente.
- **Añadir más sensores al vehículo**. La plataforma actual permite la inclusión de multitud de sensores nuevos que aporten nuevas funcionalidades.
- **Integrar módulos básicos del stack de AUTOSAR**. El stack de AUTOSAR contiene multitud de módulos de lo cuales, algunos de ellos son básicos en cualquier desarrollo AUTOSAR. Por ejemplo, podrían añadirse en un futuro módulos relacionados con diagnósticos.
- **Diseño e implementación de una PCB**. El actual montaje hardware podría verse mejorado con el diseño e implementación de una pequeña PCB.

REFERENCIAS

[1] Daehyun Kum, G. Park, Seonghun Lee and Wooyoung Jung, "AUTOSAR migration from existing automotive software," 2008 International Conference on Control, Automation and Systems, 2008, pp. 558-562, doi: 10.1109/ICCAS.2008.4694565.

[2] H. Bo, D. Hui, W. Dafang and Z. Guifan, "Basic Concepts on AUTOSAR Development," 2010 International Conference on Intelligent Computation Technology and Automation, 2010, pp. 871-873, doi: 10.1109/ICICTA.2010.571.

[3] Z. Wang and D. Yin, "Design and Implementation of Vehicle Control System for Pure Electric Vehicle Based on AUTOSAR Standard," 2019 22nd International Conference on Electrical Machines and Systems (ICEMS), 2019, pp. 1-5, doi: 10.1109/ICEMS.2019.8921856.

[4] M. Soltani and E. Knauss, "Challenges of Requirements Engineering in AUTOSAR ecosystems," 2015 IEEE 23rd International Requirements Engineering Conference (RE), 2015, pp. 294-295, doi: 10.1109/RE.2015.7320445.

[5] S. Dersten, J. Axelsson and J. Froberg, "Effect Analysis of the Introduction of AUTOSAR: A Systematic Literature Review," 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011, pp. 239-246, doi: 10.1109/SEAA.2011.44.

[6] T. Hermans, P. Ramaekers, J. Denil, P. D. Meulenaere and J. Anthonis, "Incorporation of AUTOSAR in an Embedded Systems Development Process: A Case Study," 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011, pp. 247-250, doi: 10.1109/SEAA.2011.45.

[7] V. T. Popović, M. Vulić, A. Davidović and I. Kaštelan, "Modeling and Development of AUTOSAR Software Components," 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT), 2019, pp. 313-316, doi: 10.1109/ISCE.2019.8901035.

[8] G. Park, D. Ku, S. Lee, W. -J. Won and W. Jung, "Test methods of the AUTOSAR application software components," 2009 ICCAS-SICE, 2009, pp. 2601-2606.

[9] NXP® Semiconductors Official Site | Home. (2021). NXP. https://www.nxp.com/

[10] Technologies Ag, I. (2021). Semiconductor & System Solutions - Infineon Technologies. Copyright Infineon Technologies AG - all rights reserved. https://www.infineon.com/

[11] Vector Group | Vector. (2021). Vector: Software + Services for Automotive Engineering. https://www.vector.com/es/es/

[12] A.D. (2021). AUTOSAR. AUTOSAR. https://www.autosar.org/

[13] Quadrature Encoder Basics Part 1: Theory. (2018, 2 octubre). Quadrature Encoder. https://www.rs-online.com/designspark/quadrature-encoder-basics-part-1-theory

[14] MIT App Inventor | Explore MIT App Inventor. (2021). MIT App Inventor. https://appinventor.mit.edu/

[15] Erika Enterprise RTOS v3. (2021). Erika Enterprise RTOS v3. https://erika-enterprise.com/

[16] S32 Design Studio IDE. (2021). NXP Semiconductors. https://www.nxp.com/design/software/development-software/s32-design-studio-ide:S32-DESIGN-STUDIO-IDE

[17] A. Zahir, "OIL-OSEK implementation language," IEE Seminar on OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), 1998, pp. 8/1-8/3, doi: 10.1049/ic:19981079.