Centre universitari adscrit a la

TecnoCampus

upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO Y PUESTA EN MARCHA DE UNA BANCADA PARA EL DESARROLLO DE SOLUCIONES INDUSTRIALES INTEGRALES AVANZADAS CON MICROCONTROLADORES ARM CORTEX-M4

Memoria

Hamza Choulli Samadi

PONENT: Dr. Julián Horrillo Tello

JUNIO 2024

TecnoCampus Mataró-Maresme

Resum

La cerca d'eficiència i competitivitat impulsa l'adopció de tecnologies avançades com la IA en processos industrials per a optimitzar producció, reduir costos i millorar la qualitat. En resposta a aquestes necessitats, es proposa dissenyar i desenvolupar una bancada per a solucions industrials avançades. Aquest projecte s'estructura en cinc fases: anàlisi de tecnologies, selecció de components, disseny de la bancada, desenvolupament de la bancada i creació d'una aplicació específica. Cada fase és crucial per a crear una bancada que faciliti la realització del desenvolupament i proves de diferents aplicacions.

Resumen

La búsqueda de eficiencia y competitividad impulsa la adopción de tecnologías avanzadas como la IA en procesos industriales para optimizar producción, reducir costos y mejorar la calidad. En respuesta a estas necesidades, se propone diseñar y desarrollar una bancada para soluciones industriales avanzadas. Este proyecto se estructura en cinco fases: análisis de tecnologías, selección de componentes, diseño de la bancada, desarrollo de la bancada y creación de una aplicación específica. Cada fase es crucial para crear una bancada que facilite la realización del desarrollo y pruebas de diferentes aplicaciones.

Abstract

The pursuit of efficiency and competitiveness drives the adoption of advanced technologies such as AI in industrial processes to optimize production, reduce costs, and improve quality. In response to these needs, it is proposed to design and develop a testbed for advanced industrial solutions. This project is structured in five phases: technology analysis, component selection, testbed design, testbed development, and creation of a specific application. Each phase is crucial to create a testbed that facilitates the development and testing of various applications.

Tabla de contenidos.

Índice de figuras. III

Índice de tablas. VII

Glosario de términos. IX

1 Objetivos del proyecto. 11

1.1 Propósito. 11

1.2 Finalidad. 11

1.3 Objeto. 11

1.4 Alcance. 11

1.5 Contexto en las líneas de investigación y transferencia de conocimiento del Tecnocampus 11

2 Introducción. 13

3 Marco teórico. 15

3.1 Marco conceptual. 15

3.2 Marco contextual. 24

4 Objetivo de detalle y especificaciones técnicas. 31

5 Alcance de detalle. 33

6 Solución tecnológica. 35

6.1 Diseño de la bancada. 35

7 Desarrollo de la aplicación. 65

7.1 Instalación CCS y TI-RTOS. 65

7.2 Creación de un proyecto con el SO. 69

7.3 Funcionamiento del SO. 71

8 Impacto medioambiental. 93

9 Perspectiva de género. 95

10 Planificación. 97

10.1 Diagrama de Gantt 98

10.2 Análisis de riesgo. 99

10.3 Plan de contingencia. 99

11 Conclusiones. 101

12 Bibliografía. 103

Índice de figuras.

Fig 1. Funcionalidades principales de los microprocesadores [Fuente: Google.com] ... 18
Fig 2. Arquitectura de los microcontroladores [Fuente: EP] ... 20
Fig 3. Arquitectura de los DSP [Fuente: EP] ... 22
Fig 4. Ventana Getting Started. [Font: EP] ... 36
Fig 5. Ventana CCS. [Fuente: EP]. ... 37
Fig 6. Ventana Resource Explorer. [Fuente: EP] ... 38
Fig 7. Ventana Resource Explorer con ejemplos de la placa TM4C1294XL. [Fuente: EP] ... 39
Fig 8. Ventana Project Explorer y explorador de archivos Windows. [Fuente: [11]] ... 40
Fig 9. Ventana Project Explorer con un proyecto de la placa TM4C1294XL. [Fuente: EP] ... 41
Fig 10. Ventana Properties de CCS. [Fuente: EP] ... 41
Fig 11. Ventana Properties pestaña Builder. [Fuente: EP] ... 42
Fig 12. Ventana Properties pestaña Variables. [Fuente: EP] ... 43
Fig 13. Conexión/configuración hardware de los componentes. [Fuente: EP] ... 44
Fig 14. CCS debugger. [Fuente:[11]] ... 44
Fig 15. Ventana de depuración del CSS. [Fuente: [11]] ... 45
Fig 16. Ventana Debug. [Fuente:EP] ... 45
Fig 17. Ventana Variables. [Fuente: EP] ... 46
Fig 18. Ventana Expressions. [Fuente: EP] ... 46
Fig 19. Ventana Registers. [Fuente: EP] ... 46
Fig 20. Ventana del sistema de depuración. [Fuente: [11]] ... 47
Fig 21. Barra de herramientas del sistema de depuración. [Fuente: EP] ... 48
Fig 22. Ventana de la herramienta XGCONF. [Fuente: EP] ... 50
Fig 23. Orden de prioridad de los threads. [Fuente: [13]]. ... 51
Fig 24. Comportamiento de los diferentes threads. [Fuente: [13]] ... 53
Fig 25. Microcontrolador TM4C1294NCPDT. [Fuente: [12]] ... 55
Fig 26. Diagrama de bloques de alto nivel del microcontrolador. [Fuente: [12]] ... 57
Fig 27. Diagrama de bloques de la memoria interna. [Fuente: [12]] ... 58
Fig 28. Diagrama de bloques UART. [Fuente: [12]] ... 59
Fig 29. Diagrama de bloques I^2C. [Fuente: [12]] ... 59
Fig 30. Diagrama de bloques de los módulos de PWM. [Fuente: [12]] ... 60

Fig 31. Diagrama de bloques del generador de PWM. [Fuente: [12]] 60
Fig 32. Diagrama de bloques del UART. [Fuente: [12]] 61
Fig 33. Diagrama de bloques del módulo ADC. [Fuente: [12]] 61
Fig 34. Diagrama de bloques del procesador. [Fuente: [12]] 63
Fig 35. Ventana de descarga de CCS. [Fuente: EP]..... 65
Fig 36. Archivo ejecutable de CCS. [Fuente: EP] 65
Fig 37. Pasos a seguir para la instalación del entorno de desarrollo. [Fuente: EP]..... 66
Fig 38. Ventana para seleccionar el entorno de trabajo. [Fuente: EP]..... 67
Fig 39. Ventana Getting Started. [Fuente: EP] 67
Fig 40. Ventana para la descarga del TI-RTOS. [Fuente: EP] 68
Fig 41. Pasos a seguir para la instalación del SO. [Fuente: EP]..... 68
Fig 42. Ventana Getting Started seleccionando la opción New Project. [Fuente: EP] ... 69
Fig 43. Parte superior de la ventana New CCS Project. [Fuente: EP]..... 69
Fig 44. Parte inferior de la ventana New CCS Project. [Fuente: EP]..... 70
Fig 45. Ventana Project Explorer con el proyecto creado. [Fuente: EP] 70
Fig 46. Explicación de la cabecera del ejemplo TI-RTOS. [Fuente: EP] 71
Fig 47. Explicación de la Task que se ejecuta en el ejemplo TI-RTOS. [Fuente: EP] ... 71
Fig 48. Explicación del programa principal que se ejecuta en el ejemplo TI-RTOS. [Fuente: EP]..... 72
Fig 49. Opciones de copiar y pegar en la ventana Project Explorer. [Fuente: EP]..... 72
Fig 50. Pasos a seguir para abrir el kernel del proyecto. [Fuente: EP]..... 73
Fig 51. Ventana del kernel de TI-RTOS. [Fuente: EP] 73
Fig 52. Contenido del main sin las deficiniones de las variables de la Task. [Fuente: EP] 74
Fig 53. Kernel Task, opción: Module. [Fuente: EP]..... 74
Fig 54. Kernel del Task, opción: Advanced. [Fuente: EP] 75
Fig 55. Kernel Task, opción: Instance. [Fuente: EP]..... 75
Fig 56. Kernel Hwi, opción: Module. [Fuente: EP] 76
Fig 57.Kernel Hwi, opción: Advanced. [Fuente: EP]..... 76
Fig 58. Kernel Hwi, opción: Instance. [Fuente: EP] 77
Fig 59. Kernel Swi, opción: Module. [Fuente: EP]..... 77
Fig 60. Kernel Swi, opción: Instance. [Fuente: EP] 77
Fig 61. Cabecera del archivo empty.c. [Fuente: EP] 78
Fig 62. Modificación del thread heartBeatFxn. [Fuente: EP]..... 78

Fig 63. Thread Swi y Hwi. [Fuente: EP] 78
Fig 64. Modificaciones de la función main. [Fuente: EP] 79
Fig 65. Kernel Swi/Hwi, opción: Instance. [Fuente: EP] 80
Fig 66. Kernel Clock, opción: Module. [Fuente: EP] 80
Fig 67. Kernel Clock, opción: Instance. [Fuente: EP] 80
Fig 68. Función del thread clockFxn0. [Fuente: EP] 81
Fig 69. Creación nueva Task. [Fuente: EP] 81
Fig 70. Kernel semáforo, opción: Module. [Fuente: EP] 82
Fig 71. Kernel semáforo, opción: Instance. [Fuente: EP] 82
Fig 72. Ventana Available Products, opción: LoggingSetup. [Fuente: EP] 82
Fig 73. Kernel LoggingSetup, opción: Logging. [Fuente: EP] 83
Fig 74. Código del archivo empty.c. [Fuente: EP] 83
Fig 75. Ventana Available Products, opción: Event. [Fuente: EP] 84
Fig 76. Kernel Event, opción: Instance. [Fuente: EP] 84
Fig 77. Definición de los diferentes eventos del programa y contenido de la función main. [Fuente:EP] 84
Fig 78. Contenido de la función: heartBeatFxn. [Fuente: EP] 85
Fig 79. Contenido de la función: taskFxn1. [Fuente: EP] 85
Fig 80. Mensajes de la consola a la hora de ejecutar el programa. [Fuente: EP] 85
Fig 81. Available Products y Outline del módulo GateAll. [Fuente: EP] 86
Fig 82. Encabezado del archivo empty.c. [Fuente: EP] 86
Fig 83. Definición de la variable estructura y el nombre del Gate. [Fuente: EP] 86
Fig 84. Funciones de ledOn y ledOff. [Fuente: EP] 87
Fig 85. Función main del archivo empty.c. [Fuente: EP] 87
Fig 86. Contenido de las funciones de las Tasks threads. [Fuente: EP] 87
Fig 87. Consola de la aplicación con la herramienta Gate. [Fuente: EP] 88
Fig 88. Mailbox kernel, opción: Module. [Fuente: EP] 88
Fig 89. Mailbox kernel, opción: Instance. [Fuente: EP] 89
Fig 90. Estructura del mensaje. [Fuente: EP] 89
Fig 91. Contenido de la función heartBeatFxn. [Fuente: EP] 89
Fig 92. Contenido de la función taskFxn1. [Fuente: EP] 90
Fig 93. Consola de la aplicación con la herramienta Mailbox. [Fuente: EP] 90
Fig 94. Queue kernel, opción: Instance. [Fuente: EP] 90
Fig 95. Estructura de la Queue. [Fuente: EP] 91

Fig 96. Contenido de la función heartBeatFxn. [Fuente: EP] 91
Fig 97. Contenido de la función taskFxn1. [Fuente: EP] 91
Fig 98. Consola de la aplicación con la herramienta Mailbox. [Fuente: EP] 92
Fig 99. Diagrama de Gantt. [Fuente: EP] 98

Índice de tablas.

Tabla 1. Características técnicas de microcontroladores disponibles en el mercado. [Fuente: EP] 28
Tabla 2. Características técnicas de DSP disponibles en el mercado. [Fuente: EP] 30
Tabla 3. Características del microcontrolador TMC1294NCPDT. [Fuente: [12]] 56
Tabla 4. Características eléctricas del microcontrolador. [Fuente: [12]] 62
Tabla 5. Tareas y distribución de horas del proyecto. [Fuente: EP] 97

Glosario de términos.

ADC	Analog-to-Digital Convertor (Conversor de analógico a digital)
ADC	Digital-to-Analog Convertor (Conversor de digital a analógico)
API	Application Programming Interface
ARM	Advanced RISC Machines
C/C++	Lenguaje de programación de sistemas
CCS	Code Composer Studio
DMA	Direct Memory Access (Acceso Directo a Memoria)
DSP	Digital Signal Processor (Procesador de Señales Digitales)
E/S	Entradas/Salidas
EP	Elaboración Propia
GPIO	General-Propuse Input/Output (E/S de propósito general)
Hwi	Hardware interrupt (Interrupción por hardware)
IA	Inteligencia Artificial
IOS	Operating System de Apple Inc.
IoT	Internet of Things (Internet de las Cosas)
LCD	Liquid Crystal Display (Pantalla de Cristal Líquido)
MAC	Unidades de Multiplicación-Acumulación
OS/SO	Operating System/Sistema Operativo
PIB	Producto Interior Bruto
PWM	Pulse Width Modulator
RAM	Random Access Memory (Memoria de Acceso Aleatorio)
Swi	Software interrupt (Interrupción por software)
TI	Texas Instruments

1 Objetivos del proyecto.

1.1 Propósito.

El objetivo del proyecto consiste en diseñar e implementar una bancada que facilite la realización del desarrollo y pruebas de diversas aplicaciones. Esta bancada proporcionará un entorno controlado y seguro para la evaluación de sistemas, dispositivos y procesos.

1.2 Finalidad.

La finalidad del proyecto es proveer una plataforma robusta y adaptable que facilite la investigación, prototipado y validación de diferentes aplicaciones para el desarrollo de diferentes soluciones tecnológicas.

1.3 Objeto.

El objeto del proyecto consiste en el diseño y puesta en marcha de una bancada para el desarrollo de soluciones industriales integrales avanzadas. Se aborda desde la selección de los componentes *hardware* y *software* hasta la puesta en marcha de la bancada. Esta solución se desarrolla a través de cinco etapas que abarcan desde el análisis tecnológico disponible en el mercado hasta el desarrollo de la bancada.

1.4 Alcance.

El alcance del proyecto se centra en las diferentes etapas para el diseño y puesta en marcha de la bancada para el desarrollo de soluciones industriales avanzadas. Esto incluye la selección del *hardware* y *software*, el diseño y desarrollo de la bancada, así como el desarrollo de una aplicación específica. Se delimitan las exclusiones, como el desarrollo y la implementación de la IA y la implementación de la aplicación en empresas manufactureras.

1.5 Contexto en las líneas de investigación y transferencia de conocimiento del Tecnocampus

El trabajo de final de grado realizado está relacionado con la actividad del grupo de investigación en *"*Fabricació *Intel·ligent i Innovació Industrial (FI4.0)"*, que orienta su

actividad al estudio de los nuevos modelos industriales, resultantes del proceso de transformación digital de la empresa y actividad económica. Las líneas de investigación del grupo son: innovación y desarrollo territorial, sostenibilidad, analítica de datos inteligentes, robótica avanzada, i eficiencia energética y fiabilidad.

Este trabajo de final de grado se realiza dentro de la línea de análisis de datos inteligentes, permitiendo, a través del desarrollo de aplicaciones de digitalización la incorporación de la inteligencia artificial a la gestión de la función de la función de producción en el contexto de la industria 4.0.

2 Introducción.

La búsqueda constante de eficiencia operativa y competitividad impulsa la adopción de tecnologías avanzadas como la IA. La integración de la IA en los procesos industriales se ha convertido en un elemento crucial para optimizar la producción, reducir costos y mejorar la calidad de los productos. En respuesta a esta necesidad, surge el proyecto de diseño y puesta en marcha de una bancada para el desarrollo de soluciones industriales integrales avanzadas. Este proyecto se basa en la creación de una plataforma versátil que permita diseñar, implementar y desplegar aplicaciones industriales innovadoras y eficientes, a través de un microcontrolador y la IA.

El desarrollo de este proyecto se estructura en cinco fases:

1. Análisis de tecnologías disponibles: se llevará a cabo un análisis de las diferentes tecnologías disponibles en el mercado, incluyendo sensores, tecnologías de procesamiento y actuadores. El objetivo es identificar las opciones más adecuadas para la integración en la bancada.
2. Selección de componentes: se procede a la selección de los elementos más adecuados, evaluando su compatibilidad entre ellos y su capacidad para soportar aplicaciones orientadas al entorno industrial.
3. Diseño de la bancada: se realizará el diseño preliminar de la bancada, definiendo la distribución de los elementos y estableciendo las conexiones necesarias para su funcionamiento óptimo.
4. Desarrollo de la bancada: se procede al desarrollo de la bancada, incluyendo la configuración de los elementos desde el microcontrolador hasta los algoritmos de IA. Esta etapa culminará con la puesta en marcha de la bancada y la posibilidad de desarrollar aplicaciones industriales.
5. Desarrollo de una aplicación: esta fase, se enfoca en la creación de una aplicación específica que aproveche la infraestructura y capacidades de la bancada diseñada.

Cada una de estas fases contribuirá de manera significativa al éxito del proyecto, permitiendo la creación de una herramienta innovadora y eficiente para el desarrollo de soluciones industriales avanzadas. La cuidadosa selección de componentes, combinada con el desarrollo de algoritmos basados en IA, posicionará a las empresas en la vanguardia de la eficiencia operativa y la competitividad en el sector industrial.

3 Marco teórico.

3.1 Marco conceptual.

El proyecto se enmarca en un contexto donde la industria busca adaptarse a los avances tecnológicos para mejorar la eficiencia operativa y aumentando la competitividad. La integración de la IA en los procesos industriales se presenta como una oportunidad para optimizar la producción, reducción de costes y mejora en la calidad de los productos.

El diseño y puesta en marcha de la bancada se basa en la selección de componentes, *hardware* y *software* que cumplan con los requisitos de rendimiento, fiabilidad y compatibilidad necesarios para el funcionamiento en entornos industriales. El proyecto se desarrolla en un contexto de rápida evolución tecnológica, donde la automatización y la IA están transformando la forma en que se diseñan, producen y gestionan la gran mayoría de bienes y servicios.

La implementación no solo responde a la necesidad de preparar las plantas industriales para el futuro, sino también a la demanda creciente de soluciones que mejoren la productividad y la eficiencia en un mercado cada vez más exigente y competitivo.

3.1.1 IA.

A lo largo de las últimas décadas han surgido varias definiciones de IA, ninguna definitiva. John McCarthy en el 2007 ofreció la siguiente definición en una entrevista, “Es la ciencia y la ingeniería para crear máquinas inteligentes, especialmente programas informáticos inteligentes. Está relacionada con la tarea similar de utilizar ordenadores para comprender la inteligencia humana, pero la IA no tiene por qué limitarse a métodos que sean biológicamente observables”. En dicha entrevista nos ofrece una definición de inteligencia, “La inteligencia es la parte computacional de la capacidad para alcanzar objetivos en el mundo. Las personas, muchos animales y algunas máquinas presentan distintos tipos y grados de inteligencia”.

Sin embargo, décadas antes de esta definición, el nacimiento de la conversación sobre IA se destacó en la obra de Alan Turing, "Maquinaria computacional e inteligencia", publicada en 1950. En este artículo, Turing, a menudo llamado "padre de la informática", hace la siguiente pregunta: "¿Las máquinas pueden pensar?" A partir de ahí, propone una prueba, la ahora conocida como la "Prueba de Turing", en la que un interrogador humano intentaría distinguir entre la respuesta de un ordenador y la de un texto humano. Aunque

esta prueba ha sido objeto de mucho escrutinio desde su publicación, sigue siendo una parte importante de la historia de la IA, así como un concepto actual dentro de la filosofía, ya que utiliza ideas en torno a la lingüística.

Stuart Russell y Peter Norvig publicaron después IA: Un enfoque moderno, que se convirtió en uno de los principales libros de texto en el estudio de la IA. En él, profundizan en cuatro posibles objetivos o definiciones de la IA, que diferencian los sistemas informáticos en función de la racionalidad y el pensamiento frente a la actuación:

Enfoque humano:

- Sistemas que piensan como humanos: se enfoca en la emulación de la inteligencia humana, tanto en términos de comportamiento como de pensamiento.
- Sistemas que actúan como humanos: se enfocan en la emulación de la inteligencia humana, pero en términos de comportamiento.

Enfoque ideal:

- Sistemas que piensan racionalmente: se enfocan en la resolución de problemas de manera lógica y racional.
- Sistemas que actúan racionalmente: se enfocan en la toma de decisiones y la acción en el mundo.

La definición de Alan Turing habría caído en la categoría de "sistemas que actúan como humanos".

A lo largo de los años, la IA ha pasado por muchos ciclos de auge, pero incluso para los escépticos, el lanzamiento de ChatGPT de OpenAI parece marcar un punto de inflexión. La última vez que la IA generativa tuvo tanta atención, los avances se produjeron en la visión por ordenador, pero ahora el salto adelante está en el procesamiento de lenguajes naturales.

Las aplicaciones de esta tecnología crecen día a día, y apenas se empieza a explorar sus posibilidades.

Tipos de IA.

Una posible forma de diferenciar a las IA, según su potencia, es la siguiente:

- IA débil: se define como la IA racional que se centra típicamente en una tarea muy definida o estrecha, este tipo de IA es limitado, pero ahorra mucho tiempo a la especie humana, ya que puede automatizar tareas como por ejemplo Siri, de IOS.
- IA fuerte: es una forma teórica de IA utilizada para describir una mentalidad determinada de desarrollo de IA. Es el tipo de inteligencia que iguala o excede la inteligencia humana promedio, es decir, la inteligencia de una máquina que puede realizar con éxito cualquier tarea intelectual, siendo consciente de sí misma con la capacidad de resolver problemas, aprender y planificar el futuro.

La diferenciación entre las dos es la capacidad y amplitud, como aquellas especializadas en tareas específicas o que aspiran a alcanzar un nivel de inteligencia comparable o superior a la humana.

Machine Learning.

El *Machine Learning* o aprendizaje automático, subcampo de la IA que se centra en el desarrollo de algoritmos que permiten a las maquinas aprender patrones y tomar decisiones sin ser explícitamente programados. Hay varios tipos de aprendizaje automático:

- Aprendizaje supervisado: los algoritmos trabajan con datos etiquetados, intentando encontrar una función que, dadas las variables de entrada, le asigne la etiqueta de salida adecuada. El algoritmo se entrena con un base de datos, aprendiendo a asignar la etiqueta de salida, aprendiendo a predice el valor de salida.
- Aprendizaje no supervisado: tiene lugar cuando no se dispone de datos etiquetados para el entrenamiento. Sólo se conocen los datos de entrada, pero no existen valores de salida deseados, esto nos sirve para describir la estructura de datos, simplificando el análisis de los datos de entrada.

El aprendizaje automático depende en gran medida de la selección adecuada de algoritmos, la optimización de parámetro y de la calidad y cantidad de datos que se utilizan para entrenar a la máquina.

Deep Learning y redes neuronales.

El *Deep Learning* o aprendizaje profundo, subcampo del aprendizaje automático donde todos los algoritmos se inspira en el funcionamiento de las neuronas o las redes neuronales.

Las redes neuronales generalmente se dividen en capas. La primera **es** donde se introducen los datos en la red y se conoce como capa de entrada. El conjunto de nodos que proporciona la salida de la red se llama capa de salida. Y, el conjunto de nodos entre estas dos capas se denomina capas ocultas.

3.1.2 Tecnologías de procesamiento.

Las tecnologías de procesamiento son el pilar principal para el desarrollo de nuevas tecnologías, permitiendo el análisis, la interpretación y generación de nuevos conocimientos o nuevas soluciones. Permitiendo impulsar tecnologías como la IA, el IoT, el Edge Computing y la nube.

Microprocesadores.

Los microprocesadores están diseñados para realizar una amplia variedad de tareas y aplicaciones. Son los encargados de ejecutar el SO de cualquier aplicación, pero solo ejecuta instrucciones en lenguaje máquina realizando operaciones aritméticas y de lógica simple. Un microprocesador incorpora tres bloques funcionales principales:

- Registro: es una memoria de alta velocidad y poca capacidad, que permite guardar transitoriamente y acceder a valores muy usados, generalmente en operaciones matemáticas.
- Unidad de control: su función es buscar las instrucciones en la memoria principal, decodificarlas y ejecutarlas.
- Unidad aritmética lógica: es un circuito digital que realiza operaciones aritméticas y operaciones lógicas entre los valores de los argumentos. Puede incluir una unidad de coma flotante, puede realizar operaciones de cálculo en coma flotante. Esto incluye desde las sumas y multiplicaciones más usuales, hasta realizar cálculos trigonométricos o exponenciales.

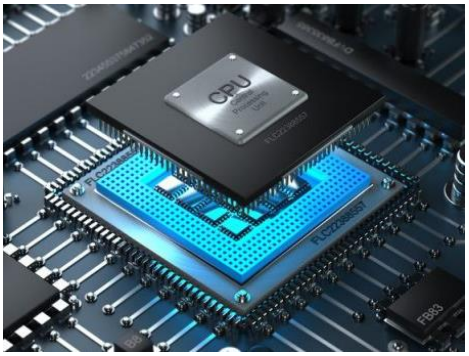


Fig 1. Funcionalidades principales de los microprocesadores [Fuente: Google.com]

Para el correcto funcionamiento de un microprocesador hay que incluir otros elementos *hardware* en el sistema. Algunos de los elementos clave son: memoria RAM, placa base, refrigeración, fuente de alimentación, etc. Hay que incluir componentes *software* como un SO para poder realizar las funciones deseadas.

En el mercado se encuentan varios modelos:

- Intel Core Series: tiene un uso amplio en ordenadores personales, portátiles o estaciones de trabajo.
- IBM POWER Series: utilizados en sistemas más grandes y servidores empresariales.
- Apple Silicon: microprocesador diseñado para los productos de Apple.

Cada tipo de procesador tiene sus características propias, capacidades, ventajas e inconvenientes, la elección depende de las necesidades específicas de la aplicación y del usuario.

3.1.3 Microcontroladores.

Un microcontrolador es un dispositivo compacto y autónomo que integra en un solo chip varios componentes esenciales para el procesamiento de información y el control de dispositivos electrónicos. Está diseñado para realizar tareas especifica, donde la eficiencia y la optimización de recursos son esenciales.

Los microcontroladores están formados por *hardware* y *software*. La primera incluye los componentes específicos para la aplicación como la tarjeta de vídeo, audio, etc., y el *software* (usado para el diseño de las funciones específicas que requiera la aplicación) está diseñado para realizar las funciones específicas por la aplicación.

Arquitectura.

Los sistemas integrados constan de varios componentes que forman su arquitectura, en este apartado se analizan los más esenciales:

- Microprocesador: es el elemento fundamental para un sistema integrado, responsable de ejecutar y coordinar las operaciones del sistema y elementos que le rodean.
- Memoria: proporciona un almacenamiento para el código del programa que el sistema ejecuta, también se encuentra los datos temporales y resultados. Su característica principal es que debe tener un acceso de lectura y escritura lo más rápido posible para que el microprocesador no pierda tiempo en tareas que no son meramente de cálculo.

- E/S: este elemento permite la interacción del sistema integrado con su entorno y otros dispositivos. Se encuentran una gran variedad de periféricos permitiendo adaptar el sistema integrado a diferentes aplicaciones, desde la adquisición de datos hasta el control de dispositivos externos.
- Comunicación: para la conexión con otros dispositivos es esencial una interfaz de comunicación, que facilitará la trasferencia de datos entre sistemas integrados, otros componentes del propio sistema o dispositivos externos.
- Reloj: el control del tiempo es crucial para coordinar las operaciones y asegurar que las operaciones se realicen el orden y momento deseado. Este elemento incluye un reloj que proporciona pulsos para la sincronización de las operaciones, un temporizador que genera interrupciones o activa eventos después de un intervalo de tiempo, entre otros usos.
- SO: algunos sistemas integrados ejecutan un SO en tiempo real (RTOS) o *firmware* especializado que gestiona las operaciones del sistema que se ha diseñado para una aplicación.
- Sensores y actuadores: son especialmente importantes para la interacción con el entorno físico. Los sensores recopilan datos del entorno, mientras que los actuadores realizan acciones, basados en los datos recogidos por los sensores o programados por el *software*.
- Fuente de alimentación: suministra energía al sistema. Puede incluir baterías, fuentes de alimentación externas o sistemas de gestión de energía.

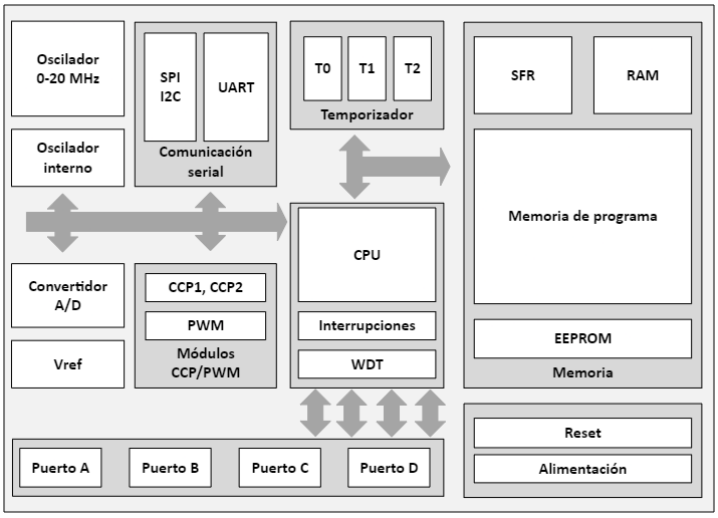


Fig 2. Arquitectura de los microcontroladores [Fuente: EP]

La combinación de estos elementos depende de la aplicación y los requisitos del sistema integrado. Un sistema integrado en un dispositivo médico tendrá una configuración diferente en comparación con un sistema en un automóvil.

Ventajas e inconvenientes.

Los sistemas integrados ofrecen varias ventajas que los hacen adecuados para una gran variedad de aplicaciones, estas ventajas incluyen:

- Eficiencia en el coste: los sistemas integrados están diseñados para aplicaciones específicas, permitiendo cumplir con los requisitos de la solución utilizando solo los componentes necesarios. Esto permite un coste muy reducido en comparación a una solución de propósito general.
- Consumo de energía: al tener un diseño tan especifico se puede optimizar el consumo de energía y prolongar la duración de la batería en dispositivos portátiles.
- Tamaño compacto: los sistemas integrados pueden ser más compactos en comparación con las soluciones de propósito general, lo que los hace ideales para dispositivos o entornos con espacios muy reducidos.
- Tiempo real: los sistemas integrados se pueden utilizar en aplicaciones donde la capacidad de respuesta es crítica, garantizando respuestas rápidas que son esenciales en entornos de sistemas de control industrial, automóviles y dispositivos médicos.

Aunque ofrece muchas ventajas, también pueden tener algunos inconvenientes, estos incluyen:

- Suelen tener recursos limitados en términos de capacidad de procesamiento, memoria y almacenamiento. Esto puede ser una restricción cuando se trata de ejecutar aplicaciones más complejas o exigentes en términos de recursos.
- Debido a su integración profunda con dispositivos específicos, los sistemas integrados pueden ser difíciles de actualizar.
- Debido a la complejidad, los ciclos de desarrollo de sistemas integrados pueden ser más largos en comparación con otros.

Teniendo en cuenta todos estos puntos, se pueden aplicar los sistemas integrados en diferentes ámbitos de la sociedad.

3.1.4 Procesadores de señales digitales.

Un DSP es un tipo especializado de microcontrolador diseñado específicamente para realizar operaciones de procesamiento de señales digitales. A diferencia de otros microcontroladores que se utilizan para una amplia gama de aplicaciones, los DSP están optimizados para manipular datos digitales, como señales de audio, video, imágenes y otras formas de información que pueden representarse de manera digital. La diferencia esencial entre un DSP y un microprocesador es que el DSP tiene características diseñadas para soportar tareas de altas prestaciones, repetitivas y numéricamente intensas.

Su arquitectura y diseño están centrados en optimizar operaciones específicas asociadas con el procesamiento de señales, ideales para aplicaciones especializadas en este ámbito, con la ayuda de entornos de desarrollo que permitan exprimir al máximo las capacidades del DSP.

Arquitectura.

La arquitectura de un DSP es diferente de la arquitectura de un procesador de ámbito general o microcontrolador. Algunos de los aspectos claves son:

- Pipeline de datos: permite el procesamiento de múltiples instrucciones simultáneamente. Esto optimiza el rendimiento para aplicaciones de procesamiento de señal que requieren un alto rendimiento en el tiempo.
- Unidades de procesamiento vectorial: permite las operaciones matemáticas en paralelo en múltiples datos, ideal para el procesamiento de señales que implica operaciones repetitivas en un conjunto de datos grande, como un filtro digital y trasformadas.
- Memoria especializada: permite la optimización del acceso a datos, que incluye múltiples bancos de memoria, almacenamiento en buffer circular y DMA.
- MAC: realiza operaciones de multiplicación y acumulación en paralelo, funciones de redondeo y saturación, optimizando y acelerando operaciones para el procesamiento de señales como convoluciones y filtrado.

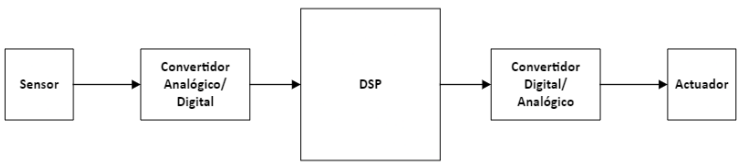


Fig 3. Arquitectura de los DSP [Fuente: EP]

La arquitectura está diseñada para optimizar el rendimiento y la eficiencia en aplicaciones de procesamiento de señales digitales, y permitiendo el desarrollo y funcionamiento de aplicaciones en tiempo real de un régimen fuerte, es decir, con un tiempo de respuesta inmediato.

3.1.5 SO.

Un SO es un *software* que actúa de intermediario entre el *hardware* de una computadora y el usuario o las aplicaciones que se ejecutan. Su función principal es gestionar los recursos del sistema y proporcionar una interfaz amigable para la interacción entre el usuario y la máquina. Las funciones de un SO son:

- Gestión de recursos: controla y asigna los recursos del *hardware* para garantizar un uso eficiente y equitativo.
- Interfaz de usuario: proporciona una interfaz amigable que permite a los usuarios interactuar con la máquina.
- Gestión de archivos: administra la creación, lectura, escritura y eliminación de archivos del sistema.
- Gestión de procesos: controla la ejecución de procesos y aplicaciones, asignando recursos y asegurando el orden de ejecución de las tareas.
- Gestión de memoria: administra el uso de la memoria RAM garantizando que los programas tengan acceso a la cantidad de memoria que necesitan y prevenir conflictos.
- Seguridad: proporciona mecanismos para proteger el sistema y los datos almacenados, como la autenticación de usuarios y el control de accesos.
- Comunicación entre *hardware* y *software*: facilita la comunicación y funcionamiento conjuntos de los elementos.

Los microcontroladores suelen tener requisitos específicos y, por lo tanto, utilizan SO especialmente diseñados para estos entornos. Es importante seleccionar el SO que mejor se adapte a los requisitos específicos de la aplicación y las limitaciones de recursos del microcontrolador. Algunos microcontroladores también pueden ejecutar aplicaciones sin un SO completo, dependiendo de la complejidad y las necesidades de la aplicación.

3.1.6 Entorno de desarrollo

Un entorno de desarrollo es un conjunto de herramientas y recursos que se utilizan para desarrollar el *software* de una aplicación de manera eficiente. Este entorno proporciona un conjunto de funciones que simplifican y automatizan tareas comunes durante el proceso de desarrollo. Estas son las características de un entorno de desarrollo:

- Editor de código: incluye un editor de código que resalta la sintaxis del lenguaje de programación, facilitando la lectura y escritura del código.
- Herramientas de compilación: proporciona herramientas para transformar el código en un formato que entiende la máquina para la ejecución del *software*.
- Depuración: ofrece la capacidad de identificar y corregir errores en el código.
- Herramientas de pruebas: ofrece herramientas de ejecución y automatización de pruebas para la verificación del código. También ofrece un entorno controlador y facilita la configuración de servidores locales.
- Navegador de proyectos: facilita la organización y navegación a través de los archivos del proyecto.
- Integración de bibliotecas: ofrece una amplia librería del lenguaje de programación, facilitando el desarrollo de *software* reutilizando código.
- Documentación: facilita la creación y visualización de documentación del código, incluyendo comentarios, generación automática de documentación y acceso rápido a información relevante.
- Personalización: permite al desarrollador personalizar el entorno según sus preferencias.

El uso de un entorno de desarrollo ayuda a la productividad y proporcionar un conjunto de herramientas para todas las capas del *software*. La elección de un entorno de desarrollo suele depender del lenguaje de programación y las preferencias del desarrollador o del equipo.

3.2 Marco contextual.

El sector industrial se refiere a la parte de la economía que se dedica a la elaboración de productos mediante la transformación de la materia prima o de la fabricación de otros elementos. Este sector abarca una amplia gama de actividades, desde la producción de

elementos físicos hasta la producción de energía. El sector industrial contribuye significativamente al PIB del país, ya que genera empleo e impulsa la innovación.

El proyecto se desarrolla en un contexto marcado por la creciente digitalización e innovación tecnológica en el sector industrial. Las plantas manufactureras buscan constantemente mejorar sus procesos y adaptarse a las demandas del mercado globalizado, donde la eficiencia operativa y la calidad son fundamentales para mantener la competitividad.

El proyecto se desarrolla en un contexto donde la digitalización, la automatización y la IA son aspectos clave para la competitividad y el crecimiento de las empresas industriales. La bancada para el desarrollo de soluciones industriales integrales avanzadas surge como una respuesta a las demandas y desafíos de este contexto, ofreciendo una herramienta innovadora y eficaz para impulsar la eficiencia y la competitividad en el sector industrial.

3.2.1 Aplicaciones de la IA en el sector industrial.

La IA tiene un impacto significativo en el sector industrial, mejorando la eficiencia, la productividad y la toma de decisiones. Algunas de las aplicaciones incluyen:

- Mantenimiento predictivo: Utilización de algoritmos que ayudan a predecir fallos en maquinaria y equipos industriales, permitiendo realizar el mantenimiento, reduciendo los tiempos de inactividad no planificados.
- Optimización de la cadena de suministro: usando algoritmos para prever la demanda, optimizar rutas, gestionar inventarios y mejorar la eficiencia de la cadena de suministro.
- Automatización de procesos de fabricación: implementación de robots y sistemas autónomos que pueden aprender y adaptarse a nuevas tareas.
- Control de calidad automatizado: sistemas de visión y algoritmos para inspeccionar y clasificar productos de forma rápida y precisa, reduciendo los errores de inspección humana.
- Gestión de energía: uso óptimo de energía en instalaciones industriales, identificando patrones de uso y proponiendo ajustes para reducir los costos y mejorar la eficiencia.
- Diseño de prototipos: herramientas de diseño que utilizan la IA para generar prototipos y mejorar el diseño de productos existentes.

- Sistemas de monitorización de activos: implementación de sensores y sistemas que motorizan el estado de los activos industriales en tiempo real, facilitando la detección de problemas y la optimización de su rendimiento.
- Planificación de la producción: ayudan a la planificación y programación de la producción, teniendo en consideración la demanda del mercado, la capacidad de la planta, el mantenimiento, los tiempos de producción, etc.
- Colaboración humano-robot: la integración de robots colaborativos que trabajan junto con los empleados.
- Análisis de datos: utilización de técnicas para extraer información valiosa a partir de grandes conjuntos de datos industriales, facilitando la toma de decisiones.

Estas aplicaciones muestran cómo la inteligencia artificial está transformando el sector industrial, permitiendo una mayor automatización, eficiencia y eficacia en las operaciones y la adaptación a entornos cambiantes.

3.2.2 Sistemas integrados.

Un sistema integrado o *embedded system*, es un sistema de computación basado en un microprocesador diseñado para realizar una o algunas funciones dedicadas, frecuentemente en un sistema de computación real. En estos sistemas la mayoría de los componentes se encuentras incluidos en la placa (tarjeta de vídeo, audio, etc).

Los sistemas integrados se pueden programar directamente en el lenguaje ensamblador del microcontrolador, o utilizando los compiladores específicos que suelen utilizar lenguajes como el C/C++.

Estos sistemas se encuentran en una amplia variedad de dispositivos y aplicaciones como electrodomésticos, dispositivos médicos, automóviles, sistema de control de acceso, sistemas de control industrial, etc.

3.2.3 Microcontroladores en el mercado

En la tabla 1, se representan algunos de los microcontroladores más utilizados en el mercado. Estos dispositivos son ampliamente utilizados en una gran variedad de aplicaciones. En la tabla se incluyen varios detalles de cada microcontrolador.

Fabricante	**Modelo**	**Arquitectura**	**Frecuencia (MHz)**	**Memoria flash (KB)**	**RAM (KB)**	**Periféricos integrados**	**Entorno de desarrollo**	**SO**
STMicro	STM32F407VGT6	ARM Cortex-M4	168	1024	192	UART, SPI, I^2C, PWM, ADC, DAC, Comparadores, temporizadores, USB	STM32CubeIDE	FreeRTOS, RTX, µC/OS
Microchip	PIC16F877A	8-bit	20	14	368	UART, SPI, I^2C, PWM, ADC, comparadores, temporizadores	MPLAB X	No soporta SO
Analog Devices	ADuCM3029	ARM Cortex-M3	26	128	64	UART, SPI, I^2C, PWM, ADC, DAC, Comparadores,	CroosCore Embedded Studio	No soporta SO

Fabricante	**Modelo**	**Arquitectura**	**Frecuencia (MHz)**	**Memoria flash (KB)**	**RAM (KB)**	**Periféricos integrados**	**Entorno de desarrollo**	**SO**
						temporizadores, LCD		
Atmel	ATmega328P	8-bit	20	32	2	UART, SPI, I^2C, PWM, ADC, comparadores, temporizadores	Atmel Studio	No soporta SO
NXP	LPC1768	ARM Cortex-M3	100	512	64	UART, SPI, I^2C, PWM, ADC, DAC, Comparadores, temporizadores, Ethernet	LPCXpresso	FreeRTOS, RTX, µC/OS
TI	TM4C1294XL	ARM Cortex-M4F	120	1024	2256	UART, SPI, I^2C, PWM, ADC, DAC, Comparadores, temporizadores, Ethernet	CCS	FreeRTOS, TI-RTOS, µC/OS

Tabla 1. Características técnicas de microcontroladores disponibles en el mercado. [Fuente: EP]

3.2.4 DSP en el mercado

En la tabla 2, se presenta una selección de DSP de diferentes fabricantes. Estos DSP se pueden encontrar en una variedad de aplicaciones que requieren procesamiento de señales a tiempo real.

Fabricante	**Modelo**	**Arquitectura**	**Frecuencia (MHz)**	**Memoria flash (KB)**	**RAM (KB)**	**Periféricos integrados**	**Entorno de desarrollo**	**SO**
Texas Instruments	TMS320C6748	C6000	300	320	0	DMA, UART, SPI, I^2C, PWM, ADC; DAC, temporizador	CCS	No soporta SO
Analog Devices	ADSP-21489	SHARC	400	544	4	DMA, UART, SPI, I^2C, PWM, ADC; DAC, temporizador	VisualDSP++	No soporta SO
NXP	LCP54102	ARM Cortex-M4	100	96	512	DMA, UART, SPI, I^2C, PWM, ADC; DAC, temporizador	MCUXpresso	FreeRTOS, RTX, µC/OS
Renesas Electronics	RZ/A1H	ARM Cortex-A9	400	128	0	DMA, UART, SPI, I^2C, PWM, ADC; DAC, temporizador	Renesas e^2 studio	FreeRTOS, Linux

Fabricante	**Modelo**	**Arquitectura**	**Frecuencia (MHz)**	**Memoria flash (KB)**	**RAM (KB)**	**Periféricos integrados**	**Entorno de desarrollo**	**SO**
Xilinx	Zynq-7000	ARM Cortex-A9	667	512	0	DMA, UART, SPI, I^2C, PWM, ADC; DAC, temporizador	Vivado	FreeRTOS, Linux
Infineon Technologies	TC3A	AURIX TriCore	300	384	6	DMA, UART, SPI, I^2C, PWM, ADC; DAC, temporizador	DAVE	FreeRTOS, AUTOSAR

Tabla 2. Características técnicas de DSP disponibles en el mercado. [Fuente: EP]

4 Objetivo de detalle y especificaciones técnicas.

Los objetivos del proyecto son 4 que se enumeran:

1) Obtención de una respuesta en tiempo real.

Obtener una respuesta rápida de la bancada conlleva ejecutar las operaciones en tiempos muy reducidos. Para lograr una respuesta en tiempo real es necesario un sistema duro, donde los procesos se cumplen en plazos muy concretos antes de la llegada del siguiente proceso. Para cumplir con tiempos tan estrictos, es necesario un dispositivo con una velocidad de procesamiento alto.

2) Desarrollo de una aplicación de bajo nivel para el sector industrial.

La selección de un entorno de desarrollo y un SO amigable nos permite desarrollar aplicaciones de bajo nivel dirigidas al ámbito industrial. Estos dos elementos con la combinación de una IA, nos permite la creación de una amplia gama de aplicaciones industriales.

3) Visualización de las respuestas obtenidas.

Poder observar la respuesta temporal del sistema, a través de una gráfica, un numero o texto, es necesario desarrollar un elemento que nos permita transformar los valores de la respuesta temporal del microcontrolador a una señal visual. El objetivo es poder graficar la respuesta temporal a través de un osciloscopio o una pantalla LCD.

4) Desarrollar el proyecto con un bajo coste.

Un objetivo clave es mantener un coste bajo sin comprometer la calidad y el rendimiento de la bancada. Para alcanzar dicho objetivo, la selección de componentes de *hardware* y *software* específico, que cubra las necesidades de la bancada y no exceder con el presupuesto teniendo opciones o características que no se requieren, pero que aumentan el coste del proyecto.

Estos son los objetivos del proyecto y sus especificaciones técnicas necesaria para lograr el diseño y puesta en marcha de una bancada para el desarrollo de soluciones industriales con un microcontrolador e IA.

5 Alcance de detalle.

El alcance del proyecto abarca principalmente las diferentes etapas fundamentales para el diseño y la puesta en marcha de la bancada para el desarrollo de soluciones industriales integrales avanzadas con un microcontrolador e IA. Para ello, establecemos una lista de alcances:

1) Selección del *hardware*:
   - Microprocesador, microcontrolador o DSP.
   - Dispositivos para la generación de señal.
   - Dispositivo para la obtención de la señal a tratar.
   - Dispositivo para la muestra de la señal a tratar.
2) Selección del *software* compatible con el *hardware*:
   - Entorno de desarrollo.
   - SO que permita el desarrollo de aplicaciones en tiempo real.
   - IA que permita el desarrollo de aplicaciones en tiempo real, compatible con el entorno de desarrollo y el SO.
3) Diseño y desarrollo de la bancada:
   - Vista preliminar de la disposición y conexionado de los diferentes elementos que se incluye en la bancada.
   - Desarrollo del *software* para la puesta en marcha de la bancada.
4) El desarrollo de aplicaciones destinadas al entorno industrial.

Así como se ha delimitado el marco de alcance del proyecto, es necesario destacar las exclusiones:

- El desarrollo de la IA, que incluye la recopilación y preparación de los datos, la selección de algoritmos, entrenamiento del modelo, evaluación, ajustes y el despliegue.
- La implementación de la aplicación desarrollada en una empresa dedicada a la manufacturación de cualquier sector.

6 Solución tecnológica.

Se realiza un análisis basado en los objetivos establecidos y se selecciona la solución más adecuada, también se realiza dicha selección por los siguientes puntos:

1. El objetivo es desarrollar y diseñar una bancada que permita el desarrollo de aplicaciones de ámbito general en la industrial, seleccionando este microcontrolador, el abanico al desarrollar dichas aplicaciones es mayor, no solo permite el tratamiento de señales como un DSP.
2. El conocimiento previó que se ha adquirido durante el grado de ingeniería de un dispositivo Cortex-M4, reduce el tiempo de familiarización con el dispositivo y su entorno de desarrollo.

Aunque el microcontrolador no sea la opción más potente del mercado, cumple con todas las objetivos y especificaciones técnicas. TM4C1294NCPDT cumple con la mayoría de los objetivos para el diseño y desarrollo de la bancada. Tiene una velocidad de procesamiento adecuada y capaz de soportar el desarrollo de aplicaciones con IA implementada. TI ofrece periféricos con pantallas LCD, siendo una posible solución para la visualización de la respuesta en tiempo real.

6.1 Diseño de la bancada.

La bancada consta de tres componentes principales:

- Un ordenador con todo el software requerido para el funcionamiento de la bancada, incluyendo el CCS, TI-RTOS y cualquier programa necesario en el proceso de desarrollo.
- El microcontrolador, encargado de llevar a cabo las operaciones necesarias para el procesamiento de la señal obtenida a través del ADC, con el fin de mostrarla mediante un DAC en el osciloscopio.
- Un osciloscopio utilizado para realizar observaciones tanto de la señal original del motor como de la señal procesada por el microcontrolador.

El diseño de la bancada se complementa con otros dispositivos, como los osciloscopios, generadores de señales o amperímetros y voltímetros.

6.1.1 CCS

CCS es un entorno de desarrollo integrado compatible con los microcontroladores y microprocesadores de TI, se basa en el *software* de código abierto Eclipse. CCS comprende un conjunto de herramientas para desarrollar y depurar aplicaciones integradas. Incluye un compilador C/C++, un editor de código fuente, un entorno de creación de proyectos, un depurador, entre otras funciones.

CCS combina las ventajas del Eclipse con las funciones avanzadas de depuración integrada de TI, lo que resulta en un entorno de desarrollo atractivo y repleto de funciones para desarrolladores integrados.

Para el uso recomendado del CCS se requiere de un *hardware* con una memoria de 8 GB, un espacio en el disco de 5 GB, un procesador multi-core, compatible con los sistemas operativos de Windows, Linux y macOS.

Para empezar

En este apartado se detallan los pasos recomendados para iniciar el uso de CCS en el desarrollo de aplicaciones con el microcontrolador deseado. Al iniciar CCS por primera vez, se despliega una ventana llamada *Getting Started*, la cual emerge al inicializar CCS y al seleccionar la carpeta deseada.

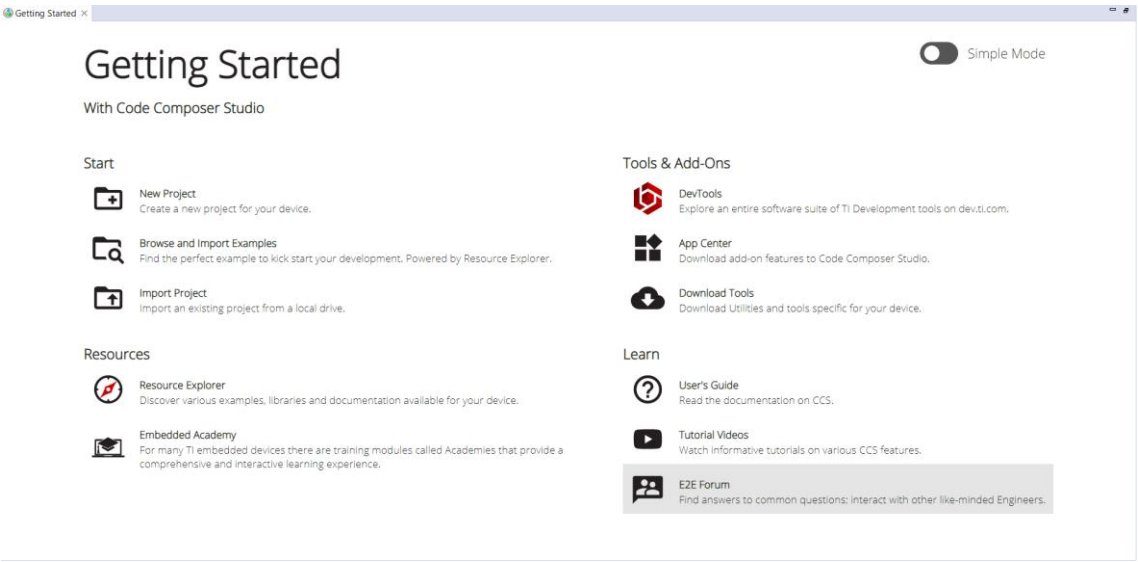


Fig 4. Ventana Getting Started. [Font: EP]

En esta ventana se pueden visualizar múltiples opciones:

- *New Project*: crear un nuevo proyecto CCS desde cero requiere un poco de conocimiento previo del dispositivo. Por lo tanto, se recomienda empezar siempre con un proyecto de ejemplo, si está disponible. Una vez que se haya adquirido cierta experiencia, se puede crear un nuevo proyecto desde cero.

  Para crear un nuevo proyecto CCS, se realizan los siguientes pasos:

  1. Se selecciona la opción *New Project*.
  2. En la ventana de *New CSS Project*, se siguen los siguientes puntos:
     - *Target:* se selecciona el dispositivo que se utiliza.
     - *Connection:* se selecciona el tipo de conexión entre el dispositivo y CCS.
     - *Project name:* se asigna un nombre al proyecto de la aplicación que se realizará.
     - *Project template/examples*: se selecciona el tipo de proyecto deseado, podemos obtener un proyecto vacío o un ejemplo que proporcione CCS.

     Para finalizar, se aprieta el botón Finish.
  3. El proyecto aparece en la ventana *Project Explorer* y se establece como proyecto activo.
  4. Dependiendo de la plantilla seleccionada, se añaden algunos archivos al proyecto para facilitar el proceso de construcción y carga del programa en el dispositivo seleccionado, como se puede observar en la fig 5.

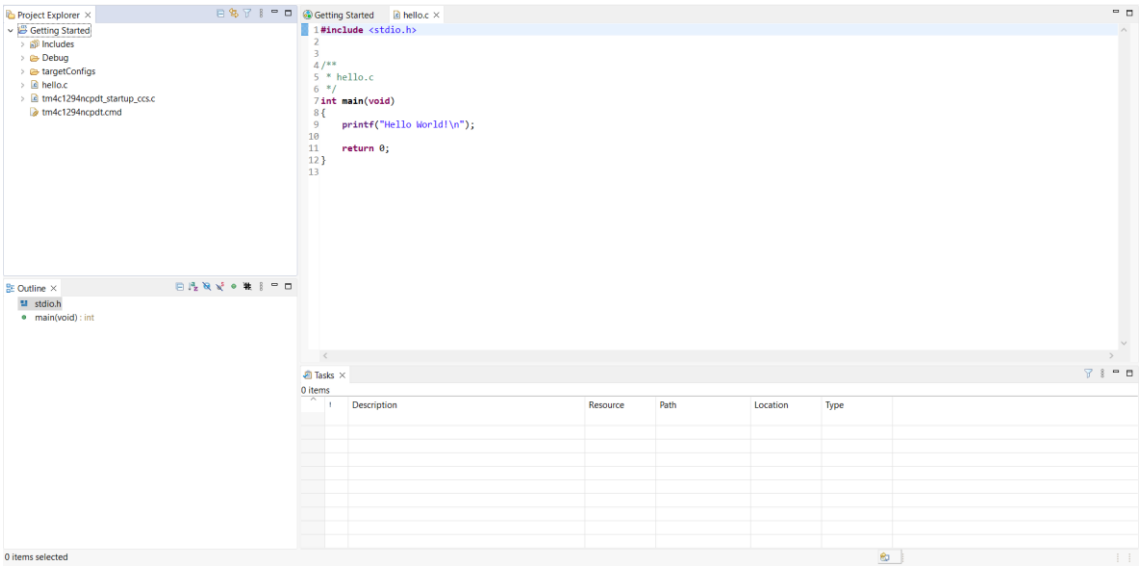


Fig 5. Ventana CCS. [Fuente: EP].

- *Import Project*: los proyectos CCS se deben de importar al espacio de trabajo antes de que se utilizen. La forma más cómoda de importar un proyecto de ejemplo es desde el *Resource Explorer*, opción que se menciona más adelante.

  En caso de que el ejemplo no este habilitado desde el *Resource Explorer*, el proyecto se puede importar manualmente siguiendo los siguientes pasos:

  1. Se selecciona la opción *Import Project*, y se abre una ventana *Select CCS Project to Import*.
  2. *Select search-directory*: se selecciona la carpeta de trabajo donde se encuentra el proyecto.
  3. *Discovered projects*: se selecciona el proyecto deseado.
  4. Se habilita la opción *Cope Project into workspace*, para realizar una copia del proyecto en la carpeta de trabajo habitual, de este modo se pueden realizar modificaciones en el proyecto, sin modificar el proyecto original.

  Una vez el proyecto se ha importado en la carpeta de trabajo habitual, será visible en la ventana *Project Explorer*.
- *Resource Explorer*: esta opción se caracteriza por la búsqueda de los últimos ejemplos disponibles, librerías, aplicaciones de demostración, entre otros, para el microcontrolador para el que se realiza la aplicación seleccionada.

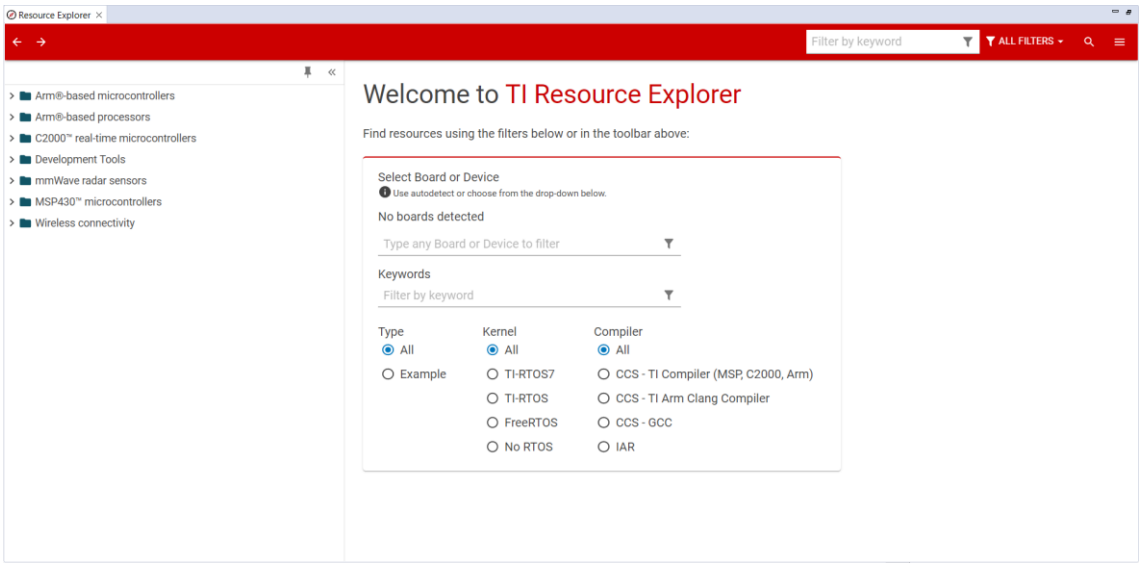


Fig 6. Ventana Resource Explorer. [Fuente: EP]

Se realiza la selección del dispositivo en *Select Board or Device*, donde se realiza la búsqueda de ejemplos o aplicaciones de demostración, por ejemplo, en este proyecto

se usa el microcontrolador de TM4C1294XL, en la fig 7 se muestra todos los ejemplos disponibles del microcontrolador.

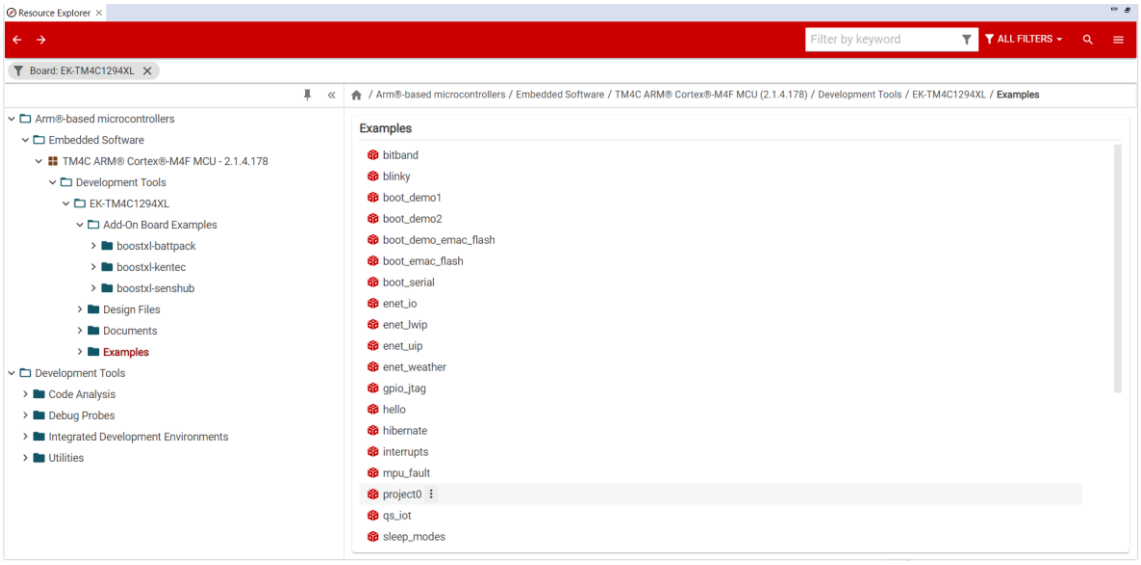


Fig 7. Ventana Resource Explorer con ejemplos de la placa TM4C1294XL. [Fuente: EP]

En la fig 7, se pueden observar las diferentes opciones que proporciona el buscador del microcontrolador TM4C1294XL, además, de una amplia lista de ejemplos.

Creación y gestión de proyectos

La organización y gestión de proyectos de desarrollo de software, especialmente en el ámbito de los sistemas integrados, pueden ser bastante complejas debido a la necesidad de atender diferentes configuraciones de hardware, opciones de compilación y el dispositivo objetivo. A continuación, se detalla cómo se puede utilizar de manera efectiva el entorno integrado de desarrollo CCS para optimizar estos procesos:

- Organización de proyectos: se puede utilizar una estructura jerárquica de carpetas dentro del proyecto CCS para organizar los archivos de manera lógica. Los archivos se agrupan en función de la funcionalidad, módulos o subsistemas para mejorar la navegabilidad y mantenibilidad.

  La ventana *Project Explorer* muestra todos los proyectos que forman parte del espacio de trabajo activo. Principalmente, representa el sistema de archivos de la carpeta del proyecto. Cuando se crea una subcarpeta y se mueven archivos a esa subcarpeta desde esta ventana, el sistema de archivos real se modifica en consecuencia. De manera similar, cualquier cambio realizado en el sistema de archivos se reflejará en la ventana *Project Explorer*, como se puede observar en la fig 8. Es importante tener en cuenta

que no todos los archivos que aparecen en la ventana existen en el sistema de archivos, y viceversa.

Esto posibilita la creación de carpetas dentro del proyecto con el fin de organizar el código fuente, archivos de encabezado, bibliotecas y otros recursos de manera efectiva.

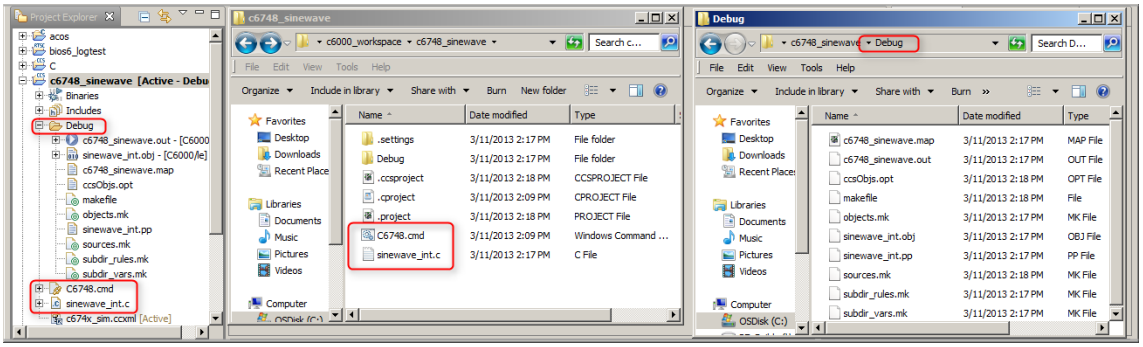


Fig 8. Ventana Project Explorer y explorador de archivos Windows. [Fuente: [11]]

- Creación de proyectos: se admiten diferentes tipos de proyectos en CCS, los cuales son seleccionados al crear un proyecto nuevo. El tipo de proyecto determinará la cadena de herramientas y configuraciones que se utilizarán, y se mostrarán en la interfaz.
  - *CCS Project*: es el tipo de proyecto más comúnmente utilizado al trabajar con la cadena de herramientas de TI, el *makefile* se genera automáticamente. Este tipo de proyecto se utiliza cuando se trabaja con procesadores integrados de TI y se desea crear un proyecto para cargar y depurar un dispositivo de esta familia.
  - *C or C++ Project*: es un proyecto estándar de Eclipse C o C++, se requiere una cadena de herramientas externa como GCC. El asistente de proyecto permite seleccionar un tipo de proyecto que puede generar automáticamente el *makefile* o no. Este tipo de proyecto se selecciona para crear un proyecto estándar de Eclipse C/C++.
  - *Makefile Project with Existing Code*: es un proyecto vacío que no crea automáticamente el *makefile.* Esto se utiliza para importar, modificar y compilar código existente basado en un *makefile*.
- Después de crear el proyecto, se pueden añadir o enlazar archivos para el proyecto, que son necesarios para el desarrollo de la aplicación. Cuando se añade un archivo se realiza una copia física en la carpeta donde se localiza el proyecto. Cuando se enlaza se crea una referencia en la carpeta del proyecto.

Para añadir o enlazar archivos, se selecciona el proyecto deseado, clic derecho y se escoge la opción *Add Files*, se realiza una búsqueda y se añaden los archivos deseados para el proyecto, que se mostraran en la ventana de *Project Explorer*.

También, se pueden añadir carpetas, para realizar este proceso, se selecciona el proyecto deseado, clic derecho y se escoge la opción *New* → *Folder*, *Advanced* y se selecciona la carpeta a través de la opción *Link to alternate location (Linked Folder)*.

CCS añade funciones como la copia o la eliminación de proyectos, entre otras opciones. También, se crean carpetas con diferentes tipos de proyectos para un mismo dispositivo o aplicación, como se observa en la fig 9.

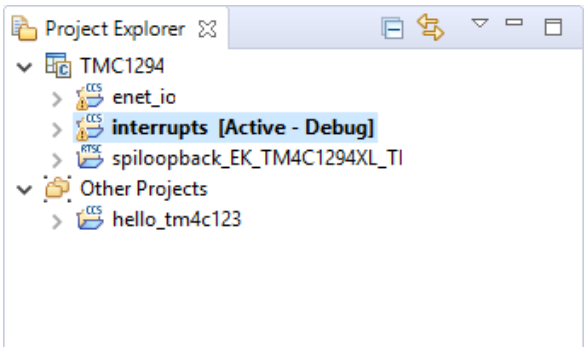


Fig 9. Ventana Project Explorer con un proyecto de la placa TM4C1294XL. [Fuente: EP]

Configuración de proyectos.

CCS tiene varias funciones para configurar un proyecto, en esta sección se describen las funciones con un uso más común. Para la configuración del proyecto, clic derecho en el proyecto y se selecciona la opción Properties, se abre una ventana parecida a la fig 10, esta ventana varía según el dispositivo y tipo de proyecto que se seleccione.

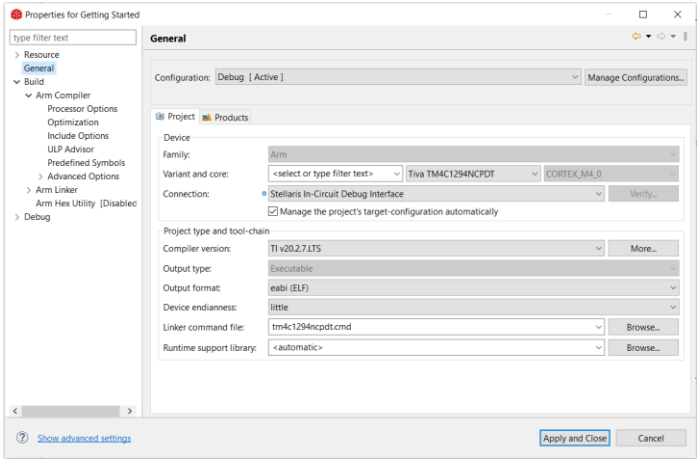


Fig 10. Ventana Properties de CCS. [Fuente: EP]

Se selecciona la opción de *Build*, a continuación, se describen las opciones que aparecen en esta ventana.

Builder: define el comando de compilación, la generación del *makefile* automática y otros ajustes de complicación, fig 11. En la mayoría de proyecto se mantiene la configuración predeterminada de esta pestaña.

Se encuentran opciones de compilación como:

- *Stop on first build error*: cuando se habilita esta función detiene la compilación del programa al encontrar el primer error.
- *Enable parallel build*: esta opción se encuentra habilitada por defecto, ya que la mayoría de PCs tienen multi-core, esto ayuda a la velocidad de compilación.

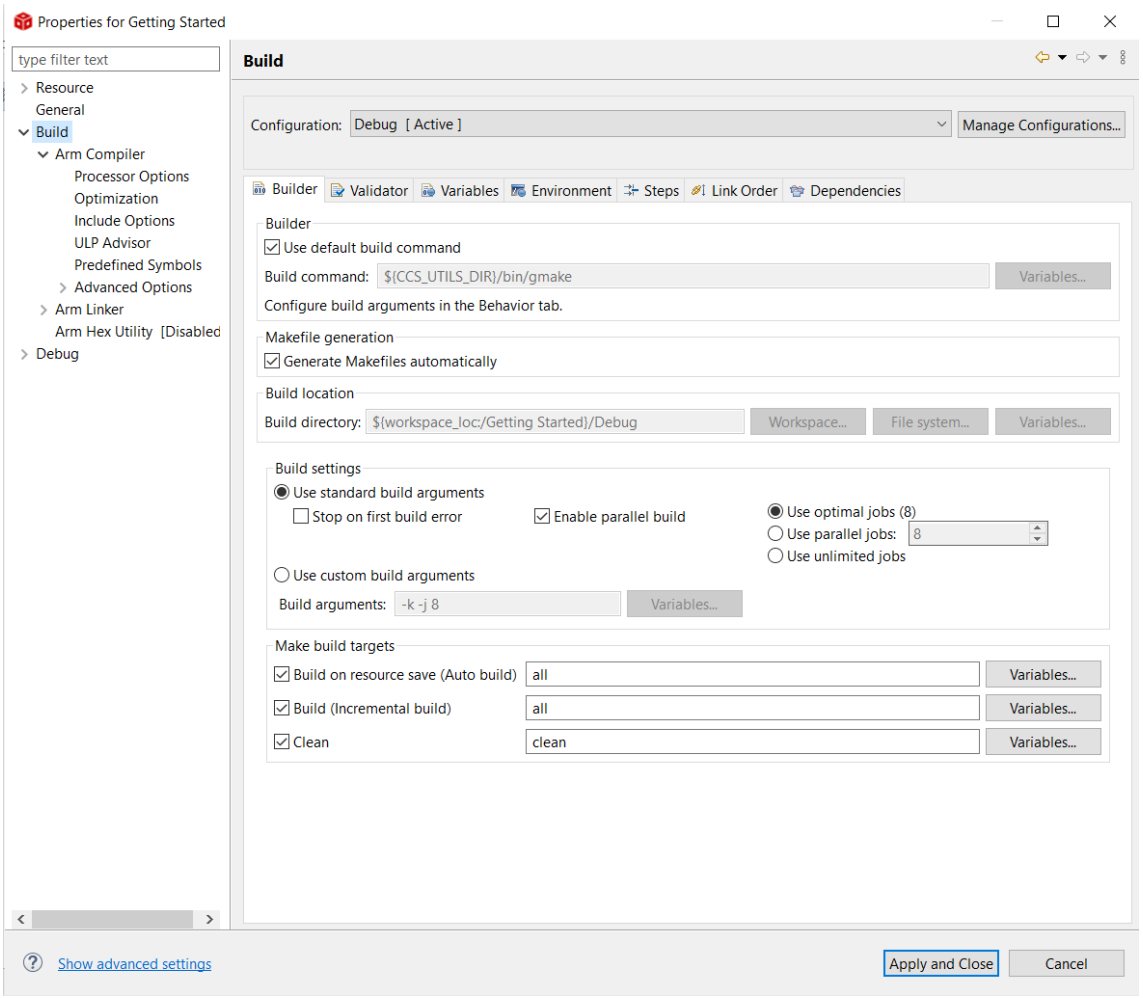


Fig 11. Ventana Properties pestaña Builder. [Fuente: EP]

- *Validator*: en esta pestaña se observa el nivel de adherencia a otras versiones de esta herramienta.

- *Steps*: en esta pestaña se encuentran los pasos que se ejecutan antes y después de la compilación del proyecto.
- *Variables*: en esta pestaña se muestran las variables de construcción.

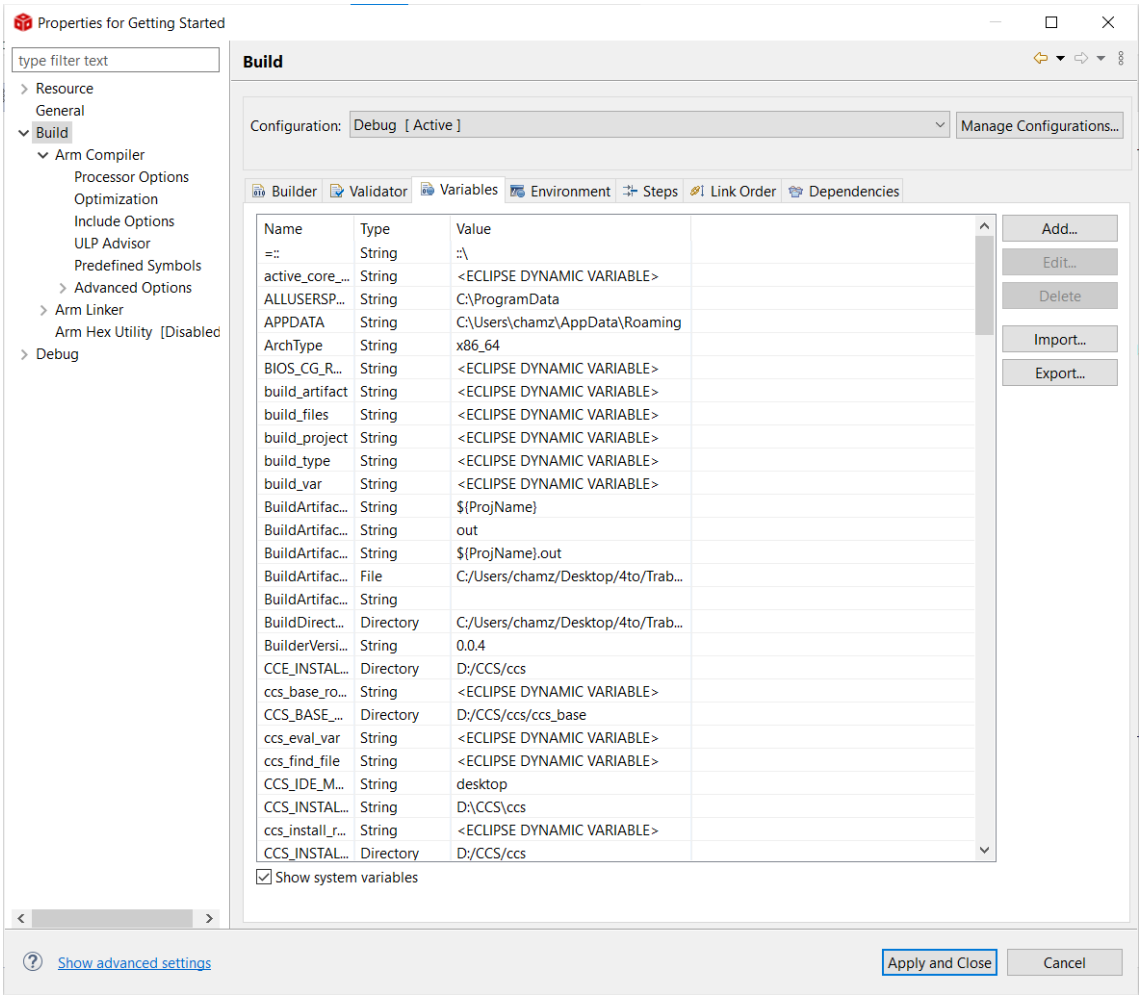


Fig 12. Ventana Properties pestaña Variables. [Fuente: EP]

En la fig 12, se observa todas las variables de compilación que alberga tanto el sistema como el proyecto.

- *Link Order*: en esta pestaña se añade y se define el orden de los archivos, se pasan al linker.
- *Dependencies*: en esta pestaña se añaden y se define las dependencias entre proyectos, permitiendo que los proyectos referenciados se construyan antes que el principal.

Sistema de depuración

En este apartado se ofrece un resumen del sistema de depuración de CCS y el proceso de depuración de un proyecto para un dispositivo integrado.

Este sistema se encarga de realizar una evaluación del comportamiento de la aplicación en el entorno deseado, este proceso permite la detección de errores o modificaciones en el comportamiento de la aplicación. CCS puede depurar un ejecutable alojado en un SO de alto nivel en el sistema integrado.

La configuración física de depuración se muestra en la fig 13, con los componentes principales. La conexión es la pieza de hardware entre el PC que ejecuta CCS, y el dispositivo donde se ejecuta el código. Esta configuración permite que el anfitrión se comunique con el dispositivo, cargue datos y código, y controle la ejecución a través de las diferentes opciones que proporciona el sistema.

Fig 13. Conexión/configuración hardware de los componentes. [Fuente: EP]

Los elementos del sistema de depuración se muestran en la fig 14, y se procesan de la siguiente manera:

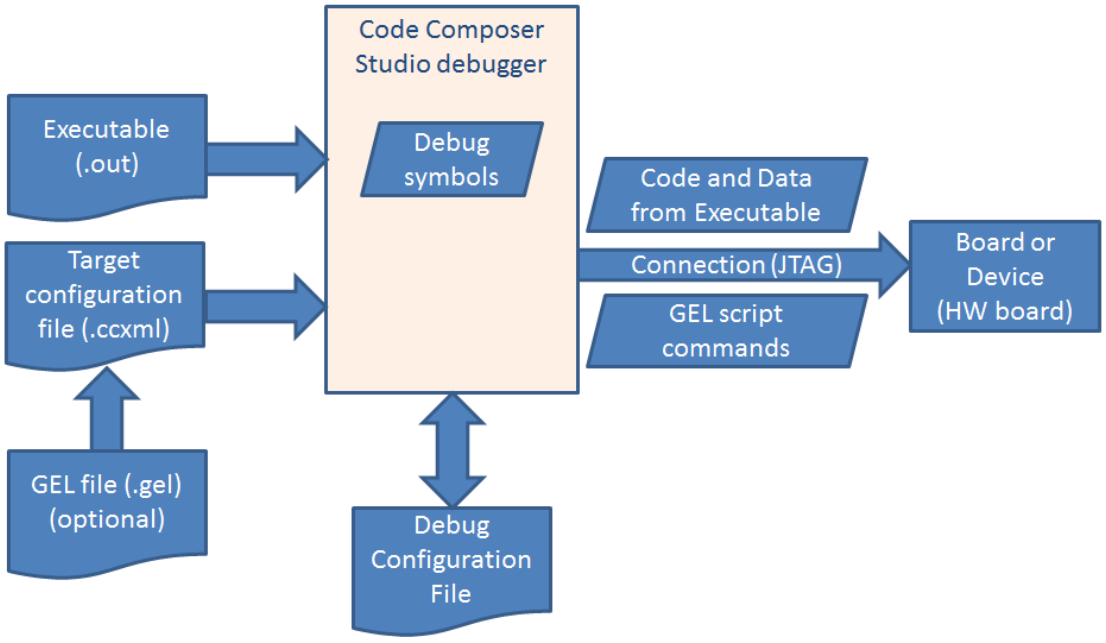


Fig 14. CCS debugger. [Fuente:[11]]

1. CCS cambia la perspectiva activa a la perspectiva de depuración, incluyendo muchas ventanas útiles para el proceso, estas vistas se observan en la fig 15.

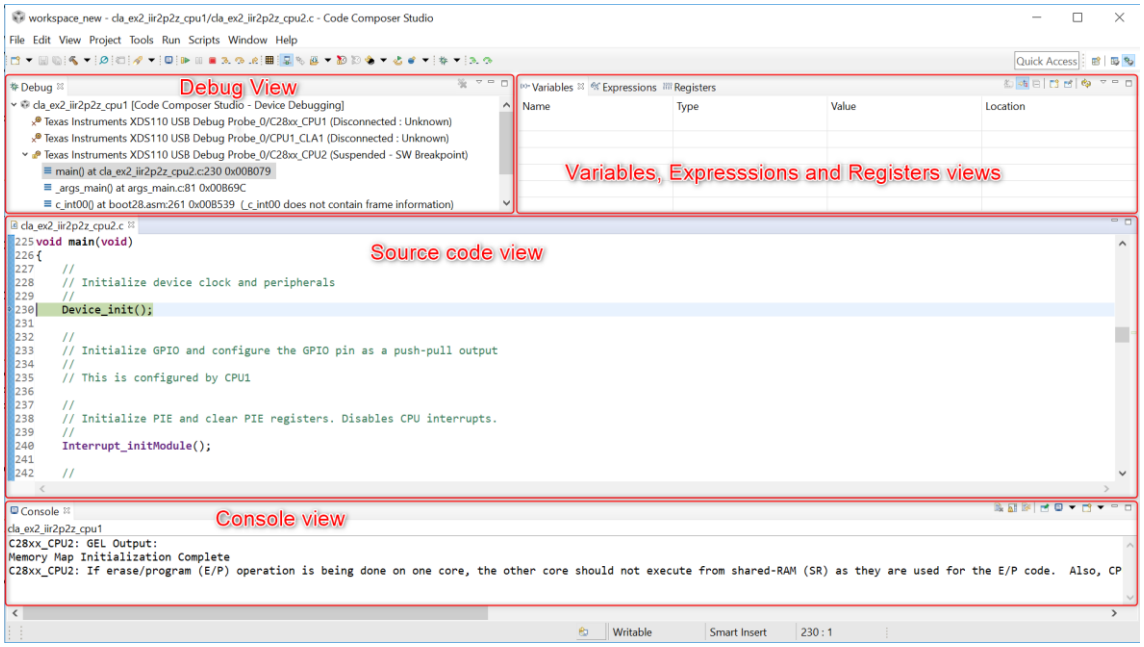


Fig 15. Ventana de depuración del CSS. [Fuente: [11]]

Cada ventana proporciona un tipo de información:

- *Debug View*: se abre por defecto la ventana mostrada en la fig 16, contiene información sobre:
    - El árbol que muestra el archivo de configuración del dispositivo utilizado y el tipo de sistema de depuración que se está utilizando.
    - El estado de cada núcleo del dispositivo, si se encuentra conectado, desconectado, etc.
    - La dirección actual del contador del programa o la pila de llamadas del programa que se encuentra actualmente ejecutado.

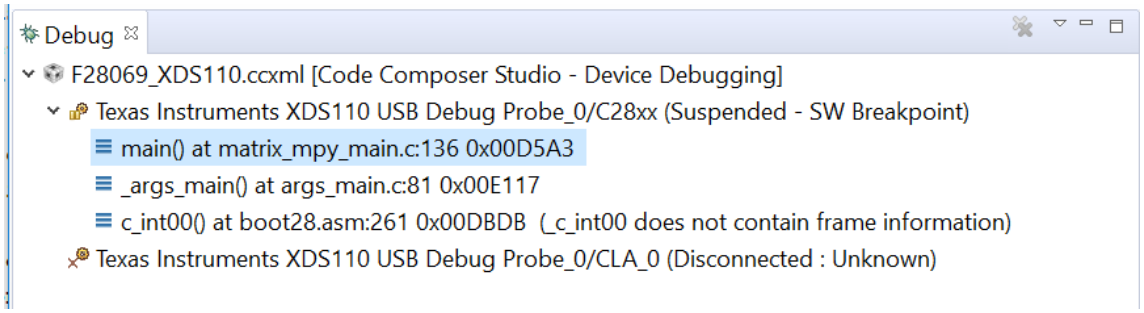


Fig 16. Ventana Debug. [Fuente:EP]

- *Source code view*: muestra el código del archivo *main*.

- *Variables, Expressions and Registers views*:

  En la pestaña *Variables*, solo muestra las variables locales que se utilizan en la ejecución de la aplicación. Esta ventana muestra el contenido de cada variable, proporcionando información sobre el nombre, el tipo de variable, el valor que contiene y la dirección en la que se encuentra en la fig 17.

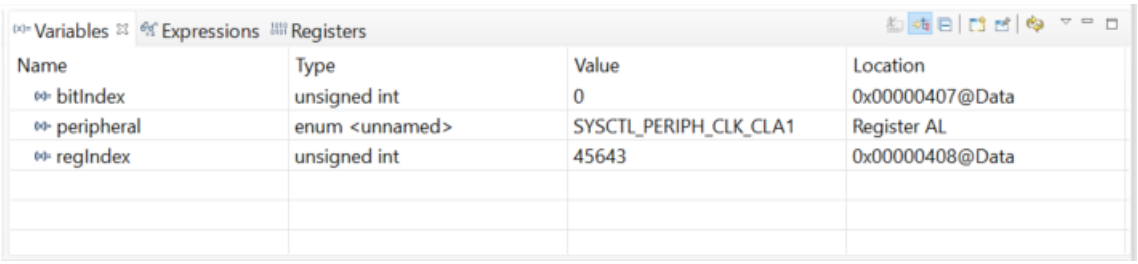


Fig 17. Ventana Variables. [Fuente: EP]

En la pestaña de *Expressions*, se puede monitorizar las variables, tanto locales, globales como estáticas, que incluye la aplicación, la distribución de esta ventana se muestra en la fig 18.

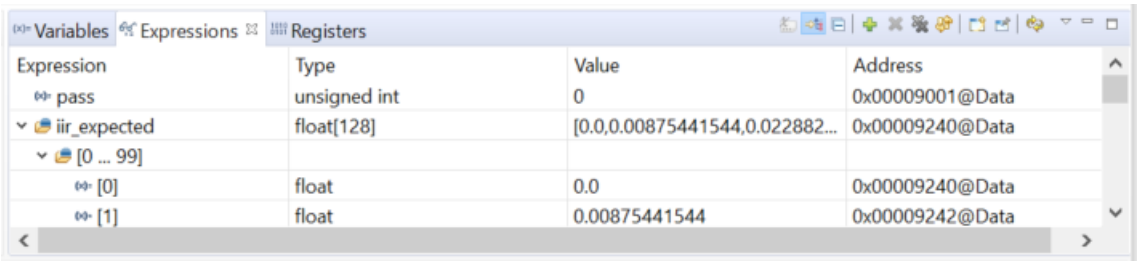


Fig 18. Ventana Expressions. [Fuente: EP]

La pestaña *Registers*, permite ver y editar el contenido de los registros del dispositivo y sus periféricos, esta ventana se muestra en la fig 19.

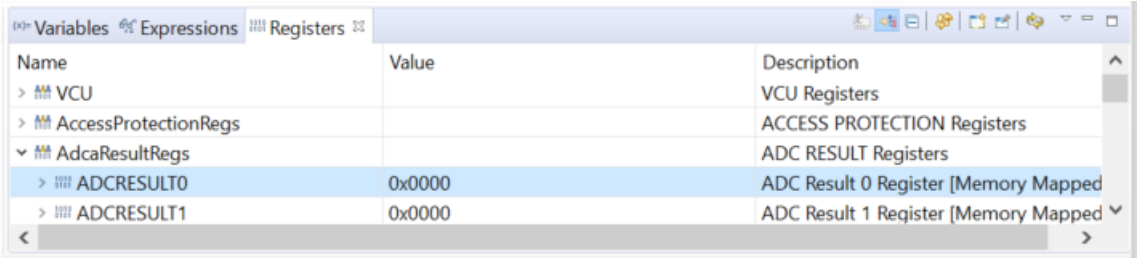


Fig 19. Ventana Registers. [Fuente: EP]

- *Console view*: se utiliza para introducir datos si es necesario en durante el funcionamiento de la aplicación.

También, se pueden observar otras ventanas como el *Disassembly View* o *Memory Browser View*, se abren desde el botón *View* y se selecciona la ventana deseada, donde se pueden observar las transformaciones de código máquina a lenguaje ensamblador, u observar la modificación de una dirección en la memoria.

2. A continuación, se analiza el archivo de configuración del dispositivo, se crea una configuración de depuración y con la ayuda de estos archivos, se conecta y se comunica con el dispositivo.
3. Una vez establecida la comunicación, se ejecuta una serie de comandos de inicialización del hardware desde el *GEL script*.
4. CCS se encarga de dividir la información en:
   - El contenido del código y los datos se envían a través de la conexión y se almacena en las ubicaciones establecidas en la memoria del dispositivo.
   - Los símbolos de depuración se mantienen en el PC para permitir la correlación de direcciones de memoria del dispositivo con el código fuente del proyecto.
   - El proceso es el mismo tanto si el código se descarga en RAM como en *Flash*.
5. El sistema de depuración realiza un último paso: establece un punto de ruptura en la función *main*, y se ejecuta en el dispositivo hasta que termina el ciclo de la aplicación.

Una vez finalizado el proceso, el depurador se mantiene a la espera de la interacción mediante los comandos de *Resume*, *Suspend, Step Into, Step Over*, entre otros.

El sistema de depuración se inicia apretando el botón F11, o en el menú *Run* → *Debug*. Cuando se termina el proceso explicado anteriormente y se conecta al dispositivo, se abre una ventana como la fig 20.

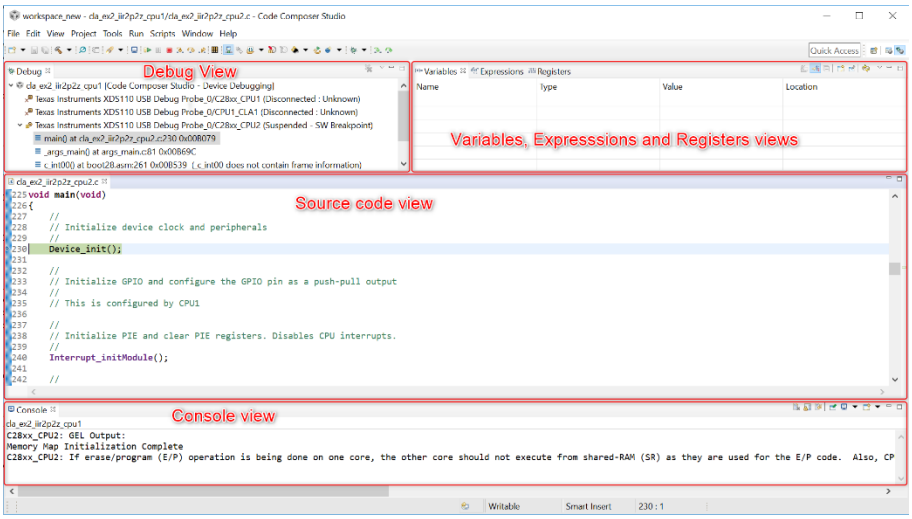


Fig 20. Ventana del sistema de depuración. [Fuente: [11]]

Las funciones del sistema de depuración se encuentran en la parte superior de la ventana. A continuación, se da una explicación de la barra de herramientas de la fig 21, de izquierda a derecha.

Fig 21. Barra de herramientas del sistema de depuración. [Fuente: EP]

- *Terminal*: se centra en mostrar las comunicaciones desde el CCS hasta el dispositivo.
- *Rusume*: se inicia la ejecución del dispositivo.
- *Suspend*: se detiene la ejecución del dispositivo.
- *Terminate*: se desconecta todo el hardware y finaliza la sesión de depuración, y se encarga de cerrar la ventana del sistema de depuración, y abrir a la ventana de edición.
- *Step Into*: ejecuta una única línea de código fuente, lo que permite ejecutar el código paso a paso.
- *Step Over*: similar al anterior, pero el salto se realiza sobre subrutinas o funciones, ejecutando su código de una sola vez.
- *Step Return*: Similar al anterior, pero ejecutando todas las líneas, subrutinas o funciones hasta llegar a la funcion anterior.
- *Runtime Object Viewer*: se encarga de iniciar una ventana para los sistemas SYSBIOS y TI-RTOS.
- *Coneect/Disconnect*: conecta o desconecta el núcleo seleccionado en la ventana Debug.
- *Restore Debug State*: se encarga de restaurar todos los ajustes de depuración del núcleo o dispositivo.
- *Load:* carga el código en el núcleo seleccionado en la ventana Debug.
- *Real-time modes*: activa/desactiva los modos de tiempo real soportados por el dispositivo.
- *Reset*: restablece el núcleo o dispositivo.
- *Restart*: emite un reinicio cuando se cargan símbolos o un programa en el dispositivo de destino. Un reinicio simplemente recoloca el registro en el punto de entrada.
- *New Breakpoint*: añade o activa un *breakpoint* en una línea señalada por el cursor del ratón.
- *Debug As*: inicia una nueva sesión del sistema de depuración.
- *Assembly Step Into*: ejecuta el código interno paso a paso.

- *Assembly Step Over*: similar al anterior, pero ejecutando su código interno de una vez, sobre una subrutina o función.
- *EnergyTrace*: inicia una sesión *EnergyTrace*.

Con esta información se garantizará un uso adecuado del sistema de depuración de CCS.

Herramientas gráficas.

CCS dispone de herramientas gráficas para la visualización de datos valiosos de tipo gráfico, los tipos de gráficas que se encuentran habilitados son:

- *Single time*: gráfico que representa los valores de la matriz en el eje "Y" y el índice de la matriz en el eje "X".
- *Dual time*: muestra dos líneas en una sola ventana.
- *FFT Magnitude*: gráfico que calcula la magnitud en el dominio frecuencia realizando una FFT en la matriz de datos.
- *FFT Magnitud & Phase*: gráfico que calcula la magnitud y fase en el dominio frecuencial de una matriz de datos, se pueden observar tanto datos reales como complejos.
- *FFT Complex*: gráfico que calcula la magnitud y fase en el dominio frecuencial de dos matrices de datos.
- *FFT Waterfall*: gráfico con múltiples líneas que calcula la magnitud en el dominio de la frecuencia, en un número arbitrario de matrices.

Esta opción se encuentra en el menú *Tools* → *Graph* → *type of graph*.

Para más información sobre CCS consultar la referencia 11.

6.1.2 Texas Instruments Real Time Operative System: Kernel

TI-RTOS es un SO en tiempo real desarrollado por TI, una extensión del CCS, que incluye un equipo de desarrollo de *software*, aparte de un *Kernel* que gestiona todas las funciones de tiempo real.

Para empezar.

Los pasos a seguir para la creación de un proyecto TI-RTOS son exactamente igual que al crear un proyecto en CCS, se selecciona el microcontrolador de la aplicación y seleccionamos el tipo de proyecto TI-RTOS.

XGCONF.

La herramiento XGCONF se forma por diferentes paneles representados en la figura 22.

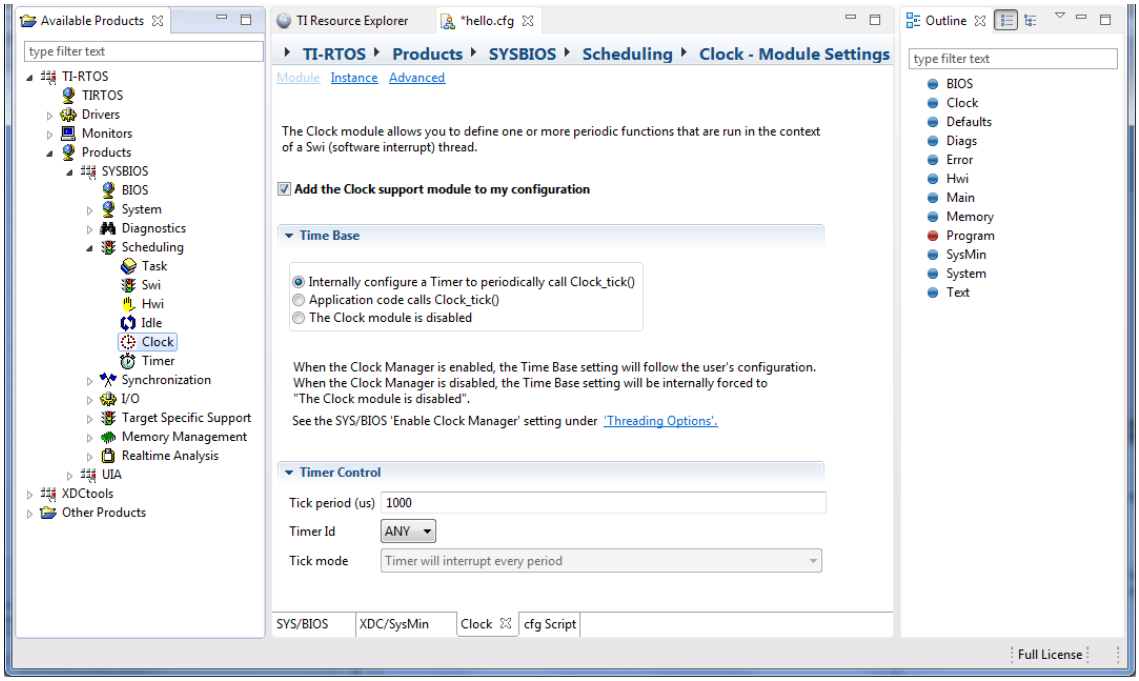


Fig 22. Ventana de la herramienta XGCONF. [Fuente: EP]

En esta herramienta se encuentran tres ventanas diferentes:

- *Available Products*: permite añadir módulos a la configuración.
- *Outline*: muestra los módulos usados en la actual configuración y permite escoger un módulo para desplegar la ventana de *Property*.
- *Property*: muestra los ajustes del módulo seleccionado y permite realizar cambios en las propiedades de dicho módulo.

Ventana Available Products

En la ventana se observan todos los módulos y paquetes disponibles para la configuración del proyecto. En la lista, los módulos se encuentran primero los de *software* y después los de hardware.

Para usar un módulo, se realiza clic derecho sobre el módulo deseado y se escoge la opción *Use <module>*. Cuando se selecciona un módulo, se puede configurar sus propiedades en la ventana *Property*. También, se puede obtener una guía seleccionando la opción *Help*, del módulo que deseamos obtener información.

Ventana Outline

Esta ventana se muestran los módulos e instancias que están disponibles para la configuración de nuestro proyecto. Esta ventana se puede mostrar de dos formas

- Configuración de usuario, en este modo la ventana se dedica a mostrar únicamente los módulos, referencias e instancias creadas en el proyecto.

- Configuración de resultados, en este modo se muestra un árbol con todos los módulos e instancias que están siendo utilizadas de manera implícita o explicita en el proyecto.

Para crear una instancia, se realiza clic derecho al módulo y se selecciona *New <module>*. El color de la bola al lado de cada módulo se encarga de indicar la disponibilidad de dicho módulo en la tarjeta conectada. En caso de que el color sea rojo, el módulo se encuentra deshabilitado en la tarjeta, en caso de ser azul, el módulo se encuentra disponible en dicho dispositivo.

Threads.

Muchas aplicaciones en tiempo real deben realizar al mismo tiempo una serie de funciones aparentemente no relacionadas, a menudo en respuesta a eventos externos como la disponibilidad de datos o la presencia de una señal de control.

Tanto las funciones realizadas como el momento en que se realizan son importantes. Estas funciones se denominan *threads*. Los distintos sistemas definen los *threads* de forma más o menos amplia. Un *threads* es un único punto de control que puede activar una llamada a función o una rutina de servicio de interrupción.

Los programas de aplicaciones de tiempo real organizados con los *threads* son más fáciles de diseñar, implementar y mantener. Se destacan cuatro tipos de t*hreads* en el SO, que se encuentran ordenados de mayor a menor prioridad, como se observa en la fig 23:

- Hwi *threads*. Hwi *threads* son los *threads* con mayor propiedad en las aplicaciones. Estas interrupciones se usan para realizar tareas críticas con un tiempo de respuesta crítico. Se activan en respuesta a eventos que se producen en el entorno. Los subprocesos Hwi siempre se ejecutan hasta su finalización.
- Swi *threads*. Swi *threads* se encuentran por debajo las Hwi *threads*, A diferencia de los Hwi, que se activan mediante interrupciones de hardware, los Swi se activan mediante programación llamando a determinadas APIs del módulo Swi. Un Swi maneja *threads* que le impede ejecutarse como tareas, pero cuyos

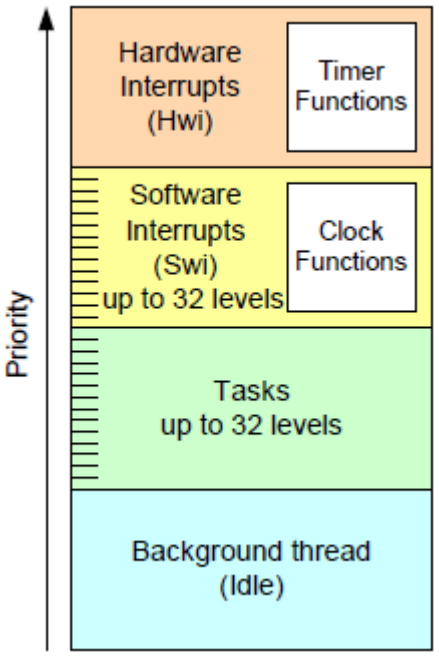


Fig 23. Orden de prioridad de los threads. [Fuente: [13]].

plazos no son tan estrictos como las interrupciones por *hardware*. Los Swis permiten a los Hwis diferir el procesamiento de *threads* menos críticos y con menor prioridad, minimizando el tiempo que la CPU pasa dentro de una rutina de servicio de interrupción, donde otros Hwis pueden ser deshabilitados.

- *Task threads*. Los *Task threads* tiene mayor prioridad que los *Idle threads*. Las *Tasks* se diferencia de las Swi en que se pueden esperar durante la ejecución hasta que los recursos necesarios estén disponibles. Para cada *Task threads* se requiere de una pila separada. El sistema proporciona una serie.

Qué tipo de threads usar.

El tipo y el nivel de prioridad que se elige para cada subproceso en un programa influye en que se ejecute a tiempo y correctamente. La configuración estática permite el cambio de un tipo de *thread* a otro.

En una aplicación se pueden usar varios tipos de *threads*. A continuación, se indican algunas reglas para decidir qué tipo utilizar para cada subproceso que se debe realizar en una aplicación.

- Swi o *task* frente a Hwi. Las Hwis se consideran para procesar interrupciones con plazos de hasta 5 microsegundos, especialmente cuando los datos se pueden sobrescribir si no se cumple el plazo. Los Swi o *tasks* se deben considerar para eventos con plazos más largos, alrededor de 100 microsegundos o más. Las funciones Hwis deben enviar Swis o *tasks* para realizar procesamientos de menor prioridad. El uso de los *threads* de menos prioridad minimiza el tiempo que las interrupciones están deshabilitadas, permitiendo que se produzcan otras Hwi.
- Swi frente a *task*. Se utilizan los Swis si las funciones tienen interdependencia e intercambio de datos. Se utilizan los *task* si los requisitos son más complejos. Mientras que los *trhreads* de mayor prioridad se adelante a los de menor prioridad, solamente los *tasks* pueden esperar a otro evento. Los *tasks* tiene más opciones que los Swis a la hora de compartir y usar datos. Todas las funciones necesarias para un Swi tienen que estar disponibles para su ejecución. Los Swi son más eficientes en memoria, ya que se ejecutan todos desde una pila.
- *Idle.* Se utilizan los *idle threads* para realizar tareas no críticas cuando no sea necesario ningún otro proceso. Estos *threads* no tienen tiempos de respuesta críticos,

se ejecutan cuando hay tiempo de procesamiento libre. Los *idle threads* se ejecutan secuencialmente con la misma prioridad.

- *Clock*. Se utiliza el *clock* cuando se desea ejecutar una función basada en múltiples interrupciones basadas en el pulso del *clock*. Las funciones de *clock* se pueden ejecutar periódicamente o solo una vez. Estas funciones se ejecutan como Swi.
- *Clock* frente a Swi. Todas las funciones *clock* se ejecutan con la misma prioridad que los Swis, por lo que una función de *clock* no puede adelantarse a otra. Sin embargo, las funciones de *clock* pueden enviar Swi *threads* de menor prioridad para un procesamiento más largo.
- *Timer*. *Timer threads* se ejecutan dentro del contexto de un Hwi *thread*, estos *threads* ejecutan la tarea en un tiempo mínimo. En caso de que se requiera más tiempo de procesamiento, se debe considerar trabajar con un Swi, de este modo se gestiona de manera eficiente la CPU.

El sistema ejecuta el *threads* de mayor prioridad, excepto en los siguientes casos:

- Si el *thread* que se está ejecutando desactiva algunas o todas las Hwis temporalmente, impidiendo que se ejecuten este tipo de interrupciones.
- Si el *thread* que se está ejecutando desactiva temporalmente las Swis. Esto evita que cualquier Swi con mayor prioridad se ejecute antes del *thread* actual, pero se pueden ejecutar los Hwis
- Si el *thread* que se está ejecutando, desactiva temporalmente los *tasks*. Esto evita que cualquier *task* con mayor prioridad se ejecute antes del *thread* actual, pero permite la ejecución de los Hwis y Swis.

En la fig 24, se muestra el gráfico de ejecución para una aplicación con los Swis y Hwis habilitados.

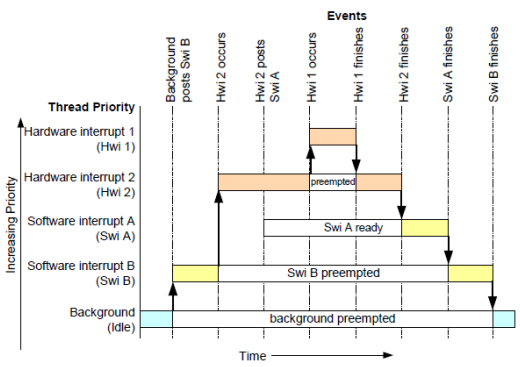


Fig 24. Comportamiento de los diferentes threads. [Fuente: [13]]

Módulos de sincronización.

Kernel proporciona un conjunto fundamental de funciones para la sincronización y comunicación entre tareas. Estos son los semáforos, eventos, gates, *mailbox* y *queue*.

- Semáforos: Se utilizan para coordinar el acceso a un conjunto de *task* que requiere el mismo recurso. Por defecto, existen los semáforos simples contadores, mantienen una cuenta interna del número de recursos disponibles. Cuando el recuento es mayor que cero, los *tasks* no se bloquean.

  Existen los semáforos binarios, su valor no puede incrementarse más allá de uno. Por lo tanto, se deben utilizar para coordinar el acceso a un recurso por dos *tasks*. Estos semáforos ofrecen mejor rendimiento que los semáforos simples.

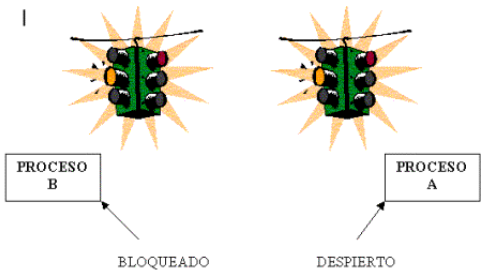


Fig 25. Funcionamiento de un semáforo. [Fuente: EP]

- Eventos: Proporcionan un medio de comunicación y sincronización entre *threads*. Funcionan de manera similar a los semáforos, excepto que permiten especificar múltiples condiciones que deben ocurrir antes de que el *thread* en espera regrese, facilitando la sincronización y la comunicación entre *threads*.
- *Gates*: Es un módulo de sincronización que se utiliza para control de acceso a secciones críticas del código.
- *Mailbox*: son estructuras de datos que permiten la comunicación entre *threads* mediante el envío y recepción de datos. Un *thread* puede colocar un mensaje en el *mailbox* para que se reciba más tarde. Esto facilita la comunicación asíncrona entre *threads.*
- *Queue*: son estructuras de datos que permiten la comunicación entre *threads* mediante el envío y recepción de datos. A diferencia de los *mailbox*, un *queue* puede contener cualquier tipo de dato.

6.1.3 Tiva™ TM4C1294NCPDT

En este apartado se explica las características técnicas del microcontrolador que se integra en la bancada de desarrollo de aplicaciones.

Arquitectura del microcontrolador.

En este punto del proyecto realizaremos un estudio del microcontrolador TM4C1294NCPDT.

Fig 26. Microcontrolador TM4C1294NCPDT. [Fuente: [12]]

Características generales del microcontrolador.

TM4C1294NCPDT es un microcontrolador con las siguientes características:

Función	Descripción
Rendimiento	
Microprocesador	ARM Cortex-M4F
Velocidad de procesamiento	120 MHz
Memoria Flash	1024 KB
SRAM	256 KB
EEPROM	6 KB
ROM interna	ROM interna con el software de TivaWare para C Series
EPI	8/16/32-bits dedicados a la interfaz de los periféricos y memoria
Seguridad	
Comprobación de redundancia cíclica (CRC) Hardware	Función Hash de 16/32-bits que permite cuatro formas de CRC
Tamper	Soporte para cuatro entradas tamper y respuesta configurable a eventos de tamper
Interfaz de comunicación	

Función	**Descripción**
UART	8 UARTS
QSSI	Cuatro módulos SSI con Bi-, Quad- y soporte avanzado de SSI
I^2C	Diez módulos con cuatro velocidades de transmisión
CAN	Dos controladores CAN 2.0 A/B
Ethernet MAC	10/100 Ethernet MAC
Ethernet PHY	PHY con soporte de hardware IEE1588 PTP
USB	USB 2.0 OTG/Host/Device con opción de interfaz ULPI y soporte Link Power Management (LPM)
integración de sistema	
uDMA	ARM PrimeCell 32 canales de control configurables de uDMA
GTM	Ocho bloques de 16/32-bits
Temporizador Watchdog	Dos temporizadores Watchdog
Módulo de hibernación	Módulo de hibernación con batería de bajo consumo
GPIO	15 bloques físicos de GPIO
Control de movimiento avanzado	
PWM	Un módulo PWM, con cuatro bloques generadores PWM y un bloque de control, para un total de 8 salidas PWM.
QEI	Un módulo de QEI
Soporte analógico	
ADC	Dos módulos ADC de 12 bits, cada uno con una frecuencia de muestreo máxima de dos millones de muestras/segundo
Controlador con comparación analógica	Tres controladores independientes
Comparador digital	16 comparadores digitales
JTAG y SWD	Un módulo JTAG con ARM SWD integrado
Información general	
Pins	128 pins
Rango de operación	-40ºC a 85ºC

Tabla 3. Características del microcontrolador TMC1294NCPDT. [Fuente: [12]]

Diagrama de bloques de alto nivel del microcontrolador.

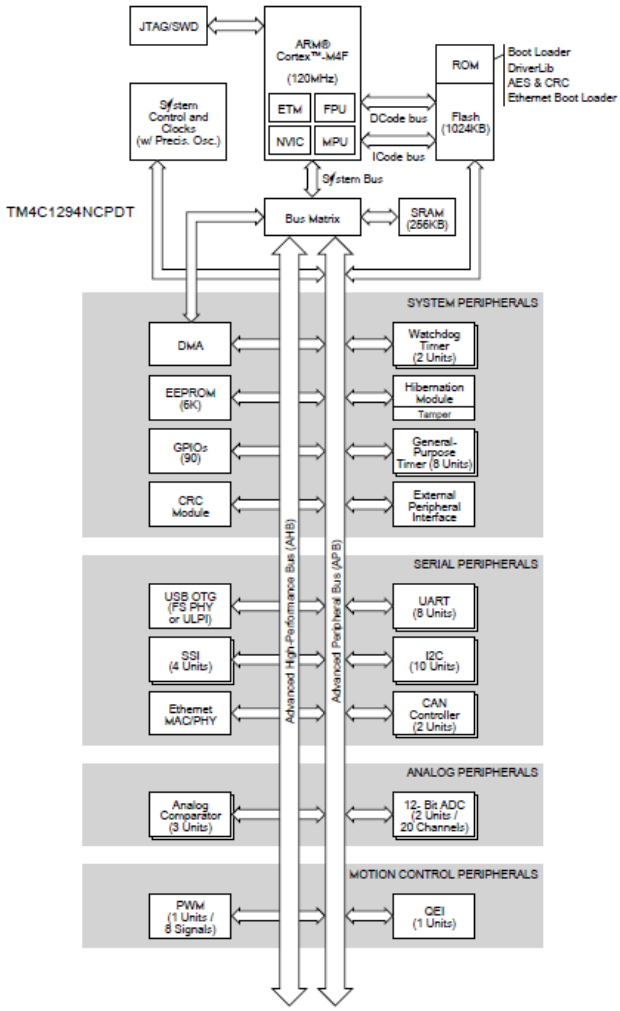


Fig 27. Diagrama de bloques de alto nivel del microcontrolador. [Fuente: [12]]

Memoria interna.

El microcontrolador tiene una memoria:

- SRAM de 256 KB
- ROM interna
- Memoria flash de 1024 kB
- EEPROM de 6 KB.

La memoria interna tiene el siguiente diagrama de bloques:

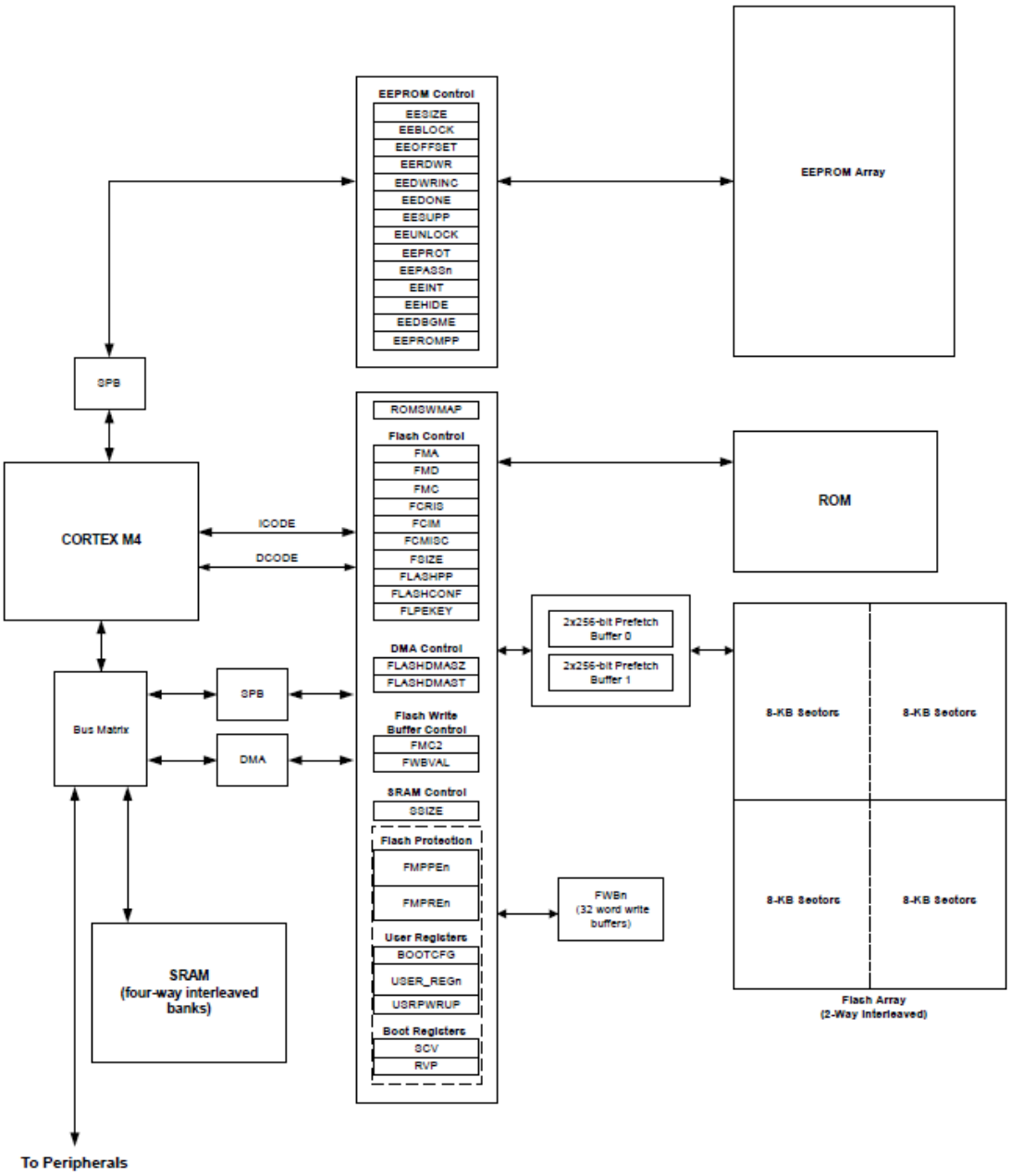


Fig 28. Diagrama de bloques de la memoria interna. [Fuente: [12]]

UART.

UART es una función estándar de comunicación para la transmisión y recepción de datos entre los diferentes elementos que se encuentra en el microcontrolador.

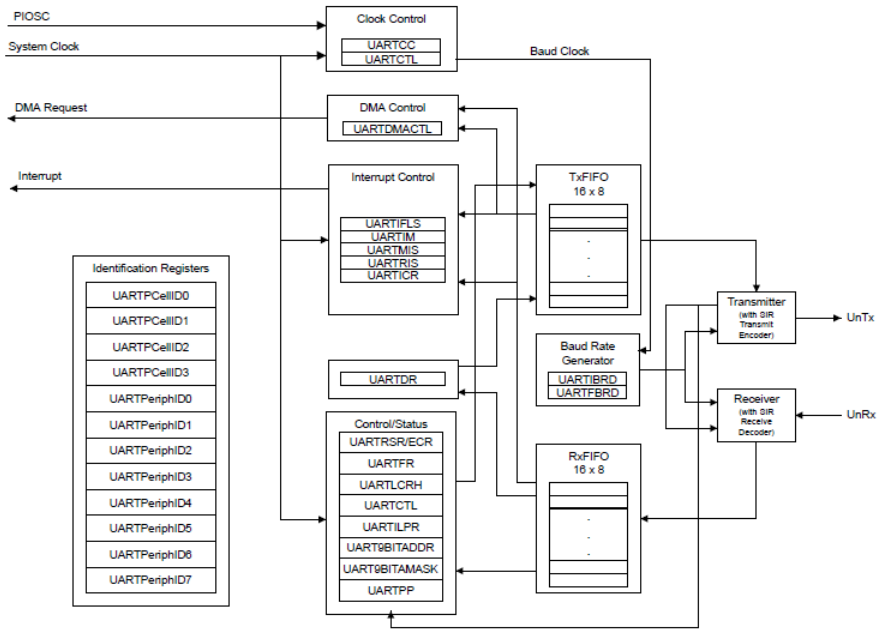


Fig 29. Diagrama de bloques UART. [Fuente: [12]]

I²C

Bus que permite la transferencia bidireccional a través de un diseño de dos hilos.

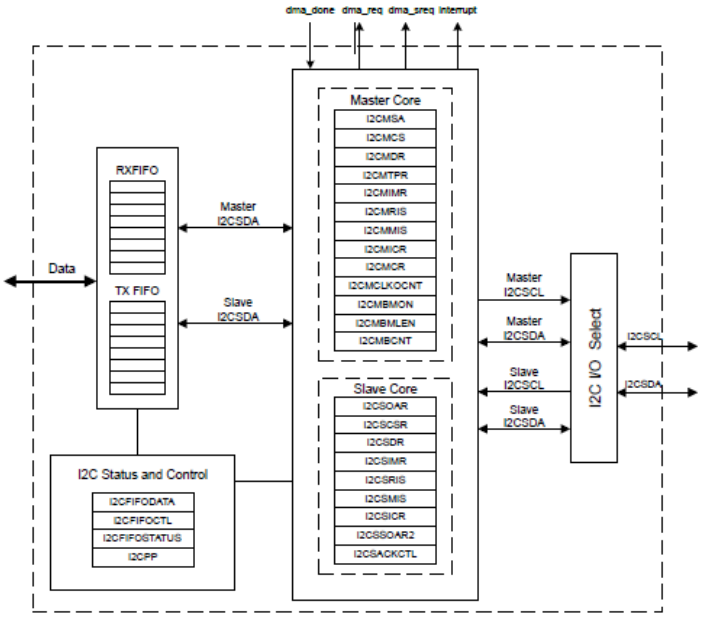


Fig 30. Diagrama de bloques I²C. [Fuente: [12]]

PWM.

Herramienta para codificar digitalmente señales analógicas. Se utilizan contadores de alta resolución para generar una onda cuadrada se codifica para modular una señal analógica.

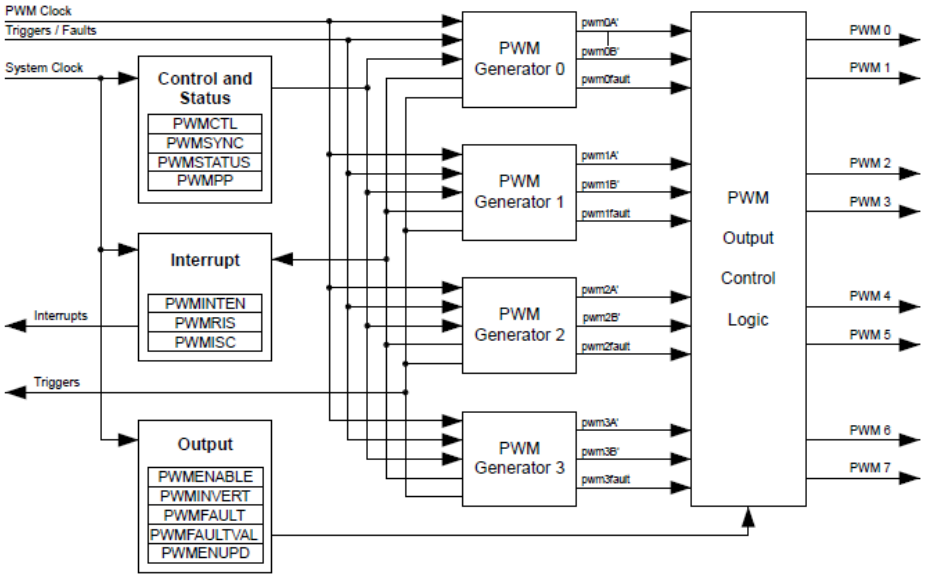


Fig 31. Diagrama de bloques de los módulos de PWM. [Fuente: [12]]

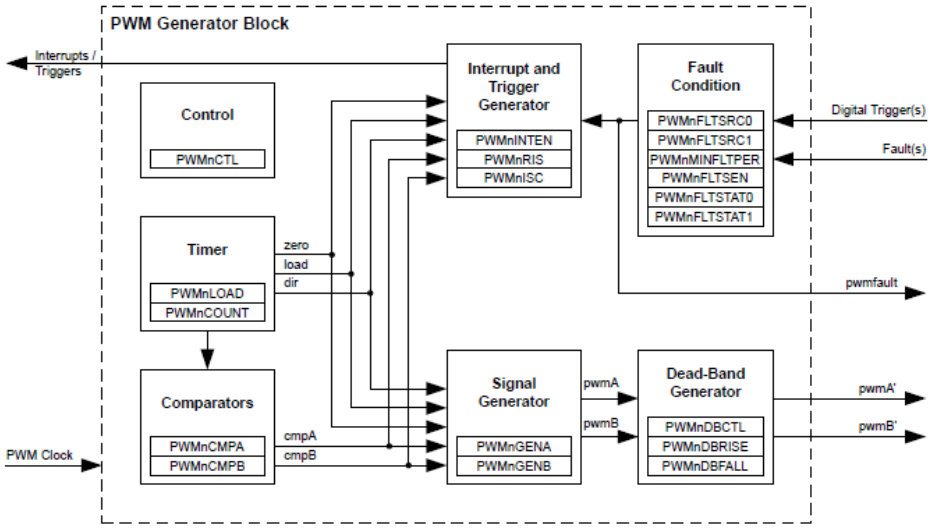


Fig 32. Diagrama de bloques del generador de PWM. [Fuente: [12]]

GPIO.

El GPIO es un módulo compuesto por 15 bloques físicos, cada uno corresponde a un puerto GPIO [Puerto A, Puerto B, …, Puerto Q]. El módulo GPIO este compuesto por 90 pines de E/S configurables, dependiendo de las funciones deseadas para la aplicación.

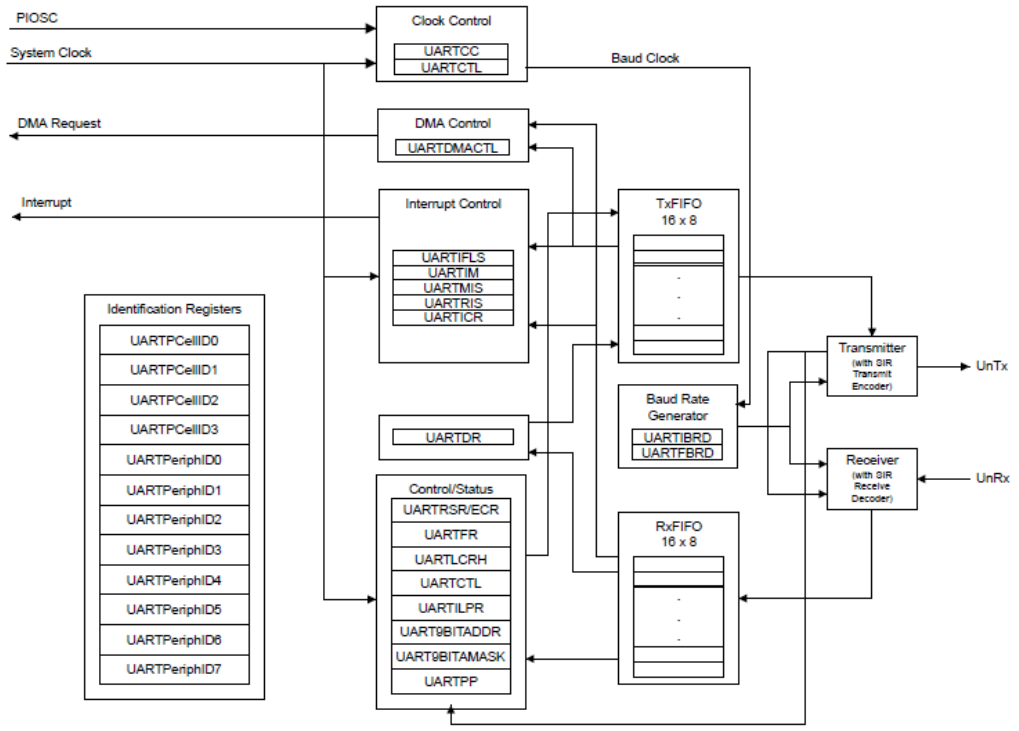


Fig 33. Diagrama de bloques del UART. [Fuente: [12]]

ADC.

TM4C1294NCPDT dispone de dos módulos ADC, estos tienen 20 canales de entrada analógica, ADC de precisión de 12 bits, sensor de temperatura interno, velocidad de muestreo máxima de dos millones de muestras por segundo, entre otras características.

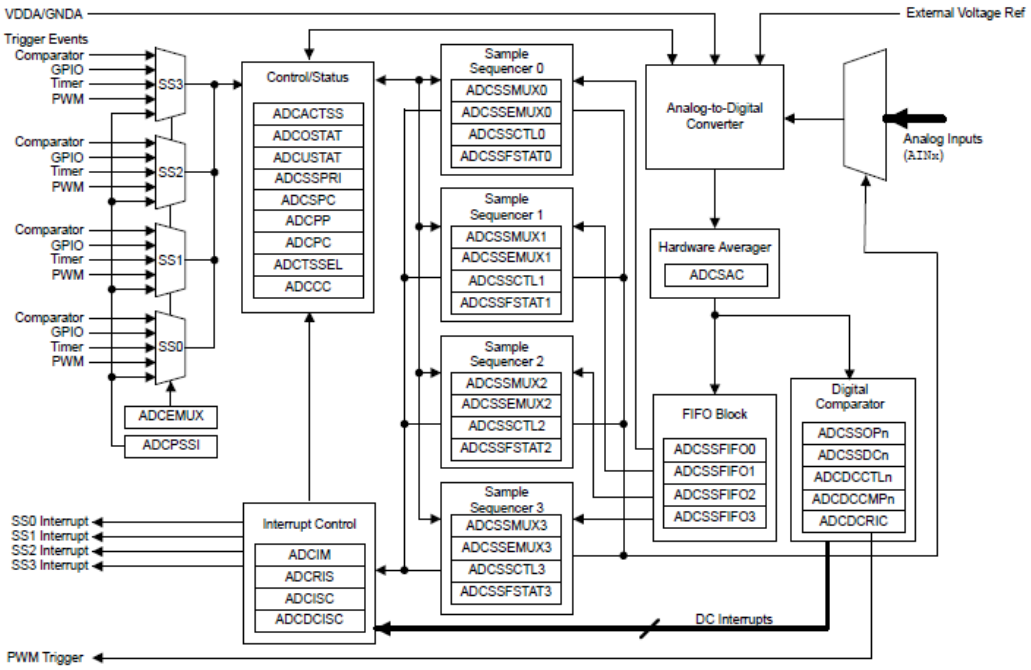


Fig 34. Diagrama de bloques del módulo ADC. [Fuente: [12]]

Características eléctricas.

Las ratios máximas que puede soportar el microcontrolador son:

Parámetro	**Valor (Mín./Máx.)**		**Unidad**
VDD tensión de alimentación	0	4	V
VDDA tensión de alimentación	0	4	V
VBAT tensión de la batería de alimentación	0	4	V
VBAT pendiente tensión de la batería de alimentación	0	0,7	V/µs
Tensión de entrada	-0,3	4	V
Corriente máxima por pin de salida	–	64	mA

Tabla 4. Características eléctricas del microcontrolador. [Fuente: [12]]

Procesador Cortex-M4F.

Características del procesador.

Cortex-M4F proporciona un procesador de alto rendimiento y bajo coste que cumple los requisitos del sistema de implementación de memoria mínima, número reducido de pines y bajo consumo de energía, al tiempo de ofrecer un rendimiento excepcional y un sistema de interrupciones. Entre sus características incluye:

- Arquitectura de 32 bits optimizada para aplicaciones integradas de tamaño reducido.
- Funcionamiento a 120 MHz; rendimiento de 150 DMPIS.
- Extraordinario rendimiento de procesamiento combinado con una rápida gestión de las interrupciones.
- Unidad de coma flotante de precisión única compatible con IEEE754.
- Unidad de procesamiento vectorial SIMD de 16 bits.
- La ejecución rápida de código permite un reloj de procesador más lento o aumenta el tiempo en modo de reposo.
- Arquitectura Harvard caracterizada por buses separados para instrucciones y datos.
- Núcleo de procesador, sistema y memoria eficiente al procesamiento digital de señales.
- Aritmética de saturación para el procesamiento de señales.
- Consumo de energía bajo con modos de reposo integrados.

Diagrama de bloques del procesador

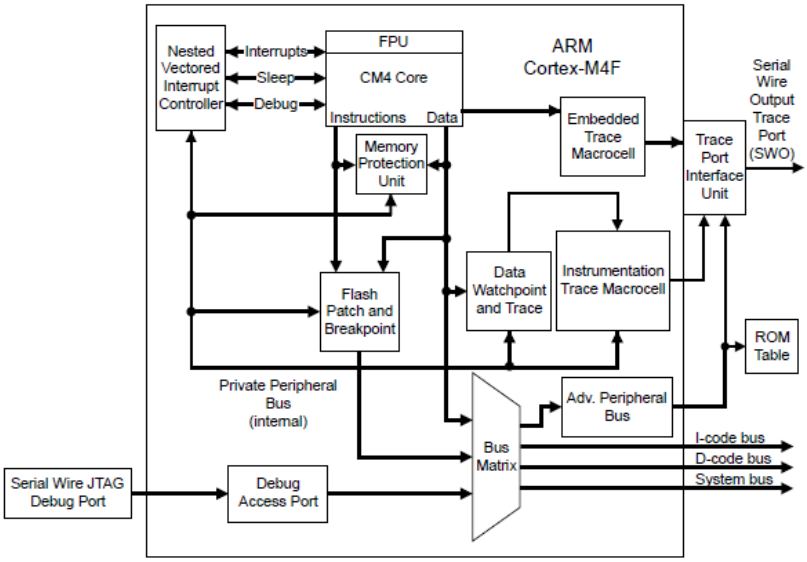


Fig 35. Diagrama de bloques del procesador. [Fuente: [12]]

Funciones del procesador.

En este apartado se desarrollan las funciones que proporciona Cortex-M4:

- Systick: proporciona un contador de 24 bits con un mecanismo de control flexible.
- Nested Vectored Interrupt Controller: facilita la gestión de interrupciones, controla la gestión de la energía y un control de registro implementado en el sistema.
- System Control Block: ofrece el control y la información del sistema, incluyendo configuración, control y reportes.
- Memory Protection Unit: protección sobre regiones de la memoria, limitando el acceso y la exportación.
- Floating-Point Unit: habilita las operaciones de ruma, resta, multiplicación, división y raíz cuadrada con números en coma flotante.

Para más información de las diferentes funciones consultar la ficha técnica en los anexos.

6.1.4 Otros elementos.

El diseño de la bancada se finaliza añadiendo dispositivos para el desarrollo de las aplicaciones, alguno de ellos: un generador de señal, voltímetro, amperímetro y osciloscopio. Estos dispositivos se adaptarán a las necesidades de la aplicación.

Generador de señales.

Un generador de señales puede generar diferentes tipos de forma de onda y frecuencias. Estas señales pueden ser utilizadas en una variedad de aplicaciones, como pruebas y mediciones en laboratorios, desarrollo de productos, etc.

El generador de señales puede generar diferentes tipos de formas de onda, como sinusoidales, cuadradas, etc. También pueden ajustar la frecuencia, amplitud, fase y otras características. El generador de señales es una herramienta fundamental en el campo de la electrónica, proporcionando señales controladas y precisas.

Osciloscopio.

Un osciloscopio es un dispositivo de medición electrónico que se utiliza para visualizar y analizar señales eléctricas en el dominio del tiempo. Este dispositivo muestra gráficamente la forma de onda de una señal eléctrica, permitiendo observar su amplitud, frecuencia, fase, entre otros parámetros.

En la pantalla de visualización, donde el eje horizontal representa el tiempo y el eje vertical representa la amplitud de la señal, los usuarios pueden ajustar tanto la escala de tiempo como la de amplitud, para examinar detalles específicos de la señal.

El osciloscopio es una herramienta esencial en la electrónica, proporcionando una forma visual de analizar y comprender el comportamiento de las señales eléctricas en el tiempo.

Voltímetro y amperímetro.

Un voltímetro es un dispositivo de medición utilizado para conocer la diferencia de potencial entre dos puntos de un circuito, se expresa en voltios. Por otro lado, un amperímetro es un dispositivo que se utiliza para medir la intensidad de la corriente eléctrica en un circuito.

Todos los dispositivos descritos en este apartado son esenciales para la medición, análisis y generación de señales eléctricas, ideales para el análisis del comportamiento de la bancada diseñada.

7 Desarrollo de aplicaciones.

7.1 Instalación CCS y TI-RTOS.

Primero se debe realizar la instalación del entorno de desarrollo, para su instalación se tendrá que seguir esta serie de instrucciones:

1. Se accede al siguiente enlace: https://www.ti.com/tool/download/CCSTUDIO
   Se abrirá la siguiente ventana con las diferentes versiones disponibles del entorno de desarrollo, en este proyecto se trabaja con la versión 12.3.0 lanzada el 07 de abril del 2023.

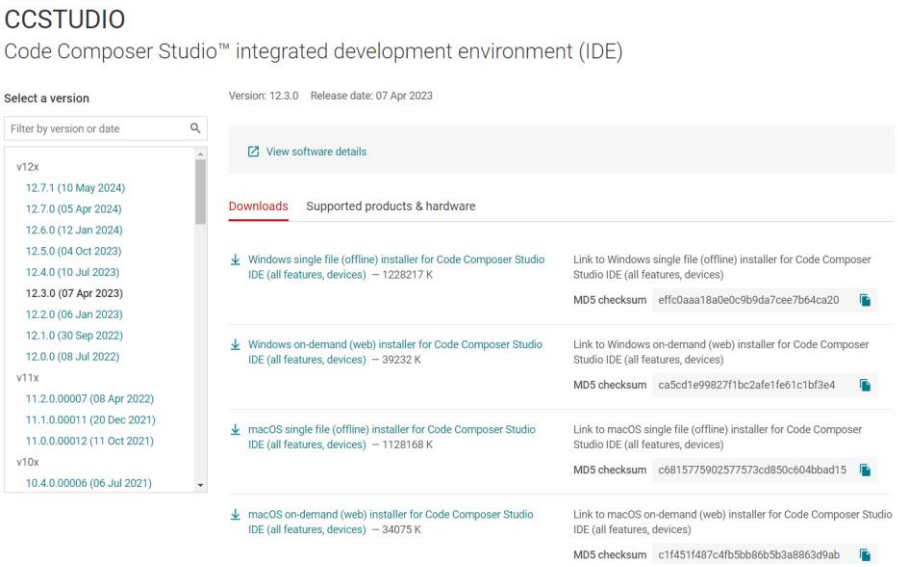


Fig 36. Ventana de descarga de CCS. [Fuente: EP]

   Se escoge la versión compatible con nuestro sistema operativo y se descarga el archivo.
2. Cuando finalice la descarga, se descomprime el archivo y se ejecuta la opción marcada.

Fig 37. Archivo ejecutable de CCS. [Fuente: EP]

A continuación, se permite los permisos necesarios para poder instalar el entorno de desarrollo.

3. Aparecerá una ventana donde se selecciona la opción *Next*, dando paso a una nueva, donde se aceptan los términos y condiciones para la instalación del CCS. Y se vuelve a seleccionar la opción *Next*. A continuación, se puede seleccionar el directorio donde se desea instalar el CCS, se recomienda no modificar la ubicación predeterminada, finalmente, se selecciona la opción recomendada para la instalación y el producto que se desea utilizar, en este caso, la opción remarcada en la imagen.

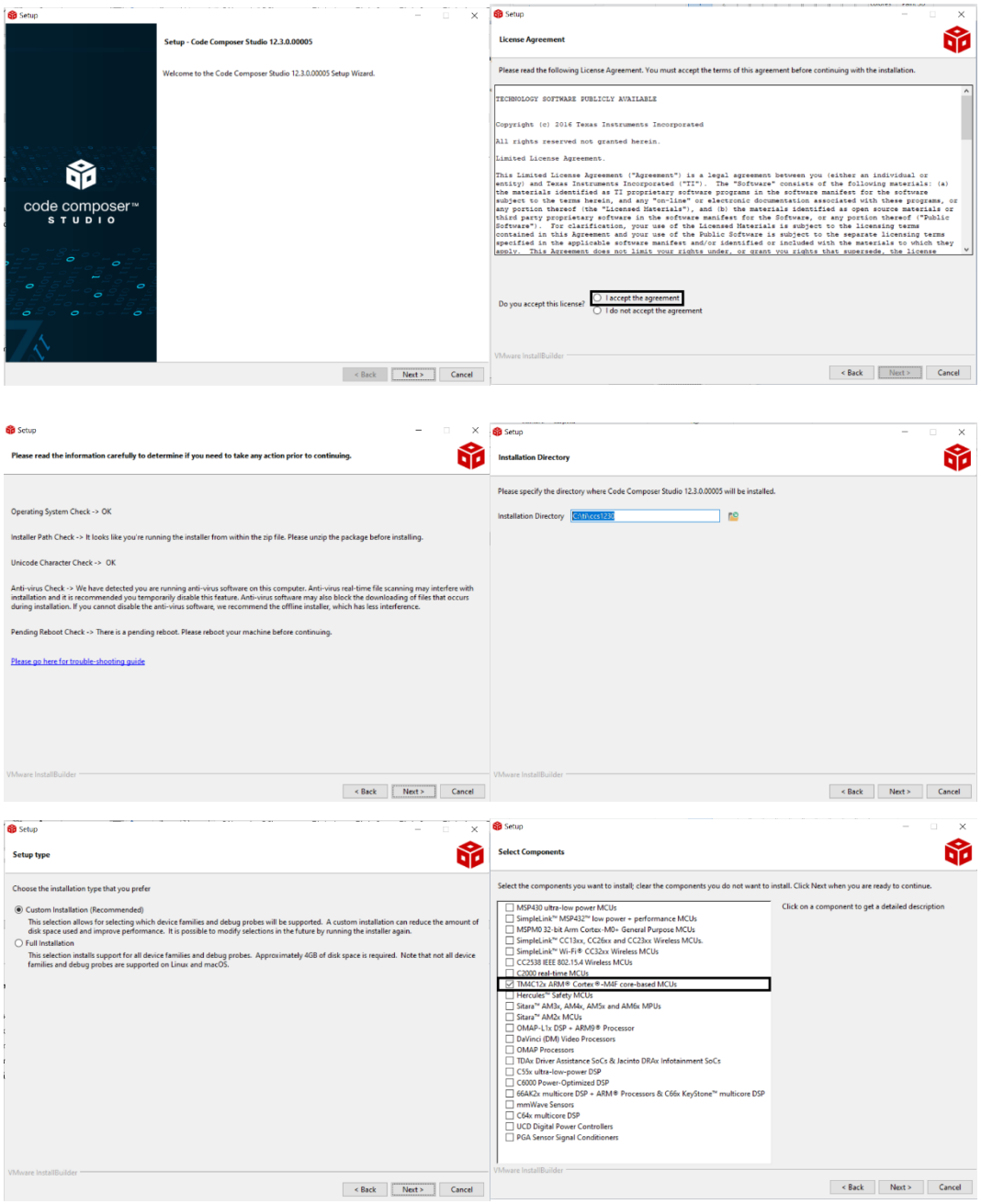


Fig 38. Pasos a seguir para la instalación del entorno de desarrollo. [Fuente: EP]

4. Al acabar la instalación, abriremos el programa CCS, donde aparecerá la siguiente ventana.

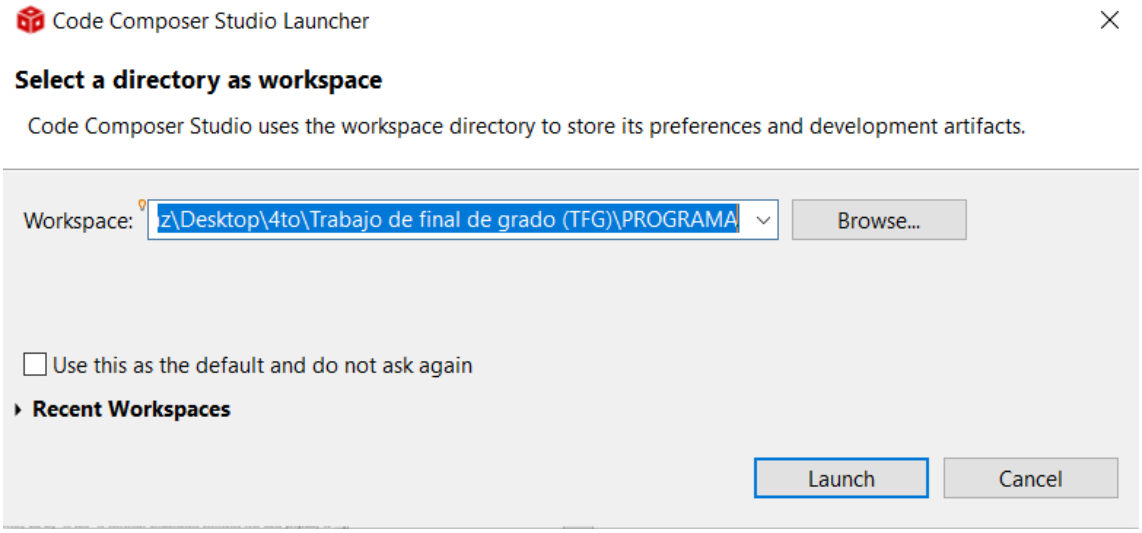


Fig 39. Ventana para seleccionar el entorno de trabajo. [Fuente: EP]

Se recomienda crear una carpeta para el directorio de los proyectos con fácil acceso, por ejemplo, una carpeta en el escritorio. Una vez seleccionado el entorno de trabajo en la opción *Workspace*, se selecciona la opción *Launch*.

5. Finalmente, aparece la siguiente ventana donde se pueden seleccionar las diferentes opciones ya mostradas en el punto *7.1.1 CCS*, como, por ejemplo, la creación, importación o búsqueda de un proyecto.

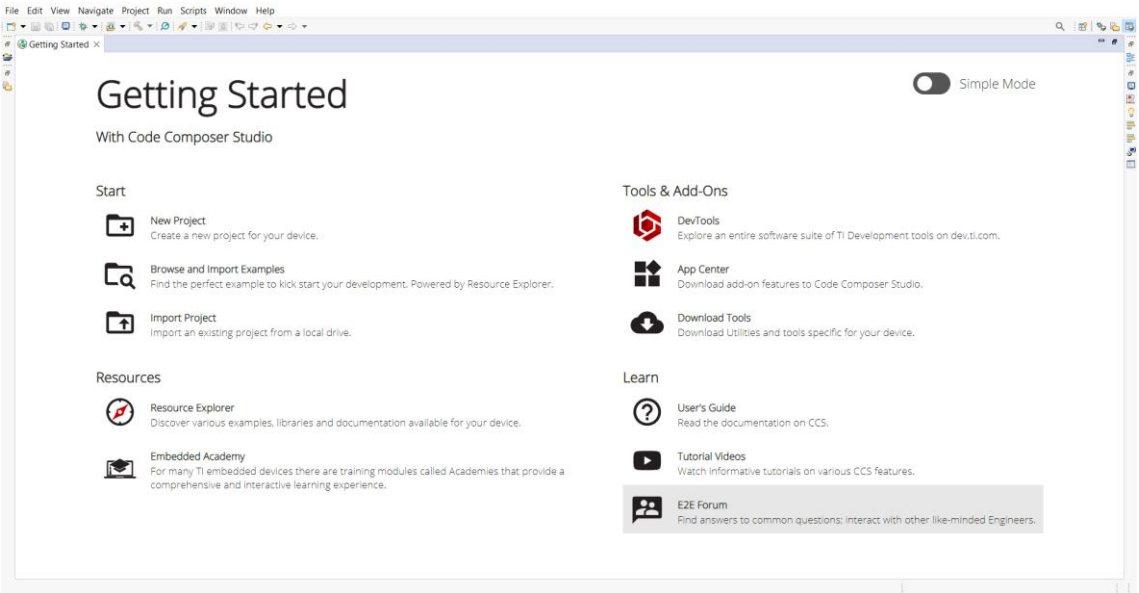
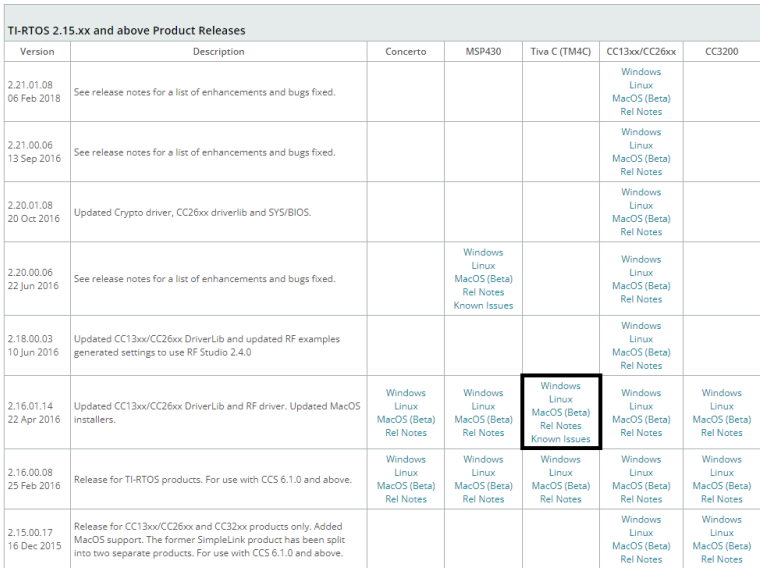


Fig 40. Ventana Getting Started. [Fuente: EP]

Con este último paso se finaliza la instalación del entorno de desarrollo CCS. Ahora se procede con la instalación del SO, para su instalación se seguirán esta serie de instrucciones:

1. Se accede el siguiente enlace:

   Se abrirá la siguiente ventana con tabla que expone las diferentes versiones disponibles del SO, en este proyecto se trabaja con un microcontrolador del fabricante TI de la familia Tiva C, es decir, que es necesario que se seleccione TI-RTOS compatible con el microcontrolador del proyecto y sistema operativo del ordenador, es decir, la opción marcada en la imagen, que es la última versión disponible del TI-RTOS.



TI-RTOS 2.15.xx and above Product Releases

Version	Description	Concerto	MSP430	Tiva C (TM4C)	CC13xx/CC26xx	CC3200
2.21.01.08 06 Feb 2018	See release notes for a list of enhancements and bugs fixed.				Windows Linux MacOS (Beta) Rel Notes	
2.21.00.06 13 Sep 2016	See release notes for a list of enhancements and bugs fixed.				Windows Linux MacOS (Beta) Rel Notes	
2.20.01.08 20 Oct 2016	Updated Crypto driver, CC26xx driverlib and SYS/BIOS.				Windows Linux MacOS (Beta) Rel Notes	
2.20.00.06 22 Jun 2016	See release notes for a list of enhancements and bugs fixed.		Windows Linux MacOS (Beta) Rel Notes Known Issues		Windows Linux MacOS (Beta) Rel Notes	
2.18.00.03 10 Jun 2016	Updated CC13xx/CC26xx DriverLib and updated RF examples generated settings to use RF Studio 2.4.0				Windows Linux MacOS (Beta) Rel Notes	
2.16.01.14 22 Apr 2016	Updated CC13xx/CC26xx DriverLib and RF driver. Updated MacOS installers.	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes Known Issues	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes
2.16.00.08 25 Feb 2016	Release for TI-RTOS products. For use with CCS 6.1.0 and above.	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes
2.15.00.17 16 Dec 2015	Release for CC13xx/CC26xx and CC32xx products only. Added MacOS support. The former SimpleLink product has been split into two separate products. For use with CCS 6.1.0 and above.				Windows Linux MacOS (Beta) Rel Notes	Windows Linux MacOS (Beta) Rel Notes

Fig 41. Ventana para la descarga del TI-RTOS. [Fuente: EP]

2. Al finalizar la descarga, se muestra una ventana donde se selecciona la opción *Next*, se aceptan los términos y condiciones, y la carpeta donde se desea instalar, se recomienda no modificar este directorio. Por último, solo queda esperar a que se acabe la instalación del producto.

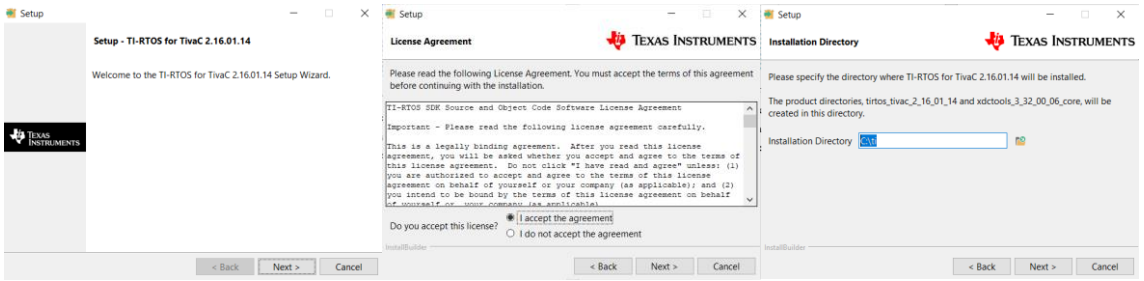


Fig 42. Pasos a seguir para la instalación del SO. [Fuente: EP]

7.2 Creación de un proyecto con el SO.

Para poder trabajar con el SO que nos proporciona TI, primero se tiene que crear un proyecto que permita trabajar con las herramientas que proporciona TI-RTOS. Para ello, es necesario seguir esta serie de instrucciones:

1. Se ejecuta el programa CCS y aparecerá la ventana *Getting Started*, se selecciona la opción de *New Project*. En caso de no aparecer esta ventana, se realiza clic en *View* de la barra de herramientas y se selecciona *Getting Started*.

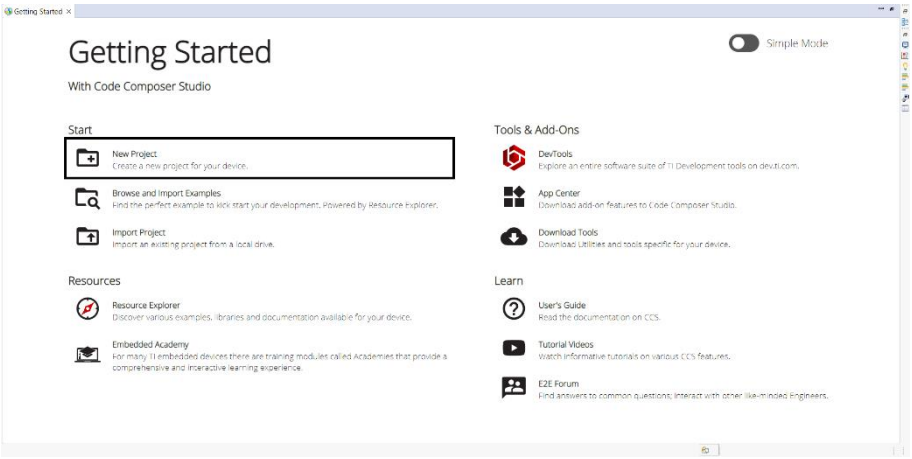


Fig 43. Ventana Getting Started seleccionando la opción New Project. [Fuente: EP]

2. Al seleccionar la opción *New Project*, se muestra la siguiente ventana, donde se selecciona el microcontrolador con el que se desea trabajar, en la opción *Target*. El tipo de conexión que hay entre el ordenador y el dispositivo, en la opción *Connection*, el nombre del proyecto, *Project Name*. Y, por último, se selecciona el compilador con el que se trabaja durante el proyecto, *Compiler version*.

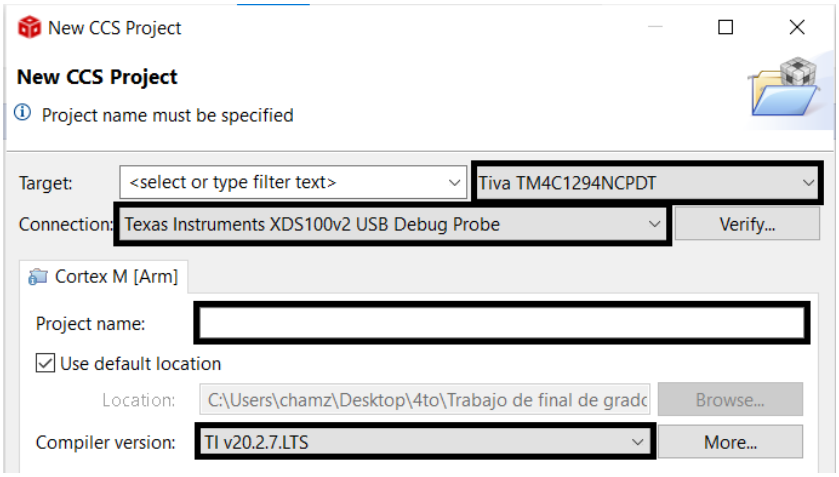


Fig 44. Parte superior de la ventana New CCS Project. [Fuente: EP]

En la parte inferior de la ventana, en la opción *Project templates and examples*, se selecciona la opción marcada en la siguiente imagen. Finalmente, se finaliza la creación del proyecto a través del botón *Finish*.

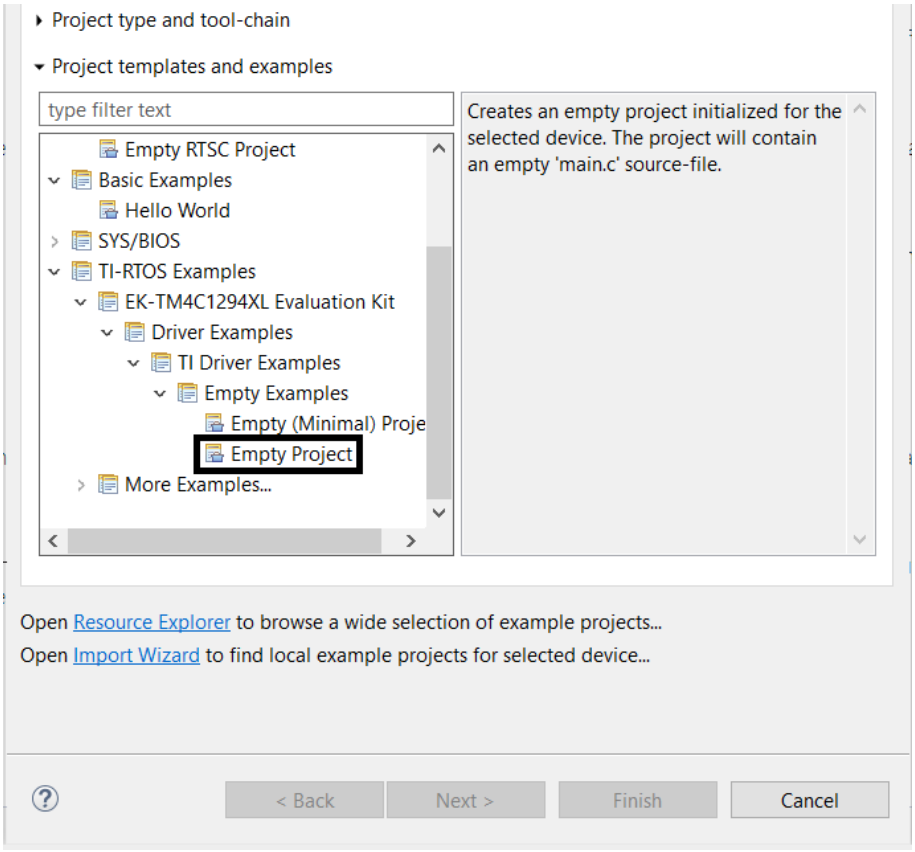


Fig 45. Parte inferior de la ventana New CCS Project. [Fuente: EP]

3. Una vez finalizado el proceso de creación del proyecto, CCS se encarga de mostrar el nuevo proyecto en la ventana *Project Explorer*, el nombre de la carpeta que se utiliza para la explicación de los diferentes elementos que dispone el SO es TI-RTOS, como se muestra en la imagen.

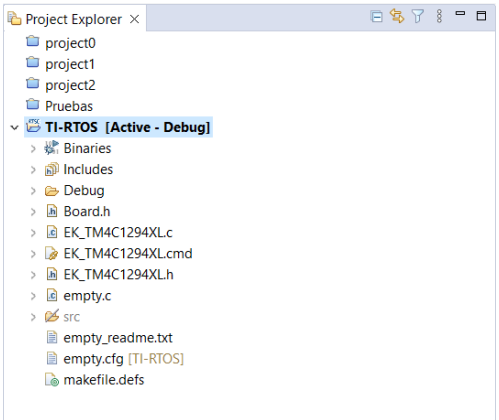


Fig 46. Ventana Project Explorer con el proyecto creado. [Fuente: EP]

7.3 Funcionamiento del SO.

Después de realizar todos los pasos previos, se puede trabajar en la aplicación que se desea hacer, pero no antes de entender las herramientas más importantes que nos proporciona el SO.

Para ello, se utilizará el proyecto creado en el punto anterior, y explicando este pequeño ejemplo que proporciona TI para el dispositivo TM4C1294NCPDT. Este proyecto se divide en tres partes:

1. La primera es la cabecera donde se incluyen todos los archivos y las variables que se utilizan para el ejemplo. En la imagen, se explica el contenido del código línea por línea.

```
/* XDCtools Header files */
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el SO en tiempo real.

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

#define TASKSTACKSIZE   512 //Se define una macro con un valor de 512.

Task_Struct task0Struct; //Se declara una estructura de Task.
Char task0Stack[TASKSTACKSIZE]; //Se declara una matriz
                                //con un tamaño especificado por TASKSTACKSIZE
```

Fig 47. Explicación de la cabecera del ejemplo TI-RTOS. [Fuente: EP]

2. La segunda parte es la definición de la función *heartBeatFxn*, donde se define que hará la tarea durante su ejecución. Es una tarea sencilla, donde se modifica el estado de un LED específico del dispositivo.

```
void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos argumentos
                                        //La función tiene tipo de retorno void,
                                        //lo que significa que no devuelve ningún valor
{
    while (1) { //Se ejecuta un bucle infinito para el funcionamiento de la Task
        Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de la tarea
                                        //durante un periodo de tiempo determinado.
        GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle para cambiar el estado,
                                    //es decir, un parpadeo de un LED específico del microcontrolador.
    }
}
```

Fig 48. Explicación de la Task que se ejecuta en el ejemplo TI-RTOS. [Fuente: EP]

3. Por último, encontramos el *main* del programa.

```
int main(void)  //Se declara la función principal del programa
{
    Task_Params taskParams; //Se declara una variable del tipo Task_Params,
                            //que se utiliza para especificar los parámetros de configuración de una tarea.
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

    /* Construct heartBeat Task  thread */
    Task_Params_init(&taskParams);  //Se inicializa la estructura con los valores predeterminados de los parámetros
                                    //de la Task. Esto asegura que todos los parámetros de la Task se inicialicen
                                    //correctamente antes de configurarlos de manera individual.
    taskParams.arg0 = 1000; //Se establece el primer argumento de la Task.
    taskParams.stackSize = TASKSTACKSIZE; //Se establece el tamaño de la pila de la Task.
    taskParams.stack = &task0Stack; //Se establece la pila de la Task.
    Task_construct(&task0Struct, (Task_FuncPtr)heartBeatFxn, &taskParams, NULL);
    //Se contruye la tarea utilizando los parámetros especificados,
    //se le pasa la dirección de la estructura de la Task,
    //la función que la Task ejecutará,
    //los parámetros de la Task, y, por último, el NULL para indicar que no se está utilizando ninguna Task
    //como instancia de un objeto.



     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.


    System_printf("Starting the example\nSystem provider is set to SysMin. "
                  "Halt the target to view any SysMin contents in ROV.\n");
    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha ejecutado correctamente.
}
```

Fig 49. Explicación del programa principal que se ejecuta en el ejemplo TI-RTOS. [Fuente: EP]

Cuando se realiza la ejecución del programa, a través del botón *Debug*, podremos observar que el comportamiento del LED0, con el nombre de D1 en el dispositivo que se utiliza en el proyecto.

7.3.1 Configuración *Task*.

A continuación, se realiza una copia del proyecto de ejemplo TI-RTOS, y se nombrará TI-RTOS2. Para realizar este paso, en la ventana *Project Explorer*, se realiza clic derecho y se selecciona primero la opción de copiar, y a continuación la opción de pegar.

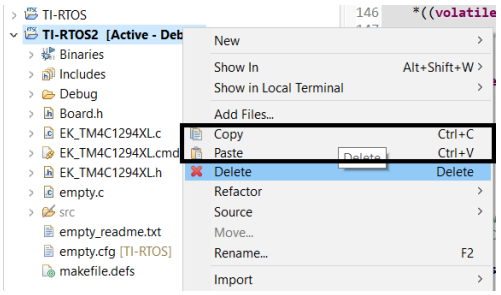


Fig 50. Opciones de copiar y pegar en la ventana Project Explorer. [Fuente: EP]

A continuación, se utilizará el *kernel* del SO. Para poder acceder a esta opción se deben seguir esta serie de pasos:

1. Se selecciona el archivo del proyecto con el nombre de *empty.cgf [TI-RTOS]*, clic derecho y se selecciona la opción *Open With* → *XGCONF*. Como se puede observar en la imagen.

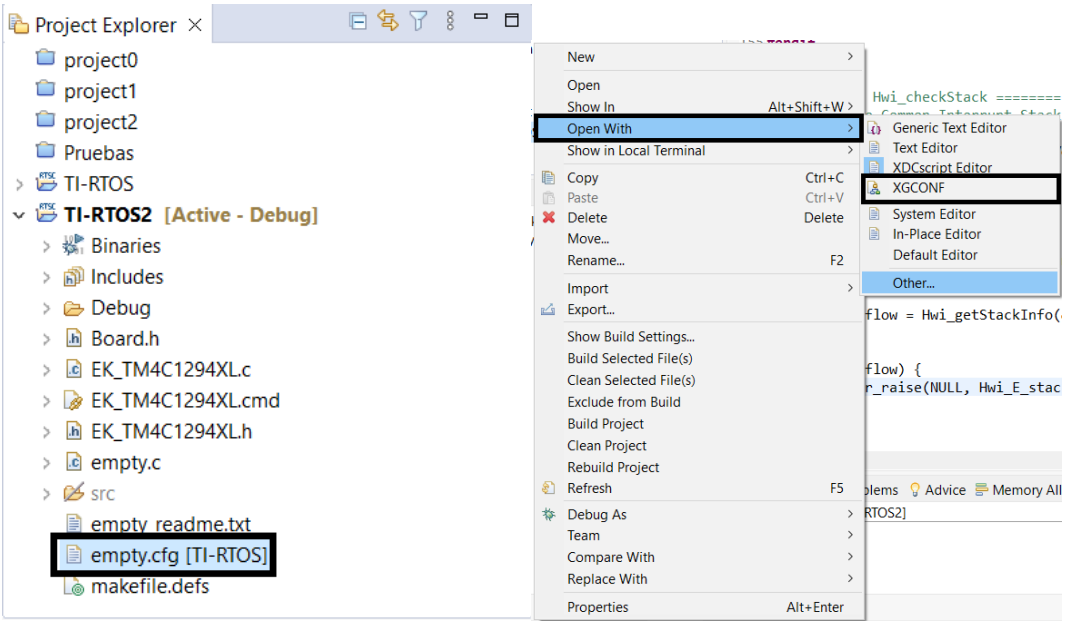


Fig 51. Pasos a seguir para abrir el kernel del proyecto. [Fuente: EP]

2. Se abre dos ventanas, la primera es el archivo que se acaba de seleccionar y la segunda ventana, *Outline*, se encuentran las opciones del *kernel* habilitadas en este proyecto.

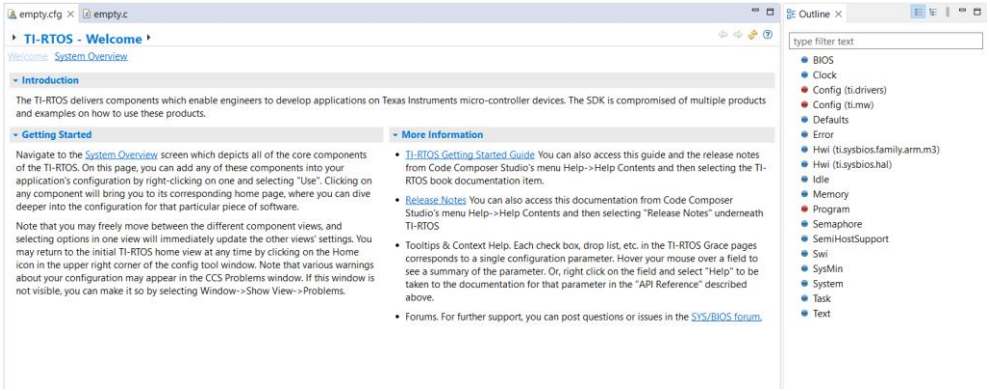


Fig 52. Ventana del kernel de TI-RTOS. [Fuente: EP]

Una vez que se ha accedido al *kernel* del proyecto, es hora de trabajar sobre el proyecto. El objetivo en este punto es crear la misma *Task* sin escribir el código en el archivo *empty.c* y solo utilizando el *kernel*.

Para ellos se realizarán esta serie de pasos:

1. Se realizan las siguientes modificaciones en el código del archivo *empty.c*, eliminando todas las líneas relacionas con la *Task* dentro de la función *main*. También, se eliminan todas las variables declaradas en la cabecera del programa.

```
int main(void)  //Se declara la función principal del programa
{
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    System_printf("Starting the example\nSystem provider is set to SysMin. "
                  "Halt the target to view any SysMin contents in ROV.\n");
    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha ejecutado correctamente.
}
```

Fig 53. Contenido del main sin las deficiniones de las variables de la Task. [Fuente: EP]

2. Se ejecuta el programa y se puede observar que en esta ocasión no hay ningún cambio en el LED0, permanece encendido, es decir, la *Task* definida en la parte superior del archivo *empty.c* no se ejecuta en ningún momento.
3. Realizada la comprobación, se procede a crear la *Task* en el *kernel*. Para ellos volvemos a seleccionar el archivo abierto anteriormente *empty.cfg [TI-RTOS]*. En la ventana *Outline*, se selecciona la opción *Task*, apareciendo la siguiente ventana.

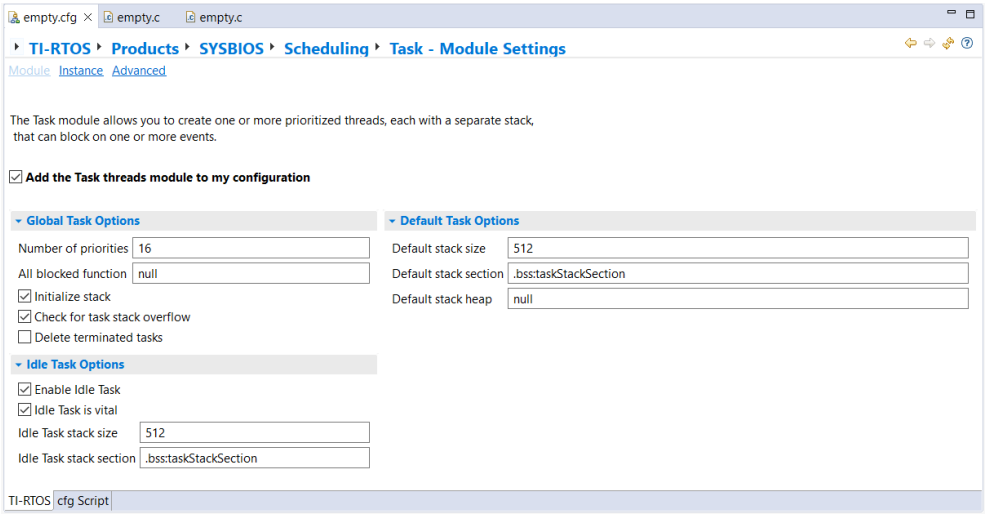


Fig 54. Kernel Task, opción: Module. [Fuente: EP]

Se encuentran varias opciones en esta ventana de las cuales no se realiza ninguna modificación. Al crear una *Task* se realizan las modificaciones necesarias para cada una de ellas de manera individual. En la opción *Advanced* se encuentra una pequeña explicación sobre todas estas variables.

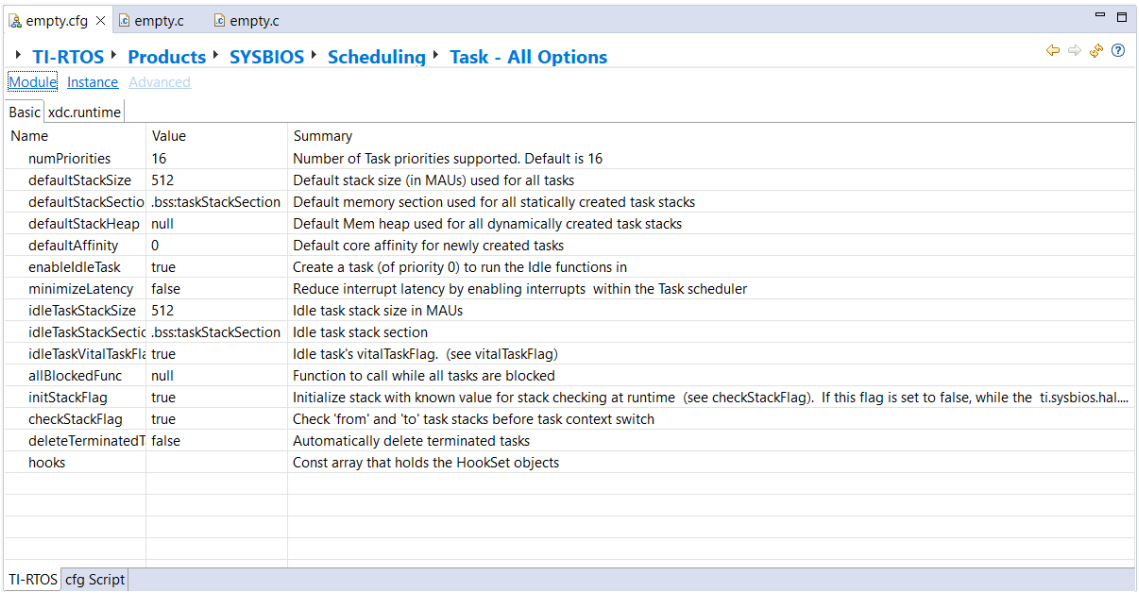


Fig 55. Kernel del Task, opción: Advanced. [Fuente: EP]

4. Se selecciona la opción de *Instance* del *kernel Task*. Se selecciona la opción *Add*, para crear la tarea que realiza el parpadeo del LED0. Una vez creada la *Task*, se realizan dos modificaciones.

   La primera se encuentra en *Function*, donde se define el nombre de la función, la segunda modificación se encuentra en *Argument 0*, donde se define la duración de la tarea, estas modificaciones se pueden observar en la siguiente imagen.

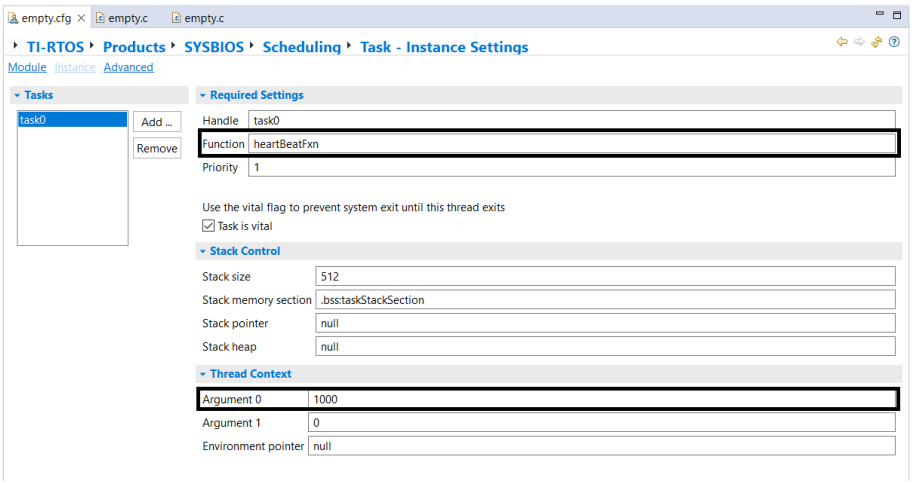


Fig 56. Kernel Task, opción: Instance. [Fuente: EP]

Por último, se realiza un guardado del proyecto para actualizar el proyecto para la detección de los cambios en el *kernel* y se ejecuta el programa. Se puede observar que el LED0 tiene el mismo comportamiento que el ejemplo que aporta TI-RTOS.

7.3.2 Configuración Swi y Hwi.

Se realiza otra copia del último proyecto con el que se ha trabajado. Esta vez, se configurarán dos herramientas habilitadas en el *kernel* de TI-RTOS, Swi y Hwi. Se siguen todos los pasos iniciales descritos en el punto 8.3.1 Configuración *Task*, para abrir el *kernel* de este proyecto.

Una vez en el *kernel*, seguiremos estos pasos para crear un *thread* del tipo Hwi:

1. En la ventana *Outline*, se selecciona la opción Hwi (ti.sysbios.hal), apareciendo la siguiente ventana.

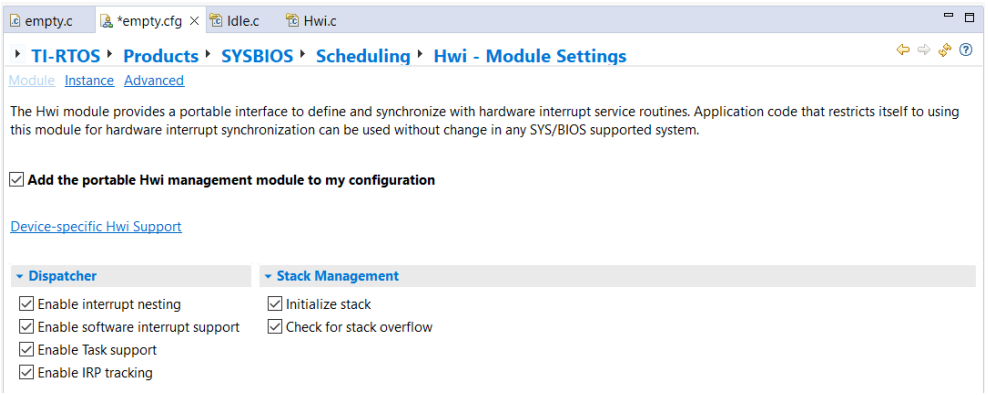


Fig 57. Kernel Hwi, opción: Module. [Fuente: EP]

Se encuentran varias opciones habilitadas en esta ventana de las cuales no se realiza ninguna modificación. En la opción *Advanced* se encuentra una lista de las opciones habilitadas del dispositivo.

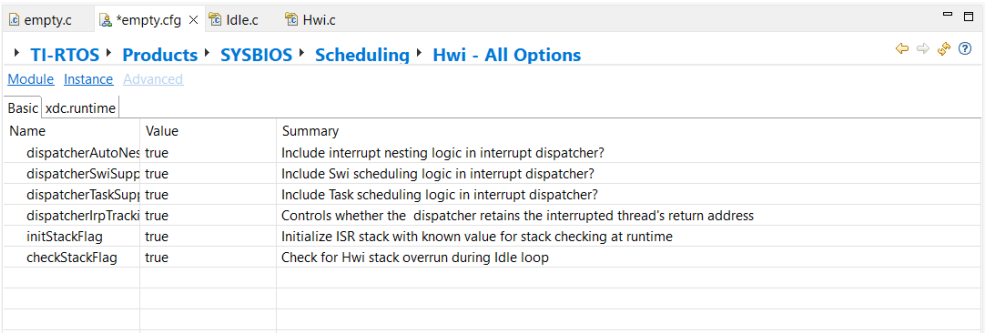


Fig 58.Kernel Hwi, opción: Advanced. [Fuente: EP]

2. En la opción *Instance* se define el nombre de la función y la prioridad de este *thread*, debe ser mayor de 15, cuanto mayor sea el número mayor es la prioridad del *thread.*

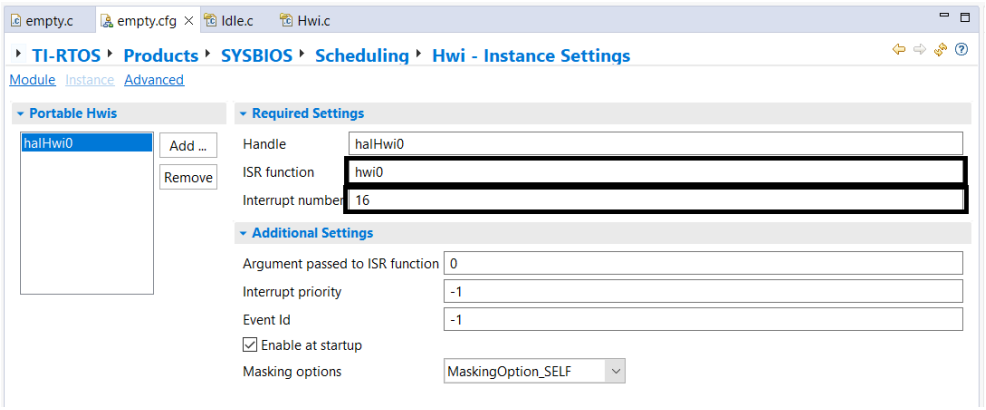


Fig 59. Kernel Hwi, opción: Instance. [Fuente: EP]

Una vez en el *kernel*, seguiremos estos pasos para crear un *thread* del tipo Swi:

3. En la ventana *Outline*, se selecciona la opción Swi, apareciendo la siguiente ventana.

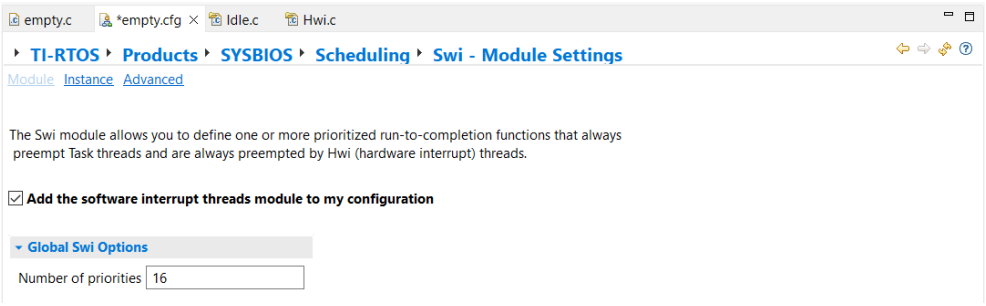


Fig 60. Kernel Swi, opción: Module. [Fuente: EP]

Se encuentran varias opciones habilitadas en esta ventana de las cuales no se realiza ninguna modificación. En la opción *Advanced* solo se encuentra el número de prioridades Swi que soporta el dispositivo.

4. En la opción *Instance* se define el nombre de la función y la prioridad de este *thread*, debe ser mayor de 15, cuanto mayor sea el número mayor es la prioridad del *thread.*

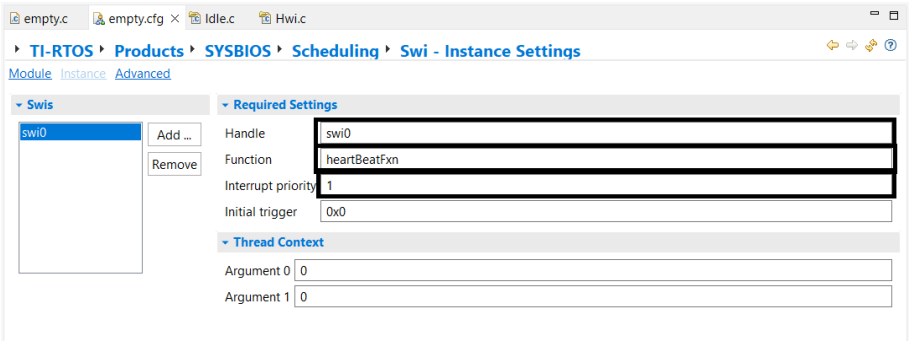


Fig 61. Kernel Swi, opción: Instance. [Fuente: EP]

A continuación, se realizan una serie de cambios en el archivo *empty.c*, para el uso de las nuevas herramientas añadidas en el *kernel*. Para la modificación del código se seguirán esta serie de pasos:

1. Se añade una nueva variable en la cabecera, que se utilizará en el *thread* Swi. También, se incluye dos archivos.

```
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el SO en tiempo real.
#include <ti/sysbios/knl/Swi.h>
#include <ti/sysbios/hal/Hwi.h>

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

volatile Bool blink = TRUE;
```

Fig 62. Cabecera del archivo empty.c. [Fuente: EP]

2. Se modifica el *thread heartBeatFxn*, para que el comportamiento del LED0 dependa de la variable *blink*, pasando del parpadeo, al estar encendido de manera permanente.

```
void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos argumentos
                                        //La función tiene tipo de retorno void,
                                        //lo que significa que no devuelve ningún valor
{
    while (1) { //Se ejecuta un bucle infinito para el funcionamiento de la Task
        if (blink == TRUE){
            Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de la tarea
                                            //durante un periodo de tiempo determinado.
            GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle para cambiar el estado,
                                        //es decir, un parpadeo de un LED específico del microcontrolador.
        }else{
            Task_sleep((unsigned int)arg0);
            GPIO_write(Board_LED0, Board_LED_ON); //Se ejecuta la función GPIO_write, para cambiar el estado
                                                  //del LED0, en este caso, permanece encendido.
        }
    }
}
```

Fig 63. Modificación del thread heartBeatFxn. [Fuente: EP]

3. Se añade las funciones de los *threads* Hwi y Swi.

```
void swiFxn0(UArg arg0, UArg arg1)
{
    blink = !blink; //Cambia el estado del blink, para modificar el comportamiento del LED0.
}

void hwi0(UArg arg0)
{
    swiFxn0(0, 0); //Se lanza el thread Swi.
}
```

Fig 64. Thread Swi y Hwi. [Fuente: EP]

4. Por último, se incluyen en la función main dos líneas de código, para habilitar la interrupción a través del botón SW1.

```
int main(void)  //Se declara la función principal del programa
{
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    GPIO_enableInt(Board_BUTTON0);  //Se habilita las interrupciones a través del botón SW1.
    GPIO_setCallback(Board_BUTTON0, hwi0);  //Se ejecuta el thread hwi0 cuando se presiona el botón SW1.

    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha ejecutado correctamente.
}
```

Fig 65. Modificaciones de la función main. [Fuente: EP]

Por último, se realiza un guardado del proyecto para actualizar el proyecto para la detección de los cambios en el *kernel* y se ejecuta el programa. Se puede observar que el LED0 tiene el mismo comportamiento que en el apartado anterior, y cuando se presiona el botón SW1, la luz del LED0 permanece encendido, y si se vuelve a presionar el botón, el LED0 vuelve a parpadear.

7.3.3 Configuración *Clock*.

La configuración de un *Clock*, permite el cambio periódico en el estado del LED0, es decir, no hace falta la intervención a través del *hardware* para realizar este cambio. Para la configuración de esta herramienta se realiza una copia del proyecto anterior.

Se deben realizar modificaciones tanto en el archivo *empty.cfg* y *empty.c*. Se comienca modificando el archivo *empty.cfg* a través de la siguiente lista de pasos.

1. Se deben eliminar las herramientas utilizadas en el apartado anterior, para ellos se hace con el clic derecho y seleccionar la opción *Delete*, o a través de la opción *Remove*, como se muestra en la *fig 65. Kernel Swi/Hwi, opción: Instance*.
2. Se crear un nuevo *thread* tipo *clock,* aparecerá una ventana como en la *fig 66. Kernel Clock, opción: Module*.

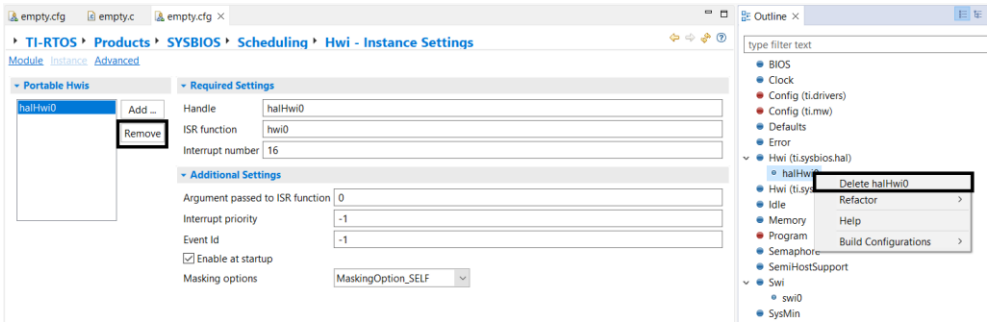


Fig 66. Kernel Swi/Hwi, opción: Instance. [Fuente: EP]

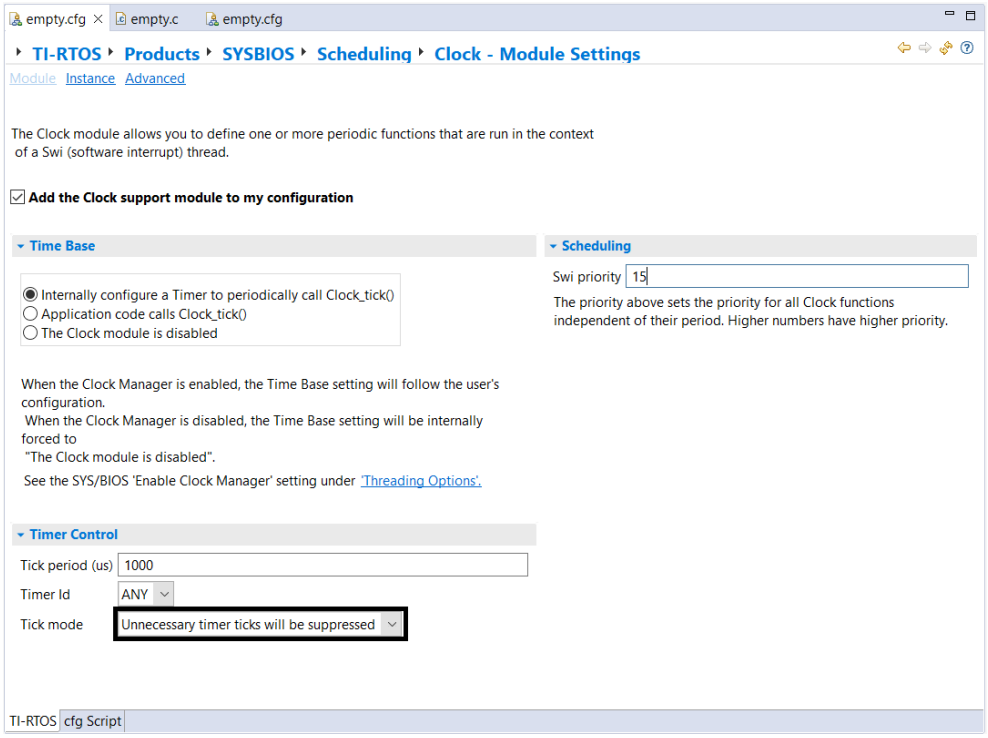


Fig 67. Kernel Clock, opción: Module. [Fuente: EP]

3. En la opción *Instance*, donde realizaremos los cambios en los parámetros *Function*, el nombre de la función, *Unitial timeout*, cuando se realiza el primer lanzamiento del *thread*, y finalmente, el periodo del tiempo para el lanzamiento del *thread*, *Period*.

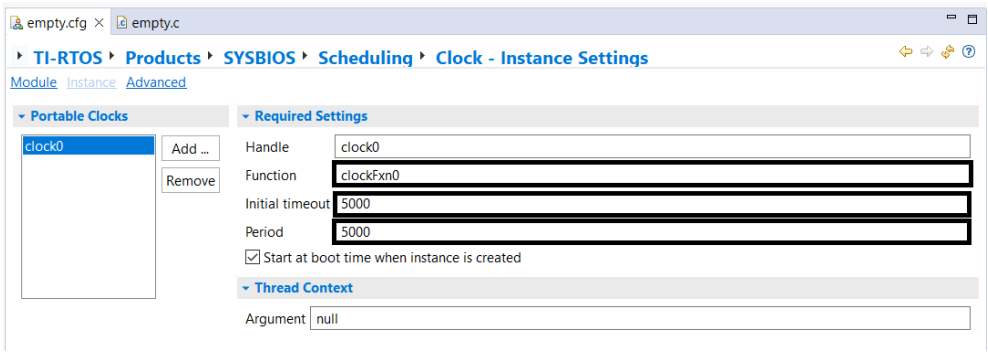


Fig 68. Kernel Clock, opción: Instance. [Fuente: EP]

En el archivo *empty.c* se realizan varios cambios, se eliminan las funciones relacionadas con los *threads* de Swi y Hwi añadidos en el apartado anterior. Se añade en la cabecera el archivo del *Clock threads*. Y se define la función del *Clock thread* que se crear en este apartado.

```
void clockFxn0(UArg arg0, UArg arg1)
{
    blink = !blink; //Cambia el estado del blink, para modificar el comportamiento del LED0.
}
```

Fig 69. Función del thread clockFxn0. [Fuente: EP]

En la función *main* se eliminan las líneas que habilitan la interrupción por hardware.

7.3.4 Configuración semáforo.

Se realiza otra copia del proyecto anterior, en esta ocasión se trabajará con la herramienta de semáforos. Esta herramienta controla el acceso de diferentes *Tasks* que comparte un mismo recurso. Este elemento de sincronización se configura siguiendo estos pasos:

1. Se debe eliminar la herramienta utilizada en el apartado anterior, para ello se hace con el clic derecho y seleccionar la opción *Delete*, o a través de la opción *Remove*.
2. Se crea una nueva *Task*.

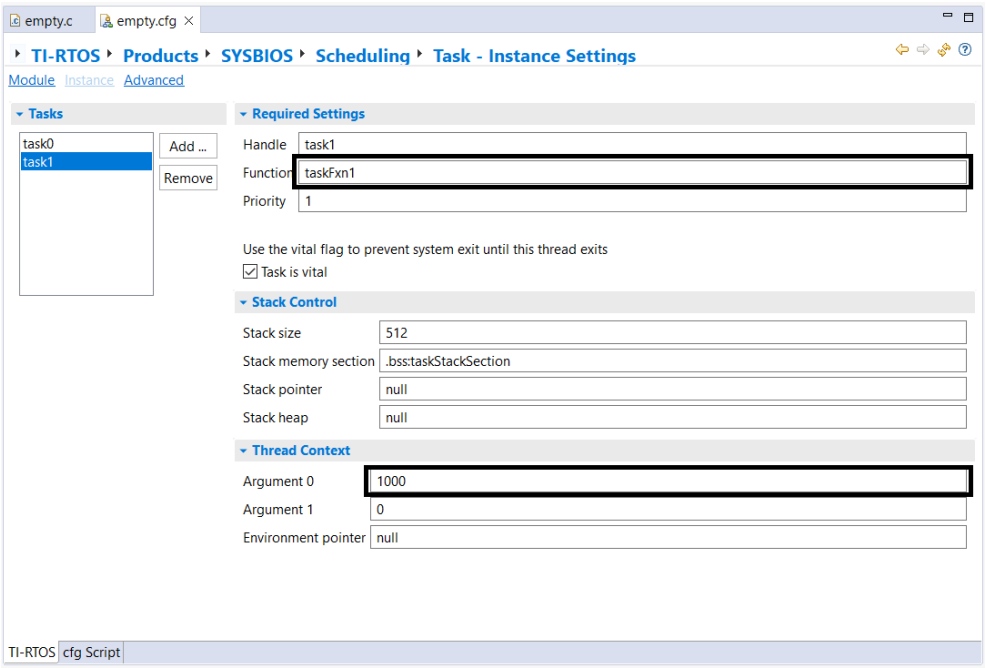


Fig 70. Creación nueva Task. [Fuente: EP]

3. Una vez terminada la creación de la nueva *Task*, se configura el semáforo que será necesario para este apartado. Para ellos, es necesarios crear un nuevo semáforo a través del *kernel* de TI-RTOS.

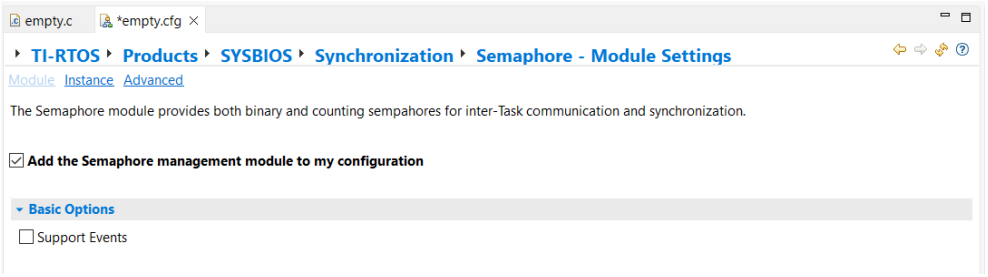


Fig 71. Kernel semáforo, opción: Module. [Fuente: EP]

4. Se modifica el *Handle* y el tipo de semáforo, en este caso es binario, ya que el recurso está siendo utilizado por una tarea o por otra.

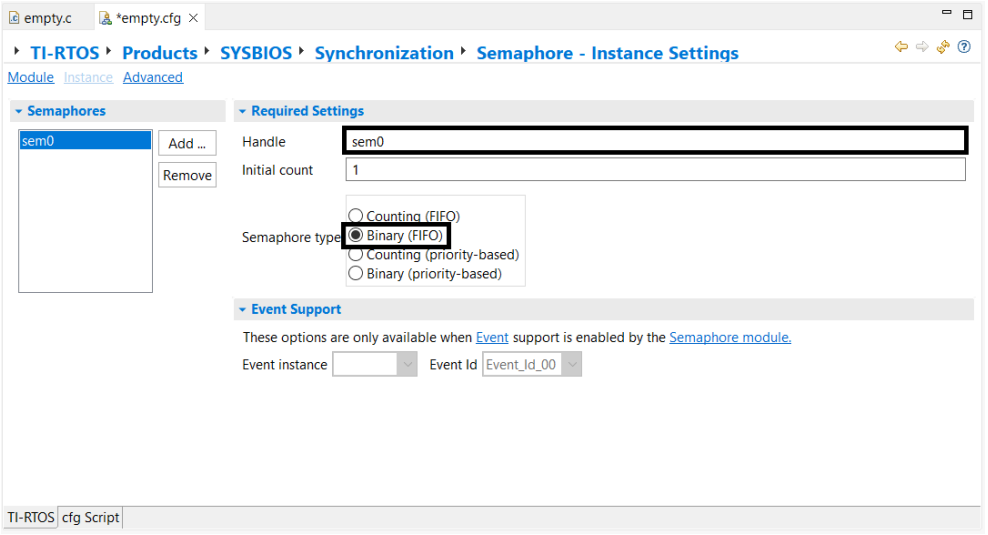


Fig 72. Kernel semáforo, opción: Instance. [Fuente: EP]

5. Para el uso de los semáforos es necesario el uso del siguiente elemento *LoggingSetup*, para habilitar esta opción, se realiza una búsqueda en la ventana *Available Products*, se selecciona el elemento y se presiona la opción *Use LoggingSetup*.

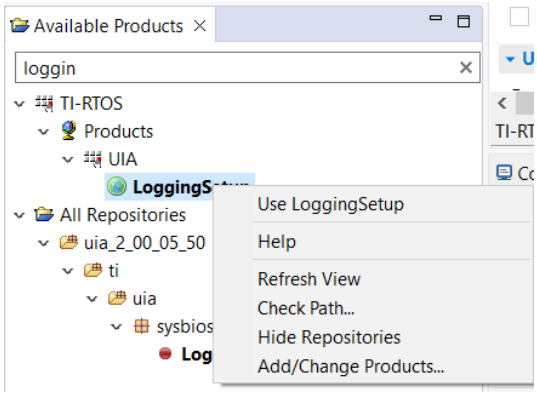


Fig 73. Ventana Available Products, opción: LoggingSetup. [Fuente: EP]

Una vez habilitado el producto se habilitan las opciones marcadas en la imagen que se encuentra debajo.

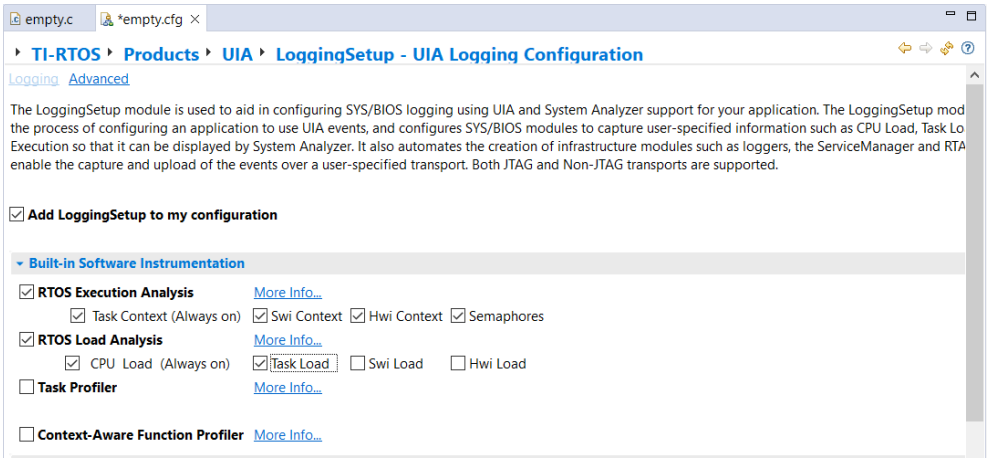


Fig 74. Kernel LoggingSetup, opción: Logging. [Fuente: EP]

En el archivo *empty.c* se elimina la función del *Clock thread* y se añade estas líneas de código.

```
void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos argumentos
                                        //La función tiene tipo de retorno void,
                                        //lo que significa que no devuelve ningún valor
{
    while(1){
        Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de la tarea
                                        //durante un período de tiempo determinado.
        GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle para cambiar el estado,
                                    //es decir, un parpadeo de un LED específico del microcontrolador.
        Semaphore_post(sem0);       //Se publica el semáfor
    }
}

void taskFxn1(UArg arg0, UArg arg1)
{
        while(1){
            Semaphore_pend(sem0, BIOS_WAIT_FOREVER); //Línea que espera la disponibilidad del recurso
            GPIO_write(Board_LED0, Board_LED_ON);   //El LED0 permanece encendido.
            Task_sleep((unsigned int)arg0);
        }
}
```

Fig 75. Código del archivo empty.c. [Fuente: EP]

Una vez ejecutado el programa, se puede observar que el LED0 empieza parpadeando y al cabo de un tiempo, sin el uso de relojes ni temporizadores, cambia su estado a encendido permanente.

7.3.5 Configuración eventos.

En este apartado se enseña la configuración de eventos. Este elemento se crea con un sencillo paso. Antes de crear este elemento de sincronización es necesario añadir este elemento, mediante la opción *Use evento*, como se muestran en la *fig 75. Ventana Available Products, opción: Event.*

Una vez añadido la herramienta, se crear un nuevo elemento de sincronización tipo *event*, aparecerá una ventana, donde solo es necesario que se otroge un nombre al evento, como se indica en la *fig 76. Kernel Event, opción: Instance.*

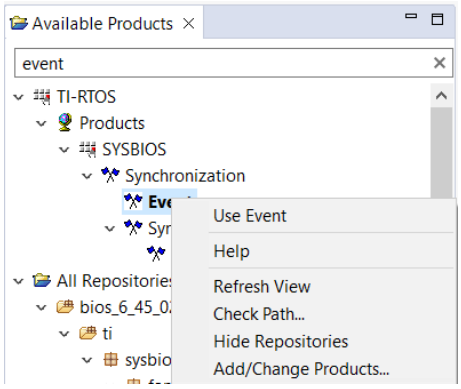


Fig 76. Ventana Available Products, opción: Event. [Fuente: EP]

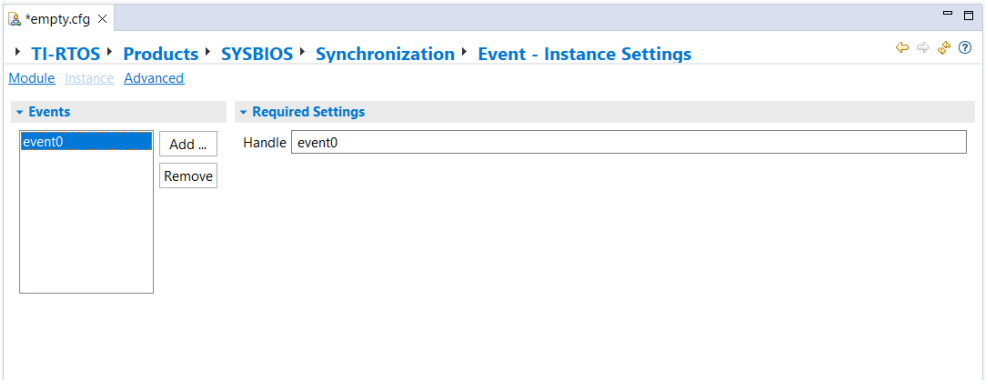


Fig 77. Kernel Event, opción: Instance. [Fuente: EP]

El objetivo en este apartado, es la creación de un evento cada vez que se apague o se encienda el led. Para ellos hay que realizar las siguientes modificaciones en el archivo *empty.c*. Se deben seguir esta serie de pasos:

1. En la cabecera el programa se definen los eventos que son necesarios para el programa.
2. El cambiar el contenido de la función *main*.

```
// Definición de eventos
#define EVENT_LED_ON 0x01 //Se define el evento de cuando el LED esta encendido
#define EVENT_LED_OFF 0x02  //Se define el evento de cuando el LED esta apagado


int main(void) {

    Board_initGeneral();
    Board_initGPIO();

    BIOS_start();

    return (0);
}
```

Fig 78. Definición de los diferentes eventos del programa y contenido de la función main. [Fuente:EP]

3. Se modifica las funciones de las *Tasks*:

```
void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED
        Task_sleep((unsigned int)arg0);   //Se espera el tiempo assignado
        GPIO_write(Board_LED0, Board_LED_OFF);  //Se apaga el LED
        Task_sleep((unsigned int)arg0);   //Se espera el tiempo assignado
        Event_post(event0, EVENT_LED_ON); //Se lanza el evento de LED encendido
        Task_sleep((unsigned int)arg0);
        Event_post(event0, EVENT_LED_OFF);  //Se lanza el evento de LED apagado
    }
}
```

Fig 79. Contenido de la función: heartBeatFxn. [Fuente: EP]

```
void taskFxn1(UArg arg0, UArg arg1) {
    UInt events;    //Se define una variable para el almacenamiento del evento.
    while (1) {
        events = Event_pend(event0, Event_Id_NONE, EVENT_LED_ON | EVENT_LED_OFF, BIOS_WAIT_FOREVER);
        //Se mantiene a la esperar de los eventos de encendido o apagado del LED
        if (events & EVENT_LED_ON){ //¿Se ha encendido el LED?
            System_printf("LED encendido\n"); //Se imprime un mensaje
            System_flush();
        }
        if (events & EVENT_LED_OFF){    //¿Se ha apagado el LED?
            System_printf("LED apagado\n"); //Se imprime un mensaje
            System_flush();
        }
    }
}
```

Fig 80. Contenido de la función: taskFxn1. [Fuente: EP]

Una vez que se han realizado todas las modificaciones en el código, se puede ejecutar el proyecto. Se observa que el comportamiento del LED no ha cambiado respecto al primer proyecto, aunque en este caso, se puede analizar en la consola del CCS que el dispositivo comunica cuando está el LED0 apagado y cuando está encendido.

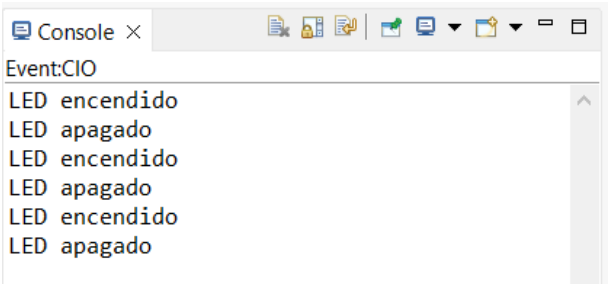


Fig 81. Mensajes de la consola a la hora de ejecutar el programa. [Fuente: EP]

7.3.6 Configuración *Gates*.

En este apartado se desarrolla la configuración de *Gates*. Este elemento se crea como todas las anteriores. Antes de crear este elemento de sincronización es necesario añadir esta herramienta, mediante la opción *Use GateAll*, se pueden observar que hay más opciones de esta función, como se muestra en la *fig 81. Available Products y Outline del módulo GateAll*, la que se ha seleccionado trabaja con *threads* de tipo Hwi, Swi y *Task*.

Para utilizar la herramienta, se realiza clic derecho en *GateAll* en la ventana *Outline* y solo es necesario hacer clic al *New GateAll*, como se muestra a la derecha de la f*ig 81. Available Products y Outline del módulo GateAll.*

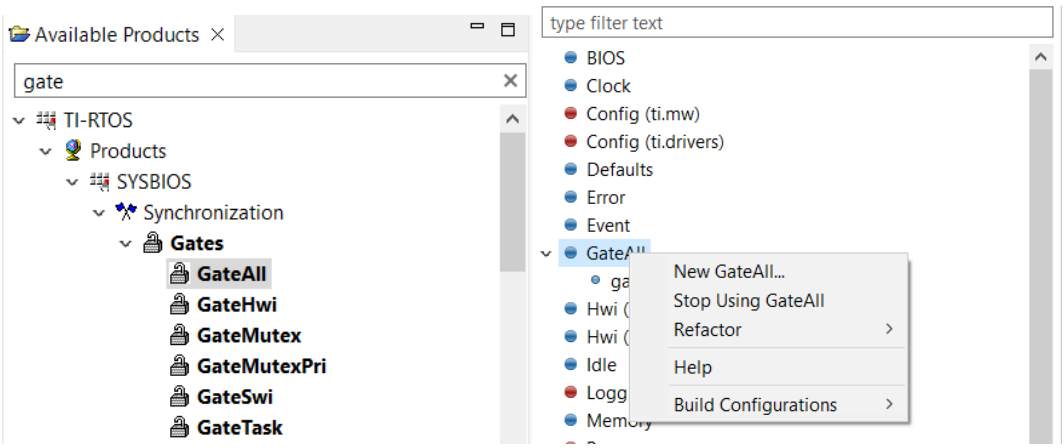


Fig 82. Available Products y Outline del módulo GateAll. [Fuente: EP]

Para observar el funcionamiento se realizará una serie de modificaciones en el archivo *main.c*. De esta manera, programa se adapta a las necesidades de esta herramienta de sincronización. Con esta herramienta el *kernel* esta limitado, y no se puede trabajar con todas las características del *GateAll*, ya que solo podemos dar nombre al *GateAll* a través del *kernel*. Teniendo en cuenta este inconveniente, es mejor crear el *GateAll* a través de código en el archivo *empty.c*.

Para obtener una aplicación para que utilice esta herramienta, se realizarán una serie de pasos para obtener un programa que nos muestre en todo momento que *Task* esta utilizando el recurso LED0.

1. Primero se añaden todos los archivos necesarios en la cabecera.

```
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/gates/GateAll.h>
#include <ti/drivers/GPIO.h>
#include <xdc/runtime/System.h>
#include "Board.h"
```

Fig 83. Encabezado del archivo empty.c. [Fuente: EP]

2. Se define el nombre y la variable estructura del *Gate* que se utiliza durante la aplicación

```
GateAll_Struct gateAll0Struct; //Se define la variable estructura del Gate
GateAll_Handle gateAll0;    //Se define el nombre del Gate
```

Fig 84. Definición de la variable estructura y el nombre del Gate. [Fuente: EP]

3. Para más comodidad se define dos funciones que realizarán el apagado y encendido del LED0. Se requiere de estas funciones para no escribir el mismo código en múltiples líneas del código, haciendo que sea más sencillo su modificación.

```
void ledOn(void) {
    GPIO_write(Board_LED0, Board_LED_ON);
    System_printf("LED encendido\n");
    System_flush();
}

void ledOff(void) {
    GPIO_write(Board_LED0, Board_LED_OFF);
    System_printf("LED apagado\n");
    System_flush();
}
```

Fig 85. Funciones de ledOn y ledOff. [Fuente: EP]

4. Se modifica el contenido de la función *main*.

```
int main(void) {
    Board_initGeneral();
    Board_initGPIO();

    GateAll_construct(&gateAll0Struct, NULL); //Se pone la dirección de gateAll0Struct
    gateAll0 = GateAll_handle(&gateAll0Struct); //Se obtiene el handle desde la estructura

    BIOS_start();

    return (0);
}
```

Fig 86. Función main del archivo empty.c. [Fuente: EP]

5. Se modifica el contenido de las funciones de los *Task threads.*

```
void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        IArg key = GateAll_enter(gateAll0); //Se entra al Gate.
        System_printf("heartBeatFxn\n");
        System_flush();
        ledOn();    //Se llama a la función ledOn.
        GateAll_leave(gateAll0, key);   //Se sale del Gate.
        Task_sleep((unsigned int)arg0);
        key = GateAll_enter(gateAll0);  //Se entra al Gate.
        System_printf("heartBeatFxn\n");
        System_flush();
        ledOff();   //Se llama a la función ledOff
        GateAll_leave(gateAll0, key);   //Se sale del Gate.
        Task_sleep((unsigned int)arg0);
    }
}

void taskFxn1(UArg arg0, UArg arg1) {
    while (1) {
        Task_sleep((unsigned int)arg0);
        IArg key = GateAll_enter(gateAll0); //Se entra al Gate
        static Bool ledState = FALSE;   //Se define la variable ledState
        if (ledState) {
            System_printf("taskFxn1\n");
            System_flush();
            ledOff();   //Se llama a la función ledOff.
        } else {
            System_printf("taskFxn1\n");
            System_flush();
            ledOn();    //Se llama a la función ledOn.
        }
        ledState = !ledState;   //Se niega el estado de la variable
        GateAll_leave(gateAll0, key);   //Se sale del Gate
        Task_sleep((unsigned int)arg0);
    }
}
```

Fig 87. Contenido de las funciones de las Tasks threads. [Fuente: EP]

Por último, se guarda y se ejecuta el proyecto para observar el comportamiento de la aplicación. Se sigue observando que no hay cambio en el comportamiento del LED0, pero en la consola de CCS, se puede observar los mensajes que proporciona el dispositivo.

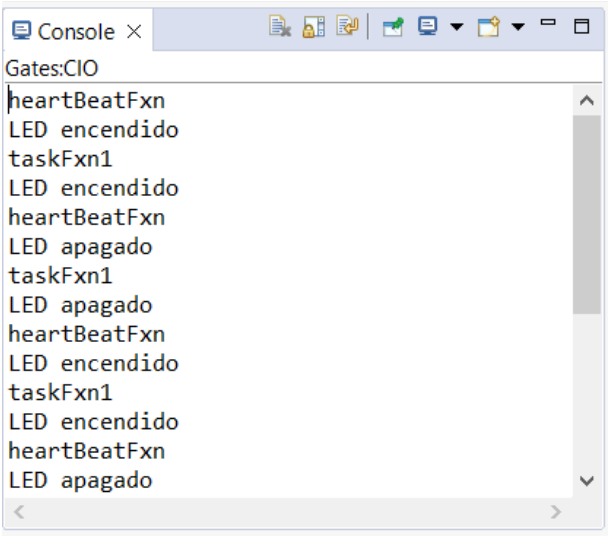


Fig 88. Consola de la aplicación con la herramienta Gate. [Fuente: EP]

7.3.7 Configuración *Mailbox*.

En este apartado se desarrolla la configuración de *Mailbox*. Este elemento se crea como todas las anteriores. Antes de crear este elemento de sincronización es necesario añadir esta herramienta, mediante la opción *Use Mailbox.*

Primero, se realiza una copia del proyecto y se configura la herramienta siguiendo esta serie de pasos:

1. Se añade la herramienta a través de la ventana *Available Products* y habilitar la herramienta en el kernel. Se selecciona la herramienta y aparece la siguiente ventana.

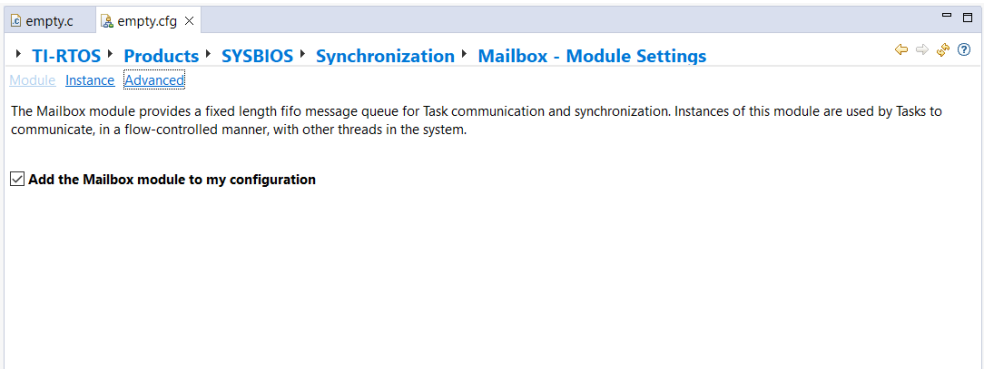


Fig 89. Mailbox kernel, opción: Module. [Fuente: EP]

2. Se crea el *Mailbox*, cambiado las variables *Handle*, se encarga de darle nombre a la herramienta, *Size of messages (chars)*, el tamaño del mensaje, y, por último, *Max number of messages*, el número de mensaje que se envían a traves de este *Mailbox.*

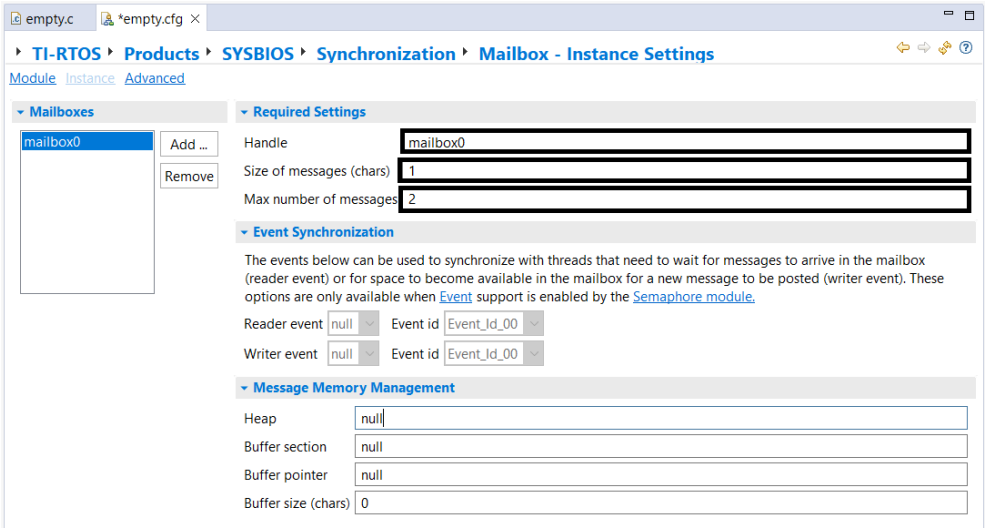


Fig 90. Mailbox kernel, opción: Instance. [Fuente: EP]

El objetivo de este punto es crear una aplicación que se encargue de modificar el estado del LED0 después de recibir el mensaje.

1. Se crea la estructura del *Mailbox* en la cabecera del archivo *empty.c*.

```
typedef struct {
    int ledState;   //1 para encender, 0 para apagar
} LedMsg;
```

Fig 91. Estructura del mensaje. [Fuente: EP]

2. Se modifica el contenido de las funciones relacionadas con los *Tasks threads.*

```
void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        LedMsg msg;
        while (1) {
            static Bool ledState = FALSE; //Se altera el estado del LED en el mensaje
            msg.ledState = ledState ? 0 : 1; //Se encarga de comprobar el estado del Mailbox
            ledState = !ledState;
            if (Mailbox_post(mailbox0Handle, &msg, BIOS_NO_WAIT)){// Enviar el mensaje a a través del Mailbox
                System_printf("Task 1: Mensaje enviado\n");
            } else {
                System_printf("Task 1: Fallo al enviar el mensaje\n");
            }
            System_flush();
            Task_sleep((unsigned int)arg0);
        }
    }
}
```

Fig 92. Contenido de la función heartBeatFxn. [Fuente: EP]

```
void taskFxn1(UArg arg0, UArg arg1) {
    LedMsg msg;
        while (1) {
            // Esperar a recibir un mensaje de la Mailbox
            if (Mailbox_pend(mailbox0Handle, &msg, BIOS_WAIT_FOREVER)) {
                System_printf("Task 2: Mensaje recibido\n");
                System_flush();

                // Procesar el mensaje para controlar el LED
                if (msg.ledState == 1) {
                    ledOn();
                } else {
                    ledOff();
                }
            } else {
                System_printf("Task 2: Fallo al recibir el mensaje\n");
                System_flush();
            }
        }
}
```

Fig 93. Contenido de la función taskFxn1. [Fuente: EP]

Una vez se han seguido todos los pasos se ejecuta el programa, se observa que el comportamiento de LED0 no ha cambiado, pero el mensaje que transmite la consola es el siguiente.

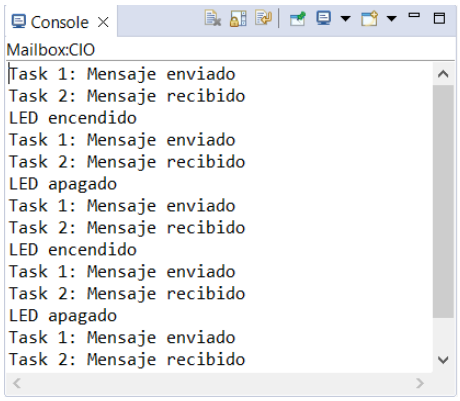


Fig 94. Consola de la aplicación con la herramienta Mailbox. [Fuente: EP]

Se observa que el estado del LED0 solo se modifica una vez enviado y recibido el mensaje.

7.3.8 Configuración *Queue*.

En este apartado se desarrolla la configuración de *Queue*. Este elemento se crea como todas las anteriores. Antes de crear este elemento de sincronización es necesario añadir esta herramienta, mediante la opción *Use Queue*.

Se realiza el mismo procedimiento utilizado anteriormente para crear una variable de esta herramienta, no hay información adicional como en otras herramientas o *threads*.

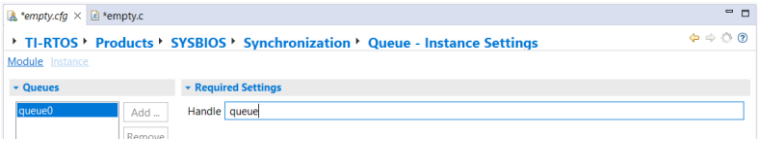


Fig 95. Queue kernel, opción: Instance. [Fuente: EP]

Para la comprobación del funcionamiento de esta herramienta, se realiza una copia del último proyecto realizado, y se realizarán los siguientes cambios en el archivo *empty.c*:

3. Se crea la estructura del *Queue* en la cabecera del archivo *empty.c*.

```
// Definición de la estructura de la Queue
typedef struct {
    Queue_Elem elem; // Elemento de Queue
    int ledState;    // 1 para encender, 0 para apagar
} LedMsg;
```

Fig 96. Estructura de la Queue. [Fuente: EP]

4. Se modifica el contenido de las funciones relacionadas con los *Tasks threads*.

```
void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        // Crear el mensaje
        LedMsg *msg = &ledMsgs[msgIndex];
        msgIndex = (msgIndex + 1) % NUMMSGS;

        // Alternar el estado del LED en el mensaje
        static Bool ledState = FALSE;
        msg->ledState = ledState ? 0 : 1;
        ledState = !ledState;

        // Enviar el mensaje a la Queue
        Queue_put(queueHandle, &msg->elem);
        System_printf("Task 1: Mensaje enviado\n");
        System_flush();
        Task_sleep((unsigned int) arg0);
    }
}
```

Fig 97. Contenido de la función heartBeatFxn. [Fuente: EP]

```
void taskFxn1(UArg arg0, UArg arg1) {
    while (1) {
        // Esperar a recibir un mensaje de la cola
        if (!Queue_empty(queueHandle)) {
            LedMsg *msg = (LedMsg *)Queue_get(queueHandle);

            // Procesar el mensaje para controlar el LED
            if (msg->ledState == 1) {
                ledOn();
            } else {
                ledOff();
            }
            System_printf("Task 2: Mensaje recibido y procesado\n");
            System_flush();
        }
        Task_sleep((unsigned int) arg0);
    }
}
```

Fig 98. Contenido de la función taskFxn1. [Fuente: EP]

Una vez se han seguido todos los pasos se ejecuta el programa, se observa que el comportamiento de LED0 no ha cambiado, pero el mensaje que transmite la consola es el siguiente.

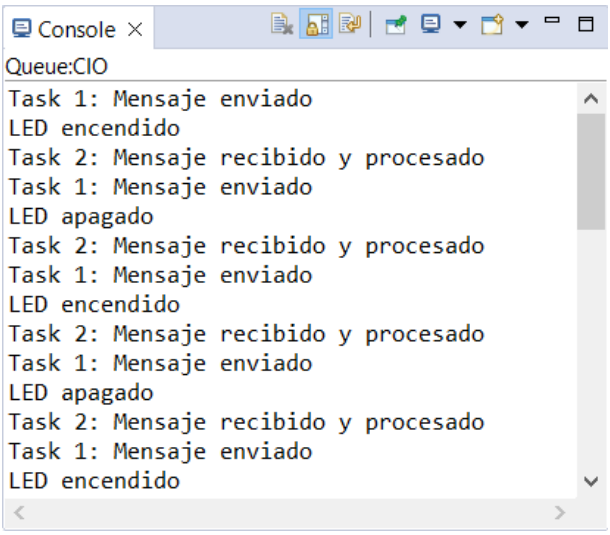


Fig 99. Consola de la aplicación con la herramienta Mailbox. [Fuente: EP]

8 Impacto medioambiental.

Para el análisis del impacto medioambiental, se utilizarán varias variables: el consumo energético, la vida útil y el reciclaje de los microcontroladores.

La primera se basa en Cortex-M4, el cual es bajo debido a que están diseñados para aplicaciones que requieren eficiencia energética. Aunque hay factores que influyen en su consumo:

- Ajustando la frecuencia de operaciones se mejora tanto el rendimiento como el consumo de energía.
- El uso de periféricos contribuye en el consumo de energía.
- La eficiencia del código o algoritmos implementados es crucial para reducir la carga de trabajo, que afecta de manera directa.

A través de distintas herramientas se puede medir y optimizar el consumo energético durante el desarrollo de la aplicación. Al tener acceso al suministro de energías renovables este influye positivamente en el impacto que tiene en el medioambiente.

La segunda variable no está limitada por un tiempo específico, hay que tener en cuenta varios factores que afectan a su función, entre ellos:

- Las condiciones ambientales donde se encuentra el microcontrolador, como la temperatura o la humedad.
- La vida útil de la memoria *flash* está determinada por la cantidad de ciclos de escritura y lectura que puede soportar.
- El avance en las tecnologías de implementación de semiconductores puede mejorar la durabilidad de los componentes.

9 Perspectiva de género.

En este apartado se trata la perspectiva de género, en este caso, el sector tecnológico incluido el ámbito industrial. Éste se ha enfrentado históricamente desafíos en términos de diversidad de género y representación equitativa en roles técnicos y de liderazgo, además, tradicionalmente, estos campos han estado dominados por hombres, lo que ha resultado en una falta de diversidad y perspectivas en el desarrollo de tecnologías y soluciones industriales.

El presente proyecto no presenta explícitamente cuestiones de género. En proyectos de desarrollo tecnológico como el descrito en el presente proyecto, es crucial considerar cómo se están abordando las barreras de género en el acceso a la educación, las oportunidades laborales y el liderazgo en el campo de la tecnología.

Es fundamental fomentar la participación y el avance de mujeres en roles técnicos, de ingeniería y de dirección en proyectos. Sin embargo, se considera una perspectiva neutral en términos de género, ya que no hay referencias específicas que excluyan o privilegien a un género sobre otro. El proyecto se enfoca principalmente en aspectos tecnológicos y empresariales sin entrar en consideraciones de género.

10 Planificación.

En este apartado se realiza una exposición de las tareas necesarias para realizar el diseño y desarrollo de la bancada.

Código	Descripción	Horas	Predecesora
A	Creación del objeto	5	–
B	Revisión de antecedentes	30	A
C	Alcance del proyecto	5	B
D	Objetivos y especificaciones técnicas	15	C
E	Generación de posibles alternativas de solución	20	D
F	Desarrollo de la alternativa más adecuada	10	E
G	Viabilidad técnica	20	F
H	Viabilidad económica	5	F
I	Viabilidad medioambiental	5	F
J	Planificación	5	G, H, I
K	Presupuesto	5	J
L	Documentación del anteproyecto	30	-
M	Corrección del anteproyecto	10	L
N	Diseño de la bancada	30	M
Ñ	Documentación de la memoria intermedia	30	N
O	Corrección de la memoria intermedia	10	Ñ
P	Explicación del microcontrolador	10	N
Q	Explicación del entorno de desarrollo	20	N
R	Explicación del SO	20	N
S	Desarrollo de aplicaciones	45	P, Q, R
T	Documentación del proyecto	50	M
U	Preparación de la presentación	20	W

Tabla 5. Tareas y distribución de horas del proyecto. [Fuente: EP]

Las tareas que pertenecen al anteproyecto, que se ha llevado a cabo desde noviembre hasta mediados de enero. El resto de tareas se realizarán durante el resto del año académico, acabando el 3 junio, con la entrega de la documentación. Quedando solo la presentación del proyecto, que se realiza en la semana de 24 de junio.

10.1 Diagrama de Gantt

El proyecto empieza el día 27 de noviembre del 2023, y finaliza el día 3 de junio del 2024.

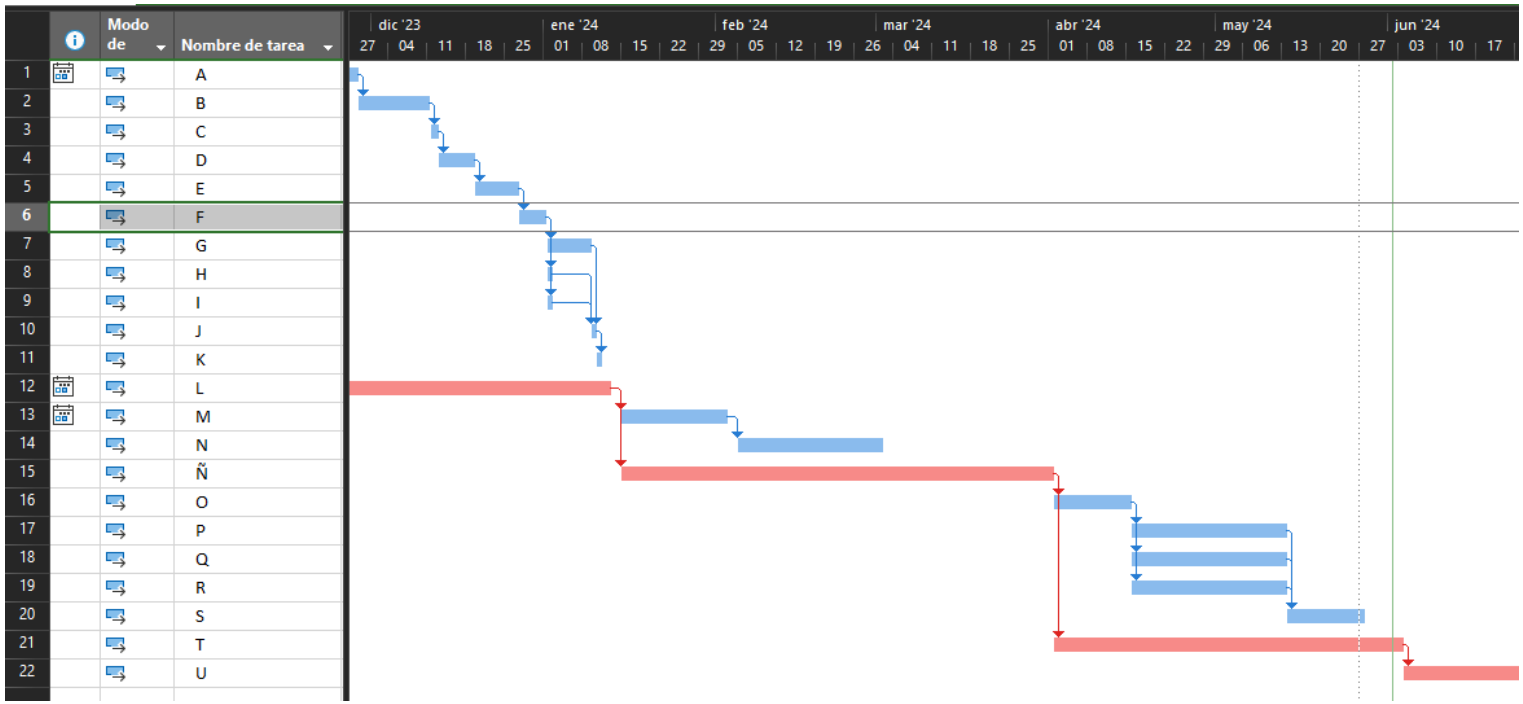


Fig 100. Diagrama de Gantt. [Fuente: EP]

La última tarea, se realiza después de la entrega de la memoria final, empezando el día 3 de junio y finalizando el día de la presentación del proyecto.

10.2 Análisis de riesgo.

Para elaborar un plan de contingencia para el proyecto, se considera crucial la identificación de los posibles riesgos que podrían surgir durante cada etapa del mismo. Se han identificado varios riesgos relevantes:

- Posibles dificultades en el diseño o desarrollo de la bancada, que podrían incluir desafíos técnicos, restricciones de recursos, cambios en los requisitos del proyecto o falta de experiencia en el desarrollo de elementos similares. Estas dificultades podrían resultar en retrasos o aumentos de costos.
- Probable retraso en la entrega del hardware, que podría deberse a problemas en la cadena de suministro, problemas de calidad o cambios en las especificaciones, entre otras causas. Este retraso podría afectar significativamente al proyecto.
- Riesgo de no compatibilidad entre componentes, que podría generar problemas de rendimiento o estabilidad en el sistema debido a diferencias en estándares de comunicación, interfaces incompatibles o conflictos de software. La falta de compatibilidad podría requerir modificaciones o reemplazos de componentes, aumentando el costo y el tiempo del proyecto.
- Posible aumento del presupuesto, que podría ocurrir debido a los riesgos mencionados anteriormente, así como a aumentos en los costos de materiales o mano de obra, errores de estimación, entre otros factores.

Es importante identificar estos riesgos y tomar medidas preventivas para mitigar su impacto y garantizar el éxito del proyecto.

10.3 Plan de contingencia.

Se proporciona un plan de contingencia para abarcar cada riesgo descrito en el apartado anterior:

- Posibles dificultades en el diseño o desarrollo de la bancada.
  Contingencia: asignación de recursos adecuados para el diseño y desarrollo.
  Acción: establecimiento de reuniones regulares para identificar y abordar las incidencias a medida que surjan. Se consultará con otros docentes si es necesario.

- Probable retraso en la entrega del hardware.

  Contingencia: identificación múltiples proveedores confiables y se mantiene una comunicación frecuente con ellos.

  Acción: realización del pedido con suficiente antelación para permitir posibles retrasos. Se explorarán opciones de envió acelerado u otras alternativas.

- Riesgo de no compatibilidad entre componentes.

  Contingencia: realización de pruebas exhaustivas de compatibilidad antes del desarrollo de la bancada.

  Acción: establecimiento de un proceso de prueba riguroso que incluya la verificación de la compatibilidad entre componentes. Se contará con soporte técnico para resolver problemas de compatibilidad que puedan surgir durante el desarrollo.

- Posible aumento del presupuesto.

  Contingencia: establecimiento de un presupuesto para posibles imprevistos con el fin de hacer frente aumentos de costes.

  Acción: realización de revisiones periódicas del presupuesto del proyecto para identificar y abordar cualquier desviación.

11 Conclusiones.

El proyecto se centra en el diseño y desarrollo de una bancada para aplicaciones industriales avanzadas, estructurándose en cinco fases cruciales: análisis de tecnologías, selección de componentes, diseño de la bancada, desarrollo de la bancada y desarrollo de aplicaciones. Cada fase contribuye significativamente a la creación de una herramienta innovadora y eficiente, con un entorno controlado y seguro para la evaluación de sistemas, dispositivos y procesos.

Se llevo a cabo un análisis exhaustivo de las tecnologías disponibles y se seleccionaron componentes que cumplen con los requisitos de rendimiento, fiabilidad y compatibilidad necesarios.

El diseño y desarrollo de la bancada incluyeron la configuración de los elementos desde el microcontrolador hasta los algoritmos necesarios para el funcionamiento optimo del TI-RTOS, en esta fase, se ha requerido mayor tiempo en la compresión, configuración y optimización del sistema operativo y el *kernel* que incluye.

Durante el diseño y el desarrollo de la bancada, se observo un incremento en el presupuesto inicial estimado, principalmente debido a la variación del precio del dispositivo seleccionado.

Se recomienda llevar a cabo una mejora continua de la bancada mediante la incorporación de nuevas tecnologías y componentes. Esto permite desarrollar la capacidad para adaptarse a diferentes aplicaciones o necesidades, incrementando así la versatilidad y utilidad en diversas áreas.

Se sugiere explorar la implementación de algoritmos de IA, como una línea de desarrollo futuro. La integración de la IA podría optimizar los procesos industriales, mejorando la eficiencia y la calidad de las aplicaciones. Este enfoque no solo ampliaría las capacidades de la bancada, sino que también permite posicionar la bancada en la vanguardia de las soluciones tecnológicas.

El proyecto no solo ha cumplido con sus objetivos inmediatos, sino que también ha establecido una base sólida para futuras innovaciones y mejoras continuas, contribuyendo de manera significativa a la eficiencia operativa de las aplicaciones en el ámbito industrial.

12 Bibliografía.

[1] “Inteligencia artificial: definición, historia, usos, peligros”. Formación en ciencia de datos | DataScientest.com. Accedido el 30 de octubre de 2023. [En línea]. Disponible: https://datascientest.com/es/inteligencia-artificial-definicion

[2] “Arm Cortex-M4 MCUs | TI.com”. Analog | Embedded processing | Semiconductor company | TI.com. Accedido el 30 de octubre de 2023. [En línea]. Disponible: https://www.ti.com/microcontrollers-mcus-processors/arm-based-microcontrollers/arm-cortex-m4-mcus/overview.html

[3] J. Pastor. “Edge Computing: qué es y por qué hay gente que piensa que es el futuro”. Xataka - Tecnología y gadgets, móviles, informática, electrónica. Accedido el 24 de enero de 2024. [En línea]. Disponible: https://www.xataka.com/internet-of-things/edge-computing-que-es-y-por-que-hay-gente-que-piensa-que-es-el-futuro

[4] “Arm-based microcontrollers | TI.com”. Analog | Embedded processing | Semiconductor company | TI.com. Accedido el 1 de noviembre de 2023. [En línea]. Disponible: https://www.ti.com/microcontrollers-mcus-processors/arm-based-microcontrollers/overview.html

[5] Colaboradores de los proyectos Wikimedia. “Microcontrolador - Wikipedia, la enciclopedia libre”. Wikipedia, la enciclopedia libre. Accedido el 15 de noviembre de 2024. [En línea]. Disponible: https://es.wikipedia.org/wiki/Microcontrolador

[6] Accedido el 24 de enero de 2024. [En línea]. Disponible: https://www.st.com/en/development-tools/stm32cubeide.html

[7] Contributors to Wikimedia projects. “ARM Cortex-M - Viquipèdia, l'enciclopèdia lliure”. Viquipèdia. Accedido el 14 de diciembre de 2023. [En línea]. Disponible: https://ca.wikipedia.org/wiki/ARM_Cortex-M

[8] Colaboradores de los proyectos Wikimedia. “Sistema embebido - Wikipedia, la enciclopedia libre”. Wikipedia, la enciclopedia libre. Accedido el 21 de enero de 2024. [En línea]. Disponible: https://es.wikipedia.org/wiki/Sistema_embebido

[9] “CCSTUDIO IDE, configuration, compiler or debugger | ti.com”. Analog | Embedded processing | Semiconductor company | TI.com. Accedido el 5 de febrero de 2024. [En línea]. Disponible: https://www.ti.com/tool/CCSTUDIO

[10] “Sector industrial”. Departament d'Economia i Hisenda. Accedido el 24 de enero de 2024. [En línea]. Disponible: https://economia.gencat.cat/ca/ambits-actuacio/economia-catalana/trets/estructura-productiva/sector-industrial/index.html

[11] "Code Composer Studio User's Guide — Code Composer Studio 12.7.0 Documentation". Accedido el 5 de febrero de 2024. [En línea]. Disponible: https://software-dl.ti.com/ccs/esd/documents/users_guide/index.html

[12] "EK-TM4C1294XL Evaluation board | TI.com". Analog | Embedded processing | Semiconductor company | TI.com. Accedido el 5 de febrero de 2024. [En línea]. Disponible: https://www.ti.com/tool/EK-TM4C1294XL#tech-docs

[13] "TI-RTOS-MCU Operating system (OS) | TI.com". Analog | Embedded processing | Semiconductor company | TI.com. Accedido el 5 de febrero de 2024. [En línea]. Disponible: https://www.ti.com/tool/TI-RTOS-MCU#tech-docs

Centre universitari adscrit a la

TecnoCampus

upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO Y PUESTA EN MARCHA DE UNA BANCADA PARA EL DESARROLLO DE SOLUCIONES INDUSTRIALES INTEGRALES AVANZADAS CON MICROCONTROLADORES ARM CORTEX-M4

Estudio económico

HAMZA CHOULLI SAMADI

PONENTE: DR. JULIÁN HORRILLO TELLO

JUNIO 2024

TecnoCampus Mataró-Maresme

Índice.

1. Presupuesto..... 3
1.1. Mediciones..... 3
1.2. Cuadro de precios. 4
1.3 Presupuesto parcial. 4
1.4. Presupuesto final. 5
2. Ejecución del presupuesto. 7
2.1. Coste final del proyecto. 7
2.2. Justificación de las desviaciones. 7

1. Presupuesto

1.1. Mediciones

En el presente capítulo se recoge las mediciones correspondientes a la ingeniería y los materiales usados para realizar el proyecto.

Descripción	Horas
Creación del objeto	5
Revisión de antecedentes	30
Alcance del proyecto	5
Objetivos y especificaciones técnicas	15
Generación de posibles alternativas de solución	20
Desarrollo de la alternativa más adecuada	10
Viabilidad técnica	20
Viabilidad económica	5
Viabilidad medioambiental	5
Planificación	5
Presupuesto	5
Documentación del anteproyecto	30
Corrección del anteproyecto	10
Diseño de la bancada	30
Documentación de la memoria intermedia	30
Corrección de la memoria intermedia	10
Explicación del microcontrolador	10
Explicación del entorno de desarrollo	20
Explicación del SO	20
Desarrollo de aplicaciones	45
Documentación del proyecto	50
Preparación de la presentación	20

Tabla 1. Tareas del proyecto. [Fuente: EP]

Descripción	Horas
Microcontrolador de TI	5
Generador de señales	30
Osciloscopio	5
Voltímetro / amperímetro	15
Material oficina (papel, bolígrafos, etc)	20

Tabla 2. Material del proyecto. [Fuente: EP]

1.2. Cuadro de precios.

Capítulo I: Elaboración del proyecto	
Descripción	**Precio unitario (€)**
Investigación	60
Documentación	40
Desarrollo	50

Tabla 3. Tabla de precios capítulo I

Capítulo II: Material del proyecto	
Descripción	**Precio unitario (€)**
Microcontrolador TI	29,99
Generador de señales	321,81
Osciloscopio	355,81
Voltímetro amperímetro	251,84
Material oficina (papel, bolígrafo, etc)	20

Tabla 4. Tabla de precios materiales. [Fuente: EP]

1.3 Presupuesto parcial.

Capítulo I: Elaboración del proyecto			
Descripción	**Horas**	**Precio**	**Coste**
Creación del objeto	5	60,00 €	300,00 €
Revisión de antecedentes	30	60,00 €	1.800,00 €
Alcance del proyecto	5	60,00 €	300,00 €
Objetivos y especificaciones técnicas	15	60,00 €	900,00 €
Generación de posibles alternativas de solución	20	60,00 €	1.200,00 €
Desarrollo de la alternativa más adecuada	10	50,00 €	500,00 €
Viabilidad técnica	20	60,00 €	1.200,00 €
Viabilidad económica	5	60,00 €	300,00 €
Viabilidad medioambiental	5	60,00 €	300,00 €
Planificación	5	50,00 €	250,00 €
Presupuesto	5	50,00 €	250,00 €
Documentación del anteproyecto	30	40,00 €	1.200,00 €
Corrección del anteproyecto	10	40,00 €	400,00 €
Diseño de la bancada	30	60,00 €	1.800,00 €
Documentación de la memoria intermedia	30	40,00 €	1.200,00 €
Corrección de la memoria intermedia	10	40,00 €	400,00 €
Explicación del microcontrolador	10	60,00 €	600,00 €
Explicación del entorno de desarrollo	20	60,00 €	1.200,00 €
Explicación del SO	20	60,00 €	1.200,00 €
Desarrollo de aplicaciones	45	50,00 €	2.250,00 €
Documentación del proyecto	50	40,00 €	2.000,00 €

Preparación de la presentación	20	50,00 €	1.000,00 €
Total costes directos			20.550,00 €
Total costes indirectos (20%)			4.110,00 €
Total costes			24.660,00 €
Marge (25%)			6.165,00 €
Total capitulo I			**30.825,00 €**

Tabla 5. Costes del capítulo I. [Fuente: EP]

Capítulo II: Materiales			
Descripción	**Unidades**	**Precio unitario**	**Coste**
Microcontrolador de Texas Instruments	1	29,99 €	29,99 €
Generador de señales	1	321,94 €	321,94 €
Osciloscopio	1	355,81 €	355,81 €
Voltímetro/Amperímetro	1	251,84 €	251,84 €
Material oficina (papel, bolígrafo, etc.)	1	20,00 €	20,00 €
Total costes directos			979,58 €
Total costes indirectos (20%)			195,92 €
Total costes			1.175,50 €
Imprevistos (15%)			176,32 €
Total capítulo II			**1.351,82 €**

Tabla 6. Costes del capítulo II. [Fuente: EP]

Capítulo III: Amortizaciones			
Descripción	**Coste inversión**	**Años**	**Coste Amort.**
Microcontrolador de Texas Instruments	966,78 €	3	79,46 €
Generador de señales	140,40 €	3	11,54 €
Osciloscopio	224,40 €	6	9,22 €
Total capítulo III			**100,22 €**

Tabla 7. Costes del capítulo III. [Fuente: EP]

1.4. Presupuesto final.

Total capítulo I	30.825,00 €
Total capítulo II	1.351,82 €
Total capítulo III	100,22 €
Total	32.277,04 €
IVA 21%	6.778,18 €
Total presupuesto	**39.055,22 €**

Tabla 8. Presupuesto total del proyecto. [Fuente: EP]

La elaboración del presente proyecto genera una suma total de **39.055,22 €**, treinta y nueve mil cincuenta y cinco euros y veintidós céntimos.

2. Ejecución del presupuesto.

2.1. Coste final del proyecto.

El presupuesto de ejecución del presupuesto es el siguiente:

Total capítulo I	30.825,00 €
Total capítulo II	1.429,11 €
Total capítulo III	100,22 €
Total	32.354,34 €
IVA 21%	6.794,41 €
Total presupuesto	**39.148,22 €**

Tabla 9. Presupuesto de ejecución. [Fuente: EP]

La ejecución del proyecto conlleva un coste total de **39.148,22 €,** treinta y nueve mil ciento cuarenta y ocho euros y veintidós céntimos.

2.2. Justificación de las desviaciones.

Se observa un incremento en el presupuesto inicial debido a la desviación generada por el coste final del dispositivo, que se refleja en el capítulo II: materiales.

Capítulo II: Materiales			
Descripción	**Unidades**	**Precio unitario**	**Coste**
Microcontrolador de Texas Instruments	1	86,00 €	86,00 €
Generador de señales	1	321,94 €	321,94 €
Osciloscopio	1	355,81 €	355,81 €
Voltímetro/Amperímetro	1	251,84 €	251,84 €
Material oficina (papel, bolígrafo, etc.)	1	20,00 €	20,00 €
Total costes directos			1.035,59 €
Total costes indirectos (20%)			207,12 €
Total costes			1.242,71 €
Imprevistos (15%)			186,41 €
Total capítulo II			**1.429,11 €**

Tabla 10. Presupuesto de ejecución capítulo II: materiales. [Fuente: EP]

La adquisición del microcontrolador a través de la empresa de TI tiene un coste de 29,99 €, pero no se pueden realizar repartos en Europa. Para resolver este imprevisto, se aplica el punto dos del plan de contingencia y se explora opciones de envió en Europa, en este caso, se hace el pedido a través de Amazon. Esta gestión conlleva un aumento en el coste del dispositivo 56 €, aumentando el presupuesto del proyecto en 93 €.

Centre universitari adscrit a la

TecnoCampus

upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO Y PUESTA EN MARCHA DE UNA BANCADA PARA EL DESARROLLO DE SOLUCIONES INDUSTRIALES INTEGRALES AVANZADAS CON MICROCONTROLADORES ARM CORTEX-M4

Anexos

HAMZA CHOULLI SAMADI
PONENT: DR. JULIÁN HORRILLO TELLO

JUNIO 2024

TecnoCampus
Mataró-Maresme

Índice.

Anexo I. Primer programa..... 1
Anexo II. Programa configuración *Task*. 5
Anexo III. Programa configuración Hwi y Swi. 7
Anexo IV. Programa configuración *Clock*. 11
Anexo V. Programa configuración semáforo. 13
Anexo VI. Programa configuración eventos. 15
Anexo VI. Programa configuración Gates. 17
Anexo VI. Programa configuración Mailbox. 19
Anexo VI. Programa configuración Queue. 21

Anexo I. Primer programa

```
/* XDCtools Header files */
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo
real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el
SO en tiempo real.

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para
interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

#define TASKSTACKSIZE   512 //Se define una macro con un valor de 512.

Task_Struct task0Struct; //Se declara una estructura de Task.
Char task0Stack[TASKSTACKSIZE]; //Se declara una matriz
                                //con un tamaño especificado por TASKSTACKSIZE

/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED0. The Task_sleep is determined by arg0 which
 *  is configured for the heartBeat Task instance.
 */
void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos
argumentos
                                        //La función tiene tipo de retorno
void,
                                        //lo que significa que no devuelve
ningún valor
{
    while (1) { //Se ejecuta un bucle infinito para el funcionamiento de la
Task
        Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de la
tarea
                                        //durante un período de tiempo
determinado.
```

```
        GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle para
cambiar el estado,
                                    //es decir, un parpadeo de un LED
específico del microcontrolador.
    }
}

/*
 *  ======== main ========
 */
int main(void)  //Se declara la función principal del programa
{
    Task_Params taskParams; //Se declara una variable del tipo Task_Params,
                            //que se utiliza para especificar los parámetros de
configuración de una tarea.
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del
dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el
microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

    /* Construct heartBeat Task  thread */
    Task_Params_init(&taskParams);  //Se inicializa la estructura con los
valores predeterminados de los parámetros
                                    //de la Task. Esto asegura que todos los
parámetros de la Task se inicialicen
                                    //correctamente antes de configurarlos de
manera individual.
    taskParams.arg0 = 1000; //Se establece el primer argumento de la Task.
    taskParams.stackSize = TASKSTACKSIZE; //Se establece el tamaño de la pila
de la Task.
    taskParams.stack = &task0Stack; //Se establece la pila de la Task.
    Task_construct(&task0Struct, (Task_FuncPtr)heartBeatFxn, &taskParams,
NULL);
    //Se contruye la tarea utilizando los parámetros especificados,
    //se le pasa la dirección de la estructura de la Task,
    //la función que la Task ejecutará,
    //los parámetros de la Task, y, por último, el NULL para indicar que no se
está utilizando ninguna Task
    //como instancia de un objeto.



     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    System_printf("Starting the example\nSystem provider is set to SysMin. "
                  "Halt the target to view any SysMin contents in ROV.\n");
```

```
    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha
ejecutado correctamente.
}
```

Anexo II. Programa configuración *Task*.

```
/*
 *  ======== empty.c ========
 */
/* XDCtools Header files */
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo
real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el
SO en tiempo real.

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para
interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED0. The Task_sleep is determined by arg0 which
 *  is configured for the heartBeat Task instance.
 */
void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos
argumentos
                                        //La función tiene tipo de retorno
void,
                                        //lo que significa que no devuelve
ningún valor
{
    while (1) { //Se ejecuta un bucle infinito para el funcionamiento de la
Task
        Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de la
tarea
                                        //durante un período de tiempo
determinado.
        GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle para
cambiar el estado,
                                    //es decir, un parpadeo de un LED
específico del microcontrolador.
```

```
    }
}

/*
 *  ======== main ========
 */
int main(void)  //Se declara la función principal del programa
{
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del
dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el
microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();


     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    System_printf("Starting the example\nSystem provider is set to SysMin. "
                  "Halt the target to view any SysMin contents in ROV.\n");
    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha
ejecutado correctamente.
}
```

Anexo III. Programa configuración Hwi y Swi.

```
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo
real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el
SO en tiempo real.
#include <ti/sysbios/knl/Swi.h>
#include <ti/sysbios/hal/Hwi.h>

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para
interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

volatile Bool blink = TRUE;

/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED0. The Task_sleep is determined by arg0 which
 *  is configured for the heartBeat Task instance.
 */

void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos
argumentos
                                        //La función tiene tipo de retorno
void,
                                        //lo que significa que no devuelve
ningún valor
{
    while (1) { //Se ejecuta un bucle infinito para el funcionamiento de la
Task
        if (blink == TRUE){
            Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de
la tarea
                                            //durante un período de tiempo
determinado.
            GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle
para cambiar el estado,
```

```
                                        //es decir, un parpadeo de un LED
específico del microcontrolador.
        }else{
            Task_sleep((unsigned int)arg0);
            GPIO_write(Board_LED0, Board_LED_ON); //Se ejecuta la función
GPIO_write, para cambiar el estado
                                                  //del LED0, en este caso,
permanece encendido.
        }
    }
}


void swiFxn0(UArg arg0, UArg arg1)
{
    blink = !blink; //Cambia el estado del blink, para modificar el
comportamiento del LED0.
}

void hwi0(UArg arg0)
{
    swiFxn0(0, 0); //Se lanza el thread Swi.
}


/*
 *  ======== main ========
 */
int main(void)  //Se declara la función principal del programa
{
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del
dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el
microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    GPIO_enableInt(Board_BUTTON0);  //Se habilita las interrupciones a través
del botón SW1.
    GPIO_setCallback(Board_BUTTON0, hwi0);  //Se ejecuta el thread hwi0 cuando
se presiona el botón SW1.

    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.
```

```
    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha
ejecutado correctamente.
}
```

Anexo IV. Programa configuración *Clock*.

```
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo
real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el
SO en tiempo real.
#include <ti/sysbios/knl/Swi.h>
#include <ti/sysbios/hal/Hwi.h>
#include <ti/sysbios/knl/Clock.h>

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para
interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

volatile Bool blink = TRUE;

/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED0. The Task_sleep is determined by arg0 which
 *  is configured for the heartBeat Task instance.
 */

void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos
argumentos
                                        //La función tiene tipo de retorno
void,
                                        //lo que significa que no devuelve
ningún valor
{
    while (1) { //Se ejecuta un bucle infinito para el funcionamiento de la
Task
        if (blink == TRUE){
            Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de
la tarea
                                            //durante un período de tiempo
determinado.
            GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle
para cambiar el estado,
```

```
                                        //es decir, un parpadeo de un LED
específico del microcontrolador.
        }else{
            Task_sleep((unsigned int)arg0);
            GPIO_write(Board_LED0, Board_LED_ON); //Se ejecuta la función
GPIO_write, para cambiar el estado
                                                  //del LED0, en este caso,
permanece encendido.
        }
    }
}


void clockFxn0(UArg arg0, UArg arg1)
{
    blink = !blink; //Cambia el estado del blink, para modificar el
comportamiento del LED0.
}
/*
 *  ======== main ========
 */
int main(void)  //Se declara la función principal del programa
{
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del
dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el
microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha
ejecutado correctamente.
}
```

Anexo V. Programa configuración semáforo.

```
#include <xdc/std.h> //Contiene definiciones de tipos de dato
#include <xdc/runtime/System.h> //Proporciona funciones para interactuar
                                //con el sistema en tiempo de ejecución.

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>    //Contiene definiciones y funciones
                                //relacionadas con la gestion del SO en tiempo
real.
#include <ti/sysbios/knl/Task.h>    //Proporciona las definiciones y funciones
                                    //necesarias para trabajar con Task en el
SO en tiempo real.
#include <ti/sysbios/knl/Swi.h>
#include <ti/sysbios/hal/Hwi.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>

/* TI-RTOS Header files */
// #include <ti/drivers/EMAC.h>
#include <ti/drivers/GPIO.h> //Proporciona definiciones y funciones para
interactuar con pines GPIO.
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/USBMSCHFatFs.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"  //Contiene las definiciones específicas
                    //de la placa de desarrollo que se utiliza en el proyecto.

/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED0. The Task_sleep is determined by arg0 which
 *  is configured for the heartBeat Task instance.
 */

void heartBeatFxn(UArg arg0, UArg arg1) //Se declara la función con dos
argumentos
                                        //La función tiene tipo de retorno
void,
                                        //lo que significa que no devuelve
ningún valor
{
    while(1){
        Task_sleep((unsigned int)arg0); //Esta línea pausa la ejecución de la
tarea
                                        //durante un período de tiempo
determinado.
        GPIO_toggle(Board_LED0);    //Se ejecuta la función GPIO_toggle para
cambiar el estado,
                                    //es decir, un parpadeo de un LED
específico del microcontrolador.
        Semaphore_post(sem0);       //Se publica el semáfor
```

```
    }
}

void taskFxn1(UArg arg0, UArg arg1)
{
    while(1){
        Semaphore_pend(sem0, BIOS_WAIT_FOREVER); //Línea que espera la
disponibilidad del recurso
        GPIO_write(Board_LED0, Board_LED_ON);   //El LED0 permanece encendido.
        Task_sleep((unsigned int)arg0);
    }
}

/*
 *  ======== main ========
 */
int main(void)  //Se declara la función principal del programa
{
    /* Call board init functions */
    Board_initGeneral();    //Se inicia la configuración general del
dispositivo.
    // Board_initEMAC();
    Board_initGPIO();   //Se inicia la configuración de los pines GPIO en el
microcontrolador,
                        //preparando las entradas y salidas.
    // Board_initI2C();
    // Board_initSDSPI();
    // Board_initSPI();
    // Board_initUART();
    // Board_initUSB(Board_USBDEVICE);
    // Board_initUSBMSCHFatFs();
    // Board_initWatchdog();
    // Board_initWiFi();

     /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED específico.

    //Se imprime un mensaje por la consola de depuración.
    /* SysMin will only print to the console when you call flush or exit */
    System_flush(); //Se limpia el buffer del sistema.

    /* Start BIOS */
    BIOS_start(); //Seinicia el SO en tiempo real.

    return (0); //El programa se encarga de devolver 0 para indicar que se ha
ejecutado correctamente.
}
```

Anexo VI. Programa configuración eventos.

```
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/drivers/GPIO.h>
#include <xdc/runtime/System.h>
#include "Board.h"


// Definición de eventos
#define EVENT_LED_ON 0x01 //Se define el evento de cuando el LED esta encendido
#define EVENT_LED_OFF 0x02  //Se define el evento de cuando el LED esta apagado


int main(void) {
    // Inicializa la placa
    Board_initGeneral();
    Board_initGPIO();

    // Inicia el sistema TI-RTOS
    BIOS_start();

    return (0);
}

void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        GPIO_write(Board_LED0, Board_LED_ON);   //Se enciende el LED
        Task_sleep((unsigned int)arg0);   //Se espera el tiempo assignado
        GPIO_write(Board_LED0, Board_LED_OFF);  //Se apaga el LED
        Task_sleep((unsigned int)arg0);   //Se espera el tiempo assignado
        Event_post(event0, EVENT_LED_ON); //Se lanza el evento de LED encendido
        Task_sleep((unsigned int)arg0);
        Event_post(event0, EVENT_LED_OFF);  //Se lanza el evento de LED apagado
    }
}

void taskFxn1(UArg arg0, UArg arg1) {
    UInt events;    //Se define una variable para el almacenamiento del evento.
    while (1) {
        events = Event_pend(event0, Event_Id_NONE, EVENT_LED_ON |
EVENT_LED_OFF, BIOS_WAIT_FOREVER);
        //Se mantiene a la esperar de los eventos de encendido o apagado del
LED
        if (events & EVENT_LED_ON){ //¿Se ha encendido el LED?
            System_printf("LED encendido\n"); //Se imprime un mensaje
            System_flush();
        }
        if (events & EVENT_LED_OFF){    //¿Se ha apagado el LED?
            System_printf("LED apagado\n"); //Se imprime un mensaje
            System_flush();
        }
    }
}
```

Anexo VI. Programa configuración Gates.

```
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/gates/GateAll.h>
#include <ti/drivers/GPIO.h>
#include <xdc/runtime/System.h>
#include "Board.h"

GateAll_Struct gateAll0Struct; //Se define la variable estructura del Gate
GateAll_Handle gateAll0;    //Se define el nombre del Gate

void ledOn(void) {
    GPIO_write(Board_LED0, Board_LED_ON);
    System_printf("LED encendido\n");
    System_flush();
}

void ledOff(void) {
    GPIO_write(Board_LED0, Board_LED_OFF);
    System_printf("LED apagado\n");
    System_flush();
}

int main(void) {
    Board_initGeneral();
    Board_initGPIO();

    GateAll_construct(&gateAll0Struct, NULL); //Se pone la dirección de
gateAll0Struct
    gateAll0 = GateAll_handle(&gateAll0Struct); //Se obtiene el handle desde la
estructura

    BIOS_start();

    return (0);
}

void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        IArg key = GateAll_enter(gateAll0); //Se entra al Gate.
        System_printf("heartBeatFxn\n");
        System_flush();
        ledOn();    //Se llama a la función ledOn.
        GateAll_leave(gateAll0, key);   //Se sale del Gate.
        Task_sleep((unsigned int)arg0);
        key = GateAll_enter(gateAll0);  //Se entra al Gate.
        System_printf("heartBeatFxn\n");
        System_flush();
        ledOff();   //Se llama a la función ledOff
        GateAll_leave(gateAll0, key);   //Se sale del Gate.
        Task_sleep((unsigned int)arg0);
    }
}

void taskFxn1(UArg arg0, UArg arg1) {
```

```
    while (1) {
        Task_sleep((unsigned int)arg0);
        IArg key = GateAll_enter(gateAll0); //Se entra al Gate
        static Bool ledState = FALSE;   //Se define la variable ledState
        if (ledState) {
            System_printf("taskFxn1\n");
            System_flush();
            ledOff();   //Se llama a la función ledOff.
        } else {
            System_printf("taskFxn1\n");
            System_flush();
            ledOn();    //Se llama a la función ledOn.
        }
        ledState = !ledState;   //Se niega el estado de la variable
        GateAll_leave(gateAll0, key);   //Se sale del Gate
        Task_sleep((unsigned int)arg0);
    }
}
```

Anexo VI. Programa configuración Mailbox.

```
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Mailbox.h>
#include <ti/sysbios/gates/GateAll.h>
#include <ti/drivers/GPIO.h>
#include <xdc/runtime/System.h>
#include "Board.h"

typedef struct {
    int ledState;   //1 para encender, 0 para apagar
} LedMsg;

// Declarar la Mailbox
Mailbox_Struct mailbox0;
Mailbox_Handle mailbox0Handle;

void ledOn(void) {
    GPIO_write(Board_LED0, Board_LED_ON);
    System_printf("LED encendido\n");
    System_flush();
}

void ledOff(void) {
    GPIO_write(Board_LED0, Board_LED_OFF);
    System_printf("LED apagado\n");
    System_flush();
}

int main(void) {
    Board_initGeneral();
    Board_initGPIO();

    Mailbox_Params mboxParams;
    Mailbox_Params_init(&mboxParams);
    Mailbox_construct(&mailbox0, sizeof(LedMsg), 10, &mboxParams, NULL);
    mailbox0Handle = Mailbox_handle(&mailbox0);

    BIOS_start();

    return (0);
}

void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        LedMsg msg;
        while (1) {
            static Bool ledState = FALSE; //Se altera el estado del LED en el
mensaje
            msg.ledState = ledState ? 0 : 1; //Se encarga de comprobar el
estado del Mailbox
            ledState = !ledState;
            if (Mailbox_post(mailbox0Handle, &msg, BIOS_NO_WAIT)){// Enviar el
mensaje a a través del Mailbox
                System_printf("Task 1: Mensaje enviado\n");
```

```
            } else {
                System_printf("Task 1: Fallo al enviar el mensaje\n");
            }
            System_flush();
            Task_sleep((unsigned int)arg0);
        }
    }
}

void taskFxn1(UArg arg0, UArg arg1) {
    LedMsg msg;
        while (1) {
            // Esperar a recibir un mensaje de la Mailbox
            if (Mailbox_pend(mailbox0Handle, &msg, BIOS_WAIT_FOREVER)) {
                System_printf("Task 2: Mensaje recibido\n");
                System_flush();

                // Procesar el mensaje para controlar el LED
                if (msg.ledState == 1) {
                    ledOn();
                } else {
                    ledOff();
                }
            } else {
                System_printf("Task 2: Fallo al recibir el mensaje\n");
                System_flush();
            }
        }
}
```

Anexo VI. Programa configuración Queue.

```
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Mailbox.h>
#include <ti/sysbios/gates/GateAll.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/GPIO.h>
#include <xdc/runtime/System.h>
#include <xdc/cfg/global.h>
#include "Board.h"

// Definición de la estructura de la Queue
typedef struct {
    Queue_Elem elem; // Elemento de Queue
    int ledState;    // 1 para encender, 0 para apagar
} LedMsg;

// Declarar la cola
Queue_Struct queue;
Queue_Handle queueHandle;

// Arreglo estático para los mensajes
#define NUMMSGS 10
LedMsg ledMsgs[NUMMSGS];
int msgIndex = 0;


void ledOn(void) {
    GPIO_write(Board_LED0, Board_LED_ON);
    System_printf("LED encendido\n");
    System_flush();
}

void ledOff(void) {
    GPIO_write(Board_LED0, Board_LED_OFF);
    System_printf("LED apagado\n");
    System_flush();
}

int main(void) {
    Board_initGeneral();
    Board_initGPIO();

    Queue_construct(&queue, NULL);
    queueHandle = Queue_handle(&queue);

    BIOS_start();

    return (0);
}

void heartBeatFxn(UArg arg0, UArg arg1) {
    while (1) {
        // Crear el mensaje
        LedMsg *msg = &ledMsgs[msgIndex];
```

```
        msgIndex = (msgIndex + 1) % NUMMSGS;

        // Alternar el estado del LED en el mensaje
        static Bool ledState = FALSE;
        msg->ledState = ledState ? 0 : 1;
        ledState = !ledState;

        // Enviar el mensaje a la Queue
        Queue_put(queueHandle, &msg->elem);
        System_printf("Task 1: Mensaje enviado\n");
        System_flush();
        Task_sleep((unsigned int) arg0);
    }
}


void taskFxn1(UArg arg0, UArg arg1) {
    while (1) {
        // Esperar a recibir un mensaje de la cola
        if (!Queue_empty(queueHandle)) {
            LedMsg *msg = (LedMsg *)Queue_get(queueHandle);

            // Procesar el mensaje para controlar el LED
            if (msg->ledState == 1) {
                ledOn();
            } else {
                ledOff();
            }
            System_printf("Task 2: Mensaje recibido y procesado\n");
            System_flush();
        }
        Task_sleep((unsigned int) arg0);
    }
}
```