Universidad Autónoma de Querétaro

Facultad de Informática

Maestría en Sistemas Computacionales

Aplicación de un modelo de comunicación inter-núcleo en microcontroladores multinúcleo

Tesis

Que como parte de los requisitos para obtener el Grado de

Maestro en Sistemas Computacionales

Presenta

Carlos Antonio Mosqueda Arvizu

Dirigido por:

Dr. Julio Alejandro Romero González

Dr. Julio Alejandro Romero González
Presidente

Dr. Ricardo Chaparro Sánchez
Secretario

Dra. Diana Margarita Córdova Esparza
Vocal

Dr. Alberto Vázquez Cervantes
Suplente

Dr. Juvenal Rodríguez Reséndiz
Suplente

Centro Universitario, Querétaro, Qro.
Septiembre, 2024
México

Esta tesis está dedicada especialmente a mis padres, a mi hija y a Ely. Gracias por motivarme siempre a perseguir mis sueños…

Agradecimientos

El contenido de esta tesis fue el resultado de una aventura de casi 3 años de aprendizajes, muchas dudas, trabajo duro y de mi regreso al ámbito académico, del cual estuve alejado por 6 años. Estudiar y trabajar no es fácil. Sin embargo, ha sido posible gracias a varias personas.

Primero que nada, quiero agradecer a mi director de tesis, el Dr. Julio Alejandro Romero González. Tuve un inicio titubeante, sobre todo al intentar abordar la problemática que se presenta en este trabajo. Sin embargo, Alejo tuvo una paciencia enorme al llevarme poco a poco estructurando mis ideas y mi trabajo. Le agradezco por su enorme esfuerzo para lograr la publicación, en conjunto, de dos artículos, estudiar mi tema, y por todo este apoyo que me brindó en la recta final de mis estudios de Maestría. Quiero agradecer también el apoyo brindado por los coautores del artículo publicado sobre el caso de estudio de LET y TDMA: la Dra. Diana Margarita Córdova Esparza, el Dr. Ricardo Chaparro Sánchez, el Dr. Juan Terven Salinas, Dr. Juvenal Rodríguez Reséndiz y de nueva cuenta al Dr. Romero. Agradecimientos especiales al Consejo Nacional de Humanidades, Ciencia y Tecnología (CONAHCYT) por el apoyo económico brindado durante mis estudios de posgrado.

Le doy las gracias a Gerardo Gómez Zepeda, Gerente de *Software* en el área de EBS (*Electronic Brake Systems*) en Continental Automotive, por darme la oportunidad de colocarme en un departamento donde se aplican bases y técnicas relacionadas al tema de mi tesis. Es aquí donde logré entender la importancia del tema y esclarecer muchas dudas que tenía al respecto.

La familia también fue parte, de una manera u otra, de este camino. Agradezco a mi madre, Rocío, por enseñarme a ser siempre curioso, persistente y apasionado en lo que hago. A mi padre, Antonio, por forjar en mi disciplina, constancia, pero a la vez enseñarme a relajarme y reír siempre. Agradezco a Ely, porque sin el empujón que me dio no me hubiera motivado a hacer un posgrado, y por todo su apoyo en mis momentos más complicados. Finalmente quiero agradecerle a mi hija Julieta, por tenerme paciencia y siempre sonreír, incluso cuando hemos tenido poco tiempo para estar juntos.

Resumen

La aparición de los microcontroladores y los procesadores multinúcleo ha sido influenciada por la necesidad de procesar una gran cantidad de datos en periodos muy cortos. Las industrias como la automotriz y la aeronáutica han iniciado la adopción de estas tecnologías con la finalidad posibilitar la implementación de nuevas funcionalidades de confort, tal como la conducción autónoma, así como para cumplir con los más recientes estándares de seguridad funcional. El uso del procesamiento concurrente supone retos para la implementación de las aplicaciones que fueron diseñadas para ser ejecutadas de manera lineal. Los recursos de los de hardware, como la memoria, se comparten ahora entre múltiples núcleos de procesamiento, lo que dificulta la predicción del flujo de datos y causa problemas de concurrencia, cuestiones que degradan el desempeño de los algoritmos implementados.

Este trabajo de tesis propone la integración de dos modelos de comunicación para mitigar los efectos adversos del procesamiento paralelo en la memoria compartida. El primer modelo es el Tiempo de Ejecución Lógica (LET), el cual se ha vuelto popular en los años recientes por su naturaleza determinista y sincronización sin mecanismos bloqueantes. El segundo es el método de Acceso Múltiple por División de Tiempo (TDMA), el cual provee asilamiento temporal, así como un método de sincronización utilizando ranuras de tiempo. El mecanismo propuesto permite que el flujo de datos entre núcleos sea predecible, así como sincronizado, mitigando los problemas generados por la concurrencia de la ejecución por núcleos paralelos.

La contribución de este documento es principalmente la integración de ambos modelos con la intención de reducir la cantidad y la variabilidad de la latencia en el flujo de datos entre núcleos de procesamiento, mientras se provee coherencia y determinismo. Esto incluye análisis de latencias y variabilidad, así como la comparación contra otras propuestas de solución. La solución propuesta se implementó y evaluó en una tarjeta de desarrollo enfocada a productos automotrices y se discuten los resultados.

Palabras clave: tiempo de ejecución lógica, acceso múltiple por división de tiempo, comunicación entre núcleos.

Abstract

The emergence of multicore microcontrollers and microprocessors has been motivated by the need to quickly process a lot of data. Automotive and aeronautic industries have started to adopt such technologies to enable the implementation of new comfort features, such as Autonomous Driving, and to comply with the latest functional safety standards for real-time systems. Concurrent processing poses challenges to the implementation of features that were designed for lineal single-core processors. Hardware resources, such as memory are now shared between multiple processing cores, which makes it more difficult to predict data flow while producing concurrency problems. This degrades the performance of implemented algorithms.

This work proposes the integration of two communication models to mitigate the adverse effects of parallel processes in data on shared memory. The first model is the Logical Execution Time, which has become popular lately due to its deterministic nature and its lockless synchronization capabilities. The second model is the Time Division Multiple Access method, which provides temporal isolation and synchronization through time slots. The proposed mechanism allows data flow between cores to be predictable and synchronized, mitigating the problems produced by concurrency of execution in parallel cores.

The principal contribution of this work is the integration of both mentioned models to reduce the quantity and variability of the latency in the data flow between cores while providing coherency and determinism. This includes latency and variability analysis and the comparison with other solution proposals. The proposed solution was implemented and evaluated by an Automotive-Oriented Development Board and results are discussed.

Keywords: logical execution time, time-division multiple access, inter-core communication.

Índice

Índice de Figuras

Índice de Tablas

Abreviaturas y Siglas

Las siguientes abreviaturas se utilizaron en el documento:

AUTOSAR	*AUTOmotive Open System ARchitecture*
DMA	*Direct Memory Access*
ECM	*Error Cuadrático Medio*
F2F	*First-to-First*
ISR	*Interrupt Service Routines*
ITM	*Instrumentation Trace Macrocell*
L2F	*Last-to-First*
L2L	*Last-to-Last*
LET	*Logical Execution Time*
MCAPI	*Multicore Communications Application Programming Interface*
MPSoC	*MultiProcessor System on Chip*
MPI	*Message Passing Interface*
MPU	*Memory Protection Unit*
NoC	*Network-on-a-Chip*
NVM	*Non-Volatile Memory*
OS	*Operative System*
OSEK	*Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug*
RAM	*Random Access Memory*
RTOS	*Real Time Operative System*
SL	*System level*
SPM	*ScratchPad Memories*
SRAM	*Static Random Access Memory*
TDMA	*Time-Division Multiple Access*
TPIU	*Trace Port Interface Unit*
VDX	*Vehicle Distributed eXecutive*

Introducción

La industria automotriz muestra necesidades crecientes respecto a sus ya complejos y sofisticados requerimientos. Con el desarrollo de novedosos sistemas de seguridad, sistemas de confort y nuevas funcionalidades, surge la necesidad de sistemas con mayor capacidad de procesamiento de datos para poder hacer frente a los requerimientos de nuevas tecnologías. De acuerdo con Beckert (2019), funcionalidades como la conducción altamente automatizada de los Sistemas de Asistencia y Conducción Autónoma (ADAS), así como las actualizaciones a través del aire (*Over-The-Air*) o la comunicación car2x, son algunos de los principales impulsores de la industria automotriz en la actualidad; por varias razones como: la mejora de la seguridad y la eficiencia, mejora la experiencia del usuario, reducción de costos de mantenimiento o la recopilación de datos y análisis del rendimiento de los vehículos en tiempo real (Beckert, 2019). Como resultado, la complejidad del software requerido para estas funciones ha crecido significativamente. Antinyan (2020) por su parte menciona que se ha observado un incremento gradual en la escala del software automotriz aproximadamente cada diez años desde el año 2000.

En la actualidad, los sistemas electrónicos de los vehículos se componen de redes complejas de computadoras y unidades de procesamiento llamadas Unidades de Control Electrónico (ECU) (Beckert, 2019, p. 173), que intercambian información continuamente entre los sensores y actuadores que monitorean y controlan el rendimiento del vehículo con sistemas específicos como el motor, la transmisión, los frenos, el sistema de climatización, los sistemas de iluminación, el sistema de sonido y otros sistemas orientados al confort. Estas unidades han ido evolucionando, de tal manera que se han hecho más complejas en *hardware* y, en consecuencia, en *software*. (Antinyan, 2020), menciona que el software implementado para un automóvil Volvo se compone de aproximadamente 100 millones de líneas de código, que se derivan de alrededor de 100.000 requisitos funcionales.

Para seguir integrando más funcionalidades en las ECUs el hardware utilizado tiene que ser reemplazado para ganar mayor poder de procesamiento (Beckert, 2019). Esta sustitución se ha hecho con procesadores con velocidades de reloj más altas y un número mayor de núcleos integrados. Dado que las ECUs implementan funcionalidades diferentes y específicas en un automóvil, se diseñan soluciones de *hardware* especialmente para cada aplicación.

En términos del *hardware*, la industria ha optado por iniciar a integrar soluciones existentes basadas en microcontroladores multinúcleo en las ECUs, comúnmente llamadas "sistemas multiprocesador en un chip" (MPSoC por sus siglas en inglés). Dichos circuitos integrados implementan nuevas funciones, como la capacidad de procesamiento paralelo y un mayor desempeño, que hacen más fácil implementar los requerimientos de seguridad funcional establecidos por el estándar internacional ISO26262 (Monot et al., 2010).

Los MPSoCs cuentan con funciones adicionales a las de un microcontrolador o un microprocesador común, mismas que dependen de la industria objetivo para la cual son comercializados. Para la industria automotriz, dichos sistemas comúnmente incluyen *hardware* para tecnologías empleadas por sus productos, como lo es la comunicación de red de controlador de área (CAN), red de interconexión local (LIN), activación de múltiples interrupciones por eventos físicos, convertidor analógico/digital y digital analógico, entre otras. Sin embargo, el desarrollo de nuevas funcionalidades ha llevado consigo un aumento en su complejidad y se han integrado más módulos de *hardware*, como lo son los bancos de memoria dual, con la capacidad de reprogramar una sección de memoria mientras otra ejecuta una aplicación, redes en un chip (NoCs), memorias de *ScratchPad*, sistemas redundantes (*Lockstep)*, entre otras. Dichas funcionalidades adicionales están orientadas al aprovechamiento del paralelismo de los núcleos de procesamiento integrados.

El diseño de *software* y los métodos de implementación deben adaptarse para poder beneficiarse del paralelismo de procesamiento. Esto implica distribuir porciones funcionales de *software* en diferentes núcleos, para *software* que tradicionalmente fue diseñado para ejecutarse en un solo núcleo, y esto tiene implicaciones en los sistemas operativos (SO), en los modelos de comunicación y las arquitecturas de *hardware* donde la memoria de acceso

aleatorio (RAM) se requiere en distintas implementaciones como: memoria compartida, memoria de *Scratchpad* y redes en chips (NoC) (Schoeberl et al., 2015).

Dado que las funciones distribuidas necesitan intercambiar datos continuamente, la necesidad de intercambio de datos entre núcleos se vuelve muy relevante. Las ECUs de tiempo real en los automóviles, al implementar sistemas de control, generalmente incluyen la implementación de cadenas de eventos de sensor-procesamiento-actuador. En dichas cadenas, la latencia, la coherencia de los datos y el flujo de datos son críticos para los algoritmos de control, que, en muchos casos, tienen que cumplir con requerimientos de seguridad funcional. Es por ello por lo que adquiere una gran importancia el estudio de la comunicación entre núcleos de procesamiento

Antecedentes

La comunicación entre distintas unidades de procesamiento es un concepto que se origina desde la creación de los sistemas distribuidos, los cuales no tienen la restricción de ejecutar tareas en tiempo real, pero que sentaron las bases de los sistemas multinúcleo, tomando como base los conceptos de protocolos como la Interfaz de Paso de Mensajes (MPI). Basado en este protocolo, la Asociación Multinúcleo (*Multicore Association*) propuso un protocolo orientado a sistemas embebidos llamado Interfaz de Programación de Aplicaciones de Comunicaciones inter-núcleo, MCAPI por sus siglas en inglés. Dicho protocolo se probó en arquitecturas x86 y PowerPC, las cuales caen en la categoría de procesadores de propósito general (Hung et al., 2012). Es por ello que ellos propusieron una librería propia basada en los fundamentos de MCAPI pero retirando los mecanismos más demandantes en términos de recursos, y la nombraron MSG, con enfoque de portabilidad al abstraer el acceso al *hardware* subyacente (Hung et al., 2012).

Dado que desarrollos como el MSG utiliza mecanismos bloqueantes para evitar accesos concurrentes a la memoria, su operación puede resultar en una degradación del desempeño en sistemas multinúcleo. Debido a que, para sistemas de tiempo real, el tiempo de respuesta y la latencia son fundamentales, se han desarrollado otros modelos que buscan

evitar el uso de mecanismos de contención y bloqueo. Una solución muy simple es el modelo de comunicación explícita, el cual consiste en utilizar un espacio de memoria, compartido entre núcleos de procesamiento, para compartir datos en variables compartidas. El problema de dicho modelo es que, a pesar de ser una solución sencilla para sistemas secuenciales de un núcleo, el paralelismo de ejecución provoca que se generen accesos concurrentes a dichas variables. De acuerdo con Hamman *et al.* (2017) esto puede afectar la consistencia de los datos, durante la ejecución de una tarea, en variables que no son solo de lectura, por lo que activaciones intercaladas resultaran en ejecuciones con distintos valores de datos.

El modelo de comunicación implícita se presenta como una de las soluciones a los problemas de concurrencia de acceso a memoria, y está definido en el estándar de AUTOSAR. La comunicación implícita se enfoca en mantener la consistencia de los datos por medio de copias locales de ellos. La ejecución de una tarea se realiza con una copia de un dato de una variable compartida, y esto impone la coherencia durante el transcurso de la ejecución. Este modelo, sin embargo, presenta desventajas en sistemas multinúcleo debido al problema de fluctuación de muestreo. Gemlau *et al.* (2021) mencionan que este problema se origina debido a que "*las operaciones de lectura y escritura pueden ocurrir en cualquier momento dentro de los intervalos de activación de las tareas*" y el tiempo de respuesta de cada tarea en el mejor y peor de los casos. Los sistemas de tiempo real requieren un comportamiento predecible y un tiempo de respuesta bajo, por lo que la fluctuación de muestreo impacta en el desempeño cuando dichos sistemas cuentan con múltiples núcleos.

El modelo de tiempo de ejecución lógica (*Logical Execution Time*), comúnmente conocido como LET, fue propuesto por primera vez por el lenguaje de programación de activación por tiempo Giotto en el año 2001, pero ha sido retomado y desarrollado en los años recientes debido a su característica determinista y su previsibilidad. La causalidad sigue siendo un problema vigente debido a que industrias como la automotriz o la aeroespacial han enfrentado la problemática de la migración de sus aplicaciones diseñadas para sistemas mononucleares a plataformas que integran múltiples núcleos de procesamiento.

Biondi *et al.* (2017), han trabajado en propuestas para la integración del modelo LET en plataformas multinúcleo, abordando las problemáticas de la asignación de memoria y

tiempo, proponiendo optimización y métodos de análisis de tiempo en un sistema de tiempo real basado en el estándar de AUTOSAR.

Martínez *et al.* (2020) desarrollaron metodologías de análisis y caracterización para cadenas de eventos utilizando los modelos de comunicación explícita, implícita y LET con la finalidad de proveer métricas de latencia. Pazzaglia *et al.* (2023) trabajaron en la optimización de la comunicación internúcleo aplicando LET en conjunto con la tecnología de Acceso Directo de Memoria (DMA) la cual delega el intercambio de datos entre núcleos de procesamiento al *hardware* de DMA, liberando carga del procesador y reduciendo la latencia en la transferencia. Beckert (2019) menciona los conceptos de Micro-LET y Macro-LET, los cuales se distinguen por proveer granularidad dependiendo de las necesidades de la aplicación, para grupos de tareas que formen parte de cadenas de eventos y distribuidos temporalmente. La propuesta del autor incluye el uso de dobles *buffers* o *buffers* de anillo para lograr integrar LET en ambientes multi núcleo.

Igarashi et al., 2020; Igarashi & Azumi, 2019 por su parte han indagado en las problemáticas temporales de la programación de tareas, utilizando modelos como el Grafo Acíclico Dirigido para la planificación de las tareas utilizando el modelo LET para el intercambio de datos entre múltiples núcleos, el cual complementan con un método heurístico para evitar contenciones de comunicación.

Pazzaglia *et al.* (2023) formularon una partición funcional y un método de optimización para la implementación de LET en sistemas multinúcleo críticos de tiempo. Gemlau *et al* (2021) han extendido el concepto de LET a nivel sistema (*System-Level Logical Execution Time*), orientando el paradigma a la implementación del modelo en sistemas distribuidos de tiempo real.

Adicional a los modelos de comunicación, se han propuesto soluciones basadas en la planeación de la ejecución de las tareas o *scheduling*, las cuales tienen como propuesta principal solucionar la concurrencia de acceso a los recursos por medio del aislamiento temporal. Beckert (2019) menciona que el Lenguaje de Definición de Tiempo, basado en Giotto, describe su ejecución en 2 máquinas, la máquina E, encargada de las operaciones

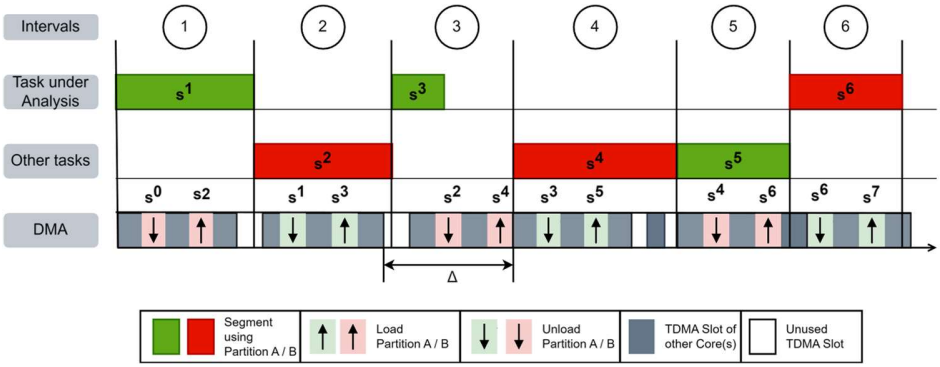
LET y la máquina S que representa la ejecución de la aplicación y que se implementa en un planeo de tareas basado en el Acceso Múltiple por División de Tiempo (TDMA), la cual es la base de varias implementaciones en el planeador de tareas llamado *scheduler*.

En la literatura, el concepto de Planificación Centrada en la Memoria (*Memory Centric Scheduling*, MSC) define un planificador de software separado para las secciones de las tareas que requieren acceso a la memoria principal. Algunas de las propuestas basadas en este concepto son las siguientes:

Soliman *et al.*(2019) propusieron un modelo de transmisión que permite el traslape de las operaciones de memoria en un esquema de tiempo segmentado utilizando el TDMA apoyándose de la tecnología del Acceso Directo de Memoria (DMA) y memorias locales (*Scratchpad*). Dicho modelo permite que las operaciones de lectura y escritura se ejecuten por el hardware: las lecturas se ejecutan previo a la ejecución de la siguiente tarea y las escrituras al inicio de la siguiente tarea. Previo a la ejecución de una tarea, el código y los datos correspondientes se cargan en la memoria local para la ejecución de la tarea y es posible cargar el código y los datos correspondientes a la siguiente tarea, si ésta es dependiente de la primera, realizando las operaciones en la memoria local. De no ser el caso, el DMA realiza la operación de carga y descarga hacia la memoria global. Un ejemplo utilizando dos tareas se puede observar en la Figura 1. Este enfoque es bastante complejo y requiere de implementación a nivel compilador y DMA, por lo que es dependiente del hardware. Tabish *et al.* (2019) proponen una solución muy similar con el enfoque general en los recursos compartidos, como el *hardware* que controla los periféricos de propósito general de un microcontrolador (GPIO). En dicha solución de manera similar el control es realizado por el sistema operativo y utilizando un esquema de TDMA para la transferencia de datos entre tareas de un mismo núcleo o de múltiples núcleos. Gemlau *et al.* (2021) por su parte especifica el uso de TDMA para el esquema de transferencia de un planificador de comunicación Ethernet entre distintos nodos de un sistema distribuido.

Figura 1

Acceso a DMA por medio de un esquema de TDMA



Nota: El diagrama representa el planificador de TDMA utilizando DMA para la transmisión de datos entre tareas en un esquema segmentado. Tomado de *Segment streaming for the three-phase execution model: Design and implementation* (p.3), por Soliman, *et al.* (2019).

Rivas *et al.* (2019) proponen una solución de MSC independiente del sistema operativo, que implementa un planificador para las fases de las operaciones de memoria y las interfaces para invocar el inicio y final de dichas fases. Los autores definen las fases como periodos de tiempo en los cuales se bloquean accesos concurrentes durante las operaciones en memoria. La propuesta define, además, que dichos bloqueos pueden ser interrumpibles para evitar los tiempos de bloqueo además de que cada fase de memoria tiene prioridades fijas. Sin embargo, esta solución demostró ser muy efectiva para las tareas de mayor prioridad y no se enfoca en la predictibilidad del flujo de datos.

Fundamentación Teórica

Sistemas Embebidos

Urbina (2020) menciona que un sistema embebido es un "sistema resultante de una combinación de componentes de hardware y software para desempeñar una o más tareas en particular" (p.7). Por otro lado, Heath (2003) define un sistema embebido como "sistema basado en un microprocesador que consiste en una unidad de control de instrucciones y un control aritmético, que fue construido para realizar una función o varias funciones y de manera diferente a una computadora personal no está diseñada para ser programada por el usuario final" (p.2). Para los sistemas de control, en industrias como la automotriz y la aeroespacial se utilizan sistemas embebidos, debido a que dichos sistemas están diseñados para implementar funciones específicas con requerimientos donde el tiempo de respuesta es relevante.

Sistemas de Tiempo Real

Kopetz (2011) menciona que:

> Un sistema computacional puede ser visto como un sistema de tiempo real cuando la exactitud de su salida no solo depende en el resultado lógico de la computación, sino que también en el tiempo físico cuando el comportamiento de salida fue producido por el sistema con respecto al tiempo global base" (p.2).

Los sistemas de control en el contexto de la industria automotriz requieren ser de tiempo real debido a los requerimientos críticos de tiempo de reacción. Dichos requerimientos temporales pueden ser relevantes debido a que ciertas funcionalidades requieren una rápida reacción en un actuador, ya sea para su correcto funcionamiento o incluso para cumplir con

requerimientos críticos de seguridad, donde el tiempo de respuesta es relevante para cumplir con estándares como los especificados por la ISO 26262, para la seguridad del usuario.

Depuradores

Para poder depurar los sistemas de tiempo real durante las etapas de desarrollo del software, se desarrollan dispositivos, llamados depuradores, capaces de acceder directamente a algunos recursos físicos de un microcontrolador o un microprocesador, el cual también cuenta con el *hardware* necesario para poder proveer un medio de muestreo de datos. Esto es útil para casos de uso donde se requiere comprobar el estado de los datos en un punto determinado de la ejecución del software, así como para extraer y recopilar los valores de variables con la finalidad de tener mediciones del comportamiento de los algoritmos con los datos.

Dependiendo de la arquitectura de cada procesador es que se diseñan las herramientas antes mencionadas por lo que es muy común que el mismo manufacturero de los circuitos integrados diseñe los depuradores, pero también, para arquitecturas estandarizadas como los núcleos de ARM, es posible encontrar diversos proveedores de depuradores.

Un ejemplo es el depurador *J-Link* de la empresa Segger, el cual está equipado con funciones básicas de depuración, así como estar dotado de funciones de perfilamiento de tareas y trazado. IAR Systems por su parte provee un ambiente completo e integrado con su solución que incluye el ambiente de desarrollo IAR Embedded Workbench, el cual tiene una interfaz con sus depuradores *I-jet* y *I-trace* que además de incluir perfilamiento de tareas y trazado de llamadas a funciones, también cuenta con la Unidad de Interfaz de Puerto de Trazado (*TPIU*) el cual extrae los datos del *hardware* de Macro-celda de Trazado de Instrumentación (*ITM*), la cual es útil para el muestreo de valores de datos asignados a espacios de memoria RAM o ROM, debido a que de acuerdo a con el Manual de Referencia de Arquitectura de ARM v7-M (2006), se implementa en registros de *hardware* los cuales al

ser escritos generan paquetes de datos que son enviados mediante comunicación serial al depurador en conjunto con las marcas de tiempo del momento en que se accedió al registro.

Arquitectura de Software

El software automotriz orientado a sistemas embebidos de tiempo real se desarrolla y se integra de acuerdo con los estándares de la Arquitectura Automotriz de Sistema Abierto o AUTOSAR por sus siglas en inglés.

AUTOSAR define una plataforma de software estandarizada para la electrónica del vehículo, la cual incluye interfaces y componentes de software para sistemas electrónicos como el motor, la transmisión, sistemas de seguridad, sistema de entretenimiento entre otros. En esta arquitectura de software distribuida interactúan distintos componentes a través de una red de comunicación interna.

Principalmente la plataforma AUTOSAR busca mejorar la calidad, la eficiencia y la interoperabilidad del software de electrónica del vehículo. Además, esta plataforma es modular, lo que permite la reutilización de software en diferentes sistemas y vehículos, por lo cual se pueden reducir los costos y mejorar la eficiencia del desarrollo.

OSEK – AUTOSAR OS

Para los sistemas embebidos de tiempo real en la industria automotriz, se utilizan sistemas operativos de tiempo real adaptados para las necesidades de las aplicaciones propias de la industria. De estas necesidades surgieron sistemas operativos como OSEK-VDX, el cual evolucionó a lo que se le conoce el día de hoy como AUTOSAR OS. Para la aplicación descrita en este documento, se utilizó AUTOSAR OS por su orientación a las aplicaciones

propias de las ECUs, por su orientación temporal, de *scheduler* estático, además de proveer la funcionalidad mínima requerida.

Arquitecturas de un solo núcleo

En los sistemas clásicos de un solo núcleo, se ejecutan todas las tareas en secuencia, lo que puede limitar el rendimiento y la capacidad de multitarea en comparación con los procesadores de múltiples núcleos. Además, los mecanismos de comunicación entre los diferentes componentes de funciones de *software* se realizan a través de la comunicación entre tareas, ya que las tareas en dichos sistemas tienen acceso al mismo espacio de direcciones de memoria global.

La comunicación entre tareas en componentes de funciones de software puede llevarse a cabo mediante varios mecanismos de comunicación, como variables compartidas, señales o eventos, colas o buffers, y sockets. El método de comunicación utilizado dependerá del contexto y los requisitos específicos del sistema o componente de software en cuestión, considerando factores como la sincronización, la eficiencia, la escalabilidad y la seguridad.

Dentro de estos mecanismos de comunicación se encuentra la comunicación explícita, en donde los componentes deben conocer la identidad y ubicación de los demás componentes con los que necesitan comunicarse, y se comunican enviando y recibiendo mensajes a través de un canal de comunicación compartido. Este tipo de comunicación permite el acceso sin restricciones a las variables compartidas entre tareas. De acuerdo con Hamann et al (2017) es una semántica que se utiliza en configuraciones de un solo núcleo y requiere de mecanismos de protección para evitar que las tareas de mayor prioridad afecten la coherencia de los datos en ambos sistemas preventivos, donde una tarea puede interrumpir a otra.

Arquitecturas de múltiples núcleos

Las arquitecturas de múltiples núcleos son sistemas con varios núcleos de procesamiento en un solo chip o procesador. Cada núcleo de procesamiento puede realizar operaciones independientes en paralelo, lo que aumenta la capacidad de procesamiento del sistema y reduce el tiempo de espera para completar las tareas. En este tipo de arquitecturas se han desarrollado varios mecanismos y modelos de comunicación tanto en *software* como en diferentes arquitecturas de *hardware* para proporcionar comunicación entre núcleos.

En sistemas multinúcleo el acceso al mismo espacio de memoria se expande a dos o más entidades de procesamiento en paralelo, que incluso pueden estar ejecutando diferentes sistemas operativos. Según Hung et al (2012), las arquitecturas multinúcleo se pueden dividir en dos categorías: a) homogéneas y b) heterogéneas.

- **Arquitecturas homogéneas:** son arquitecturas simétricas donde los núcleos de procesador son idénticos y de propósito general.
- **Arquitecturas heterogéneas:** integran diferentes núcleos para aumentar la eficiencia de los sistemas para aplicaciones específicas.

De acuerdo con Hung *et al.*, "para las aplicaciones integradas que apuntan a la eficiencia energética, generalmente se adoptan arquitecturas heterogéneas" (2012, p. 31).

Memoria

El mismo enfoque de usar el espacio de direcciones de memoria compartida puede aplicarse en sistemas multinúcleo, según la arquitectura de hardware del MPSoC. Sin embargo, esto trae a la mesa escenarios donde la coherencia de los datos puede verse afectada, debido al acceso simultáneo a una ubicación de memoria por parte de múltiples núcleos, operaciones de lectura y escritura superpuestas. Dado que la coherencia de datos

puede ser muy crítica en los sistemas de tiempo real, se han desarrollado y estudiado varias propuestas de *hardware*, de *software* y una combinación de ambos.

Para el esquema de memoria compartida, existen soluciones como *Scratchpad*, memoria junto con soluciones especializadas *Network-on-Chip* (NoC); *locks* son mecanismos de sincronización que permiten que solo un proceso acceda a una sección crítica de código a la vez, otro mecanismo son las barreras de memoria que garantizan que todos los procesos alcancen un punto de sincronización antes de continuar. También existen las variables atómicas permiten la lectura y modificación de variables sin conflictos de concurrencia, y finalmente la memoria transaccional que permite la ejecución de transacciones en paralelo. Tales soluciones varían en términos de complejidad, latencia de transmisión de datos y protección al acceso concurrente.

Comunicación

Para poder comprender mejor el contexto de la comunicación en los sistemas embebidos, en esta sección se describen los conceptos básicos que componen la comunicación, así como los modelos fundamentales y técnicas propuestas en la literatura.

Cadenas de eventos

Los sistemas de tiempo real producen eventos de salida dependientes de una entrada de datos. Es por ello por lo que generalmente implementan sistemas de control o sistemas intermedios de procesamiento de datos. Los eventos de entrada definen el inicio de un flujo de datos que se deriva en eventos de salida. Hamann et al (2017) definen que a dicha abstracción se le denomina cadena de eventos. Las cadenas de eventos en sistemas de control se representan desde la lectura de sensores, cuyos datos son enviados a algoritmos de control para ser procesados para luego servir de parámetros de salida para accionar actuadores.

Las cadenas de eventos tienen requerimientos dependiendo la criticidad de la funcionalidad a implementar, los cuales definen las tazas de muestreo y/o activación por eventos arbitrarios. Los tiempos de respuesta de las cadenas de eventos dependen tanto de la velocidad de respuesta de los recursos de hardware, como de la estructura del software, que es influenciada por la arquitectura, el gestor de tareas y la eficiencia de los algoritmos. Es por ello por lo que la latencia es una restricción importante para la selección del hardware y la definición de la estructura interna del software.

La complejidad de las cadenas y las distintas etapas de procesamiento de las que se pueden componer influyen en la latencia, sobre todo en los Sistemas Multi Tasa, los cuales, de acuerdo con Becker *et al.*, se pueden componer de distintas etapas con distintos periodos de tarea, llevando a generar condiciones de sobre muestreo y submuestreo. Es por ello por lo que es de suma importancia elegir la semántica adecuada de comunicación, así como la taza de activación de las tareas.

Comunicación basada en intercambio de mensajes

Existen soluciones en forma de bibliotecas de *software* cuyo objetivo es adaptar estándares de comunicación previamente desarrollados para sistemas distribuidos, como la interfaz de paso de masaje (MPI).

Según Hung *et al.* (2011), los estándares como MPI "no son adecuados para sistemas embebidos ya que consumen demasiados recursos de un sistema y muchas de las funciones que proporcionan son excesivas" (p. 193). La Asociación *Multicore* propuso otro estándar llamado Interfaz de programación de aplicaciones de comunicaciones multinúcleo (MCAPI), esta interfaz facilita la comunicación y la sincronización entre diferentes núcleos o procesadores en sistemas multinúcleo. Proporciona una capa de abstracción que oculta la complejidad de la comunicación entre diferentes núcleos o procesadores, permitiendo que se desarrollen aplicaciones en paralelo de una manera más fácil y eficiente, que fue diseñado específicamente para sistemas integrados.

En el trabajo desarrollado por Hung *et al.*, (2012), se menciona que el estándar apenas se probó y solo en sistemas heterogéneos que utilizan arquitecturas x86 y PowerPC, razón por la cual los autores propusieron su propia biblioteca, llamada MSG, que abstrae la implementación de paso de mensajes del *hardware* subyacente para que se vuelva portátil y por lo tanto, se use en diferentes arquitecturas de *hardware*, lo cual permite que el usuario implemente los mecanismos de capa baja, lo que podría implicar el uso de *hardware* como acceso directo a memoria (DMA), memoria compartida, NoC y otras que son soluciones basadas en *hardware* de distintas familias de microcontroladores.

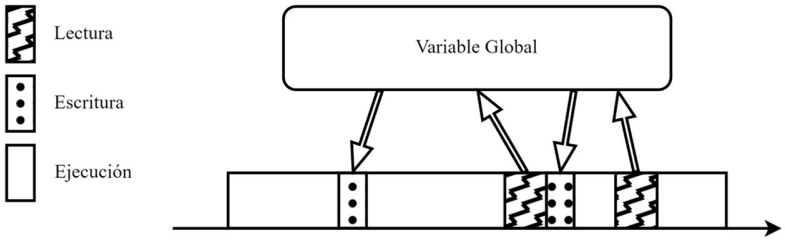
Se han propuesto métodos que se implementan sobre memoria compartida, para garantizar la coherencia de los datos durante la comunicación, ya que los diseños en sistemas de tiempo real para la industria automotriz, la incoherencia de datos puede ser un problema muy crítico que puede afectar a los sistemas de control si no se toman medidas para garantizar la coherencia de los datos. Esto puede llevar a diferentes versiones de los mismos datos en los cachés o registros de diferentes núcleos o procesadores, ya que dependen en gran medida de los datos que reciben como entrada (variables), para proporcionar la salida deseada (cadenas de ejecución de sensor-actuador).

Comunicación Explícita

La comunicación explícita es la forma más básica de intercambio de datos entre núcleos de procesamiento. Esta semántica permite el acceso sin restricciones a las variables locales. Este tipo de comunicación es útil para variables que son accedidas solamente para lectura. Para variables con accesos de lectura y escritura, esta semántica puede generar problemas de inconsistencia de datos cuando hay múltiples activaciones de tareas que entrelazan sus tiempos de acceso a las variables, dado que los valores de las variables pueden estar cambiando constantemente por escrituras procedentes de distintas tareas [Hamann, 2017].

Figura 2

Comunicación Explícita (Hamann et al., 2017)

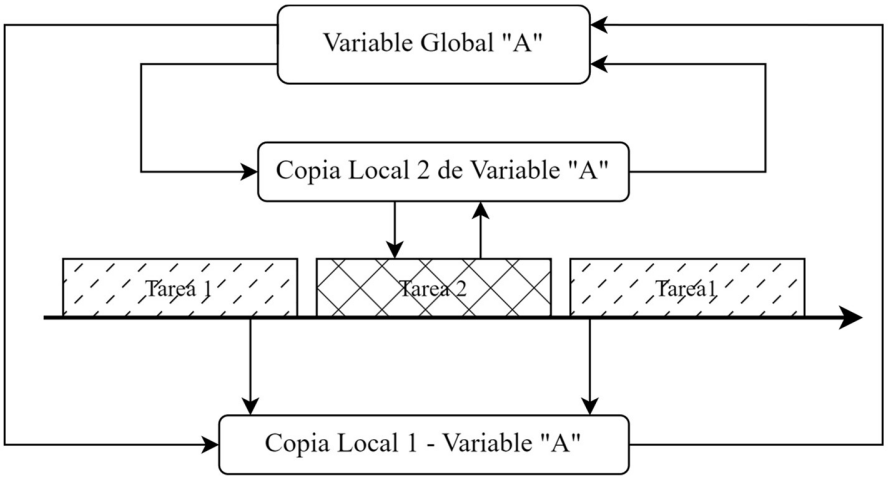


Comunicación Implícita

La comunicación implícita también se presenta en publicaciones posteriores como una solución, la cual está descrita como parte del estándar de AUTOSAR. “Esta semántica… se enfoca principalmente en mantener la consistencia de los datos para evitar las trampas de la comunicación explícita” (Hamann et al., 2017, p. 7). La comunicación implícita requiere tareas para hacer copias locales de los datos compartidos a través de variables compartidas al comienzo de su ejecución.

Figura 3

Comunicación Implícita (Hamann et al., 2017)



Según Hamann *et al.*, "esto asegura que la tarea funcione en la misma copia a lo largo de su ejecución, y también preserva el estado consistente de los datos" (2017, p. 7). El único problema para este tipo de modelo es que no es inmune a la fluctuación de muestreo, que ocurre cuando una tarea de productor se ejecuta a un ritmo más rápido que las tareas de consumidor. Dicho modelo también se conoce como tiempo de ejecución limitado "donde las operaciones de lectura y escritura pueden ocurrir en cualquier momento dentro de intervalos que dependen de la activación de la tarea, la fluctuación de muestreo, el tiempo de respuesta en el mejor y el peor de los casos" (K.-B. Gemlau et al., 2021, p. 9).

El problema de fluctuación de muestreo se intensifica cuando hay ejecución paralela de tareas, motivo por el que la comunicación se vuelve no determinista por la falta de sincronización entre los planificadores en diferentes núcleos, especialmente en sistemas heterogéneos.

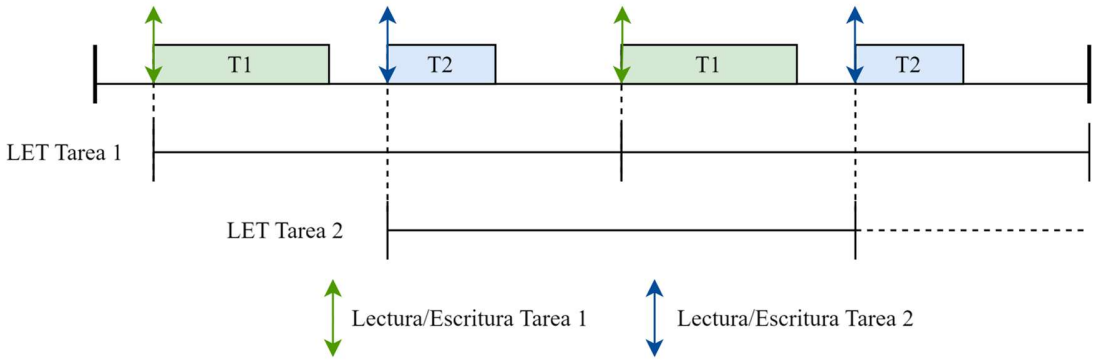
Tiempo de Ejecución Lógica

Para abordar los problemas de la comunicación implícita, el modelo de Tiempo de ejecución Lógica LET ha atraído el interés en los últimos años. "A diferencia del caso

implícito, LET impone la comunicación de la tarea en tiempos determinados, correspondientes a los tiempos de activación de la tarea" (Martínez et al., 2018, p. 13). LET se derivó del lenguaje de programación Giotto, que se activa con el tiempo y fue diseñado para sistemas de control con restricciones estrictas de tiempo real. "Es un concepto de programación en tiempo real que garantiza el determinismo temporal al desvincular la computación y la comunicación" (Hamann et al., 2017, p. 8).

Figura 4

Tiempo de Ejecución Lógica



De acuerdo con Igarashi *et al.* (2020), LET funciona fijando los tiempos de procesamiento de lectura y escritura independientemente del tiempo de ejecución de la tarea. Este modelo elimina la necesidad de métodos de sincronización como *mutex*, semáforos, áreas exclusivas, etc. También proporciona determinismo al fijar las operaciones de lectura y escritura en el tiempo, sacrificando la latencia a favor de la consistencia de los datos.

El paradigma del modelo LET se originó en un contexto con reglas definidas para sistemas en tiempo real con múltiples tareas que se ejecutan periódicamente, forzando el determinismo en las cadenas de eventos donde la consistencia de datos es un requisito. Esto es un aspecto importante necesario para los sistemas de control. Sin embargo, si este modelo se implementa con su definición original en sistemas multinúcleo, la falta de mecanismos de sincronización entre tareas que se ejecutan en paralelo, accesos de lectura y accesos de escritura pueden superponerse, según Beckert (2019).

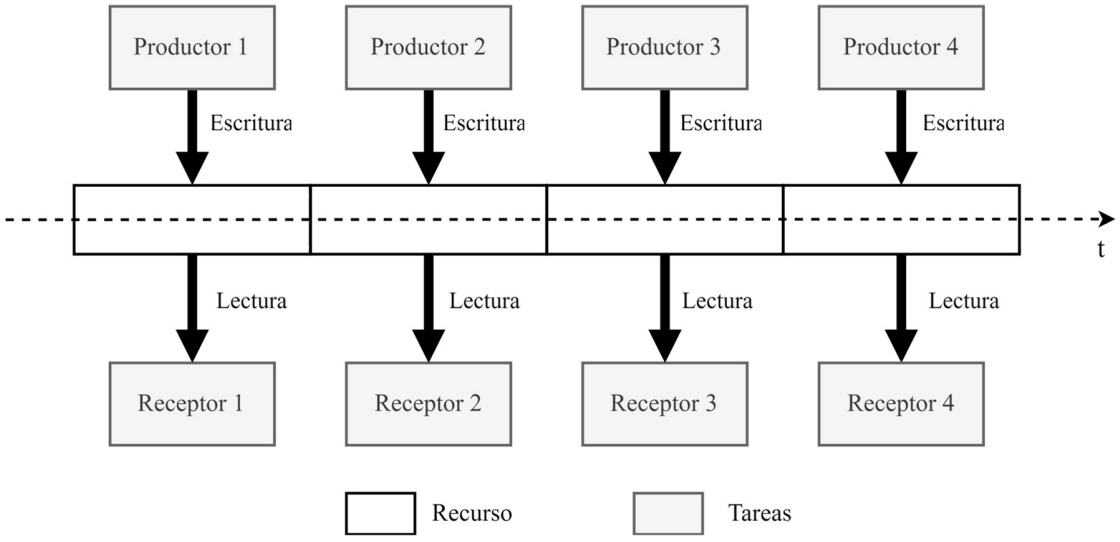
Recientemente se han presentado nuevas propuestas en las que el modelo LET, donde se combina con otros algoritmos existentes o con métodos heurísticos con el objetivo de mitigar o eliminar las desventajas y proporcionar soluciones útiles para los sistemas de tiempo real. Algunas otras propuestas, como la de Gemlau *et al.* (2021), se enfocan en derivar modelos de LET para que puedan ser aplicados en otros contextos como en aplicaciones industriales donde los sistemas distribuidos pueden beneficiarse de LET, pero desde una perspectiva diferente, debido a la latencia en tales redes es mucho mayor que la que existe entre los núcleos integrados. Por lo tanto, se propone una extensión denominada tiempo de ejecución lógica a nivel sistema (SL-LET).

Acceso Múltiple por División de Tiempo

El TDMA, de acuerdo con Kopetz (2011), se define como una tecnología de comunicación donde el eje del tiempo está estáticamente particionado en ranuras de tiempo. Cada ranura de tiempo está estáticamente asignada a un componente y solo se le permite mandar un mensaje a dicho componente durante la duración de su ranura asignada. El TDMA se ha utilizado en varias aplicaciones tecnológicas, como las comunicaciones inalámbricas donde se busca aprovechar el ancho de banda de un canal en su totalidad. Como fue discutido en la sección *Acceso Múltiple por División de Tiempo*, los métodos basados en el TDMA se han aplicado en sistemas multinúcleo de tiempo real con la finalidad de proveer determinismo al flujo de los datos en las cadenas de eventos. En los estudios "*Segment streaming for the three-phase execution model: Design and implementation"* (Soliman et al., 2019), "*A new approach to guarantee critical task schedulability in TDMA-based bus access of multicore architecture"* Lara et al. (2019) y "*A real-time scratchpad-centric OS with predictable inter/intra-core communication for multi-core embedded systems"* Tabish et al. (2019) se proponen soluciones que implican modificaciones en el planificador del sistema operativo además del uso de la tecnología DMA, mientras que Beckert (2019) propone una solución agnóstica al sistema operativo con un enfoque hacia la priorización del flujo de los datos basado en la prioridad de sus tareas correspondientes.

Figura 5

Recurso compartido por el esquema de acceso temporal de TDMA



Tiempo de Ejecución Lógica en múltiples núcleos utilizando TDMA

Pazzaglia et al. (2023) menciona que la definición original de Giotto sobre el modelo LET, menciona que cada instancia periódica de una tarea *Ti,* actualiza sus valores de entrada al momento de iniciar su activación. Dicha tarea utiliza esos valores para calcular nuevos valores de salida, que son puestos a disposición de una tarea consumidora al final de cada periodo. A estas transacciones se les denomina comunicaciones LET. Es por ello por lo que Pazzaglia define las siguientes propiedades para el modelo LET:

1. **Comunicaciones hacia y desde la misma tarea:** Esta propiedad determina que las escrituras se tienen que realizar al final de un periodo *T* y las escrituras al inicio del periodo *T+1*. En términos prácticos se puede asumir que al cumplimiento del periodo *T* se deben realizar primero las operaciones de escritura correspondientes a la tarea que acaba de finalizar su ejecución e

inmediatamente iniciar las operaciones de lectura para la ejecución periódica subsecuente.

2. **Comunicación LET intra-tarea:** Para mantener las dependencias causales entre tareas comunicándose por medio del modelo LET, las operaciones de escritura que se inician al final de cada periodo deben siempre terminarse antes de iniciar las operaciones de lectura correspondientes al periodo siguiente.
3. **Traslape:** Las comunicaciones LET que se activen en distintos instantes de tiempo no deben traslaparse entre sí. Esto bajo la suposición realista de que las operaciones siempre toman un tiempo mayor a 0.

Las propiedades mencionadas mantienen predictibilidad bajo un conjunto de tareas estáticas administradas por un *scheduler* de prioridades fijas y periodos definidos. Incluso si se asume una carga total del procesador menor o al 100%, las interrupciones por tareas de mayor prioridad no deberían afectar la temporalidad de los datos. Sin embargo, este escenario cambia en un entorno capaz de ejecutar tareas en paralelo.

El uso del modelo LET, utilizando su definición original, para comunicar tareas ejecutándose en distintos núcleos de procesamiento tiene los siguientes inconvenientes:

1. **Temporalidad variable:** La ejecución de tareas concurrentes depende de su núcleo de procesamiento directamente y su propia instancia de sistema operativo. Dicha característica puede causar variabilidad en los tiempos de ejecución de cada núcleo. Dicho esto, asumiendo que se tiene una tarea productora *Tp* ejecutándose una vez en un periodo *T* en un núcleo *X* y una tarea consumidora *Tc* ejecutándose en un periodo *T* en un núcleo *Y*, se puede decir que el tiempo de desfase entre la ejecución de *Tp* y *Tc* se definir por:

$$\Delta = |t_{Tp} - t_{Tc}|$$

donde t_{Tp} es el tiempo de activación de la tarea Tp y t_{Tc} es el tiempo de activación de la tarea Tc. Considerando que el periodo T es igual para ambas tareas, se puede asumir entonces que

$$\Delta_{max} = T$$

en ambos casos, donde

$$t_{Tc} = t_{Tp} + T$$

o

$$t_{Tp} = t_{Tc} + T$$

Bajo este escenario los datos de la tarea productora se consumen con un retraso de T.

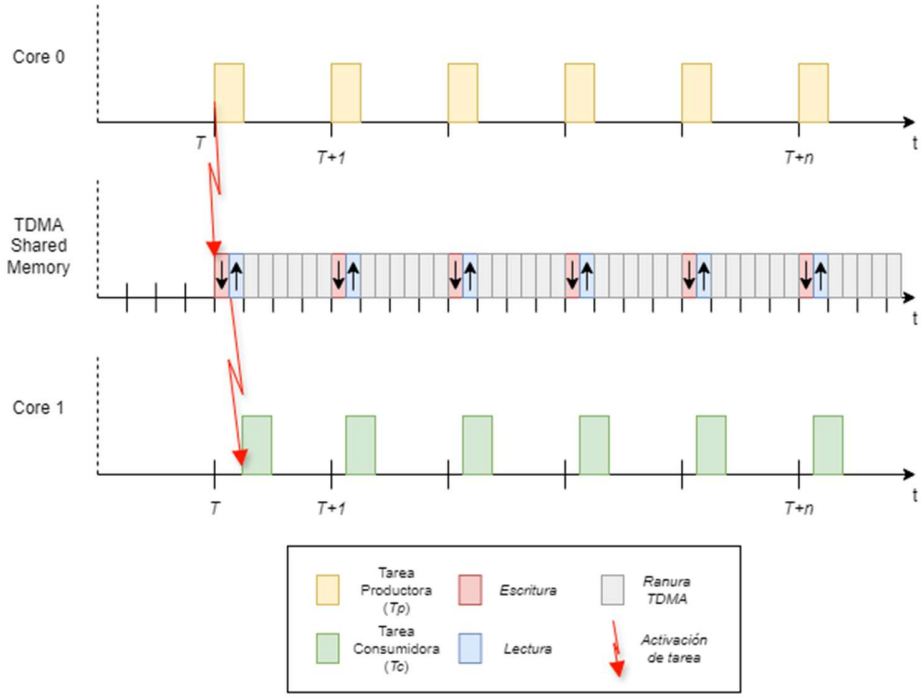
2. **Traslape de operaciones:** Considerando el escenario anterior, en el peor de los casos (Δ_{max}) se pueden generar problemas de contención al realizar operaciones en una misma dirección de memoria, y producir fluctuación de muestreo. En un ambiente real es muy común que las operaciones de lectura sean tan rápidas como un ciclo de reloj, mientras que las operaciones de escritura tienden a ser más tardadas. Dado que esto supone la inversión del orden de las operaciones de acceso a memoria dependiendo de la rapidez y las características del hardware (violando la Propiedad 1 de LET), es posible que se produzca pérdida de datos a falta de sincronización de las operaciones entre productor y consumidor.

Con la finalidad de minimizar estos problemas, este estudio propone la complementación del modelo LET con un esquema de TDMA para el acceso a la memoria compartida entre dos núcleos. Para ello se deben definir las cadenas de eventos que impliquen flujo de datos de un núcleo a otro y se deben identificar sus tareas productoras y

consumidoras, así como sus periodos de ejecución y tiempo de desfase. Del conjunto de tareas se deben identificar las tareas productoras y consumidoras que impliquen una comunicación directa entre núcleos. Con las propiedades de dichas tareas se puede definir la duración de las ranuras de tiempo bajo el esquema de TDMA, con las cuales se controla el acceso a la memoria compartida. Para definir el periodo LET de cada par de tareas se identifica el periodo de ejecución más lento entre tareas que se comunican. Por último, se debe seleccionar una tarea iniciadora para sincronizar las operaciones de las ventanas de LET en los núcleos en comunicación.

Figura 6

Modelo LET con Memoria Compartida por TDMA



Esta propuesta busca que se mantengan las propiedades del modelo LET al fijar el orden de las operaciones de lectura y escritura entre tareas en núcleos distintos, además de proveer aislamiento temporal al acceder a la memoria con el esquema TDMA.

Latencia

La latencia es una característica importante para la evaluación de los distintos métodos y modelos de comunicación, debido a que esta define los tiempos de respuesta en un sistema. De acuerdo con Krawczyk et al. (2019), para implementar una funcionalidad, se requiere cumplir con ciertas restricciones para que la reacción a un estímulo de entrada se garantice dentro de un tiempo especificado.

La latencia se mide tanto en los recursos del hardware (p. ej. latencia de acceso a la memoria) como en las entidades de software. En el software de sistemas de control, se definen las cadenas de efecto, las cuales ejecutan las distintas etapas de procesamiento de datos dada una entrada (p. ej. lecturas de un sensor), para poder realizar una acción a la salida (p. ej. la activación de un actuador). Las latencias en la respuesta de hardware generalmente tienden a ser mucho menores que las latencias en las cadenas de efecto, por lo que pocas veces tiene impacto en los cálculos de la latencia de punto a punto.

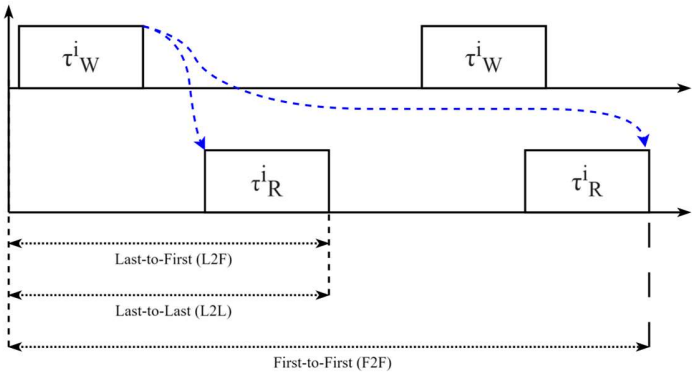
Existen varios tipos de latencia dependiendo los tiempos que se quieran calcular y medir en las distintas partes del software. Para la latencia de punto a punto, Feiertag et al. (2008) presentan las siguientes semánticas para el análisis punto a punto (*End-to-End*):

1. **Latencia de Máxima Edad (*L2L*):** Esta latencia considera el tiempo desde la última entrada de datos no sobrescrita hasta la última salida de datos generará por la misma entrada, considerando duplicados de esta.
2. **Latencia de Edad (*L2F*):** Esta latencia considera el tiempo desde la última entrada de datos no sobrescrita hasta la primera salida generada por la misma entrada.
3. **Latencia reactiva (*F2F*):** Esta latencia considera el tiempo entre la primera entrada de datos que no es sobrescrita hasta la primera salida de datos generada por la siguiente entrada.

Estas latencias son útiles dependiendo el contexto de las cadenas de efecto y por lo tanto forman parte de los análisis de latencia punto a punto y determinismo en los sistemas de tiempo real, como parte de la información que ayuda a determinar problemas de coherencia y consistencia de datos. Esta información además ayuda a medir el desempeño de algoritmos implementados en distintas configuraciones del entorno del sistema.

Figura 7

Distintos tipos de latencia (Mosqueda-Arvizu et al., 2024)



Caracterización de la latencia punto a punto

Las semánticas mencionadas en la sección anterior son útiles en distintos contextos, dependiendo los requerimientos del sistema a desarrollar. Martínez et al. (2018b) mencionan que, por ejemplo, para sistemas de control, las semánticas de edad son útiles para determinar por cuanto tiempo la entrada de datos de un sensor es válida para manejar la salida a los actuadores (p.ej. sistema de inyección de combustible). Por otro lado, en las aplicaciones de control de chasis y carrocería, el tiempo de reacción con una nueva entrada de datos es importante (p.ej. tiempo de cierre de los seguros de la puerta, tiempo de activación de las bolsas de aire).

Para poder medir los efectos de los distintos modelos de comunicación en las latencias es necesario definir los momentos en el que se deben efectuar las operaciones de lectura y escritura. Para los modelos de comunicación explícita e implícita, dicho análisis involucra factores variables, como la duración de ejecución de las tareas, así como los momentos en los que se accede a las variables durante la misma ejecución. Por otro lado, el modelo de LET requiere de actualizar de manera lógica las entradas y salidas en momentos específicos de lectura y escritura respectivamente. Esto facilita la aplicación de las semánticas de latencia debido al determinismo del modelo, posibilitando que las latencias sean constantes, independientemente del tiempo de ejecución y asignación de la aplicación a un determinado núcleo. Sin embargo, estas propiedades solo se cumplen cuando los periodos de las tareas realizan una comunicación síncrona armónica (HSC).

Se dice que dos tareas que se comunican tienen periodos síncronos si el periodo de una de ellas es un múltiplo entero de la otra (Martínez et al., 2020). Es posible entonces decir que la relación que existe entre ambas tareas se puede definir calculando el mínimo común múltiplo *MCM* entre ellas, siendo el periodo máximo el resultado del cálculo ver (ver Ecuación 1).

$$MCM(T_l, T_e) = T_{max} \tag{1}$$

Dada esta relación, es posible entonces caracterizar los puntos de escritura y lectura entre ambas tareas utilizando las Ecuaciones 2 y 3 propuestas por Martínez et al. (2018a).

$$P_{W,R}^{n} = \left\lfloor \frac{nT_{max}}{T_W} \right\rfloor T_W \tag{2}$$

$$Q_{W,R}^{n} = \left\lceil \frac{nT_{max}}{T_R} \right\rceil T_R \tag{3}$$

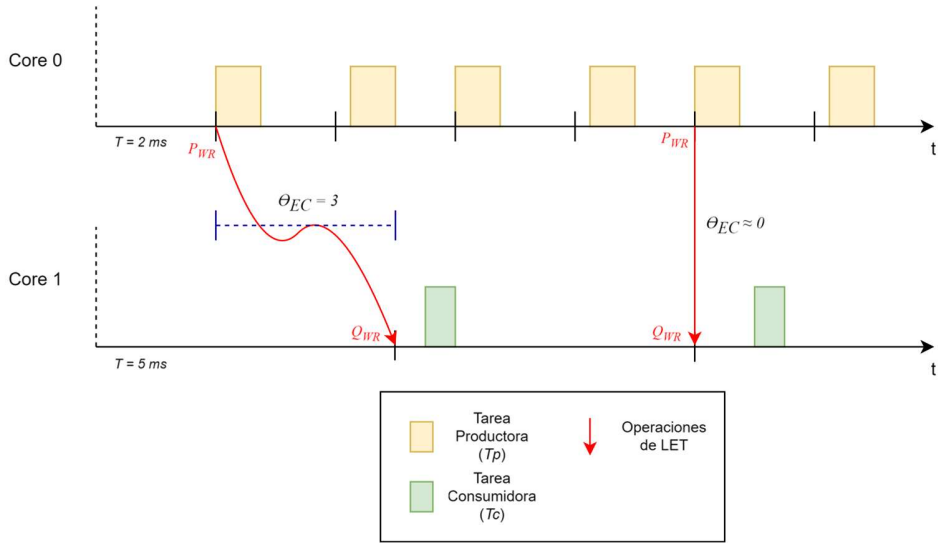
Dichas ecuaciones se pueden generalizar de la siguiente forma para el caso de *HSC*:

$$P_{W,R}^{n} = Q_{W,R}^{n} = nT_{max} \tag{4}$$

Cuando las tareas no cumplen las propiedades de la *HSC* se dice que realizan comunicación síncrona no armónica (*NHSC*). En este caso las propiedades determinísticas del modelo LET no se cumplen, dado que las latencias varían dependiendo la definición de los puntos de escritura y lectura, los cuales varían a través del tiempo. Un ejemplo de esto se ilustra en la Figura 8, dónde se muestra una cadena compuesta por dos tareas. La tarea 1 (T_1) tiene un periodo de 2 ms mientras que la tarea 2 (T_2) tiene un periodo de ejecución de 5 ms.

Figura 8

Comunicación Síncrona No Armónica (NHSC)



Como se puede apreciar, el mínimo común múltiplo de T_1 y T_2 no es igual a T_2, dado que $MCM(2,5) \neq 5$, como se especifica en la Ecuación 1. Sin embargo, agregando tiempos de desfase en las tareas, es posible ajustar los tiempos de latencia para que sean constantes. En el trabajo desarrollado por Martínez et al. (2018a), se propusieron las Ecuaciones 5 y 6 para considerar los desfases.

$$P_{W,R}^{n} = O_w + \left\lceil \frac{nT_{max} + O_{max} - O_w}{T_W} \right\rceil T_W \tag{5}$$

$$Q_{W,R}^{n} = O_R + \left\lceil \frac{nT_{max} + O_{max} - O_R}{T_R} \right\rceil T_R \tag{6}$$

Se consideran los tiempos de desfase de cada tarea con la finalidad de poder calcular los tiempos de las operaciones de escritura ($P_{W,R}^{n}$) y lectura ($Q_{W,R}^{n}$).

Una vez obtenidos los tiempos de escritura y lectura se puede calcular la latencia de edad considerando la ecuación 7 propuesta por Martínez et al. (2020).

$$\alpha^n = T_1 + \theta_{EC}^{n} + \dot{Q}_{\eta-1,\eta}^{n+1} - \dot{Q}_{\eta-1,\eta}^{n} \tag{7}$$

Dónde θ_{EC}^{n} se define por

$$\theta_{EC}^{n} = \dot{Q}_{\eta-1,\eta}^{n} - \dot{P}_{1,2}^{n} \tag{8}$$

Para calcular la latencia de reacción, por su parte puede ser calculada agregando el término del periodo T_η

de la última tarea de la cadena:

$$\delta^n = \alpha^n + T_\eta \tag{9}$$

Por último, la latencia máxima de edad, de acuerdo con Martínez et al. (2020), se puede obtener de la mayor de las latencias de edad de todos los caminos básicos dados en un hiper-periodo de la cadena de eventos.

$$\alpha(EC) = max_{\forall n \in H_{EC}} \alpha^n \tag{10}$$

Hipótesis

La implementación del modelo de comunicación Tiempo de ejecución lógica (LET) en sistemas heterogéneos en conjunto con el método de Acceso Múltiple por División de Tiempo (TDMA) reduce la latencia producida cuando hay un evento de traslape de operaciones.

Objetivo General

Aplicar un modelo de comunicación internúcleo en conjunto con un mecanismo de sincronización, mediante el desarrollo de una librería de software embebido, que implemente el modelo LET el cual pueda ser integrada en sistemas heterogéneos. con la finalidad de reducir la variabilidad de la latencia de las operaciones de lectura y escritura entre núcleos.

Objetivos específicos

- Identificar un mecanismo de sincronización para reducir la latencia producida por los eventos de traslape del modelo LET.

- Diseñar el modelo LET y el método de TDMA en software para observar su comportamiento.
- Implementar el software en un microcontrolador multinúcleo para realizar mediciones de los tiempos de operaciones de lectura y escritura.
- Analizar la latencia de comunicaciones internúcleo mediante el cálculo de la latencia de edad, la latencia reactiva y la longitud del camino básico planteado para la experimentación.

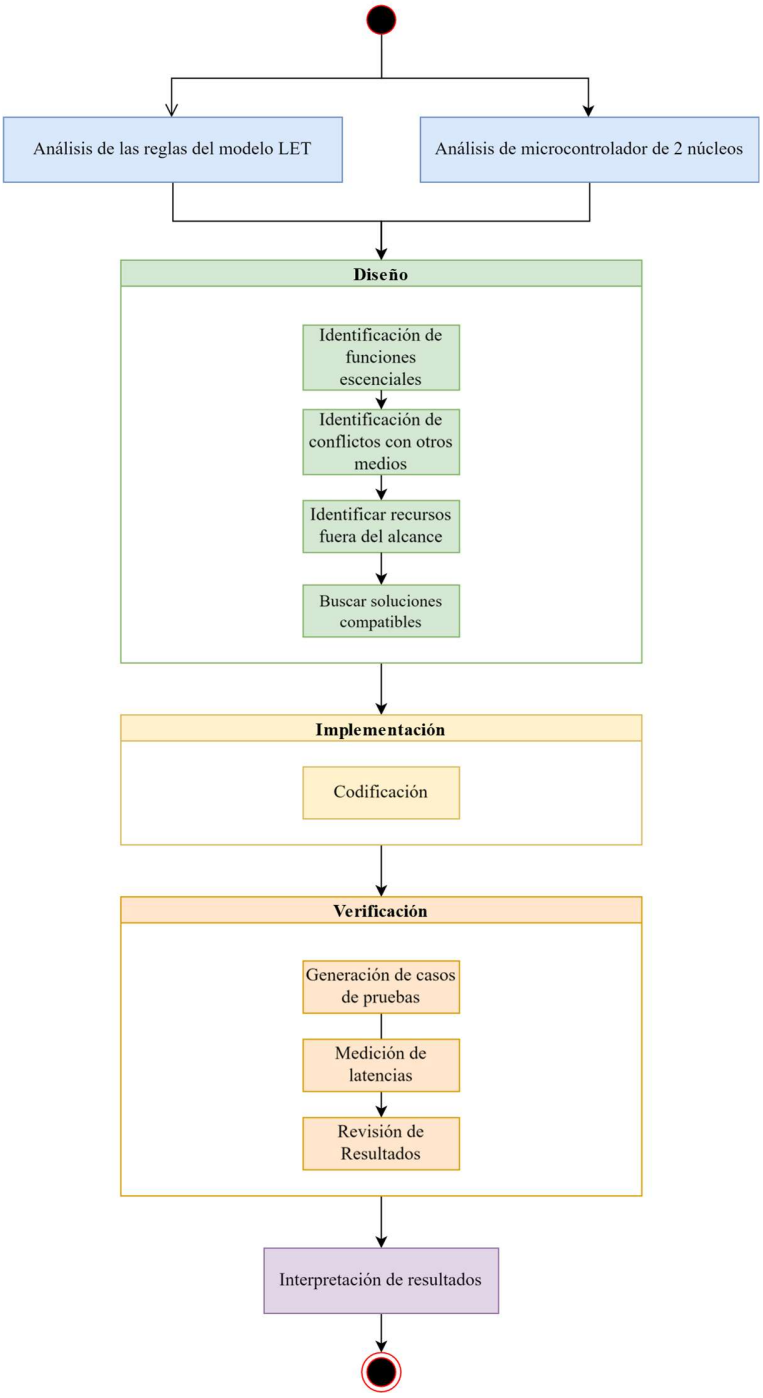
Metodología

La metodología utilizada, descrita por la Figura 9, está basada en el modelo *Waterfall* Pressman (2000), donde se definen las siguientes etapas:

a) Planteamiento de los requerimientos de acuerdo con el problema de estudio
b) Se diseña la solución
c) Se realiza la implementación
d) Se hace verificación del software implementado
e) Se interpretan los resultados

Figura 9

Diagrama de la Metodología implementada

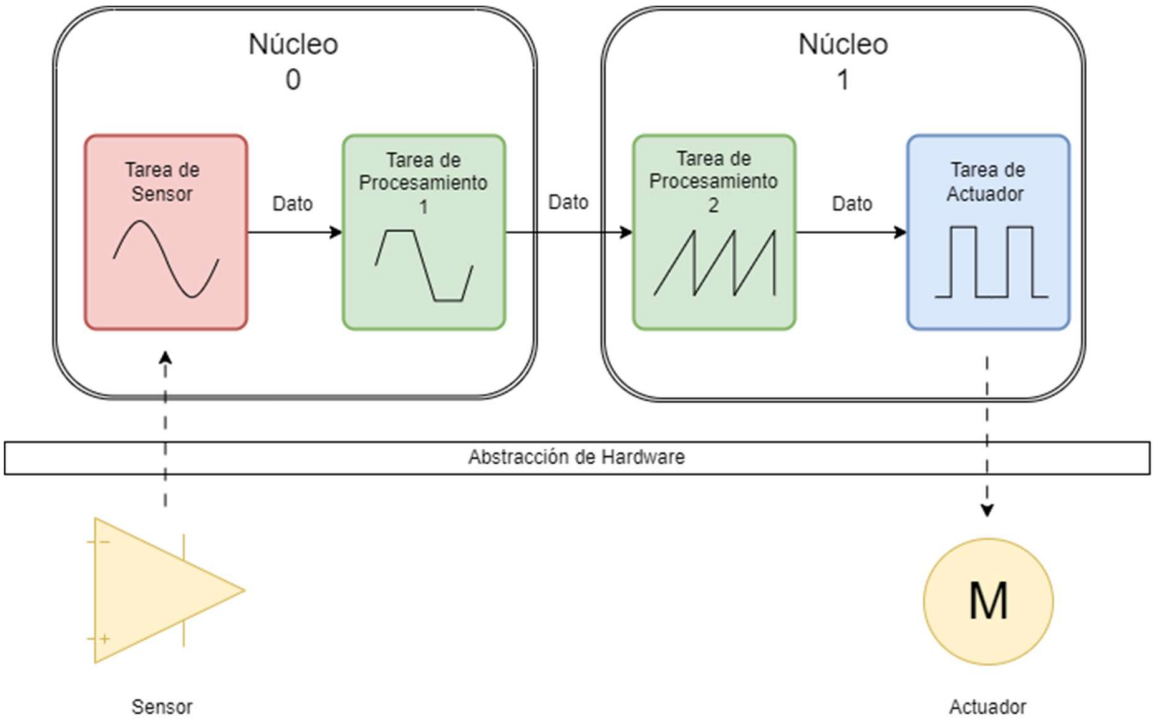


Para poder utilizar el modelo LET para tareas alojadas en distintos núcleos el primer paso es definir las cadenas de eventos que impliquen flujo de datos de un núcleo a otro. En dichas cadenas se deben identificar sus tareas productoras y consumidoras, así como sus periodos de ejecución y tiempo de desfase.

Del conjunto de tareas seleccionador se deben identificar las tareas productoras T_P y consumidoras T_C que impliquen una comunicación directa entre núcleos. En la Figura 10 se puede observar un par de tareas que comunican información del procesamiento de una señal de entrada provista por un sensor, la cual servirá para procesarse posteriormente para generar una salida a un actuador.

Figura 10

Cadena de eventos con intercambio internúcleo

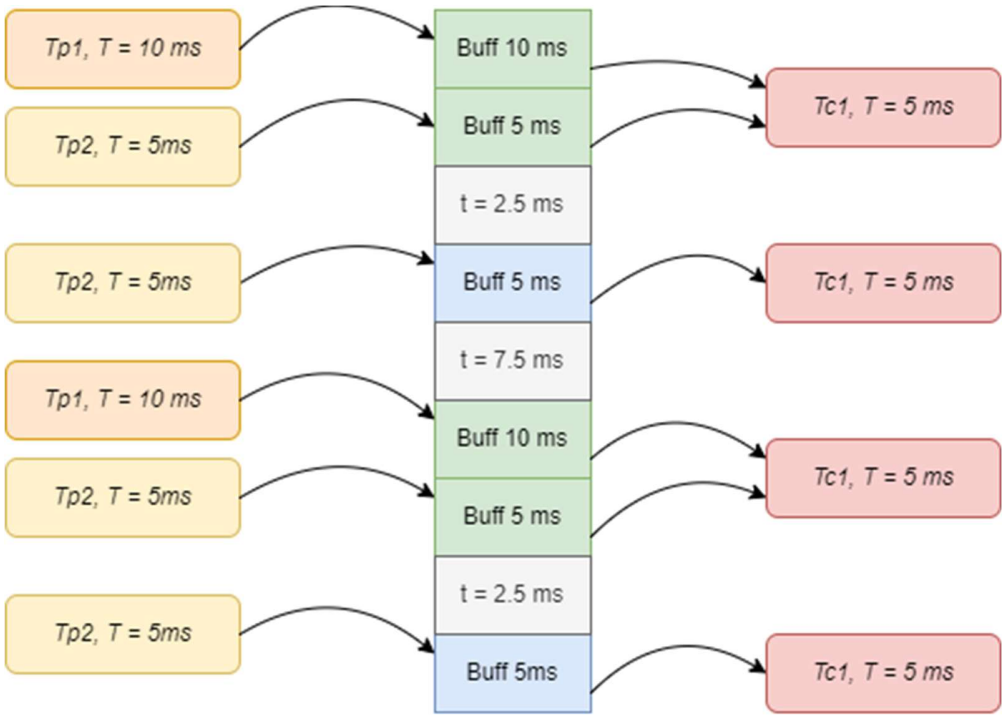


Con las propiedades de dichas tareas se puede definir la duración de las ranuras de tiempo bajo el esquema de TDMA, con las cuales se controla el acceso a la memoria

compartida que se debe asignar de manera estática a cada ranura de tiempo. Si los periodos de las tareas T_P y T_C no son iguales, se debe seleccionar el periodo más corto con la finalidad de mantener la resolución de muestreo en caso de que existan más tareas consumidoras. En la Figura 11 se puede apreciar un ejemplo que consta de dos tareas productoras, T_{P1} y T_{P2}, con periodos de 10 ms y 5 ms respectivamente. Los datos producidos por dichas tareas se consumen por una tarea consumidora *Tc*, con un periodo de 5 ms. Dado que el periodo más corto es el de 5 ms, se establece que la duración de cada ranura de memoria será de 5 ms, aunque la memoria utilizada por T_{P2} solo se accede cada 10 ms.

Figura 11

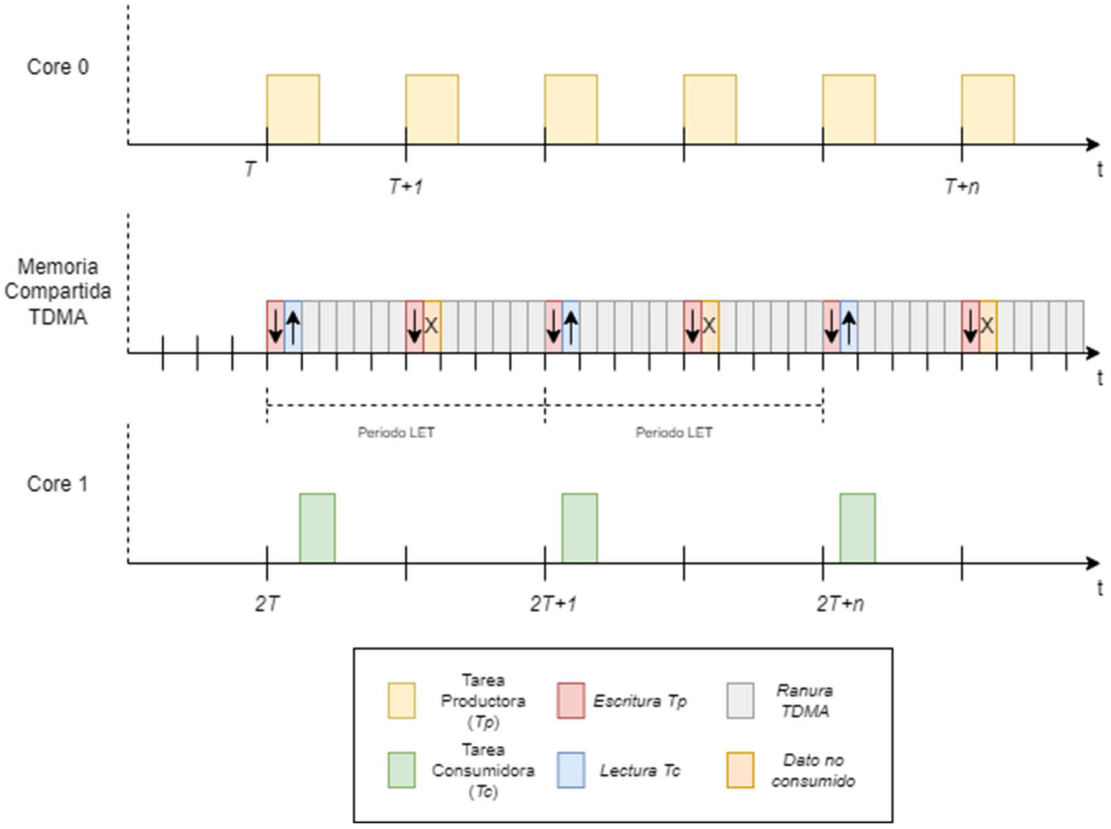
Ejemplo de acceso a memoria TDMA



Para definir el periodo LET de cada par de tareas T_PT_C se identifica el periodo de ejecución más lento entre estas tareas. Idealmente los periodos deben ser idénticos para evitar pérdidas de muestreo.

Figura 12

Periodo LET para una tarea $T_{Tc} = 2T_{Tp}$



Finalmente se debe seleccionar una de las tareas del par $T_P T_C$ como la iniciadora del evento para la activación de su tarea correspondiente en el núcleo adyacente. Esto solo se debe realizar una vez al momento de arrancar del planificador de tareas. En la

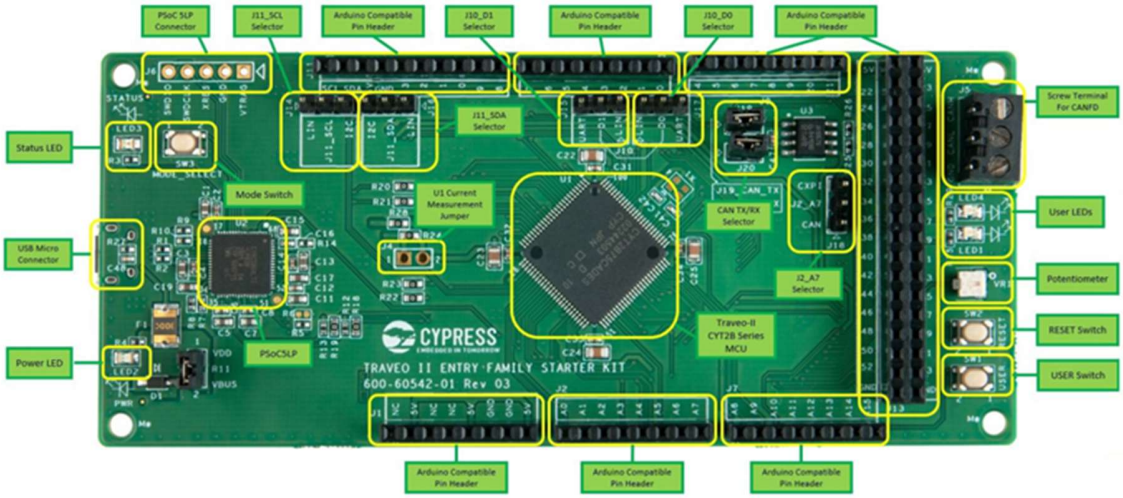
Figura 12 se puede apreciar que al inicio la tarea productora activa al inicio la tarea consumidora después de ejecutar las operaciones de lectura y escritura.

Arquitectura del Hardware

La Figura 13 provee una descripción general de la tarjeta de desarrollo Traveo II Entry Family Starter Kit de Cypress. Las tarjetas de desarrollo contienen módulos de hardware e interfaces con la finalidad de poder acceder a los periféricos del microcontrolador y poder hacer aplicaciones de prueba. Esta tarjeta en particular contiene *hardware* específico para la industria automotriz, como un transceptor de CAN, una cabecera para conectar accesorios diseñados para Arduino, entradas y salidas de propósito general para acceder al bus de LIN, I2C y SPI, un potenciómetro, 2 LEDs de prueba, además de la posibilidad de agregar un cristal externo. También incluye el PSoC 5LP, un chip que provee la capacidad de reprogramar el microcontrolador.

Figura 13

Tarjeta de desarrollo Cypres Traveo II Starter Kit (Cypress Semiconductor, 2021)



Evaluación de tiempos

En el contexto de los sistemas embebidos de tiempo real, la latencia de intercambio de datos se tiene que evaluar con la finalidad de validar los tiempos de los modelos y técnicas con respecto a sus requisitos, sobre todo para la seguridad funcional. Es por eso por lo que en la siguiente sección se describe el tipo de latencia a medir y el impacto de la solución propuesta en ella.

Latencia

Cuando los datos se almacenan en un medio para ponerlos a disposición de procesos consumidores, existe un tiempo desde que el dato comienza el proceso de almacenamiento y cuando este está disponible, el cual está denominado como latencia de buffer. En los sistemas digitales el medio más común es la memoria, que funciona como un búfer para el intercambio de datos. De acuerdo con Heath (2003), para un sistema de tiempo real, la latencia de búfer define el momento más pronto en el cual la información puede ser procesada, y por lo tanto cualquier respuesta a dicha información será retrasada por dicha latencia.

Las cadenas de eventos de sistemas de tiempo real definen cadenas de operaciones de lectura/escritura, donde de acuerdo con Martínez et al. (2020), una tarea escribe en una variable, que luego es leída por una segunda tarea, que a su vez procesa el dato de dicha variable y escribe el resultado en otra variable, la cual puede ser leída por una tercera tarea. En estas cadenas se define como Latencia de Extremo a Extremo (*End-to-End Latency*), al tiempo que transcurre desde el primer evento de entrada hasta el final de la cadena. Si la latencia es muy grande o varía mucho, esta puede afectar al rendimiento de un algoritmo. Es por eso por lo que es importante mantener un valor de latencia bajo y poco variable, para poder mantener el flujo de datos predecible y el rendimiento de la aplicación óptimo.

El modelo LET sacrifica la latencia en las cadenas de eventos a favor de predictibilidad. Esta latencia se vuelve mayor y variable al tener comunicación entre núcleos,

por lo que requiere de un mecanismo adicional para mantener la variabilidad mediante la sincronización de la ejecución de las tareas y la disponibilidad de espacios de memoria aislados temporalmente. En el caso ideal de LET con memoria TDMA, la latencia máxima está dada por la Ecuación 10, en dónde tiempos están fijos por el planificador de tareas de sus respectivos núcleos, pero pueden ser ligeramente alterados por retrasos inducidos por interrupciones o tareas de mayor prioridad interrumpiendo la tarea actual. Para este estudio, el cálculo más relevante es la latencia máxima de edad, con la finalidad de comparar la rapidez de respuesta del modelo y comprobar la mejora en tiempo de respuesta.

Implementación de la solución

Para realizar la implementación y validación de la solución propuesta, se seleccionó el *hardware* a utilizar como ambiente de desarrollo y las herramientas de hardware para programar y extraer los datos para validar. También se seleccionó el software a utilizar, tanto en la implementación, como para la realización de las mediciones.

Ambiente de desarrollo

Para validar la aplicación del modelo LET con TDMA, se utilizó el siguiente ambiente para la experimentación:

1. Tarjeta de desarrollo Traveo II Entry Family Starter Kit de Cypress con el microcontrolador Traveo II CYT2B75 enfocado a aplicaciones de bajo consumo y cristal de 16 MHz.
2. Un depurador I-jet con capacidad de trazado y acceso a memoria de depuración por medio del protocolo ITM.
3. IAR EW como ambiente de software para reprogramar el software y para la extracción de los datos.

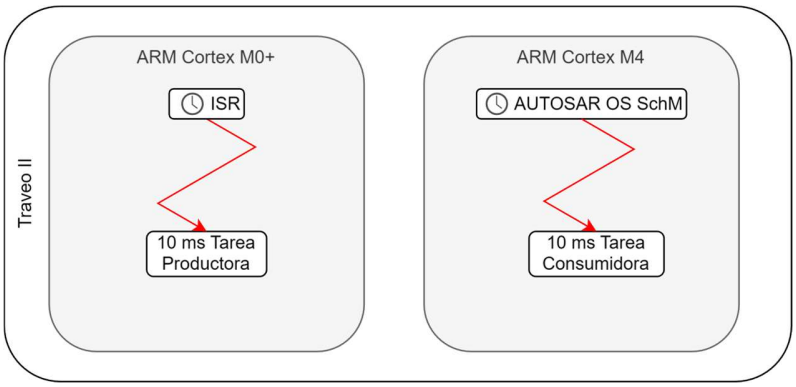
Configuración del ambiente

En el núcleo primario (M0+) se implementó un *scheduler* de ventana manejado por una interrupción a 10 ms. En este núcleo se implementó una tarea productora, la cual calcula una función rampa, la cual es configurable para escribir en buffers de tamaño de datos de 8 bits, 16 bits o 32 bits.

En el núcleo secundario (M4) se integró un sistema operativo de tiempo real (RTOS) de AUTOSAR OS. Se configuró una tarea consumidora a 10 ms, la cual se encarga de leer los datos de recepción y posteriormente escribirlos en el buffer de instrumentación para poderlos recopilar por medio del depurador.

Figura 14

Configuración de Tareas



Arquitectura del software

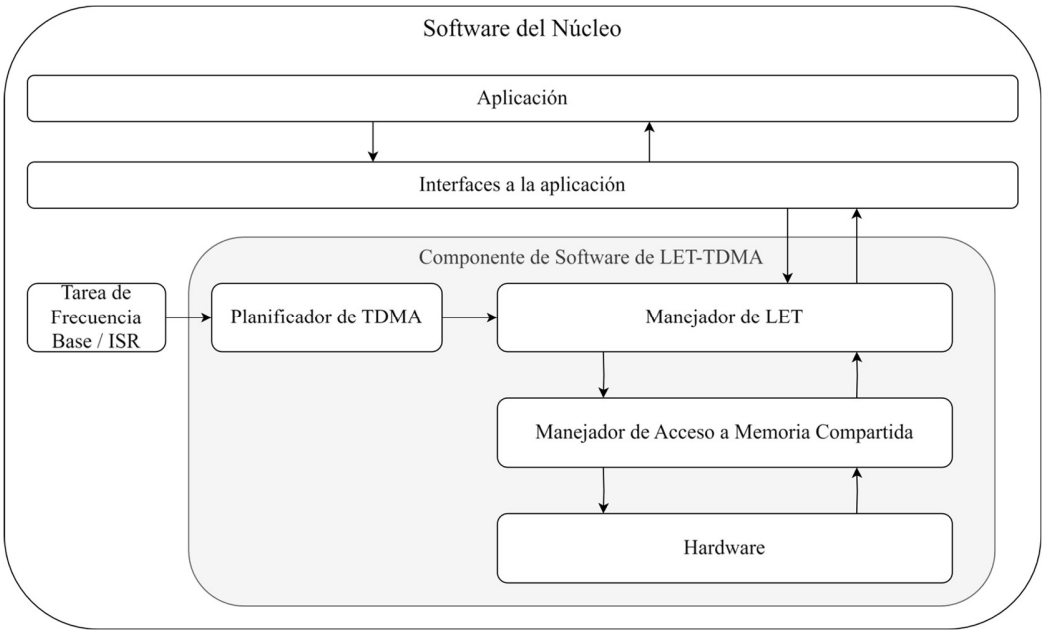
La Figura 15 muestra la solución que se implementó en una estructura de 3 capas de software, la cual consta de las siguientes capas:

1. **Capa de abstracción memoria:** Representa la parte del software que define y gestiona las secciones de memoria compartida.

2. **Manejador de LET:** Se encarga de definir las interfaces para realizar las operaciones LET. Provee operaciones de enfilado, inicio de escrituras y lectura.
3. **Planificador TDMA:** Se encarga de gestionar los tiempos de acceso a la memoria compartida entre núcleos. Provee además una extensión para implementar sincronización entre tareas.

Figura 15

Integración de LET-TDMA en Software de un núcleo del microcontrolador



Capa de abstracción de Memoria

Se implementó un gestor de memoria compartida, el cual desde la configuración reserva una sección de memoria fija en el espacio de la RAM, el cual es utilizado como memoria global para asignar los espacios correspondientes a la memoria de intercambio de datos.

Manejador de LET

Para el manejo de los tiempos de LET, se implementó un componente manejador el cual recibe la configuración de cada tarea, con su respectivo período y tamaño de buffer. El manejador tiene 3 operaciones principales:

1. Enfilar: Función encargada de enfilar escrituras de memoria local a memoria compartida. Dicha función recibe la dirección de memoria local que servirá posteriormente para el proceso de escritura en el tiempo definido por el periodo LET de la tarea correspondiente.
2. Disparo: Función encargada de revisar si hay datos pendientes por escribir. Si hay operaciones pendientes, procede al copiado de los datos a las secciones de memoria compartida correspondientes.
3. Lectura: Función encargada de acceder a la memoria compartida. Recibe el identificador del periodo LET configurado para identificar la sección de memoria a leer.

Planificador de memoria TDMA

Para el manejo de la memoria TDMA, se implementó un gestor que provee las APIs para iniciar peticiones de lectura y escritura entre núcleos. Se integran las APIs provistas por la capa de LET con la finalidad de gestionar sus tiempos de activación y acceso a la ranura de tiempo establecida.

Resultados

En esta sección se presentan los resultados obtenidos de la implementación del modelo LET y el modelo LET-TDMA. La implementación del modelo LET se realizó a manera referencia para comparar los resultados contra el modelo LET-TDMA en términos de variabilidad. Ambas se realizaron en el ambiente descrito en la sección de *Arquitectura del Hardware*. Adicionalmente se realizó la implementación del modelo LET-TDMA para poder contrastar la implementación del modelo en un ambiente simulado y un ambiente en un hardware real con la finalidad de validar la precisión de LET-TDMA.

Para las implementaciones en *hardware*, se definieron los siguientes parámetros de configuración:

Tabla 1

Configuración de las tareas

Núcleo	***Frecuencia de Reloj***	***Frecuencia Pre-escalada***	***Manejador de Tareas***	***Periodo Base***	***Tareas***	***Tipo de Memoria***	***Núm. de Buffers***	***Tamaño en bits***
Cortex M0+	80 MHz	1 MHz	Interrupción	10 ms	1	SRAM	3	8, 16, 32
Cortex M4	160 MHz	1 KHz	SO AUTOSAR	1 ms	2		3	8, 16, 32

Las tareas del núcleo M4 se implementaron como tareas básicas no interrumpibles de acuerdo con el estándar de AUTOSAR 4.0.3 SC1 con prioridad fija. La tarea implementada en el núcleo M0+ se implementó a través de una función *callback* como parte de una rutina de servicio de interrupción manejada por un *Timer*. Ambas tareas implementaron una cadena de eventos con la posibilidad de usar los 3 tamaños de buffer disponibles. La tarea productora de la cadena ejecuta la implementación de una función rampa con pendiente $m = 1$ y ordenada de origen $b = 0$.

Se realizaron mediciones de los tiempos en los que se realizaron las operaciones de escritura por parte de la tarea productora implementada en el núcleo M0+, así como de las operaciones de lectura de la tarea consumidora configurada en el núcleo M4. Para poder extraer los datos de los tiempos de las operaciones, se utilizó le *hardware* de instrumentación *ITM* disponible en el núcleo M4, en conjunto con el depurador *I-jet* de *IAR Systems* y el ambiente de desarrollo integrado *IAR Embedded Workbench*.

Para la parte de la simulación, se utilizó una configuración que consta de una tarea proveedora y una consumidora, ambas a una frecuencia de 10 ms. Ambas tareas son simuladas en *Simulink* con la ayuda de la extensión *SoC Blockset*, el cual es capaz de modelar distintos eventos de *hardware*, así como el procesamiento de núcleos paralelos.

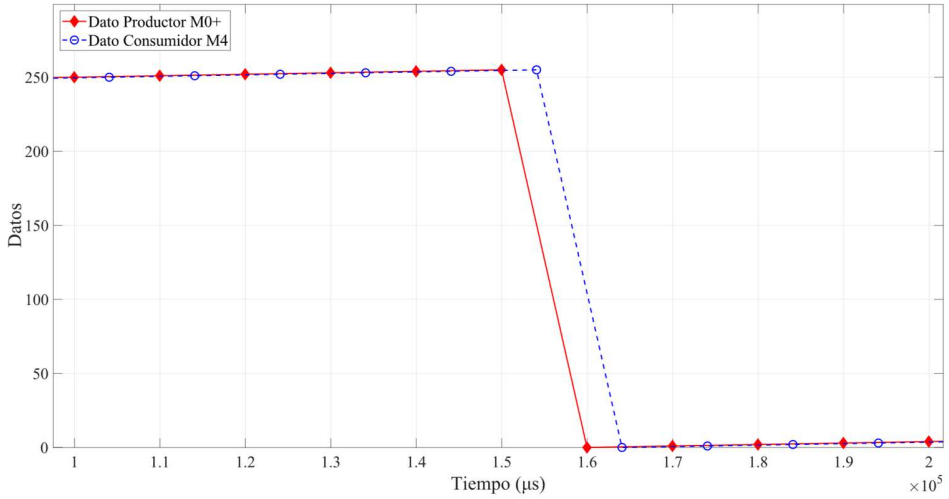
Implementación de LET en tarjeta de desarrollo

Como referencia inicial se implementó el modelo LET, utilizando la cadena de eventos propuesta para la experimentación. En la

Figura 16 se puede observar los tiempos de las operaciones de lectura y escritura utilizando el valor de la función rampa. Dicha medición corresponde a la configuración de una tarea productora, una tarea consumidora y un búfer compartido con tamaño de datos de 8 bits.

Figura 16

Tiempos de operaciones con LET



Se puede observar que el tiempo entre operaciones de lectura y escritura no es variable, sin embargo, este comportamiento solo es válido durante ese ciclo de ejecución. En ciclos posteriores se puede observar que la latencia es variable entre ciclos de ejecución, debido a que los tiempos de inicialización pueden variar, ya sea por factores internos, como la carga de datos desde memoria no volátil (NVM) así como por factores externos, como la inicialización de adquisición de datos del depurador.

Implementación de LET-TDMA en Simulink

Se realizó la implementación del modelo LET-TDMA en un ambiente simulado con la finalidad de proveer una referencia comparativa para la implementación en la tarjeta de desarrollo. Para esto se utilizó el ambiente de modelado y simulación *SoC Blockset* de *Simulink*, que cuenta con elementos para simular sistemas de tiempo real, incluido procesamiento multinúcleo.

El caso base se modeló configurando una tarea productora con un periodo de ejecución de 10 ms en el núcleo productor, activada por un evento externo simulando el manejador de tareas. La tarea productora implementa una función rampa con una pendiente de una unidad y produce una salida cada ciclo de ejecución. Adicionalmente, se configuró una tarea consumidora con la misma configuración de la tarea productora. La memoria compartida se modeló mediante un Canal de Datos Interproceso (*IPC*), el cual adicionalmente modela la sincronización entre tareas con un tiempo de desfasamiento de 1 ms mediante un generador de eventos con la finalidad de representar el comportamiento del TDMA para las dos tareas modeladas únicamente. La Figura 17 ilustra la configuración descrita.

Figura 17

Modelado de LET-TDMA con 2 núcleos

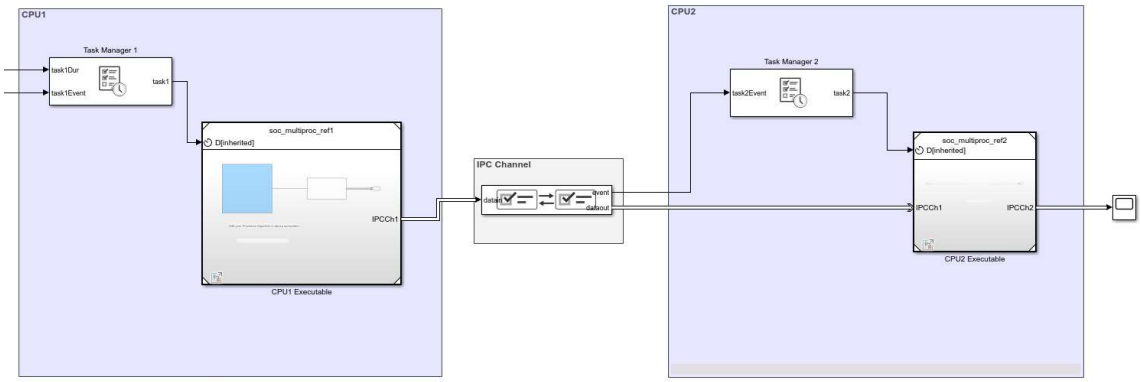
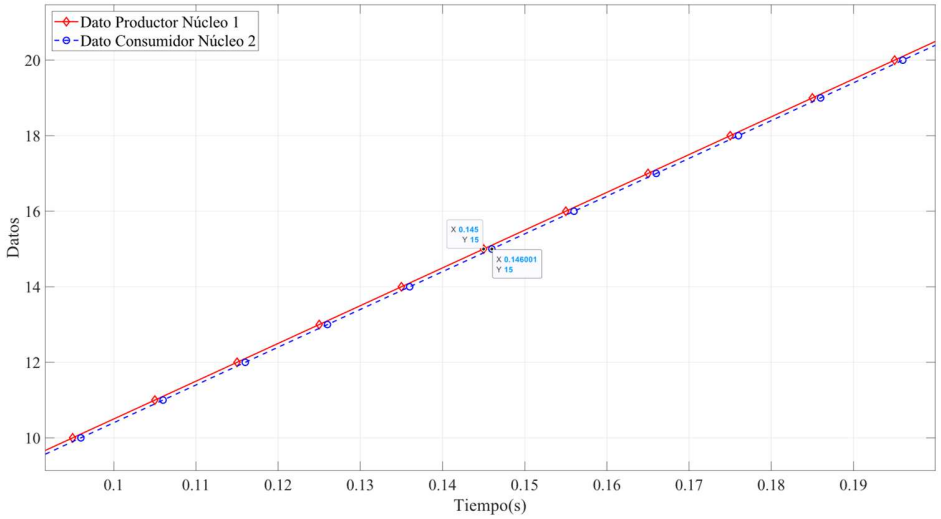


Figura 18

Tiempos de operaciones en simulación de LET-TDMA



En la Figura 18 se puede observar el comportamiento de la simulación donde el tiempo entre operaciones de lectura y escrituras es de 1 ms. Esto muestra el comportamiento ideal esperado de LET-TDMA, el cual sirvió de referencia para calcular el Error Cuadrático Medio (ECM) con los resultados arrojados por la implementación descrita en la sección *Implementación de LET-TDMA en tarjeta de desarrollo*.

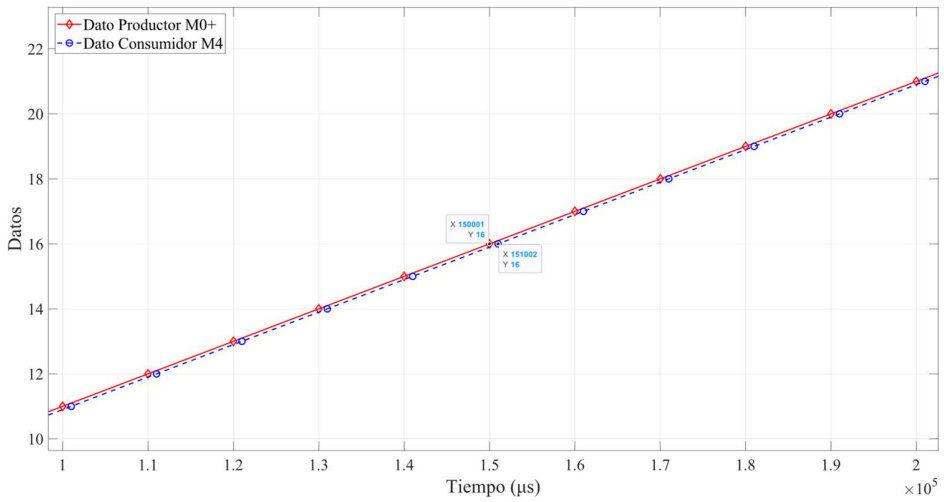
Implementación de LET-TDMA en tarjeta de desarrollo

Se realizaron mediciones de los tiempos de operación de escritura y lectura utilizando la solución de LET-TDMA utilizando buffers de 8, 16 y 32 bits.

En la Figura 19, se puede apreciar la latencia prácticamente constante de 1 ms entre la tarea productora y la tarea consumidora utilizando un buffer de 8 bits. El tiempo entre operaciones fue fijado a 1 ms en la configuración de la capa de software que gestiona las ranuras de tiempo de TDMA.

Figura 19

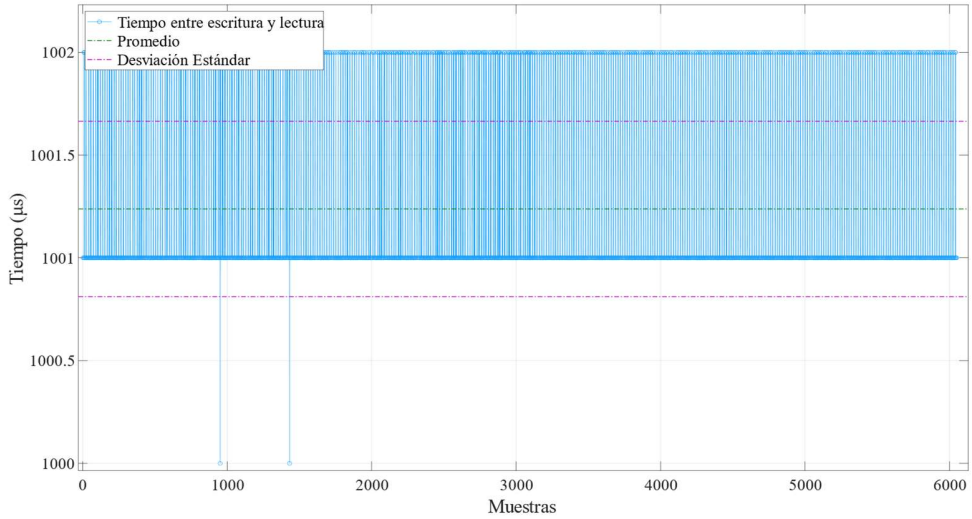
Tiempos de operaciones con LET-TDMA con búfer de 8 bits



Para poder observar con mayor detalle la baja variabilidad de la latencia, se realizó el cálculo de las diferencias de los tiempos de transmisión entre las tareas. En la Figura 20 se muestra la variabilidad de las muestras de transmisiones realizadas durante aproximadamente 1 minuto. La latencia mínima medida fue de 1 ms mientras la máxima fue de 1.002 ms utilizando un buffer de tamaño de datos de 8 bits.

Figura 20

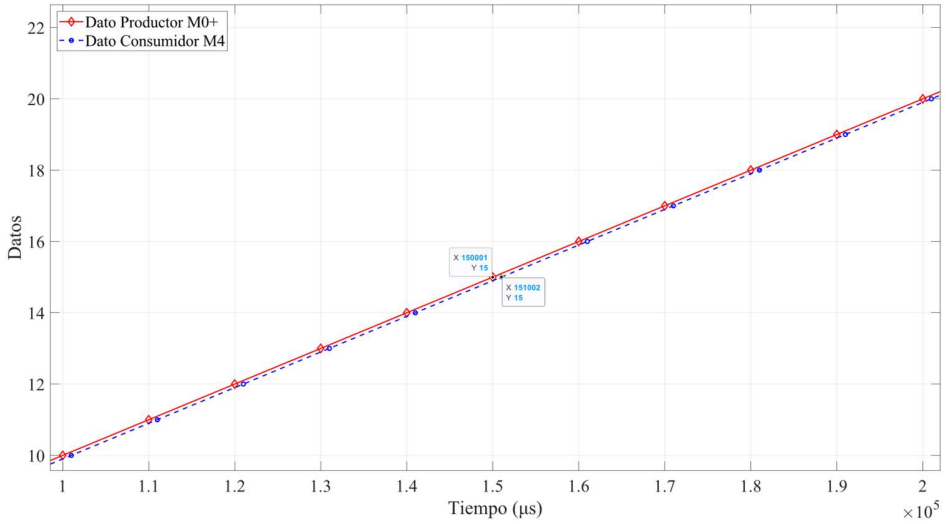
Latencia entre escrituras y lecturas en búfer de 8 bits con LET-TDMA



En la Figura 21, se puede apreciar la latencia de 1 ms entre la tarea productora y la tarea consumidora utilizando un buffer de 16 bits. Como se puede observar, el tamaño de datos del buffer no influye en los tiempos de transmisión, manteniendo la latencia con muy baja variabilidad. Tampoco existe una variación entre ejecuciones con distintas configuraciones, como sucede con implementaciones de LET en distintos núcleos.

Figura 21

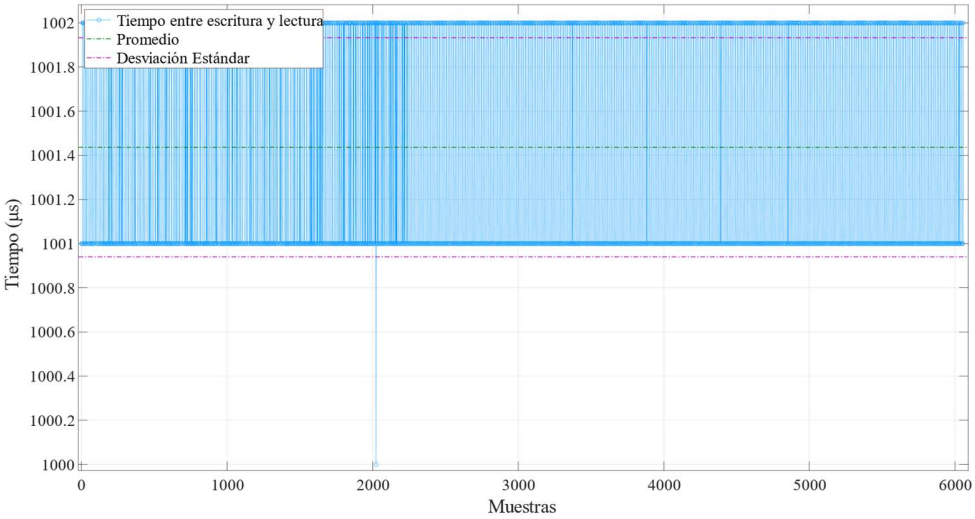
Tiempos de operaciones con LET-TDMA con búfer de 16 bits



En la Figura 22 se puede observar que la variabilidad se mantiene baja en las muestras realizadas durante aproximadamente 1 minuto utilizando un buffer de 16 bits. Se observa un máximo de 1.002 ms y un mínimo de 1 ms en las diferencias de tiempo de las operaciones de escritura y lectura.

Figura 22

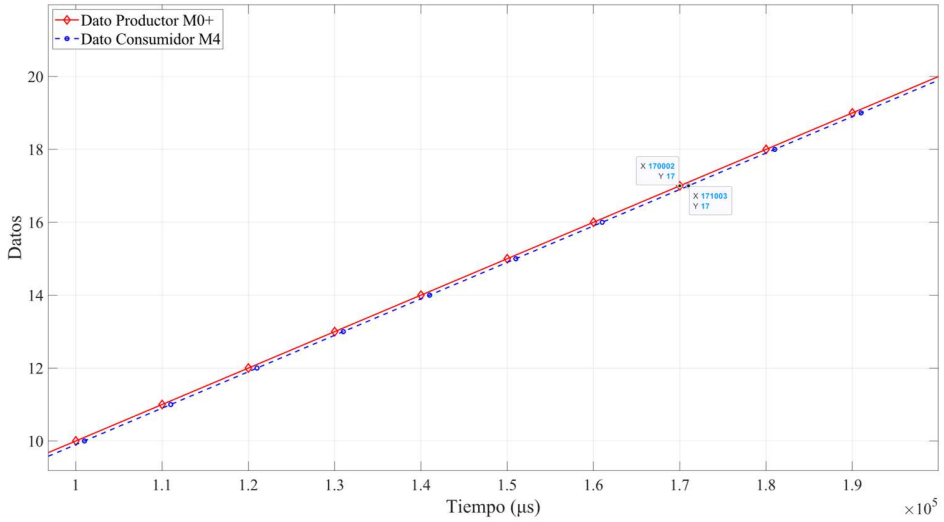
Latencia entre escrituras y lecturas en búfer de 16 bits con LET-TDMA



La Figura 23 muestra la latencia constante de 1 ms entre la tarea productora y la tarea consumidora utilizando un buffer de 32 bit, confirmando los resultados anteriores, utilizando el tipo de dato del tamaño de palabra del procesador.

Figura 23

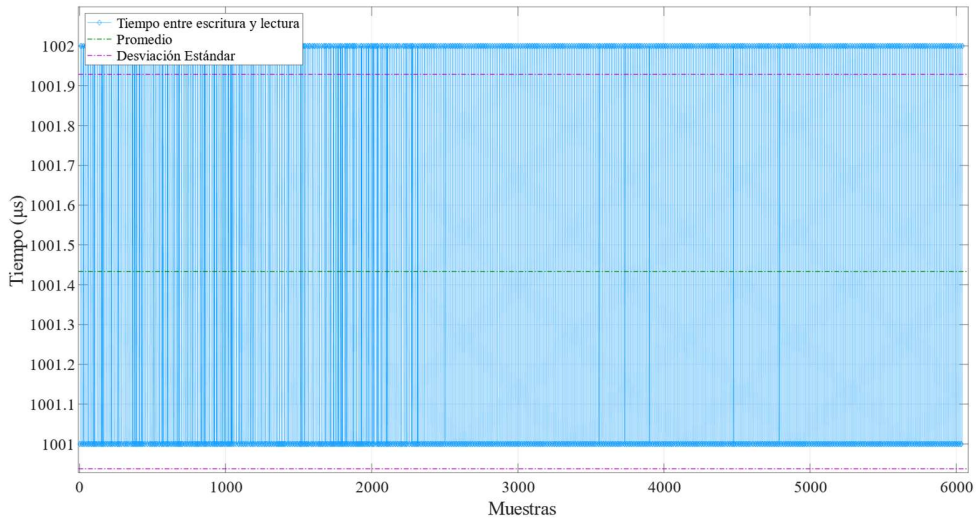
Tiempos de operaciones con LET-TDMA con búfer de 32 bits



En la Figura 24 se muestran las diferencias de tiempo entre las operaciones de lectura y escritura utilizando un buffer de 32 bits. Es posible apreciar que las latencias se mantienen entre 1.001 ms y 1.002 milisegundos, manteniendo un comportamiento muy similar a los casos observados en la Figura 20 y en la Figura 22.

Figura 24

Latencia entre escrituras y lecturas en búfer de 32 bits con LET-TDMA



Caracterización de la latencia punto a punto

La Tabla 2 muestra capturas de los tiempos de lectura y escritura obtenidos de la ejecución de la aplicación utilizando el modelo LET-TDMA con *buffers* de tamaño de 8, 16 y 32 bits. Estas muestras representan los tiempos de publicación ($P^n_{W,R}$) y los tiempos de lectura ($Q^n_{W,R}$) descritos por las Ecuaciones 5 y 6. Estos tiempos son relevantes para caracterizar los puntos de intercambio de datos, que son necesarios para definir la secuencia de ejecución de las operaciones, así como para mantener la sincronización entre cadenas de eventos y tener un flujo de datos determinista.

Tabla 2

Tiempos (μs) de operaciones utilizando búferes de 8, 16 y 32 bits

T_{W8}	T_{R8}	T_{W16}	T_{R16}	T_{W32}	T_{R32}
19,075,741	19,076,742	27,903,041	27,904,042	22,567,907	22,568,908
19,085,741	19,086,742	27,913,041	27,914,042	22,577,907	22,578,908
19,095,741	19,096,743	27,923,041	27,924,042	22,587,907	22,588,908
19,105,741	19,106,743	27,933,041	27,934,043	22,597,907	22,598,908
19,115,741	19,116,743	27,943,041	27,944,043	22,607,907	22,608,909
19,125,742	19,126,743	27,953,041	27,954,043	22,617,907	22,618,909
19,135,742	19,136,743	27,963,041	27,964,043	22,627,907	22,628,909
19,145,742	19,146,743	27,973,041	27,974,043	22,637,907	22,638,909
19,155,742	19,156,743	27,983,041	27,984,043	22,647,907	22,648,909
19,165,742	19,166,743	27,993,041	27,994,043	22,657,907	22,658,909
19,175,742	19,176,743	28,003,042	28,004,043	22,667,908	22,668,909
19,185,742	19,186,743	28,013,042	28,014,043	22,677,908	22,678,909
19,195,742	19,196,743	28,023,042	28,024,043	22,687,908	22,688,909
19,205,742	19,206,743	28,033,042	28,034,043	22,697,908	22,698,909
19,215,742	19,216,743	28,043,042	28,044,043	22,707,908	22,708,909
19,225,742	19,226,743	28,053,042	28,054,043	22,717,908	22,718,909
19,235,742	19,236,743	28,063,042	28,064,043	22,727,908	22,728,909
19,245,742	19,246,743	28,073,042	28,074,043	22,737,908	22,738,909
19,255,742	19,256,744	28,083,042	28,084,044	22,747,908	22,748,909
19,265,742	19,266,744	28,093,042	28,094,044	22,757,908	22,758,910
19,275,742	19,276,744	28,103,042	28,104,044	22,767,908	22,768,910
19,285,742	19,286,744	28,113,042	28,114,044	22,777,908	22,778,910
19,295,743	19,296,744	28,123,042	28,124,044	22,787,908	22,788,910
19,305,743	19,306,744	28,133,042	28,134,044	22,797,908	22,798,910
19,315,743	19,316,744	28,143,043	28,144,044	22,807,908	22,808,910
19,325,743	19,326,744	28,153,042	28,154,044	22,817,908	22,818,910
19,335,743	19,336,744	28,163,043	28,164,044	22,827,908	22,828,910
19,345,743	19,346,744	28,173,043	28,174,044	22,837,909	22,838,910
19,355,743	19,356,744	28,183,043	28,184,044	22,847,909	22,848,910
19,365,743	19,366,744	28,193,043	28,194,044	22,857,909	22,858,910
19,375,743	19,376,744	28,203,043	28,204,044	22,867,909	22,868,910
19,385,743	19,386,745	28,213,043	28,214,044	22,877,909	22,878,910
19,395,743	19,396,745	28,223,043	28,224,045	22,887,909	22,888,911
19,405,743	19,406,745	28,233,043	28,234,044	22,897,909	22,898,911
19,415,744	19,416,745	28,243,043	28,244,044	22,907,909	22,908,911
19,425,744	19,426,745	28,253,043	28,254,045	22,917,909	22,918,911

19,435,743	19,436,745	28,263,043	28,264,045	22,927,909	22,928,911
19,445,743	19,446,745	28,273,043	28,274,045	22,937,909	22,938,911
19,455,744	19,456,745	28,283,043	28,284,045	22,947,909	22,948,911
19,465,744	19,466,745	28,293,044	28,294,045	22,957,909	22,958,911
19,475,744	19,476,745	28,303,044	28,304,045	22,967,910	22,968,911
19,485,744	19,486,745	28,313,044	28,314,045	22,977,910	22,978,911
19,495,744	19,496,745	28,323,044	28,324,045	22,987,910	22,988,911
19,505,744	19,506,745	28,333,044	28,334,045	22,997,910	22,998,911
19,515,744	19,516,745	28,343,044	28,344,045	23,007,910	23,008,911
19,525,744	19,526,745	28,353,044	28,354,045	23,017,910	23,018,911
19,535,744	19,536,745	28,363,044	28,364,045	23,027,910	23,028,911
19,545,744	19,546,745	28,373,044	28,374,045	23,037,910	23,038,911
19,555,744	19,556,745	28,383,044	28,384,045	23,047,910	23,048,911
19,565,744	19,566,745	28,393,044	28,394,045	23,057,910	23,058,911
19,575,744	19,576,746	28,403,044	28,404,045	23,067,910	23,068,911
19,585,744	19,586,746	28,413,044	28,414,046	23,077,910	23,078,911
19,595,744	19,596,746	28,423,044	28,424,046	23,087,910	23,088,912
19,605,744	19,606,746	28,433,044	28,434,046	23,097,910	23,098,912
19,615,745	19,616,746	28,443,044	28,444,046	23,107,910	23,108,912
19,625,745	19,626,746	28,453,044	28,454,046	23,117,910	23,118,912
19,635,745	19,636,746	28,463,044	28,464,046	23,127,910	23,128,912
19,645,745	19,646,746	28,473,044	28,474,046	23,137,910	23,138,912
19,655,745	19,656,746	28,483,045	28,484,046	23,147,910	23,148,912
19,665,745	19,666,746	28,493,045	28,494,046	23,157,911	23,158,912
19,675,745	19,676,746	28,503,045	28,504,046	23,167,911	23,168,912
19,685,745	19,686,746	28,513,045	28,514,046	23,177,911	23,178,912
19,695,745	19,696,746	28,523,045	28,524,046	23,187,911	23,188,912
19,705,745	19,706,746	28,533,045	28,534,046	23,197,911	23,198,912
19,715,745	19,716,746	28,543,045	28,544,046	23,207,911	23,208,912
19,725,745	19,726,746	28,553,045	28,554,046	23,217,911	23,218,912
19,735,745	19,736,747	28,563,045	28,564,046	23,227,911	23,228,912
19,745,745	19,746,747	28,573,045	28,574,047	23,237,911	23,238,913
19,755,745	19,756,747	28,583,045	28,584,047	23,247,911	23,248,913
19,765,745	19,766,747	28,593,045	28,594,047	23,257,911	23,258,913

La Tabla 3, la Tabla 4 y la Tabla 5 ponen a la par de los tiempos derivados de los parámetros de configuración utilizados para los cálculos de $P_{W,R}^{n}$ y $Q_{W,R}^{n}$. Los parámetros base para estos tiempos son los periodos de las tareas (T_P y T_C), los tiempos de desfase

aplicados a cada una (O_W y O_R). Como parámetros derivados se calcula T_{max}, el cual corresponde al periodo más grande entre las tareas de la cadena, así como O_{max}, que corresponde al desfase mayor entre los desfases de cada tarea de la cadena.

Tabla 3

Tiempos de operaciones medidos y parámetros de configuración y cálculo (8-bits)

$P^n_{W,R,8b}$	$Q^n_{W,R,8b}$	T_{P8}	T_{C8}	O_W	O_R	T_{max}	O_{max}
19,075,741	19,076,742	10,000	10,000	0	1,000	10,000	1,000
19,085,741	19,086,742	10,000	10,000	0	1,000	10,000	1,000
19,095,741	19,096,743	10,000	10,000	0	1,000	10,000	1,000
19,105,741	19,106,743	10,000	10,000	0	1,000	10,000	1,000
19,115,741	19,116,743	10,000	10,000	0	1,000	10,000	1,000
19,125,742	19,126,743	10,000	10,000	0	1,000	10,000	1,000
19,135,742	19,136,743	10,000	10,000	0	1,000	10,000	1,000
19,145,742	19,146,743	10,000	10,000	0	1,000	10,000	1,000
19,155,742	19,156,743	10,000	10,000	0	1,000	10,000	1,000
19,165,742	19,166,743	10,000	10,000	0	1,000	10,000	1,000
19,175,742	19,176,743	10,000	10,000	0	1,000	10,000	1,000
19,185,742	19,186,743	10,000	10,000	0	1,000	10,000	1,000
19,195,742	19,196,743	10,000	10,000	0	1,000	10,000	1,000
19,205,742	19,206,743	10,000	10,000	0	1,000	10,000	1,000
19,215,742	19,216,743	10,000	10,000	0	1,000	10,000	1,000
19,225,742	19,226,743	10,000	10,000	0	1,000	10,000	1,000
19,235,742	19,236,743	10,000	10,000	0	1,000	10,000	1,000
19,245,742	19,246,743	10,000	10,000	0	1,000	10,000	1,000
19,255,742	19,256,744	10,000	10,000	0	1,000	10,000	1,000
19,265,742	19,266,744	10,000	10,000	0	1,000	10,000	1,000
19,275,742	19,276,744	10,000	10,000	0	1,000	10,000	1,000
19,285,742	19,286,744	10,000	10,000	0	1,000	10,000	1,000
19,295,743	19,296,744	10,000	10,000	0	1,000	10,000	1,000
19,305,743	19,306,744	10,000	10,000	0	1,000	10,000	1,000
19,315,743	19,316,744	10,000	10,000	0	1,000	10,000	1,000
19,325,743	19,326,744	10,000	10,000	0	1,000	10,000	1,000
19,335,743	19,336,744	10,000	10,000	0	1,000	10,000	1,000
19,345,743	19,346,744	10,000	10,000	0	1,000	10,000	1,000
19,355,743	19,356,744	10,000	10,000	0	1,000	10,000	1,000
19,365,743	19,366,744	10,000	10,000	0	1,000	10,000	1,000
19,375,743	19,376,744	10,000	10,000	0	1,000	10,000	1,000

19,385,743	19,386,745	10,000	10,000	0	1,000	10,000	1,000
19,395,743	19,396,745	10,000	10,000	0	1,000	10,000	1,000
19,405,743	19,406,745	10,000	10,000	0	1,000	10,000	1,000
19,415,744	19,416,745	10,000	10,000	0	1,000	10,000	1,000
19,425,744	19,426,745	10,000	10,000	0	1,000	10,000	1,000
19,435,743	19,436,745	10,000	10,000	0	1,000	10,000	1,000
19,445,743	19,446,745	10,000	10,000	0	1,000	10,000	1,000
19,455,744	19,456,745	10,000	10,000	0	1,000	10,000	1,000
19,465,744	19,466,745	10,000	10,000	0	1,000	10,000	1,000
19,475,744	19,476,745	10,000	10,000	0	1,000	10,000	1,000
19,485,744	19,486,745	10,000	10,000	0	1,000	10,000	1,000
19,495,744	19,496,745	10,000	10,000	0	1,000	10,000	1,000
19,505,744	19,506,745	10,000	10,000	0	1,000	10,000	1,000
19,515,744	19,516,745	10,000	10,000	0	1,000	10,000	1,000
19,525,744	19,526,745	10,000	10,000	0	1,000	10,000	1,000
19,535,744	19,536,745	10,000	10,000	0	1,000	10,000	1,000
19,545,744	19,546,745	10,000	10,000	0	1,000	10,000	1,000
19,555,744	19,556,745	10,000	10,000	0	1,000	10,000	1,000
19,565,744	19,566,745	10,000	10,000	0	1,000	10,000	1,000
19,575,744	19,576,746	10,000	10,000	0	1,000	10,000	1,000
19,585,744	19,586,746	10,000	10,000	0	1,000	10,000	1,000
19,595,744	19,596,746	10,000	10,000	0	1,000	10,000	1,000
19,605,744	19,606,746	10,000	10,000	0	1,000	10,000	1,000
19,615,745	19,616,746	10,000	10,000	0	1,000	10,000	1,000
19,625,745	19,626,746	10,000	10,000	0	1,000	10,000	1,000
19,635,745	19,636,746	10,000	10,000	0	1,000	10,000	1,000
19,645,745	19,646,746	10,000	10,000	0	1,000	10,000	1,000
19,655,745	19,656,746	10,000	10,000	0	1,000	10,000	1,000
19,665,745	19,666,746	10,000	10,000	0	1,000	10,000	1,000
19,675,745	19,676,746	10,000	10,000	0	1,000	10,000	1,000
19,685,745	19,686,746	10,000	10,000	0	1,000	10,000	1,000
19,695,745	19,696,746	10,000	10,000	0	1,000	10,000	1,000
19,705,745	19,706,746	10,000	10,000	0	1,000	10,000	1,000
19,715,745	19,716,746	10,000	10,000	0	1,000	10,000	1,000
19,725,745	19,726,746	10,000	10,000	0	1,000	10,000	1,000
19,735,745	19,736,747	10,000	10,000	0	1,000	10,000	1,000
19,745,745	19,746,747	10,000	10,000	0	1,000	10,000	1,000
19,755,745	19,756,747	10,000	10,000	0	1,000	10,000	1,000
19,765,745	19,766,747	10,000	10,000	0	1,000	10,000	1,000

Tabla 4

Tiempos de operaciones medidos y parámetros de configuración y cálculo (16-bits)

$P^{n}_{W,R,16b}$	$Q^{n}_{W,R,16b}$	T_{P16}	T_{C16}	O_W	O_R	T_{max}	O_{max}
27,903,041	27,904,042	10,000	10,000	0	1,000	10,000	1,000
27,913,041	27,914,042	10,000	10,000	0	1,000	10,000	1,000
27,923,041	27,924,042	10,000	10,000	0	1,000	10,000	1,000
27,933,041	27,934,043	10,000	10,000	0	1,000	10,000	1,000
27,943,041	27,944,043	10,000	10,000	0	1,000	10,000	1,000
27,953,041	27,954,043	10,000	10,000	0	1,000	10,000	1,000
27,963,041	27,964,043	10,000	10,000	0	1,000	10,000	1,000
27,973,041	27,974,043	10,000	10,000	0	1,000	10,000	1,000
27,983,041	27,984,043	10,000	10,000	0	1,000	10,000	1,000
27,993,041	27,994,043	10,000	10,000	0	1,000	10,000	1,000
28,003,042	28,004,043	10,000	10,000	0	1,000	10,000	1,000
28,013,042	28,014,043	10,000	10,000	0	1,000	10,000	1,000
28,023,042	28,024,043	10,000	10,000	0	1,000	10,000	1,000
28,033,042	28,034,043	10,000	10,000	0	1,000	10,000	1,000
28,043,042	28,044,043	10,000	10,000	0	1,000	10,000	1,000
28,053,042	28,054,043	10,000	10,000	0	1,000	10,000	1,000
28,063,042	28,064,043	10,000	10,000	0	1,000	10,000	1,000
28,073,042	28,074,043	10,000	10,000	0	1,000	10,000	1,000
28,083,042	28,084,044	10,000	10,000	0	1,000	10,000	1,000
28,093,042	28,094,044	10,000	10,000	0	1,000	10,000	1,000
28,103,042	28,104,044	10,000	10,000	0	1,000	10,000	1,000
28,113,042	28,114,044	10,000	10,000	0	1,000	10,000	1,000
28,123,042	28,124,044	10,000	10,000	0	1,000	10,000	1,000
28,133,042	28,134,044	10,000	10,000	0	1,000	10,000	1,000
28,143,043	28,144,044	10,000	10,000	0	1,000	10,000	1,000
28,153,042	28,154,044	10,000	10,000	0	1,000	10,000	1,000
28,163,043	28,164,044	10,000	10,000	0	1,000	10,000	1,000
28,173,043	28,174,044	10,000	10,000	0	1,000	10,000	1,000
28,183,043	28,184,044	10,000	10,000	0	1,000	10,000	1,000
28,193,043	28,194,044	10,000	10,000	0	1,000	10,000	1,000
28,203,043	28,204,044	10,000	10,000	0	1,000	10,000	1,000
28,213,043	28,214,044	10,000	10,000	0	1,000	10,000	1,000
28,223,043	28,224,045	10,000	10,000	0	1,000	10,000	1,000
28,233,043	28,234,044	10,000	10,000	0	1,000	10,000	1,000
28,243,043	28,244,044	10,000	10,000	0	1,000	10,000	1,000
28,253,043	28,254,045	10,000	10,000	0	1,000	10,000	1,000

28,263,043	28,264,045	10,000	10,000	0	1,000	10,000	1,000
28,273,043	28,274,045	10,000	10,000	0	1,000	10,000	1,000
28,283,043	28,284,045	10,000	10,000	0	1,000	10,000	1,000
28,293,044	28,294,045	10,000	10,000	0	1,000	10,000	1,000
28,303,044	28,304,045	10,000	10,000	0	1,000	10,000	1,000
28,313,044	28,314,045	10,000	10,000	0	1,000	10,000	1,000
28,323,044	28,324,045	10,000	10,000	0	1,000	10,000	1,000
28,333,044	28,334,045	10,000	10,000	0	1,000	10,000	1,000
28,343,044	28,344,045	10,000	10,000	0	1,000	10,000	1,000
28,353,044	28,354,045	10,000	10,000	0	1,000	10,000	1,000
28,363,044	28,364,045	10,000	10,000	0	1,000	10,000	1,000
28,373,044	28,374,045	10,000	10,000	0	1,000	10,000	1,000
28,383,044	28,384,045	10,000	10,000	0	1,000	10,000	1,000
28,393,044	28,394,045	10,000	10,000	0	1,000	10,000	1,000
28,403,044	28,404,045	10,000	10,000	0	1,000	10,000	1,000
28,413,044	28,414,046	10,000	10,000	0	1,000	10,000	1,000
28,423,044	28,424,046	10,000	10,000	0	1,000	10,000	1,000
28,433,044	28,434,046	10,000	10,000	0	1,000	10,000	1,000
28,443,044	28,444,046	10,000	10,000	0	1,000	10,000	1,000
28,453,044	28,454,046	10,000	10,000	0	1,000	10,000	1,000
28,463,044	28,464,046	10,000	10,000	0	1,000	10,000	1,000
28,473,044	28,474,046	10,000	10,000	0	1,000	10,000	1,000
28,483,045	28,484,046	10,000	10,000	0	1,000	10,000	1,000
28,493,045	28,494,046	10,000	10,000	0	1,000	10,000	1,000
28,503,045	28,504,046	10,000	10,000	0	1,000	10,000	1,000
28,513,045	28,514,046	10,000	10,000	0	1,000	10,000	1,000
28,523,045	28,524,046	10,000	10,000	0	1,000	10,000	1,000
28,533,045	28,534,046	10,000	10,000	0	1,000	10,000	1,000
28,543,045	28,544,046	10,000	10,000	0	1,000	10,000	1,000
28,553,045	28,554,046	10,000	10,000	0	1,000	10,000	1,000
28,563,045	28,564,046	10,000	10,000	0	1,000	10,000	1,000
28,573,045	28,574,047	10,000	10,000	0	1,000	10,000	1,000
28,583,045	28,584,047	10,000	10,000	0	1,000	10,000	1,000
28,593,045	28,594,047	10,000	10,000	0	1,000	10,000	1,000

Tabla 5

Tiempos de operaciones medidos y parámetros de configuración y cálculo (32-bits)

$P^{n}_{W,R,32b}$	$Q^{n}_{W,R,32b}$	T_{P32}	T_{C32}	O_W	O_R	T_{max}	O_{max}
22,567,907	22,568,908	10,000	10,000	0	1,000	10,000	1,000
22,577,907	22,578,908	10,000	10,000	0	1,000	10,000	1,000
22,587,907	22,588,908	10,000	10,000	0	1,000	10,000	1,000
22,597,907	22,598,908	10,000	10,000	0	1,000	10,000	1,000
22,607,907	22,608,909	10,000	10,000	0	1,000	10,000	1,000
22,617,907	22,618,909	10,000	10,000	0	1,000	10,000	1,000
22,627,907	22,628,909	10,000	10,000	0	1,000	10,000	1,000
22,637,907	22,638,909	10,000	10,000	0	1,000	10,000	1,000
22,647,907	22,648,909	10,000	10,000	0	1,000	10,000	1,000
22,657,907	22,658,909	10,000	10,000	0	1,000	10,000	1,000
22,667,908	22,668,909	10,000	10,000	0	1,000	10,000	1,000
22,677,908	22,678,909	10,000	10,000	0	1,000	10,000	1,000
22,687,908	22,688,909	10,000	10,000	0	1,000	10,000	1,000
22,697,908	22,698,909	10,000	10,000	0	1,000	10,000	1,000
22,707,908	22,708,909	10,000	10,000	0	1,000	10,000	1,000
22,717,908	22,718,909	10,000	10,000	0	1,000	10,000	1,000
22,727,908	22,728,909	10,000	10,000	0	1,000	10,000	1,000
22,737,908	22,738,909	10,000	10,000	0	1,000	10,000	1,000
22,747,908	22,748,909	10,000	10,000	0	1,000	10,000	1,000
22,757,908	22,758,910	10,000	10,000	0	1,000	10,000	1,000
22,767,908	22,768,910	10,000	10,000	0	1,000	10,000	1,000
22,777,908	22,778,910	10,000	10,000	0	1,000	10,000	1,000
22,787,908	22,788,910	10,000	10,000	0	1,000	10,000	1,000
22,797,908	22,798,910	10,000	10,000	0	1,000	10,000	1,000
22,807,908	22,808,910	10,000	10,000	0	1,000	10,000	1,000
22,817,908	22,818,910	10,000	10,000	0	1,000	10,000	1,000
22,827,908	22,828,910	10,000	10,000	0	1,000	10,000	1,000
22,837,909	22,838,910	10,000	10,000	0	1,000	10,000	1,000
22,847,909	22,848,910	10,000	10,000	0	1,000	10,000	1,000
22,857,909	22,858,910	10,000	10,000	0	1,000	10,000	1,000
22,867,909	22,868,910	10,000	10,000	0	1,000	10,000	1,000
22,877,909	22,878,910	10,000	10,000	0	1,000	10,000	1,000
22,887,909	22,888,911	10,000	10,000	0	1,000	10,000	1,000
22,897,909	22,898,911	10,000	10,000	0	1,000	10,000	1,000
22,907,909	22,908,911	10,000	10,000	0	1,000	10,000	1,000
22,917,909	22,918,911	10,000	10,000	0	1,000	10,000	1,000

22,927,909	22,928,911	10,000	10,000	0	1,000	10,000	1,000
22,937,909	22,938,911	10,000	10,000	0	1,000	10,000	1,000
22,947,909	22,948,911	10,000	10,000	0	1,000	10,000	1,000
22,957,909	22,958,911	10,000	10,000	0	1,000	10,000	1,000
22,967,910	22,968,911	10,000	10,000	0	1,000	10,000	1,000
22,977,910	22,978,911	10,000	10,000	0	1,000	10,000	1,000
22,987,910	22,988,911	10,000	10,000	0	1,000	10,000	1,000
22,997,910	22,998,911	10,000	10,000	0	1,000	10,000	1,000
23,007,910	23,008,911	10,000	10,000	0	1,000	10,000	1,000
23,017,910	23,018,911	10,000	10,000	0	1,000	10,000	1,000
23,027,910	23,028,911	10,000	10,000	0	1,000	10,000	1,000
23,037,910	23,038,911	10,000	10,000	0	1,000	10,000	1,000
23,047,910	23,048,911	10,000	10,000	0	1,000	10,000	1,000
23,057,910	23,058,911	10,000	10,000	0	1,000	10,000	1,000
23,067,910	23,068,911	10,000	10,000	0	1,000	10,000	1,000
23,077,910	23,078,911	10,000	10,000	0	1,000	10,000	1,000
23,087,910	23,088,912	10,000	10,000	0	1,000	10,000	1,000
23,097,910	23,098,912	10,000	10,000	0	1,000	10,000	1,000
23,107,910	23,108,912	10,000	10,000	0	1,000	10,000	1,000
23,117,910	23,118,912	10,000	10,000	0	1,000	10,000	1,000
23,127,910	23,128,912	10,000	10,000	0	1,000	10,000	1,000
23,137,910	23,138,912	10,000	10,000	0	1,000	10,000	1,000
23,147,910	23,148,912	10,000	10,000	0	1,000	10,000	1,000
23,157,911	23,158,912	10,000	10,000	0	1,000	10,000	1,000
23,167,911	23,168,912	10,000	10,000	0	1,000	10,000	1,000
23,177,911	23,178,912	10,000	10,000	0	1,000	10,000	1,000
23,187,911	23,188,912	10,000	10,000	0	1,000	10,000	1,000
23,197,911	23,198,912	10,000	10,000	0	1,000	10,000	1,000
23,207,911	23,208,912	10,000	10,000	0	1,000	10,000	1,000
23,217,911	23,218,912	10,000	10,000	0	1,000	10,000	1,000
23,227,911	23,228,912	10,000	10,000	0	1,000	10,000	1,000
23,237,911	23,238,913	10,000	10,000	0	1,000	10,000	1,000
23,247,911	23,248,913	10,000	10,000	0	1,000	10,000	1,000
23,257,911	23,258,913	10,000	10,000	0	1,000	10,000	1,000

La Tabla 6, la Tabla 7 y la Tabla 8 por su parte presentan las latencia de edad y de reacción (ver Ecuaciones 7 y 9 respectivamente) de las cadenas utilizando búferes de 8, 16 y 32 bits con respecto a la longitud de su camino básico θ_{EC}^{n}, (ver Ecuación 8) tiempo entre las operaciones de escritura y lectura.

Tabla 6

Longitud de límite y latencias para operaciones con búferes de 8 bits

θ_{EC}^{n}	α^{n}	δ^{n}	Q_{n-1}^{n+1}	$P_{1,2}^{n}$
1,001	21,001	31,001	19,086,742	19,075,741
1,001	21,002	31,002	19,096,743	19,085,741
1,002	21,002	31,002	19,106,743	19,095,741
1,002	21,002	31,002	19,116,743	19,105,741
1,002	21,002	31,002	19,126,743	19,115,741
1,001	21,001	31,001	19,136,743	19,125,742
1,001	21,001	31,001	19,146,743	19,135,742
1,001	21,001	31,001	19,156,743	19,145,742
1,001	21,001	31,001	19,166,743	19,155,742
1,001	21,001	31,001	19,176,743	19,165,742
1,001	21,001	31,001	19,186,743	19,175,742
1,001	21,001	31,001	19,196,743	19,185,742
1,001	21,001	31,001	19,206,743	19,195,742
1,001	21,001	31,001	19,216,743	19,205,742
1,001	21,001	31,001	19,226,743	19,215,742
1,001	21,001	31,001	19,236,743	19,225,742
1,001	21,001	31,001	19,246,743	19,235,742
1,001	21,002	31,002	19,256,744	19,245,742
1,002	21,002	31,002	19,266,744	19,255,742
1,002	21,002	31,002	19,276,744	19,265,742
1,002	21,002	31,002	19,286,744	19,275,742
1,002	21,002	31,002	19,296,744	19,285,742
1,001	21,001	31,001	19,306,744	19,295,743
1,001	21,001	31,001	19,316,744	19,305,743
1,001	21,001	31,001	19,326,744	19,315,743
1,001	21,001	31,001	19,336,744	19,325,743
1,001	21,001	31,001	19,346,744	19,335,743
1,001	21,001	31,001	19,356,744	19,345,743
1,001	21,001	31,001	19,366,744	19,355,743
1,001	21,001	31,001	19,376,744	19,365,743

1,001	21,002	31,002	19,386,745	19,375,743
1,002	21,002	31,002	19,396,745	19,385,743
1,002	21,002	31,002	19,406,745	19,395,743
1,002	21,002	31,002	19,416,745	19,405,743
1,001	21,001	31,001	19,426,745	19,415,744
1,001	21,001	31,001	19,436,745	19,425,744
1,002	21,002	31,002	19,446,745	19,435,743
1,002	21,002	31,002	19,456,745	19,445,743
1,001	21,001	31,001	19,466,745	19,455,744
1,001	21,001	31,001	19,476,745	19,465,744
1,001	21,001	31,001	19,486,745	19,475,744
1,001	21,001	31,001	19,496,745	19,485,744
1,001	21,001	31,001	19,506,745	19,495,744
1,001	21,001	31,001	19,516,745	19,505,744
1,001	21,001	31,001	19,526,745	19,515,744
1,001	21,001	31,001	19,536,745	19,525,744
1,001	21,001	31,001	19,546,745	19,535,744
1,001	21,001	31,001	19,556,745	19,545,744
1,001	21,001	31,001	19,566,745	19,555,744
1,001	21,002	31,002	19,576,746	19,565,744
1,002	21,002	31,002	19,586,746	19,575,744
1,002	21,002	31,002	19,596,746	19,585,744
1,002	21,002	31,002	19,606,746	19,595,744
1,002	21,002	31,002	19,616,746	19,605,744
1,001	21,001	31,001	19,626,746	19,615,745
1,001	21,001	31,001	19,636,746	19,625,745
1,001	21,001	31,001	19,646,746	19,635,745
1,001	21,001	31,001	19,656,746	19,645,745
1,001	21,001	31,001	19,666,746	19,655,745
1,001	21,001	31,001	19,676,746	19,665,745
1,001	21,001	31,001	19,686,746	19,675,745
1,001	21,001	31,001	19,696,746	19,685,745
1,001	21,001	31,001	19,706,746	19,695,745
1,001	21,001	31,001	19,716,746	19,705,745
1,001	21,001	31,001	19,726,746	19,715,745
1,001	21,002	31,002	19,736,747	19,725,745
1,002	21,002	31,002	19,746,747	19,735,745
1,002	21,002	31,002	19,756,747	19,745,745
1,002	21,002	31,002	19,766,747	19,755,745
1,002	21,002	31,002	19,776,747	19,765,745

Tabla 7

Longitud de límite y latencias para operaciones con búferes de 16 bits

θ_{EC}^{n}	α^{n}	δ^{n}	Q_{n-1}^{n+1}	$P_{1,2}^{n}$
1,001	21,001	31,001	27,914,042	27,903,041
1,001	21,001	31,001	27,924,042	27,913,041
1,001	21,002	31,002	27,934,043	27,923,041
1,002	21,002	31,002	27,944,043	27,933,041
1,002	21,002	31,002	27,954,043	27,943,041
1,002	21,002	31,002	27,964,043	27,953,041
1,002	21,002	31,002	27,974,043	27,963,041
1,002	21,002	31,002	27,984,043	27,973,041
1,002	21,002	31,002	27,994,043	27,983,041
1,002	21,002	31,002	28,004,043	27,993,041
1,001	21,001	31,001	28,014,043	28,003,042
1,001	21,001	31,001	28,024,043	28,013,042
1,001	21,001	31,001	28,034,043	28,023,042
1,001	21,001	31,001	28,044,043	28,033,042
1,001	21,001	31,001	28,054,043	28,043,042
1,001	21,001	31,001	28,064,043	28,053,042
1,001	21,001	31,001	28,074,043	28,063,042
1,001	21,002	31,002	28,084,044	28,073,042
1,002	21,002	31,002	28,094,044	28,083,042
1,002	21,002	31,002	28,104,044	28,093,042
1,002	21,002	31,002	28,114,044	28,103,042
1,002	21,002	31,002	28,124,044	28,113,042
1,002	21,002	31,002	28,134,044	28,123,042
1,002	21,002	31,002	28,144,044	28,133,042
1,001	21,001	31,001	28,154,044	28,143,043
1,002	21,002	31,002	28,164,044	28,153,042
1,001	21,001	31,001	28,174,044	28,163,043
1,001	21,001	31,001	28,184,044	28,173,043
1,001	21,001	31,001	28,194,044	28,183,043
1,001	21,001	31,001	28,204,044	28,193,043
1,001	21,001	31,001	28,214,044	28,203,043
1,001	21,002	31,002	28,224,045	28,213,043
1,002	21,001	31,001	28,234,044	28,223,043
1,001	21,001	31,001	28,244,044	28,233,043
1,001	21,002	31,002	28,254,045	28,243,043
1,002	21,002	31,002	28,264,045	28,253,043

1,002	21,002	31,002	28,274,045	28,263,043
1,002	21,002	31,002	28,284,045	28,273,043
1,002	21,002	31,002	28,294,045	28,283,043
1,001	21,001	31,001	28,304,045	28,293,044
1,001	21,001	31,001	28,314,045	28,303,044
1,001	21,001	31,001	28,324,045	28,313,044
1,001	21,001	31,001	28,334,045	28,323,044
1,001	21,001	31,001	28,344,045	28,333,044
1,001	21,001	31,001	28,354,045	28,343,044
1,001	21,001	31,001	28,364,045	28,353,044
1,001	21,001	31,001	28,374,045	28,363,044
1,001	21,001	31,001	28,384,045	28,373,044
1,001	21,001	31,001	28,394,045	28,383,044
1,001	21,001	31,001	28,404,045	28,393,044
1,001	21,002	31,002	28,414,046	28,403,044
1,002	21,002	31,002	28,424,046	28,413,044
1,002	21,002	31,002	28,434,046	28,423,044
1,002	21,002	31,002	28,444,046	28,433,044
1,002	21,002	31,002	28,454,046	28,443,044
1,002	21,002	31,002	28,464,046	28,453,044
1,002	21,002	31,002	28,474,046	28,463,044
1,002	21,002	31,002	28,484,046	28,473,044
1,001	21,001	31,001	28,494,046	28,483,045
1,001	21,001	31,001	28,504,046	28,493,045
1,001	21,001	31,001	28,514,046	28,503,045
1,001	21,001	31,001	28,524,046	28,513,045
1,001	21,001	31,001	28,534,046	28,523,045
1,001	21,001	31,001	28,544,046	28,533,045
1,001	21,001	31,001	28,554,046	28,543,045
1,001	21,001	31,001	28,564,046	28,553,045
1,001	21,002	31,002	28,574,047	28,563,045
1,002	21,002	31,002	28,584,047	28,573,045
1,002	21,002	31,002	28,594,047	28,583,045
1,002	21,002	31,002	28,604,047	28,593,045

Tabla 8

Longitud de límite y latencias para operaciones con búferes de 32 bits

θ_{EC}^{n}	α^{n}	δ^{n}	Q_{n-1}^{n+1}	$P_{1,2}^{n}$
1,001	21,001	31,001	22,578,908	22,567,907
1,001	21,001	31,001	22,588,908	22,577,907
1,001	21,001	31,001	22,598,908	22,587,907
1,001	21,002	31,002	22,608,909	22,597,907
1,002	21,002	31,002	22,618,909	22,607,907
1,002	21,002	31,002	22,628,909	22,617,907
1,002	21,002	31,002	22,638,909	22,627,907
1,002	21,002	31,002	22,648,909	22,637,907
1,002	21,002	31,002	22,658,909	22,647,907
1,002	21,002	31,002	22,668,909	22,657,907
1,001	21,001	31,001	22,678,909	22,667,908
1,001	21,001	31,001	22,688,909	22,677,908
1,001	21,001	31,001	22,698,909	22,687,908
1,001	21,001	31,001	22,708,909	22,697,908
1,001	21,001	31,001	22,718,909	22,707,908
1,001	21,001	31,001	22,728,909	22,717,908
1,001	21,001	31,001	22,738,909	22,727,908
1,001	21,001	31,001	22,748,909	22,737,908
1,001	21,002	31,002	22,758,910	22,747,908
1,002	21,002	31,002	22,768,910	22,757,908
1,002	21,002	31,002	22,778,910	22,767,908
1,002	21,002	31,002	22,788,910	22,777,908
1,002	21,002	31,002	22,798,910	22,787,908
1,002	21,002	31,002	22,808,910	22,797,908
1,002	21,002	31,002	22,818,910	22,807,908
1,002	21,002	31,002	22,828,910	22,817,908
1,002	21,002	31,002	22,838,910	22,827,908
1,001	21,001	31,001	22,848,910	22,837,909
1,001	21,001	31,001	22,858,910	22,847,909
1,001	21,001	31,001	22,868,910	22,857,909
1,001	21,001	31,001	22,878,910	22,867,909
1,001	21,002	31,002	22,888,911	22,877,909
1,002	21,002	31,002	22,898,911	22,887,909
1,002	21,002	31,002	22,908,911	22,897,909
1,002	21,002	31,002	22,918,911	22,907,909
1,002	21,002	31,002	22,928,911	22,917,909

1,002	21,002	31,002	22,938,911	22,927,909
1,002	21,002	31,002	22,948,911	22,937,909
1,002	21,002	31,002	22,958,911	22,947,909
1,002	21,002	31,002	22,968,911	22,957,909
1,001	21,001	31,001	22,978,911	22,967,910
1,001	21,001	31,001	22,988,911	22,977,910
1,001	21,001	31,001	22,998,911	22,987,910
1,001	21,001	31,001	23,008,911	22,997,910
1,001	21,001	31,001	23,018,911	23,007,910
1,001	21,001	31,001	23,028,911	23,017,910
1,001	21,001	31,001	23,038,911	23,027,910
1,001	21,001	31,001	23,048,911	23,037,910
1,001	21,001	31,001	23,058,911	23,047,910
1,001	21,001	31,001	23,068,911	23,057,910
1,001	21,001	31,001	23,078,911	23,067,910
1,001	21,002	31,002	23,088,912	23,077,910
1,002	21,002	31,002	23,098,912	23,087,910
1,002	21,002	31,002	23,108,912	23,097,910
1,002	21,002	31,002	23,118,912	23,107,910
1,002	21,002	31,002	23,128,912	23,117,910
1,002	21,002	31,002	23,138,912	23,127,910
1,002	21,002	31,002	23,148,912	23,137,910
1,002	21,002	31,002	23,158,912	23,147,910
1,001	21,001	31,001	23,168,912	23,157,911
1,001	21,001	31,001	23,178,912	23,167,911
1,001	21,001	31,001	23,188,912	23,177,911
1,001	21,001	31,001	23,198,912	23,187,911
1,001	21,001	31,001	23,208,912	23,197,911
1,001	21,001	31,001	23,218,912	23,207,911
1,001	21,001	31,001	23,228,912	23,217,911
1,001	21,002	31,002	23,238,913	23,227,911
1,002	21,002	31,002	23,248,913	23,237,911
1,002	21,002	31,002	23,258,913	23,247,911
1,002	21,002	31,002	23,268,913	23,257,911

Comparación entre LET y LET-TDMA

Utilizando la configuración referenciada en la Tabla 1, se procedió a extraer y analizar tiempos de las operaciones de lectura y escritura en las implementaciones de LET y LET-TDMA utilizando tamaños de buffer de 8, 16 y 32 bits. Con esto se buscó evaluar la variabilidad del tiempo de latencia entre transmisiones mediante el cálculo del error cuadrático medio, utilizando como referencia el tiempo de desfase O_R definido para los tiempos de lectura $Q_{W,R}^n$ de la caracterización de tiempos.

La Tabla 9 muestra los cálculos del error cuadrático medio con los tiempos medidos de la implementación de LET utilizando cada tamaño de buffer de los antes mencionados. Se midieron los tiempos utilizando dos métodos distintos de gestión de tareas (Sistema Operativo y Rutina de Servicio de Interrupción) con la finalidad de mostrar la variabilidad introducida por el sistema operativo. En la implementación del modelo LET utilizando la combinación de manejadores por interrupción y sistema operativo fue perceptible el incremento gradual del tiempo en la latencia entre ambas tareas en cada ejecución de la tarea consumidora. El incremento medido fue de 62.5 ns en cada activación de la tarea consumidora. La razón se origina del manejo de la tarea consumidora por medio de un sistema operativo, dado que ambas tareas se configuraron con el mismo periodo de ejecución. Sin embargo, el uso de un sistema operativo agrega una carga adicional de procesamiento al núcleo, lo que hace que exista un incremento de tiempo. En contraste, haciendo el manejo de la tarea productora por medio de una interrupción, la ejecución de la tarea inicia de manera muy rápida por lo que no hay un agregado de tiempo entre ambos gestores de tareas de sus correspondientes núcleos, y la variabilidad se determina meramente por los tiempos de inicialización de cada manejador.

La implementación del modelo LET-TDMA por su parte muestra una latencia casi constante entre operaciones, con una latencia promedio de 1.00143 ms. Esto se debe a la sincronización entre tareas manejada por las ranuras de tiempo de TDMA, la cual mantiene a las cadenas sincronizadas, alineando los tiempos de activación de la tarea productora y la

tarea consumidora. La Tabla 9 muestra los cálculos del error cuadrático medio (ECM) de los tiempos medidos de la implementación de LET-TDMA utilizando cada tamaño de buffer.

Tabla 9

Valores calculados de ECM utilizando búferes de distintos tamaños

Método	Gestor en Cortex M0+	Gestor en Cortex M4	ECM (8 bits)	ECM (16 bits)	ECM (32 bits)
LET	ISR	AUTOSAR	3.285 ms	8.926 ms	0.234 ms
	ISR	ISR	9.168 ms	7.991 ms	1.407 ms
LET-TDMA	ISR	AUTOSAR	1 ms	1 ms	1 ms

Comparación entre resultados de resultados de estudio

Se realizó un análisis de distintos estudios donde se implementaron distintos modelos y métodos de comunicación con la finalidad de comparar los resultados de este estudio contra otros similares. La Tabla 10 muestra los distintos estudios. El modelo que muestra menores latencias es la comunicación explícita, sobre todo en términos de latencia reactiva, pero tiene una variabilidad mayor comparada con los demás modelos. En contraste la comunicación implícita resuelve parcialmente el problema de la variabilidad al utilizar copias locales de los datos recibidos para el procesamiento de la tarea, otorgando coherencia en el intercambio de datos. Sin embargo, el tiempo de ejecución de cada tarea determina la capacidad de predecir el comportamiento del flujo de datos, que va a ser directamente dependiente de la complejidad de los algoritmos ejecutados, y por lo tanto la variabilidad va a ser tan alta como la cantidad de ramas de decisión de un algoritmo. Por otro lado, se puede observar que el modelo LET tiene latencias significativamente mayores a las de otros modelos, pero proveyendo tiempos de respuesta fijos y predecibles. Sin embargo, su implementación en sistemas de múltiples núcleos tiende a ser más complicada dados las restricciones de tiempo

y las limitaciones de las ejecuciones paralelas. La alternativa de LET-DMA utilizando SPM para la copia de los buffers provee una solución más eficiente al delegar la copia de memoria a los *drivers* de DMA. No obstante, su implementación es muy específica a la plataforma donde se implemente, y el tamaño de buffers es directamente dependiendo del tamaño de las memorias SPM. La implementación de LET-TDMA por su parte combina la predictibilidad y consistencia propias del modelo LET con la componibilidad del Acceso Múltiple por División de Tiempo, logrando una sincronización entre tareas de cadenas de eventos distribuidas en núcleos diferentes. Esta implementación es agnóstica al *hardware* y fácilmente extensible al uso de recursos propios de una plataforma, por lo que es más flexible en términos de implementación.

Tabla 10

Comparativo de técnicas y métodos

Autor	Modelo	Núcleos	Tareas	Edad Máxima	L2F	Latencia Reactiva	Longitud de Cadena
Tabish et al.	TDMA-DMA con SPM	3	5-20	-	400	-	-
Biondi et al.	Comunicación Explícita	2	4	-	12.74	22.746	4, 3 búferes
			3	-	6	-	3, 2 búferes
	Comunicación Implícita		5	154.234	-	-	14 búferes
					-		
	LET						
Hamann et al.	Comunicación Explícita	4	3	-	-	8.6	10,000 búferes
		4	3	-	-	36.9	
	Comunicación Implícita	4	3	-	-	111.97	
	LET						
Martínez et al.	Comunicación Explícita (CE1)	4	3	123.718	-	125.710	3, 2 búferes
		4	3	154.988	-	151.855	3, 2 búferes
	Comunicación Implícita (CE1)	4	3	210	-	212	3, 2 búferes
		4	3	2.844	-	64.894	3, 2 búferes
	LET (CE1)	4	3		-	66.33	

	Comunicación Explícita (CE2) Comunicación Implícita (CE2) LET (CE2)	4	3	6.54 53.597	-	102.597	3, 2 búferes 3, 2 búferes
Maia & Fohler	LET WCR-LET Maia-Fohler	4 4 4	2-5 2-5 2-5	4040 - 3237	5000 4000 4197	420.43 - -	38 búferes
Wang et al.	fLETEnum fLETSBacktracking fLETSymbOp	4 4 4	21 1 1	2725 2725 2725	3685 3685 3685	- - -	31-36 búferes
Günzel et al.	D19 K18 B17 Günzel	2-5 2-5 2-5 2-5	21 21 21 21	3250 2650 2650 1750	4750 2700 - 3250	- - - -	30 a 60 búferes
Mosqueda-Arvizu et al.	LET LET-TDMA	2 2	2 2	[20∼40] 10	[10∼40] 10	[10∼50] 20	2, 1 búfer

Dada sus complejidades y por su alta demanda de recursos, los modelos basados en intercambio de mensaje quedan fuera del alcance de la tabla comparativa principal. Sin embargo, la Tabla 11 muestra los resultados de los documentos que estudian los modelos de intercambio de mensajes a manera de referencia.

Tabla 11

Comparativo de métodos basados en intercambio de mensajes

Autor	Modelo	Núcleos	Tareas	Edad Máxima	L2F	Latencia Reactiva	Longitud de Cadena
Bathe	Message Passing	4	2	-	100 ms	-	2, 1 mensaje
Hung et al.	MSG MemCpy SD MSG MemCpy LD MSG MemCpy LS	3	2	-	≈2.4 ms ≈2.3 ms ≈2.2 ms	-	2, 1 mensaje
Hung et al.	MSG EMDMA SD MSG EMDMA LD MSG EMDMA LS	3	2	-	≈4.75 ms ≈4.6 ms ≈4.8 ms	-	2, 1 mensaje
Hung et al.	MSG memcpy CELL SD MSG PPE CELL MSG SPE CELL	9	2	-	≈0.001 µs ≈0.003 µs ≈0.0075 µs	-	2, 1 mensaje

Los sistemas de tiempo real requieren de poca de manda de recursos por la necesidad de rápida respuesta y los modelos como el MSG (Hung et al., 2012) y el MCAPI tienen mayores implicaciones a la hora de implementar sus protocolos que incluyen comunicación síncrona.

Discusión

Existen distintas propuestas para implementar la comunicación en sistemas embebidos multinúcleo, que van desde la adaptación de protocolos utilizados en sistemas distribuidos, hasta los métodos heurísticos y propuestas basadas en semánticas de comunicación punto a punto. En este sentido, la aplicación del modelo LET en conjunto con el método de TDMA representa una mejora con respecto a los tiempos de respuesta de las cadenas de eventos y el determinismo del flujo de datos. El modelo LET por sí solo no considera la implementación en sistemas de múltiples núcleos, por lo que no cumple con su característica determinística

al implementarse en estos sistemas. En los resultados de la implementación de LET se puede observar que la latencia depende de otros factores, como lo son los tiempos de inicialización de cada núcleo y eventos externos que puedan agregar desfase entre las ejecuciones paralelas de los núcleos.

Los resultados obtenidos en este estudio demuestran la posibilidad de implementar LET en cadenas de eventos distribuidas en distintos núcleos a través de la gestión de los buffers mediante la técnica de TDMA, manteniendo la consistencia de los datos y la sincronización del flujo de estos, características que son de extrema importancia en sistemas de tiempo real. Como resultado, se anula la pérdida de datos por fluctuación de los periodos de ejecución de las tareas.

Adicionalmente, la gestión de los buffers compartidos utilizando TDMA puede ayudar a reducir el uso de recursos de memoria, debido a la característica de aislamiento de tiempo, que permite reutilizar los mismos buffers entre distintas tareas que implementen cadenas de eventos en otras ranuras de tiempo. A su vez, esto permite que se pueda prescindir de métodos clásicos de sincronización, como los semáforos.

Mantener la latencia baja es crucial en los sistemas críticos de tiempo real debido a los estrictos requerimientos relacionados al tiempo de respuesta que dichos sistemas requieren, donde la tolerancia es baja, y la predictibilidad es clave para el modelado y pruebas de las funcionalidades. La latencia baja garantiza también un mejor desempeño de los algoritmos que implementan las cadenas de eventos, ya que la respuesta a un evento de entrada es procesada en menor tiempo. Es por esto por lo que la integración del modelo LET con el multiplexado en tiempo del TDMA provee una solución eficaz al hacer posible la predicción y modelado de las cadenas de eventos cuyas tareas se encuentran distribuidas en diferentes núcleos de procesamiento, además de evitar la pérdida de datos por fluctuación de ejecución de procesamiento paralelo.

Conclusiones

En este estudio se desarrolló la aplicación del modelo LET en conjunto con el esquema de TDMA para mejorar la predictibilidad de la comunicación entre núcleos de un sistema de tiempo real y reducir la latencia entre la comunicación. El problema planteado se estudió desde la perspectiva de un modelo de sistema compuesto por un microcontrolador de dos núcleos para aplicaciones de la industria automotriz. Posteriormente se implementó en software un gestor de comunicación LET para distintas tareas, independiente del núcleo en el que se encuentran. Se le agregó además un gestor de acceso a memoria bajo el esquema de TDMA, para controlar las operaciones.

La implementación fue evaluada mediante la integración de sus componentes de software en la plataforma de desarrollo basada en el microcontrolador Traveo II de Cypress, la cual cuenta con dos núcleos heterogéneos basados en la arquitectura ARM Cortex-M, y en los cuales uno ejecuta aplicaciones mediante un planificador sencillo basado en interrupción, mientras el otro ejecuta aplicaciones mediante un RTOS de AUTOSAR. Se realizó el análisis de los tiempos de transferencia entre una tarea productora y una tarea consumidora, ambas ubicadas en núcleos diferentes, utilizando el modelo LET y el modelo propuesto de LET-TDMA con la finalidad de comparar los comportamientos entre ambas implementaciones. El flujo de los datos se definió como una simple señal de rampa, donde el contador desborda a 0 al llegar a su límite. Mediante la comparación de ambas implementaciones se pudo observar la reducción de la latencia de transmisión de un dato de una tarea productora en el núcleo 1 a una tarea consumidora en el núcleo 2, así como determinismo debido a que el modelo propuesto reduce la variabilidad de la latencia y la mantiene cercana a ser constante en distintos ciclos de ejecución.

La implementación es independiente del microcontrolador, del sistema operativo y el tipo de memoria, a diferencia de otras propuestas de la literatura que fueron mencionadas en este documento de tesis, dónde se requieren modificaciones al sistema operativo y/o al compilador, o en su defecto existe dependencia a *hardware* específico como son las

memorias locales o de *Scratchpad*, cuyas implementaciones cuales pueden variar entre microcontroladores de distinta arquitectura y proveedor.

Los resultados muestran las siguientes mejoras:

- Tiempo de respuesta: La configuración fija de las operaciones de lectura y escritura influyó en el tiempo de respuesta, el cual permaneció prácticamente constante, con un tiempo de 1 ms entre escritura del productor y lectura del consumidor.
- Determinismo temporal: Las operaciones de escritura y lectura se realizaron en los mismos intervalos de tiempo a lo largo de la ejecución del software por lo que es posible predecir y modelar el flujo de los datos.
- Consistencia de datos: Una característica importante del TDMA es la temporalidad fija, la cual provee de sincronización a las tareas ubicadas en distintos núcleos, por lo que se elimina la pérdida de datos por fluctuación.
- Reducción de uso de recursos de memoria: El aislamiento temporal del modelo TDMA permite que los buffers de memoria compartida se puedan compartir entre distintas cadenas de eventos, debido a que el acceso se controla mediante ranuras de tiempo, por lo que se puede reducir la cantidad de memoria reservada para el intercambio de datos entre núcleos de procesamiento.

Referencias

Antinyan, V. (2020). Revealing the complexity of automotive software. *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1525–1528. https://doi.org/10.1145/3368089.3417038

Bathe, S. (2016). *Secure Communication in a Multi OS Environment*.

Beckert, M. (2019). *Scheduling Mechanisms for Efficient and Safe Automotive Systems Integration* [Technischen Universität Braunschweig]. https://doi.org/https://doi.org/10.24355/dbbs.084-201911070747-5

Biondi, A., Pazzaglia, P., Balsini, A., & Natale, M. Di. (2017). Logical Execution Time Implementation and Memory Optimization Issues in AUTOSAR Applications for Multicores. *8th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. http://retis.sssup.it/~marco/papers/2017/watersch.pdf

Feiertag, N., Richter, K., Nordlander, J., & Jonsson, J. (2008). A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. *Real-Time Systems Symposium*, 1–8.

Gemlau, K. B., Kohler, L., & Ernst, R. (2021). A Platform Programming Paradigm for Heterogeneous Systems Integration. *Proceedings of the IEEE*, *109*(4), 582–603. https://doi.org/10.1109/JPROC.2020.3035874

Gemlau, K.-B., Köhler, L., Ernst, R., & Quinton, S. (2021). System-Level Logical Execution Time: Augmenting the Logical Execution Time Paradigm for Distributed Real-Time Automotive Software. *ACM Transactions on Cyber-Physical Systems*, *1*(1), 1–27.

Günzel, M., Chen, K. H., Ueter, N., Von Der Brüggen, G., Dürr, M., & Chen, J. J. (2023). Compositional Timing Analysis of Asynchronized Distributed Cause-effect Chains.

ACM Transactions on Embedded Computing Systems, *22*(4). https://doi.org/10.1145/3587036

Hamann, A., Dasari, D., Kramer, S., Pressler, M., & Wurst, F. (2017). Communication centric design in complex automotive embedded systems. *Leibniz International Proceedings in Informatics, LIPIcs*, *76*(10), 101–1020. https://doi.org/10.4230/LIPIcs.ECRTS.2017.10

Heath, S. (2003). *Embedded Systems Design* (Second). Newnes.

Hung, S. H., Chiu, P. H., & Shih, C. S. (2012). Building and optimizing a scalable and portable message-passing library for embedded multicore systems. *Information*, *15*(7), 3039–3057.

Hung, S. H., Chiu, P. H., Tu, C. H., Chou, W. T., & Yang, W. L. (2014). Message-passing programming for embedded multicore signal-processing platforms. *Journal of Signal Processing Systems*, *75*(2), 123–139. https://doi.org/10.1007/s11265-013-0732-8

Hung, S. H., Tu, C. H., & Yang, W. L. (2011). A portable, efficient inter-core communication scheme for embedded multicore platforms. *Journal of Systems Architecture*, *57*(2), 193–205. https://doi.org/10.1016/j.sysarc.2010.11.003

Igarashi, S., & Azumi, T. (2019). Work in progress: Considering heuristic scheduling for NoC-Based clustered many-core processor using LET model. *Proceedings - Real-Time Systems Symposium*, *2019-Decem*, 516–519. https://doi.org/10.1109/RTSS46320.2019.00053

Igarashi, S., Ishigooka, T., Horiguchi, T., Koike, R., & Azumi, T. (2020). Heuristic Contention-Free Scheduling Algorithm for Multi-core Processor using LET Model. *Proceedings of the 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2020*. https://doi.org/10.1109/DS-RT50469.2020.9213582

Kopetz, H. (2011). Real-time systems: Design principles for distributed embedded applications. En *Computers & Mathematics with Applications* (2nd ed., Número 10). Springer US. https://doi.org/10.1016/S0898-1221(97)90277-7

Krawczyk, L., Bazzal, M., Govindarajan, R. P., & Wolff, C. (2019). Model-based timing analysis and deployment optimization for heterogeneous multi-core systems using eclipse APP4MC. *Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2019*, 44–53. https://doi.org/10.1109/MODELS-C.2019.00013

Lara, E., Debon, G., Goerl, R., Villa, P., Schramm, D., Poehls, L. B., & Vargas, F. (2019). A new approach to guarantee critical task schedulability in TDMA-based bus access of multicore architecture. *LATS 2019 - 20th IEEE Latin American Test Symposium*, 1–6. https://doi.org/10.1109/LATW.2019.8704572

Limited, A. (2006). *Arm ® v7-M Architecture Reference Manual*. http://www.arm.com

Maia, L., & Fohler, G. (2024). Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains for the LET Model. *ERTS 2024-12th European Congress Embedded Real Time System. June 2024*. https://doi.org/https://doi.org/10.48550/arXiv.2305.02121

Martinez, J., Sanudo, I., & Bertogna, M. (2018a). Analytical characterization of end-to-end communication delays with logical execution time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(11), 2244–2254. https://doi.org/10.1109/TCAD.2018.2857398

Martinez, J., Sanudo, I., & Bertogna, M. (2018b). Analytical characterization of end-to-end communication delays with logical execution time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(11), 2244–2254. https://doi.org/10.1109/TCAD.2018.2857398

Martinez, J., Sañudo, I., & Bertogna, M. (2020). End-to-end latency characterization of task communication models for automotive systems. En *Real-Time Systems* (Vol. 56, Número 3). Springer US. https://doi.org/10.1007/s11241-020-09350-3

Monot, A., Navet, N., Bavoux, B., & Simonot-Lion, F. (2010). Multicore scheduling in automotive ECUs. *Embedded Real Time Software and Systems - ERTSS 2010, May 2010, Toulouse, France.*, *2010*. https://hal.inria.fr/inria-00543179

Mosqueda-Arvizu, C.-A., Romero-González, J.-A., Córdova-Esparza, D.-M., Terven, J., Chaparro-Sánchez, R., & Rodríguez-Reséndiz, J. (2024). Logical Execution Time and Time-Division Multiple Access in Multicore Embedded Systems: A Case Study. *Algorithms*, *17*(7), 294. https://doi.org/10.3390/a17070294

Pazzaglia, P., Casini, D., Biondi, A., & Natale, M. Di. (2023). Optimizing Inter-Core Communications Under the LET Paradigm using DMA Engines. *IEEE Transactions on Computers*, *72*(1), 127–139. https://doi.org/10.1109/TC.2022.3191739

Pressman, R. (2000). Software Engineering: a Practitioner's Approach. En *Software Engineering Journal*.

Rivas, J. M., Goossens, J., Poczekajlo, X., & Paolillo, A. (2019). Implementation of memory centric scheduling for COTS multi-core real-time systems. *Leibniz International Proceedings in Informatics, LIPIcs*, *133*. https://doi.org/10.4230/LIPIcs.ECRTS.2019.7

Schoeberl, M., Sorensen, R. B., & Sparso, J. (2015). Models of Communication for Multicore Processors. *Proceedings - 2015 IEEE 18th International Symposium on Real-Time Distributed Computing Workshops, ISORCW 2015*, 9–16. https://doi.org/10.1109/ISORCW.2015.57

Soliman, M. R., Gracioli, G., Tabish, R., Pellizzoni, R., & Caccamo, M. (2019). Segment streaming for the three-phase execution model: Design and implementation. *Proceedings - Real-Time Systems Symposium*, *2019-December* (December), 260–273. https://doi.org/10.1109/RTSS46320.2019.00032

Tabish, R., Mancuso, R., Wasly, S., Pellizzoni, R., & Caccamo, M. (2019). A real-time scratchpad-centric OS with predictable inter/intra-core communication for multi-core

embedded systems. *Real-Time Systems*, *55*(4), 850–888. https://doi.org/10.1007/s11241-019-09340-0

Urbina, M. (2020). *TIMEA : Time-Triggered Message-based Multicore Architecture for AUTOSAR*. University of Siegen.

Wang, S., Li, D., Sifat, A. H., Huang, S.-Y., Deng, X., Jung, C., Williams, R., & Zeng, H. (2023). *Optimizing Logical Execution Time Model for Both Determinism and Low Latency*. http://arxiv.org/abs/2310.19699