UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

FACULTAD CIENCIAS DE LA INGENIERÍA

CARRERA SOFTWARE

INTEGRANTES:

CORONEL TOALA CRISTHIAN ALEXIS – ccoronelt2@uteq.edu.ec
LLERENA ABRIL ANGELINA JULEXY – allerenaa@uteq.edu.ec
NARANJO FLORES ANDERSON JEAMPIERE – anaranjof2@uteq.edu.ec
NIVELA SAAVEDRA JOSE RICARDO – jnivelas2@uteq.edu.ec

DOCENTE:

PHD. GUERRERO ULLOA GLEISTON CICERON

CURSO Y PARALELO:

SEGUNDO SOFTWARE “B”

GRUPO:

D

ASIGNATURA:

ARQUITECTURA DE COMPUTADORAS

TEMA:

COMPONENTES DEL COMPUTADOR

PERIODO ACADÉMICO:

SPA 2025 – 2026

ÍNDICE

FUNDAMENTO TEÓRICO

1. MEMORIA (CONCEPTOS BÁSICOS)

La memoria es un componente esencial para los sistemas computacionales, ya que es la encargada de almacenar y recuperar la información que se requiere en la realización de las tareas. Esta revisión técnica describe los tipos de memoria de las computadoras y sus características más importantes, tales como su capacidad, velocidad y jerarquía, así como la evolución reciente de la memoria. Para ello, fue sintetizada y puesta en relación la información de 5 artículos que han sido publicados en el periodo de los años 2018 a 2025, destacando así la aportación de Ryabko, Gbedawo, Ielmini, Molas y Fantini.

1.1. Introducción

La memoria es el elemento funcional central de un sistema computacional, y Boris Ryabko y Rakitskiy (2018) explicaron que el concepto de Computer Capacity que es una métrica teórica que permite medir el rendimiento de un procesador, demostrando que parámetros como el número de registros y la arquitectura de las instrucciones tiene un mayor impacto que el tamaño de la memoria caché [1].

1.2. Tipos de memoria utilizados en computadoras

Gbedawo et al. (2023) [2], distribuyen la memoria en tres niveles, como se aprecia en la siguiente lista enumerada:

• Memoria Primaria: registros, caché (L1, L2, L3), RAM.
• Memoria Secundaria: HDD, SSD, NAND Flash.
• Memoria Terciaria: almacenamiento óptico, WORM, sistemas en red.

Paolo Fantini (2025) [3], presenta una ampliación de esta visión y destacar en líneas generales con novedades como la HBM3E, y CXL que permiten diluir los límites entre memoria y almacenamiento.

1.3. Características

Las características principales son las siguientes:

- Capacidad: medida en bytes; la cual mide la cantidad de información. Según Fantini [3], NAND Flash ha evolucionado y es capaz de soportar ya más de 230 capas, lo que permite lograr densidades de terabytes para cada chip.
- Velocidad: tiempo de acceso, tiempo de escritura y tiempo de lectura. Molas et al. (2021) [4] avisan que tecnologías de última generación como PCM y RRAM pueden ofrecer tiempos de escritura de menos de 100ns.
- Volatilidad: determinar si puede retener la información en ausencia de corriente. MRAM y FeRAM son ejemplos de memorias no volátiles [4].
- Jerarquía: el orden de las distintas memorias se relaciona con su velocidad y su cercanía al CPU. Gbedawo y otros. [2] nos muestran la importancia de la jerarquía de memoria para conseguir un rendimiento alto.

1.4. La jerarquía de memoria

La jerarquía de memoria se centra en conseguir equilibrar entre velocidad, capacidad y precio. Gbedawo et al. (2023) [2] nos muestran en la *Tabla 1* y *Figure 1* la jerarquía organizada de modo que:

Tabla 1: Jerarquía de memoria

Nivel	Ciclos de acceso aproximados
Registros	1 ciclo
Caché	~10 ciclos

RAM	~100 ciclos
SSD/Flash	~1 millón de ciclos
Disco duro	~10 millones de ciclos

Ryabko y Rakitskiy [1] señalan que el número de registros influye más en el rendimiento que el tamaño de la caché, lo que apoya la relevancia de los niveles más próximos al procesador.

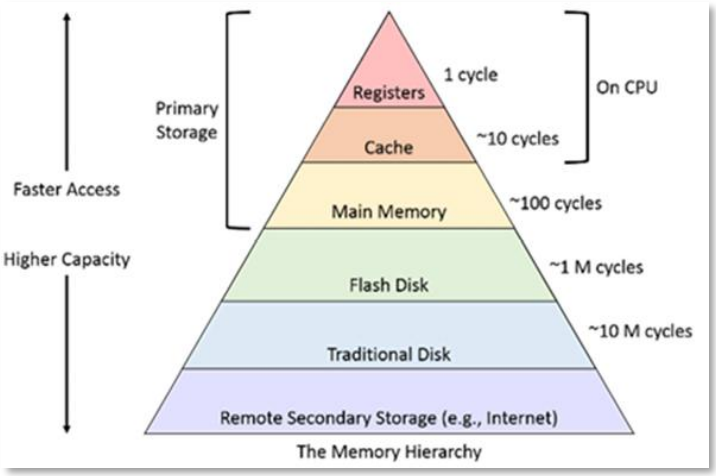


Figure 1: La jerarquía de la memoria

Para reforzar la idea de la jerarquía de memoria de un modo más visual, invitamos al lector a consultar la imagen que hemos incluido, la cual ha sido extraída del artículo Gbedawo et al. (2023) [2]. En la imagen se puede observar de qué forma están organizados los diferentes niveles de memoria como función de la rapidez de acceso a los mismos y su proximidad al procesador. Los registros y la caché son los niveles de memoria de máxima rapidez, pero no son capaces de almacenar gran cantidad de datos; en la parte superior de la imagen localizamos su representación. Cuando bajamos por la jerarquía de memoria, la capacidad de almacenamiento va creciendo, pero también la latencia desplazándose entre niveles de memoria; podemos encontrar, por ejemplo, la RAM, los SSD o los discos duros en niveles de memoria más bajos.

Esta representación gráfica sirve para describir de forma muy directa, por qué los sistemas de computación modernos están compuestos por múltiples niveles de memoria que deben ser equilibrados en rendimiento, coste y capacidad. Además, de entender por qué ciertas tecnologías emergentes han tratado de acercar el almacenamiento al procesamiento evitando la latencia de acceder a la memoria, con el impacto que ello tiene en el rendimiento de la energía.

1.5. Evolución y tendencias

Desde los años anteriores hasta la actualidad, la memoria en los equipos computacionales ha dejado de ser un simple espacio para almacenar y ha pasado a ser igualmente un elemento estratégico que determina el rendimiento y la arquitectura de los equipos. Los cinco documentos revisados coinciden en que estamos asistiendo a un cambio radical en cómo se entiende y en cómo se utiliza la memoria.

Por ejemplo, Ryabko y Rakitskiy (2018) [1]abordan y analizan la evolución de los procesadores Intel y descubren que no siempre el tamaño de la caché es el que más influye en el rendimiento, sino que lo es el número de registros disponibles y la complejidad de las instrucciones que puede gestionar el procesador, lo que cambia la forma de evaluar las mejoras arquitectónicas.

Por otro lado, Ielmini et al. (2020) [5] lanzan una propuesta subversiva, ¿y si pudiéramos realizar operaciones directamente en la memoria, sin tener que mover los datos de un sitio a otro, entre el procesador y el almacenamiento? Esta es la tradición de los sistemas de computación en memoria (IMC), es decir, una propuesta que busca eliminar el típico cuello de botella de la arquitectura de Von Neumann.

Molas y Nowak (2021) [4] proponen arquitecturas emergentes como RRAM, PCM, MRAM y FeRAM ya que, no sólo brindan mejoras en cuanto a velocidad o eficiencia energética, sino que promueven la generación de sistemas computacionales más semejantes a los humanos y por tanto propician la ejecución de determinadas aplicaciones al interior de la inteligencia artificial o la computación bioinspirada;

finalmente, Fantini (2025) [3] ofrece una visión más allá, donde enfatiza que las arquitecturas computacionales actuales no son organizadas en torno al procesador, sino a los datos, y donde la memoria juega un papel central o relevante al rendimiento del sistema, integrando de manera mucho más eficiente el procesamiento y almacenamiento bajo arquitecturas como HBM3E, CXL o chiplets híbridos.

2. MEMORIA CACHÉ:

En la jerarquía de memorias la memoria cache es el elemento que tiene la más alta velocidad incluso casi comparada con la velocidad del CPU, funciona almacenando instrucciones y variables en si misma que debido a sus velocidades y cercanía física con el procesador resulta en una respuesta más rápida al procesador preguntar directamente a esta y no a otras memorias volátiles más lentas, este proceso consiste en que el procesador pregunta por una variable o instrucción y esta pregunta primero pasa por la caché preguntando si esa información se encuentra en ella si es así se genera un acierto y se toma la información directamente de ella haciendo el proceso más rápido y eficiente, en caso de un error, la información se pide a la memoria principal y al regresar algo de esa información se instancia en la caché para que una vez que se vuelva a hacer la misma instrucción o búsqueda la recolección de esos datos será más rápido porque ya está en los datos de la caché [6].

El viaje de estas peticiones se realiza mediante el uso de buffers de rellenos que sirven para interconectar a las caches (L1, L2, L3) con el procesador y las demás memorias con menos velocidades [7] En caso de un error como antes se mencionó la pregunta se realiza a las demás memorias, este proceso se realiza cargando la pregunta en el buffer que trabaja de manera asincrónica, este tipo de trabajo genera latencia que sumado a la inefectividad de las memorias con gran peso neto se pierde una parte importante de eficiencia [7]

Para poder calcular la eficiencia de la memoria caché se toman en cuenta varios factores, entre ellos esta una fórmula matemática en donde se toman en cuenta, la cantidad de aciertos y cantidad de errores, la cantidad de aciertos se divide a la cantidad de aciertos sumada a la cantidad de errores, de forma más sencilla está dividiendo la cantidad de aciertos con la cantidad total de preguntas hechas a la caché [6]. Un ejemplo de esto es Facebook el cual informo que siguiendo estos parámetros la tasa total de aciertos positivos en su servicio es incluso superior al 95% [8]

Un ejemplo práctico del uso de la caché es en las bases de datos en donde su uso no está del todo optimizado haciendo que los centros físicos donde se guardan los datos se genere un excedente consumo de energía, aunque si existen algunas tecnologías que buscan disminuir el consumo de energía sin disminuir la efectividad y velocidad que caracteriza a la caché, estos estudios y tecnologías son escasos, debido a esa falta de interés en investigar sobre esto parece ser un tema poco resuelto o que se le da poca visibilidad [8]

Aunque no en gran escala como se daría en un proceso tan grande como lo es una base de datos en la nube, en una computadora normal la caché también consume más energía que lo que consume una memoria ROM o una memoria RAM [6]

Ya se mencionó que los datos pueden estar ubicados cerca del CPU o lejos, haciendo que los cercanos (Caché L1, L2, L3) sean los de altas velocidades, pero con poca capacidad y los lejanos o incluso no lejanos si no conectados por interconexiones sean menos veloces, pero con alto contenido neto [7]

Entonces la creación de la caché fue un intento poder romper el techo de cristal que presenta seguir la arquitectura Neumann, lo cual se consiguió de manera parcial ya que gracias a la caché las velocidades en que el procesador recibe datos de las memorias de todos los tipos, tanto HBM, GDDR, LPDDR, DRAM [7]. Aun con problemas latentes como el antes mencionado que sucedía en la nube, pero gracias a la innovación pronto la velocidad de las memorias no dará un cuello de botella a su procesador [6]

2.1. Memoria RAM

Como antes se mencionó la caché no puede trabajar sola, ahí entra la RAM o Random Access Memory que trabaja en conjunto con la caché, aunque este trabajo en conjunto genera un retardo en ciertos usos como lo pueden ser en redes neurológicas a gran escala [9]

En esencia la memoria RAM es una memoria volátil que almacena información mientras reciba energía, en la jerarquía de memorias es la siguiente después de la caché, por eso suelen trabajar en conjunto para transportar la información necesaria para el procesador, de manera básica es la que se encarga en el computador de cargar lo necesario para que funcione, cosas como aplicaciones usan memoria RAM para mantenerse funcionando [10].

El hecho de que si una computadora se apaga lo que estaba abierto se cierra pero la información en esas aplicaciones se mantiene es debido a que en ese proceso se usan 2 tipos de memorias, la RAM para mantener cargado la aplicación y cuando la pc se apaga su memoria volátil se elimina porque esta solo puede guardar información cuando recibe energía y también la memoria externa que en el caso normal de una computadora puede ser un HDD o Discos duros mecánicos, un SDD que es una unidad de estado sólido, o Nvme M.2 cada uno con diferentes velocidades pero todos un medio para guardar información no volátil y aunque para guardar la información necesitan de energía estos pueden seguir guardándola mientras no [10]

2.2. Memoria externa:

Memorias externas existen mucha variedad entre las más básicas que se encuentran en todas las computadoras como lo son los discos duros en sus diferentes presentaciones [11], que antes ya se mencionaron, pero se va a profundizar en aquí. En la actualidad los discos duros más comunes en las computadoras son las unidades de estado sólido que reemplazaron a los discos duros mecánicos debido a sus grandes velocidades y sus abaratados costos, además de los que se encuentran siempre en computadoras también se consideran memorias externas a cualquier dispositivo que pueda guardar información no volátil, como lo puede ser un USB, una tarjeta de memoria que suele abundar en celulares.

Tabla 2: Ventajas y Desventajas de HDD, SSD y M.2

	Descripción	Ventajas	Desventajas
HDD	Disco duro mecánico, después del disquete fue el predilecto para su uso en computadoras.	Por mucho tiempo fue muy barato y con grandes capacidades	Muy lento ya que al guardar información depende de un disco físico que gira.
SDD	Unidad de estado sólido, busca reemplazar al disco duro mecánico, es de lo más utilizados en la actualidad	Incluso 10 veces más rápido que el HDD	Algo más caro que el HDD
M.2	Nvme M.2 es de las versiones más actualizadas en la actualidad	Más rápido que los 2 anteriores, con la misma capacidad y mucho más pequeño.	Considerable mente más caro que los anteriores y las placas madre antiguas no lo soportan.

Normalmente en computadoras se suelen mezclar discos duros, para una mejor eficiencia a la hora de usar una computadora es recomendado el uso de los discos duros SSD y HDD, los SSD con menor capacidad de guardar información para ahorrar costos, en donde se tendrían los procesos que se necesitan realizar de manera rápida como lo es la carga del sistema operativo y en el HDD mantener la información que no es necesaria cargarla de manera constante, como fotos y videos [10]

Tabla 3: Memorias Volátiles y No Volátiles

	Concepto	Uso común	Ventajas	Desventajas
Caché	Memoria volátil ultra rápida casi tanto como un procesador que se encuentra ubicada entre el procesador y la RAM	Se usa comúnmente para acelerar la velocidad de respuestas de las computadoras	Es mucho más rápida que cualquiera de las demás memorias y consume menos energía	Es poco densa y cuando genera un error se realiza un gran consumo de energía porque se comunica con la RAM.
Memoria interna	Memoria volátil que funciona parecido a una memoria caché, pero con mucha menos velocidad y más densidad	En todas las computadoras actuales se usa	Es más densa que una memoria caché	Es mucho más lenta que la caché, genera latencia y un gran consumo de energía
Memoria externa	Memoria no volátil que puede guardar información, aunque no este recibiendo un flujo constante de energía	Para guardar información en computadoras	Es mucho más densa que las otras dos memorias siendo incluso el valor de las anteriores por 100 o por 1000 en casos como la caché	Dependiendo de la versión es mucho más lenta que las otras dos memorias, incluso en su versión más rápida no se compara a las velocidades de la RAM o de la cache

2.3. Diferencias y relaciones entre Caché, Memoria interna y Memoria externa.

Las diferencias claves entre la caché, la memoria interna y externa es la velocidad y ocupación que ejercen, primero tomare en cuenta la caché y la RAM las cuales ejercen un mismo trabajo en la computadora, pero con diferencias claves, la caché se encuentra físicamente más cerca a la PC que la RAM, haciendo que la caché la cual también es una memoria volátil sea mucho más rápida que una RAM, sin embargo, la RAM es más densa que la caché[12]

Sin embargo, aunque existen estas diferencias, la caché y la RAM suelen trabajar en conjunto, la caché recibiendo la información antes que la RAM y la RAM apoyando a la caché en ese proceso de recolección de información [12]

Ahora separando las memorias volátiles de las no volátiles, la diferencia clave entre la caché la RAM y una memoria externa como un disco duro es su funcionamiento dentro del Pc, la RAM y caché buscan ser rápidas por ello tienen menos densidad, pero los discos duros son mucho más densos pudiendo guardar mucha más información y está siendo no volátil, o sea que se mantendrá contenida, aunque el disco duro no reciba energía [12]

2.4. Importancia de la caché en el rendimiento

Como antes mencione la caché es la única memoria en el pc que puede igualar un poco la velocidad en la que trabaja el procesador y eso juntando que está más cerca del procesador que una RAM hace que la caché sea la principal memoria para mantener una velocidad alta en la computadora [6]

La caché es tan importante en el sistema que sin ella provocaría retardos gigantes en el procesamiento de todo, problemas en el impacto energético, etc. Sin la caché un programa que duraba normalmente 1 segundo en procesarse duraría entre 10 y 100 segundos y en cuestión energética puede parecer contradicción ya que antes mencione que cuando la caché da error y tiene que contactarse con la RAM esta consume mucha energía, no es contradicción por que sin la caché el procesador tendría que conectarse directamente con la RAM lo cual consumiría mucha más energía, ya que la caché consume mucha menos energía que la RAM [12]

3. ENTRADA/SALIDA (E/S): CONCEPTOS DE DISPOSITIVOS DE E/S Y SU INTERACCIÓN CON LA MEMORIA Y CPU

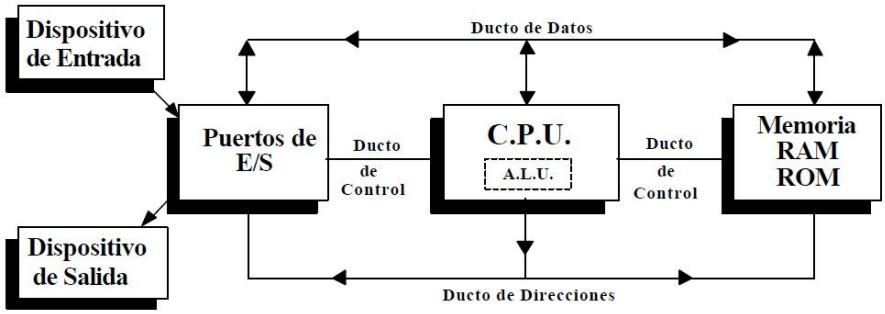


Figure 2: Diagrama general de arquitectura de computadora

La E/S (entrada/salida) en los sistemas de cómputo es la capa crucial en la conexión de los periféricos con el centro del sistema: la CPU y la memoria. Estos mecanismos no sólo consisten en la transferencia de bits, sino también en políticas de sincronización, coherencia de datos y de manejo de errores del sistema, la cual depende mucho del rendimiento final del sistema también. En los sistemas de computación de alto rendimiento tal como los sistemas de computación científica y el almacenamiento a gran escala, instrumentar y hacer visibles las rutas de E/S estima como una medida primaria para encontrar cuellos de botella y comportamientos no esperados. Neuwirth et al. (2025) tratan de plantear un marco para poder hacer la E/S explicable en HPC, subrayando la importancia de trazar flujos de datos y de hacer visibles las métricas de desempeño como los principales instrumentos de optimización de la pila completa [13].

3.1. Arquitectura general de E/S y módulos

Los módulos de E/S son los intermediarios entre los dispositivos y la memoria/CPU, llevando a cabo el controlador, la cola, las políticas de buffering, etc. La interacción de estos módulos con la jerarquía de memoria implica, decisiones sobre el tipo de mapeo (MMIO contra PMIO), la coherencia entre caches y dispositivos y la política de estructuras de colas para I/O. Xu, et al (2024) explican cómo el uso de pmem y mecanismos de caching en tránsito puede reconfigurar el diseño de módulos de E/S, afectando las latencias y la durabilidad de las operaciones de bloque mediante el uso de caches intermedias a la CPU [14].

Tabla 4: Clasificación de dispositivos de E/S

Tipo de Dispositivo	Ejemplos	Velocidad de Transferencia	Función Principal
Entrada	Teclado, Mouse	Baja (kbps)	Captura de datos

Salida	Monitor, Impresora	Media (Mbps)	Presentación de información
Entrada/Salida	Disco Duro, Pantallas Táctiles	Alta (Gbps)	Lectura y escritura de datos

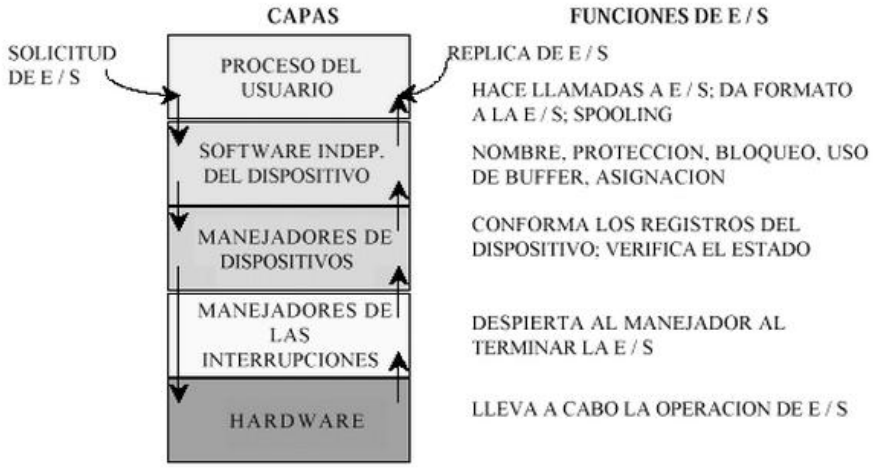


Figura 5.3: Capas del sistema de entrada / salida y las principales funciones de cada capa.

Figure 3: Esquema Global de manejadores de dispositivos

3.2. Técnicas de Entrada/Salida

3.2.1. E/S programada (Programmed I/O, PIO): fundamentos y coste

La E/S programada (PIO) supone que la CPU ejecuta instrucciones explícitas para mover datos hacia/desde los registros del dispositivo. Aunque conceptualmente simple y fácil de implementar, PIO somete a la CPU a ciclos de espera (busy-waiting) y consume recursos valiosos, lo que la vuelve ineficiente en cargas con alto ancho de banda. Investigaciones recientes muestran que, para dispositivos extremadamente rápidos y en arquitecturas con optimizaciones de caché, PIO revisitado con técnicas especializadas puede ofrecer latencias competitivas en ciertos escenarios de cola pequeña; sin embargo, el coste de CPU sigue siendo un factor limitante (Lee et al., 2022) [15]

Tabla 5: Ventajas y Desventajas de la E/S Programada

Ventajas	Desventajas
Fácil de implementar	Alto uso de CPU
Control directo sobre el hardware	Ineficiente para grandes volúmenes de datos
Bajo costo	Latencia alta

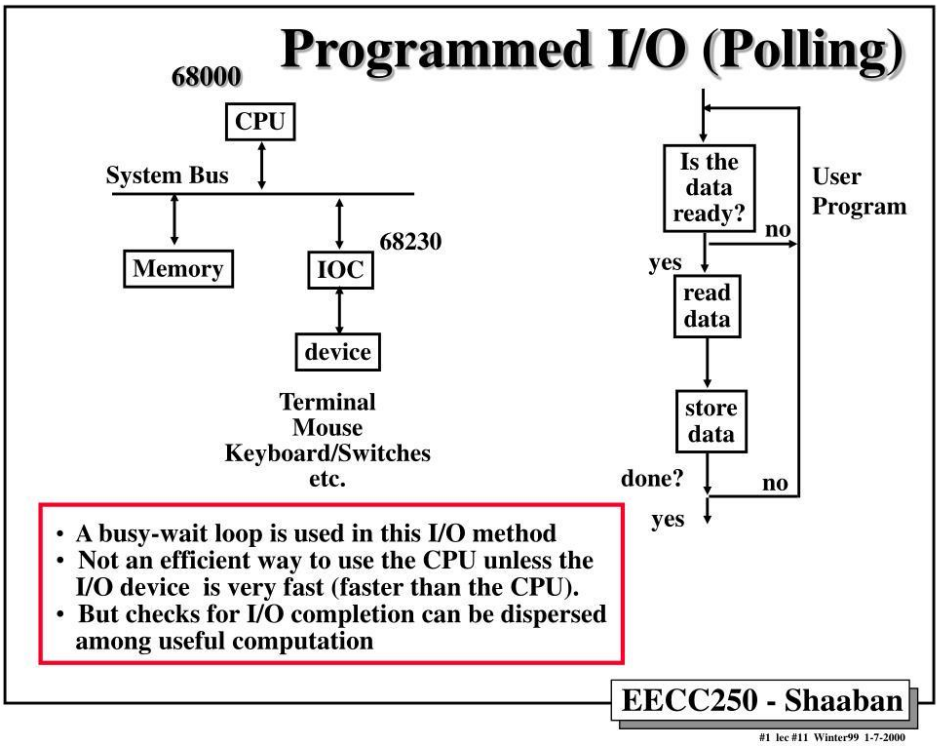


Figure 4: Ciclo de E/S programada (Polling)

3.2.2. E/S mediante interrupciones: mecanismo y complejidad

La E/S con interrupciones evita que la CPU tenga que hacer reinicio continuo porque los dispositivos informan de forma activa cuando ocurre un evento relevante. La carga se encuentra, pues, en la gestión de la interrupción, del contexto del guardado/restaurado y de los ISRs. En virtualización o en entornos de alta densidad de dispositivos la latencia por la conmutación de contexto ha sido interesante; Seo et al. (2024) evidencian cómo la E/S polled de Linux, que necesitan una larga ruta de E/S, evitan la penalización de rendimiento en usos de interrupción, pues las optimizaciones del núcleo que eliminan pasos innecesarios en la ruta de E/S [16].

Tabla 6: Comparación entre E/S Programada y E/S con Interrupciones

Característica	E/S Programada	E/S con Interrupciones
Uso de CPU	Alto	Bajo
Tiempo de respuesta	Lento	Rápido
Complejidad	Baja	Media
Ejemplos de uso	Teclado básico	Tarjeta de red

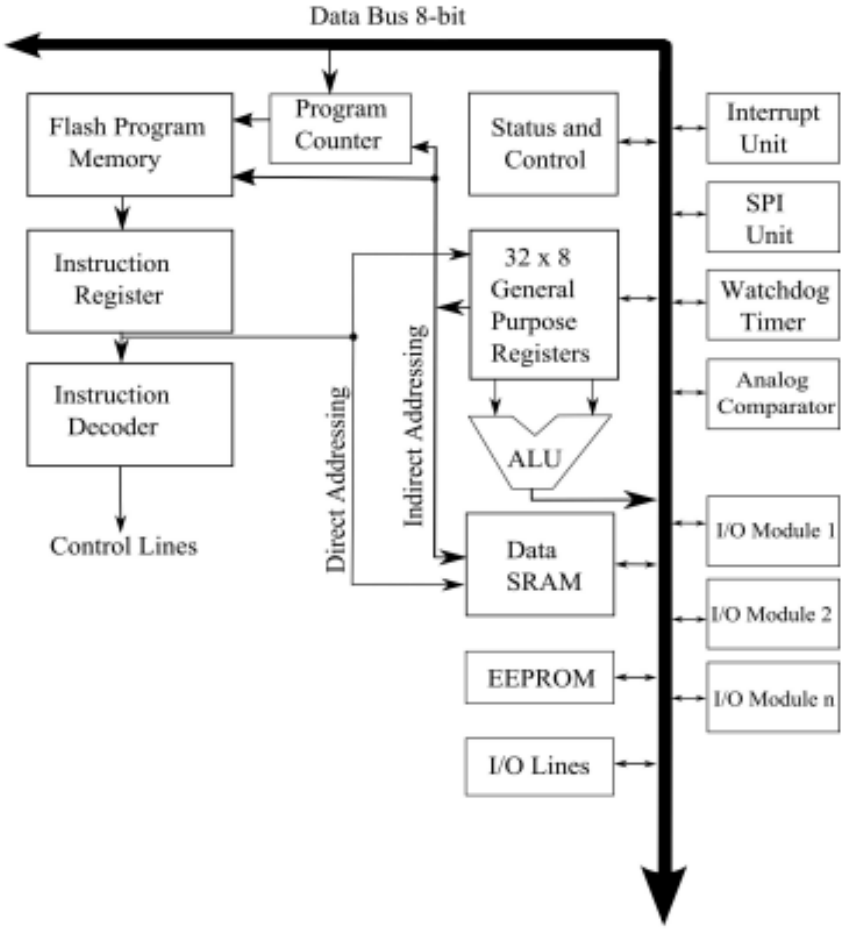


Figure 5: Manejo de interrupciones

3.2.3. Comparativa práctica: PIO vs Interrupciones

La comparación de PIO y E/S por interrupciones se ha de tener en cuenta la latencia, el uso de la CPU y la predecibilidad; en el PIO se tendrá una menor latencia en transferencias de tamaño muy pequeño, que se puede prever si es capaz de dedicarse la CPU a ello, en cambio, las interrupciones escalan mejor en entornos en multiproceso en los que la CPU tiene que atender múltiples tareas. A pesar de ello, las implementaciones del kernel han mejorado de forma que se acercan a lo mejor de los dos mundos mediante técnicas híbridas (polling con backoff). Los estudios empíricos y benchmarks muestran que la elección del mecanismo apropiado depende de la carga, de la topología de la memoria y de las capacidades del dispositivo (Lee et al., 2022) [15].

3.2.4. Acceso Directo a Memoria (DMA): concepto y flujo

El acceso directo a memoria (DMA) delega la transferencia de bloques entre dispositivo y memoria a un controlador especializado, liberando a la CPU y reduciendo overhead. El controlador DMA programa direcciones, conteos y señales de ciclo de bus para efectuar transferencias en el bus de memoria. El uso del DMA mejora la eficiencia en transferencias grandes y en operaciones de E/S continuas; además, su integración con controladores modernos permite operaciones scatter-gather y minimiza copia de datos. Li et al. (2024) realizan un estudio exhaustivo de DMA en chip y presentan optimizaciones latencia-orientadas que permiten reducir el umbral de tamaño de I/O a partir del cual el DMA es ventajoso [17].

Tabla 7: Comparación entre E/S con Interrupciones y DMA

Característica	E/S con Interrupciones	DMA
Intervención de CPU	Moderada	Mínima
Velocidad de Transferencia	Media	Alta
Complejidad del hardware	Media	Alta
Uso típico	Periféricos de velocidad media	Disco duro, tarjetas gráficas

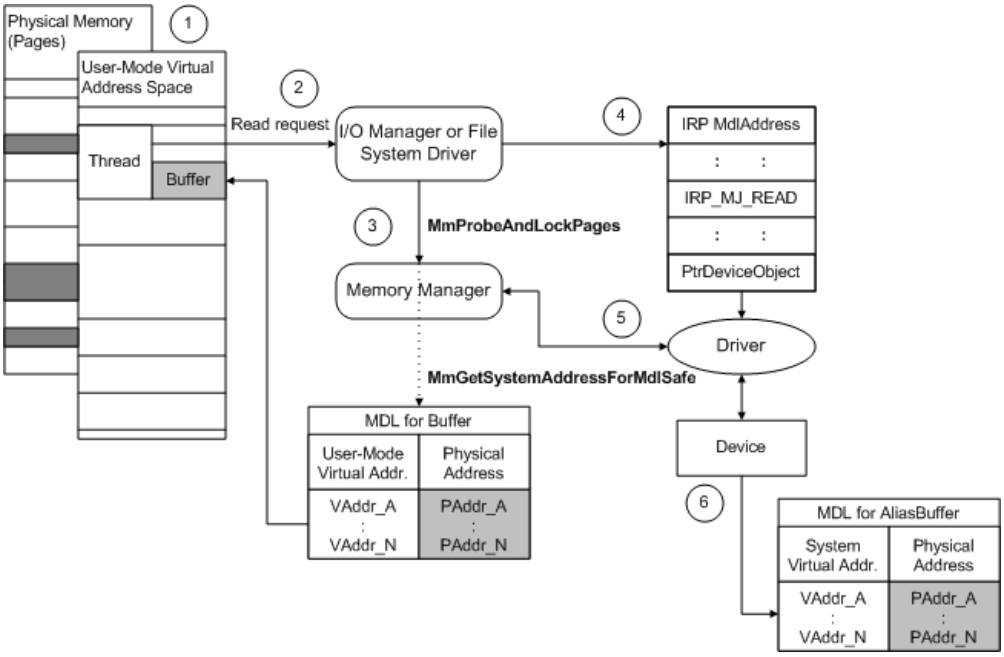


Figure 6: Diagrama de flujo DMA mostrando transferencia directa dispositivo

3.2.5. Diseño de controladores y descriptor rings para DMA

La estructura basada en descriptores y anillos en los controladores DMA resulta vital para la utilización eficiente de los sistemas de transferencia y ordenación de la transferencia. Una reestructuración contemporánea de estos anillos podría suprimir las latencias de cola y mejorar el control del acceso en los mecanismos de integración, lo que permitiría contemplar mayor flexibilidad entre la ordenación y la validación de las transferencias. Artículos recientes [18] proponen una serie de alternativas de rediseño que permiten los descriptores DMA para optimizar la concurrencia, reducir la carga de gestión y facilitar su integración con SmartNICs y aceleradores de red [18].

3.3. Problemas y técnicas avanzadas en E/S

3.3.1. Problemas de coherencia: DCA y accesos a caché

El acceso temporal directo a caché, DCA por sus siglas en inglés, y técnicas análogas permiten a dispositivos cargar los datos directamente en las cachés de la CPU y así reducir la latencia de acceso para aplicaciones, por otra parte, sensibles al tiempo de acceso. Sin embargo, DCA introduce retos con respecto a la coherencia y contención de las líneas de caché; Wang et al. (2022), tras estudiar a fondo el rendimiento y las implicaciones de DCA en multicore y redes de host evidencian que la ventaja de latencia se puede ver contrarrestada por la contención de caché y el impacto negativo en el rendimiento global de no manejarse de la forma adecuada [19].

3.3.2. E/S en entornos virtualizados y nubes IaaS

La virtualización añade capas entre el dispositivo físico y la aplicación: hipervisores, backends y el guest OS. Este apilamiento incrementa la complejidad de manejar E/S eficiente y segura.

VPRI (Guo et al., 2024) presenta un co-diseño software-hardware para manejar fallos de página de I/O (IOPFs), reduciendo la latencia al integrar mecanismos de resolución colaborativa entre dispositivo e hipervisor, lo que es clave para cargas de IaaS con acceso remoto a memoria y dispositivos virtualizados [20].

3.3.3. E/S y almacenamiento persistente: pmem y caching en tránsito

La llegada de la memoria persistente byte-adressable como arquitectura de nueva generación supuso nuevas configuraciones, ya que las operaciones de bloques tradicionales pueden ser trasladadas a pmem especializadas con tablas de traducción y capas de cache. Xu et al. (2024) informan que la cache en el tránsito (I/O transit caching, almacenamiento en el tránsito de E/S, cercano al procesador) permite una menor latencia y mejora durabilidad, junto a la descripcón de software que dirige la eviicción junto la sincronización a la memoria persistente para mantener la consistencia [14].

3.3.4. Técnicas híbridas de polling y sondeo eficiente

Las técnicas híbridas de polling funcionan con sondeo activo y utilizan mecanismos de notificación para disminuir tanto la latencia como el uso de la CPU; por ejemplo, una de estas técnicas mantiene ventanas cortas de sondeo activo en las que se chequea el dispositivo de forma constante, y si no responde, se cambia a modo de sondeo con notificación por interrupción. Lee et al. (2022) realizan un conjunto de esquemas que tienen como objetivo minimizar ciclos de CPU asociados al polling potencialmente a costa de la latencia, implementar nuevos métodos que se adapten según el patrón de carga en antiguos dispositivos de almacenamiento ultrarrápidos, pero sin sacrificar la latencia [15].

3.3.5. On-chip DMA y movimiento de datos en sistemas NVM

El rápido movimiento de datos entre DRAM, NVM y dispositivos se convierte en un cuello de botella emergente; reutilización de DMA on-chip y tuneo según el perfil de acceso se encargan de acelerar transferencias no dependientes exclusivamente de la CPU. Fastmove (Li et al., 2024) presenta un análisis del diseño de DMA on-chip y muestra ganancias en rendimiento para sistemas de almacenamiento basados en NVM, optimizando arbitraje, tamaño de la ráfaga, y rutas para el acceso que minimizan latencias pico [17].

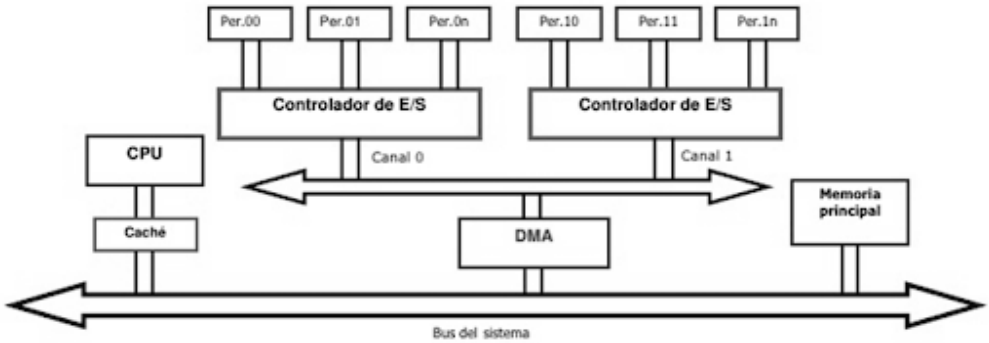


Figure 7: DMA

3.3.6. Seguridad y aislamiento en DMA: D-Box y mitigaciones

El DMA, por su capacidad de acceso directo a la memoria, presenta riesgos si periféricos maliciosos o comprometidos pueden leer/escribir regiones sensibles. D-Box [21] propone técnicas para asegurar canales DMA en sistemas embebidos mediante validación de descriptores, control de acceso y mecanismos criptográficos ligeros que preservan rendimiento sin abrir vectores de ataque que comprometan la integridad de la memoria [21].

3.3.7. SmartNICs y preprocesamiento en red con DMA

El impulso que se está dando hacia SmartNICs y funciones in-network es el que ha llevado a plantear algunos diseños donde la memoria del host y la del SmartNIC están interconectadas para permitir accesos eficientes. El trabajo In-Network Preprocessing & DMA [22] estudia cómo delegar tareas de preprocesamiento en la NIC y utilizar DMA para mover los resultados, con la ventaja de reducir la carga impuesta a la CPU y las latencias de extremo a extremo en aplicaciones de red y de almacenamiento [22].

3.3.8. Acceso a memoria en red y puentes SmartNIC-Host

Además del preprocesado, es posible implementar arquitecturas en las que el SmartNIC comparte parte del espacio de la memoria direccionable para poder disponerse de memoria para acelerar tareas distribuidas. Farooqi et al. (2025) realizan un estudio comparando las diferentes técnicas de acceso a memoria entre SmartNIC y host, ofreciendo mediciones claras en cuanto a la latencia, el ancho de banda y el coste de coherencia que permiten guiar el diseño de soluciones híbridas [23].

3.3.9. Procesamiento en memoria (PIM) y DaPPA

El paradigma PIM reduce el movimiento de datos llevando computación hacia donde residen los datos. DaPPA (Oliveira et al., 2023) presenta un framework para facilitar la programación de arquitecturas PIM (p. ej. UPMEM), automatizando particionamiento, gestión de memoria y recolección de resultados, lo que simplifica la adopción de PIM para aplicaciones data-parallel y reduce la latencia derivada del traslado DRAM↔CPU [24].

3.4. Relación con tendencias y otros paradigmas

3.4.1. Técnicas de memoria en LLMs y tendencias futuras

Si bien la E/S tradicional hace énfasis en el hardware y los dispositivos, la gestión de memoria asociada a los LLMs (Modelos de Lenguaje masivos) nos podría dejar vislumbrar la existencia de alternativas interesantes: memoria jerárquica, caché semántica o políticas de retención dinámica son ejemplos de soluciones de E/S que podrían priorizar la tratativa de datos relevantes. Trabajos recientes sobre memoria cognitiva en LLMs y sus encuestas de técnicas (Shan et al., 2025; Wu et al., 2025) hacen hincapié en expresar, indexar y recuperar estado de manera eficiente, lo que puede encontrarse en paralelo con la organización de buffers, caches y políticas de prefetching en sistemas de E/S [25][26].

El diseño de sistemas de E/S modernos exige un enfoque holístico: comprender la interacción entre dispositivos, caches, controladores DMA, SmartNICs y software de pila (kernel, hipervisor). La literatura reciente sugiere priorizar mecanismos que reduzcan movimiento innecesario de datos (PIM, on-chip DMA), proteger los canales de DMA (D-Box) y explotar arquitecturas de memoria persistente para cargas adecuadas (Caiti). Como recomendación práctica, los ingenieros deben caracterizar la carga (tamaños de I/O, latencia objetivo y patrones de acceso) y elegir el mecanismo (PIO, interrupciones, DMA, o híbridos) que ofrezca el mejor compromiso entre latencia, uso de CPU y seguridad. Para investigación futura, integrar modelos de memoria inspirados en LLMs con sistemas E/S podría abrir nuevos caminos para la priorización semántica de datos [17].

4. Interconexión con Buses

De acuerdo con Das, Palesi, Kim, Pratim Pande (2024), el concepto de la interconexión se ha vuelto más relevante que nunca. Según ellos, el reto más importante de la arquitectura de computadoras modernas ya no es la capacidad de cálculo, sino la comunicación entre los componentes, lo que hace de la interconexión un punto clave. Las interconexiones a escala de chip y de paquete son los elementos más relevantes que determinan el rendimiento, la eficiencia energética y la escalabilidad de un sistema. En el artículo se muestra cómo este papel central

de la interconexión se convierte en un patrón común en todo tipo de arquitecturas como en las de propósito general (CPU, GPU), el dominio específico (las que se utilizan en IA) y las arquitecturas de sistemas de computación cuántica. Para abordar el creciente aumento de la necesidad de comunicación, el artículo presenta las tecnologías avanzadas como los cables metálicos (la base de los enlaces cableados), la nanofotónica de alta densidad de ancho de banda y baja latencia, y los sistemas inalámbricos como las principales soluciones para los retos futuros [27].

4.1. Estructuras de buses

Según el artículo denominado “Security of Electrical, Optical and Wireless On-Chip Interconnects: A Survey” (2023) antes de que surgieran las Network-on-Chip (NoC), la topología principal para establecer la comunicación dentro de los System-on-Chip (SoC) era la arquitectura por buses. En este sentido, el artículo establece que la arquitectura de buses es sencilla, de bajo costo y opera con un conjunto de roles muy bien definidos. Aquellos tienen que ver con un maestro el cual inicia las operaciones, un esclavo que da respuestas a su solicitud, un árbitro que se encarga de controlar el uso del bus y un decodificador que envía las señales al dispositivo correspondiente. Según este estudio, aunque esta arquitectura por buses resulta ser muy útil para SoCs pequeños, a medida que los sistemas se extienden se convierten en un retraso en su comunicación. También se presentan ejemplos de estas arquitecturas, como el IBM's CoreConnect y la AMBA de ARM, así como sus vulnerabilidades frente a los ataques de canal lateral de energía [28].

4.2. Elementos de diseño

Según afirma el artículo "Multi Buses - Theory and Practical Considerations"(2020), el diseño de buses de alto rendimiento en FPGAs se basa en dos grandes conceptos, los cuales son el Multi Frame Bus (MFB) y el Multi Value Bus (MVB). El MFB está diseñado y orientado a la transferencia de varias transacciones de datos por ciclo de reloj. Este se explica a partir de cuatro atributos principales: el número de regiones que fija el número máximo de tramas por palabra, el tamaño de la región que establece el tiempo adicional necesario para tramas cortas, el tamaño del bloque que regula el tiempo de alineación y, por último, el ancho del elemento, que describe el tamaño de la unidad de datos más pequeña. El MVB queda caracterizado por el número de ítems, el cual fijará la máxima cantidad de valores transferidos, y por el ancho del ítem, que describirá el tamaño de cada valor. De este modo, en ambos casos encontramos el objetivo principal del diseño de buses que trata de llegar a una estructura de palabra de datos que permita un posicionamiento de tramas razonablemente eficiente sin comprometer la complejidad de la lógica de procesamiento y minimiza el tiempo de alineación del bus [29].

4.3. Tipos de buses

4.3.1. Bus de datos

Tal y como señalan Freund, Pirker y Dür (2024), el bus de datos cuántico es considerado como una solución flexible para las redes cuánticas. Este bus de datos cuántico lleva a cabo su operación mediante un estado de acumulación bidimensional entrelazado que actúa como su recurso clave. Con la aplicación de un esquema de medida local en un camino diagonal, es capaz de establecer varias conexiones paralelas entre distintos dispositivos de la red. Cabe señalar que un rasgo fundamental de este método consiste en que las mediciones no destruyen la estructura de entrelazamiento del estado de acumulación. Esto permite que el estado que queda pueda ser utilizado de nuevo para conexiones futuras. El artículo demuestra que esta técnica puede dar lugar a muy complejas líneas de medidas que se cruzan, giran y se fusionan, lo cual es aplicable tanto a redes de área local, como a redes de larga distancia [30].

4.3.2. Bus de direcciones

En el artículo “Design And Verification Of Apb Bridge”, se afirma que en el protocolo APB, el bus de direcciones PADDR se usa para acceder a ciertas ubicaciones y hacer lecturas y

escrituras. El protocolo APB permite que las direcciones en el bus no coincidan con la alineación con respecto al ancho del bus de datos. No obstante, un esclavo APB que recibe una solicitud con direcciones no alineadas podría no ser capaz de manejarlas, lo que conduciría a operaciones indeseadas e impredecibles. Para eliminar esta limitación, el artículo propone un puente al que denomina UTAA (Unaligned To Aligned Access), que es capaz de convertir una solicitud de dirección no alineada desde el maestro en dos solicitudes de direcciones alineadas para permitir un acceso al bus de datos. De este modo, el puente puede facilitar la reutilización de los esclavos APB que no los admiten dichas solicitudes no alineadas [31].

4.3.3. Bus de control

Según el artículo "A Novel Plane-Based Control Bus Design with Distributed Registers in 3D NAND Flash Memories", se ha propuesto un diseño de bus de control basado en planos que ahorra área. Está basado en el uso de registros distribuidos para memorias flash de tipo 3D NAND. Dicho diseño permite la reducción de los cables de enrutamiento de las señales de control hasta en un 99,47% respecto al diseño convencional. El bus de control conecta una lógica de control central con la agrupación de registros distribuidos que hay dentro de cada plano de memoria. Cada plano utiliza un bus de control con 27 señales de enrutamiento distintas para poder comunicarse. Las señales de control del bus son: un reloj de bus, señales de habilitación de lectura y de escritura, una dirección de grupo de registros de 7 bits, y buses de datos de lectura/escritura de 8 bits cada uno. El bus de control permite las operaciones: escritura/lectura de registros y direccionamiento de registros. Las operaciones de direccionamiento pueden ser continuas o bien podrían ser aleatorias. El bus de control funciona a una frecuencia de reloj de 50 MHz. En el artículo, también se incluye un esquema de plane gating basado en la dirección de grupo de registros con aproximación de 2,9 mW de potencia [32].

4.3.4. Buses internos y externos

Según el artículo “Analisis Sistem Bus USB Dan PCI Pada Organisasi Arsitektur” (2023), el bus USB es un estándar de interfaz, el cual se usa para conectar diferentes dispositivos externos a una computadora. Por otro lado, el bus PCI es un bus interno que conecta los componentes de la computadora [33].

Tabla 8: Características de buses

Característica	Bus USB (Universal Serial Bus)	Bus PCI (Peripheral Component Interconnect)
Tipos de Bus	Externo	Interno
Función Principal	Conectar diversos dispositivos externos a la computadora, como teclados, ratones, impresoras y cámaras.	Conectar componentes internos de la computadora, como tarjetas de video, tarjetas de sonido y tarjetas de red, a la placa base.
Evolución	USB 1.1 (12 Mbps), USB 2.0 (480 Mbps), USB 3.0 (5 Gbps), USB 3.1 Gen 2 (10 Gbps), USB 3.2 (20 Gbps), USB 4.0 (40 Gbps).	PCI (33 MHz, 133 MB/s), PCI-X (133 MHz, 533 MB/s), PCI Express (PCIe) con velocidades que van desde 250 MB/s por carril hasta 1 GB/s por carril y más en revisiones posteriores.
Tipo de Conexión	Comunicación en serie.	PCI y PCI-X utilizan un bus compartido, mientras

		que PCIe utiliza un enfoque de comunicación punto a punto.
Versiones de Puerto	USB Type-A (el más común), USB Type-C (más nuevo, reversible y multifuncional).	Las ranuras PCI solo aceptan tarjetas PCI. PCIe tiene diferentes tamaños de ranura (x1, x4, x8, x16)
Compatibilidad	Son compatibles con versiones más nuevas de USB, pero la velocidad de transferencia se limita a la versión más baja.	Creado por Intel y se convirtió en el estándar para sistemas Pentium

4.4. Rol en la transferencia de datos entre componentes

4.4.1. Interacción entre CPU, memoria y periféricos

Según el documento "Stop Taking the Scenic Route: the Shortest Distance Between the CPU and the NIC is MMIO" (2025), la relación entre la CPU, la memoria y los dispositivos periféricos como la NIC ha pasado a convertirse en un tema de creciente interés a causa de la oferta de enlaces de alta velocidad y la presión para interfaces de NIC eficientes para la CPU. Históricamente, la mejor manera de transportar datos de la CPU hacia la NIC ha sido por medio del acceso directo a memoria (DMA), donde el único trabajo de la CPU es iniciar la transferencia. La NIC recupera los datos de la memoria del host en forma asíncrona. Desafortunadamente, esta forma de acceso a memoria por DMA produce un acceso de memoria para cada palabra de datos, el cual es no contiguo y produce retrasos en las transferencias de datos. El artículo mencionado propone que MMIO, un tipo de E/S programada más general, supera a la implementación E/S por DMA en latencia y ancho de banda al copiar los datos directamente en los registros de la NIC. Los autores desafían la sabiduría tradicional que menosprecia el MMIO como el mecanismo más importante para mover datos y afirman que se puede conseguir un alto rendimiento de escritura con MMIO si se sueltan las restricciones del orden. Proponen un hardware eficiente para regenerar el orden en el NIC, lo que permite disfrutar las ganancias de rendimiento proveniente de las escrituras de MMIO sin orden. El estudio muestra cómo un único núcleo de CPU puede mover más de 100 Gbps de tráfico a un dispositivo por medio de MMIO con escrituras combinadas, sobrepasando la tasa de línea de la NIC, mientras que el punto de partida de las instrucciones de barrera de memoria [34].

4.4.2. Gestión del flujo de datos en sistemas multiprocesador

Tal como se menciona en el artículo "NoC-based hardware software co-design framework for dataflow thread management"(2023), la manera en que se gestionan los flujos de datos en los sistemas multiprocesador se puede entender en el marco del modelo de trabajo de un diseño de hardware y software, pero plenamente basado en la propuesta del modelo de ejecución del flujo de datos. Con esta idea se plantea implementar un sistema de gestión de hilos sobre la NoC (Red en Chip), buscando proporcionar una mejor eficiencia energética y utilización de recursos en los sistemas de muchos núcleos. El marco de trabajo que se propone opera en tres fases o pasos fundamentales: una política de distribución de hilos eficaz y rápida, la inclusión de adaptabilidad de la NoC y la exploración de una topología híbrida que combina topología de mallas 2D y anillos. Todo ello incide en que la comunicación entre los hilos en el flujo de datos es mejor y permitiendo que un hilo se ejecute solo cuando todos sus datos de entrada están disponibles, lo que es fundamental en el paradigma del flujo de datos [35].

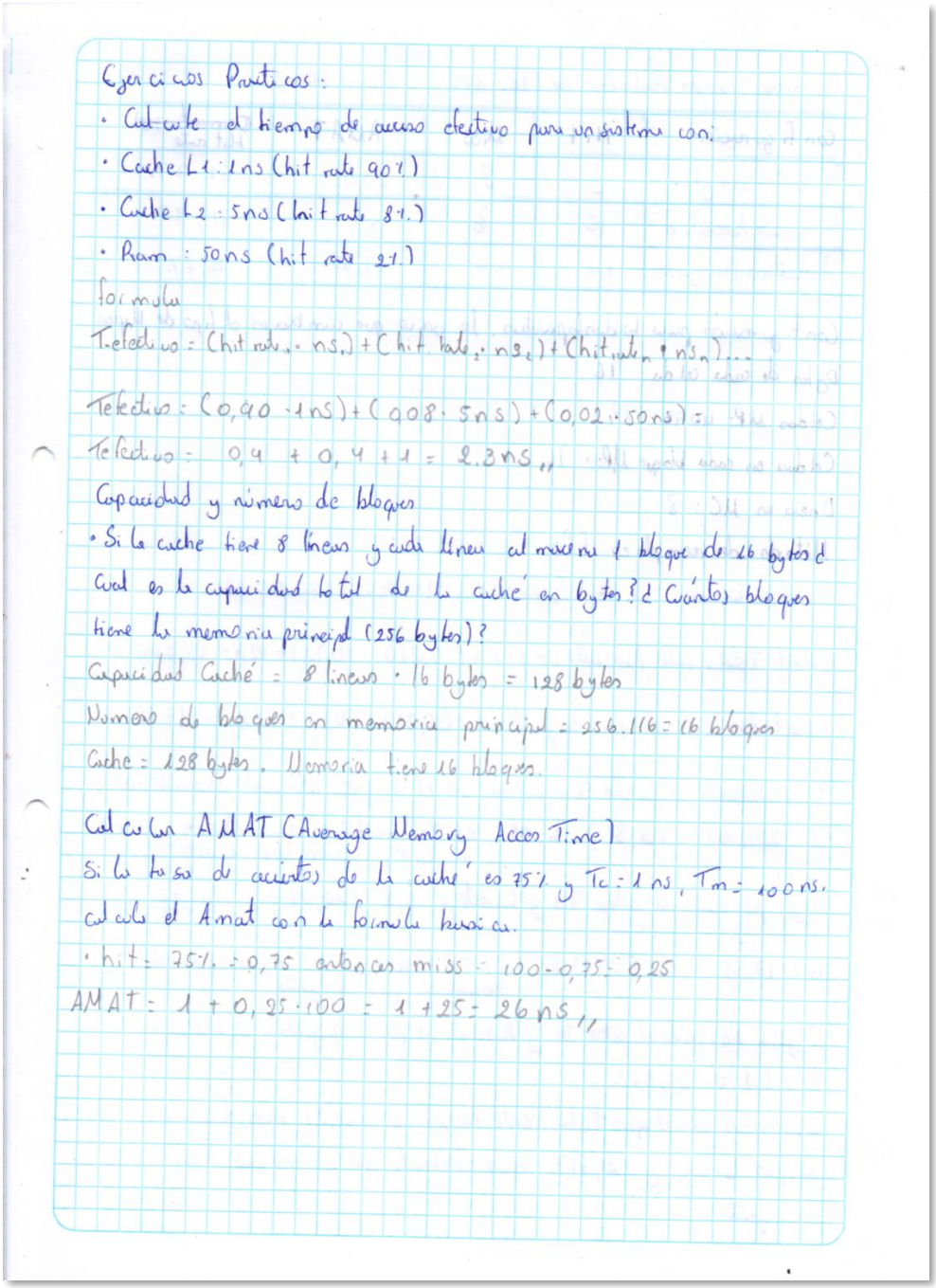
5. PROCEDIMIENTOS

5.1. Estudio de Memoria

Un ejemplo práctico de las limitaciones, usos y nuevas opciones que se han diseñado para contrarrestar las desventajas antes mencionadas de estos tipos de memoria, seria en las redes neuronales en donde se usan en conjunto la caché y la RAM, sin embargo, como antes mencione el uso de estas dos genera un gran costo de energía lo cual lleva a una perdida significativa de eficiencia en el proceso [9]

De hecho, debido a eso hay un Paper que habla sobre un método algo revolucionario donde se usan la caché, la RAM y además una memoria ROM más rápida que un disco Duro y que solo son para leer información, esta implementación que se propone hace que el consumo energético antes mencionado disminuya sin necesidad de disminuir la eficiencia del proceso [9]

5.1.1. Ejercicios prácticos y simulaciones:



Ejercicios Prácticos:
- Calcule el tiempo de acceso efectivo para un sistema con:
- Cache L1: 1ns (hit rate 90%)
- Cache L2: 5ns (hit rate 8%)
- Ram: 50ns (hit rate 2%)

formula

$T_{efectivo} = (Hit\ rate_1 \cdot ns_1) + (Hit\ rate_2 \cdot ns_2) + (Hit\ rate_n \cdot ns_n) \dots$

$T_{efectivo} = (0,90 \cdot 1ns) + (0,08 \cdot 5ns) + (0,02 \cdot 50ns) =$

$T_{efectivo} = 0,9 + 0,4 + 1 = 2,3ns$

Capacidad y número de bloques

- Si la cache tiene 8 líneas y cada línea almacena 1 bloque de 16 bytes ¿Cuál es la capacidad total de la caché en bytes? ¿Cuántos bloques tiene la memoria principal (256 bytes)?

Capacidad Caché = 8 líneas · 16 bytes = 128 bytes

Número de bloques en memoria principal = 256.116 = 16 bloques

Caché = 128 bytes. Memoria tiene 16 bloques.

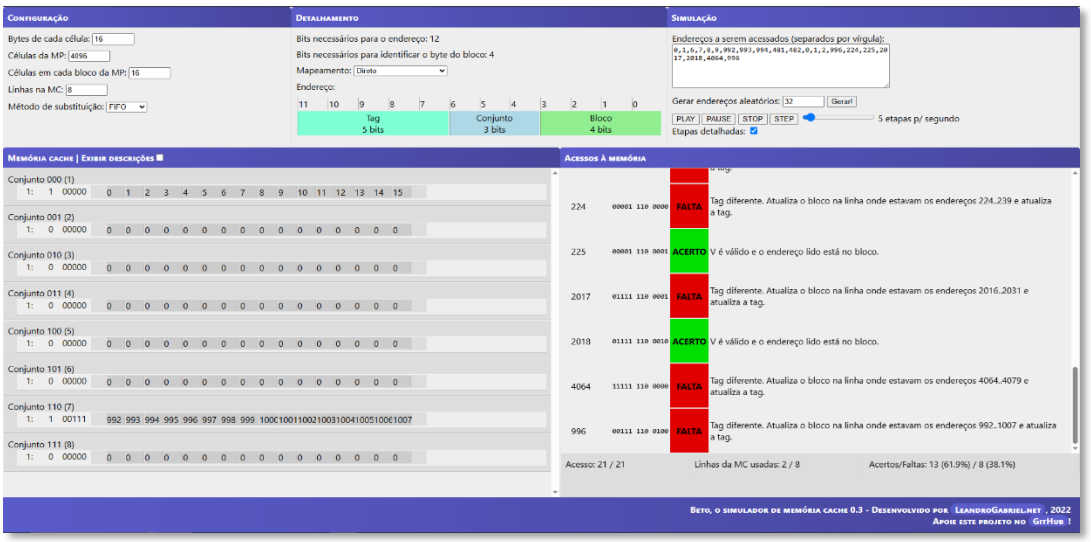
Calcular AMAT (Average Memory Access Time)

Si la tasa de aciertos de la caché es 75% y $T_c = 1ns$, $T_m = 100ns$. calcule el Amat con la formula básica.

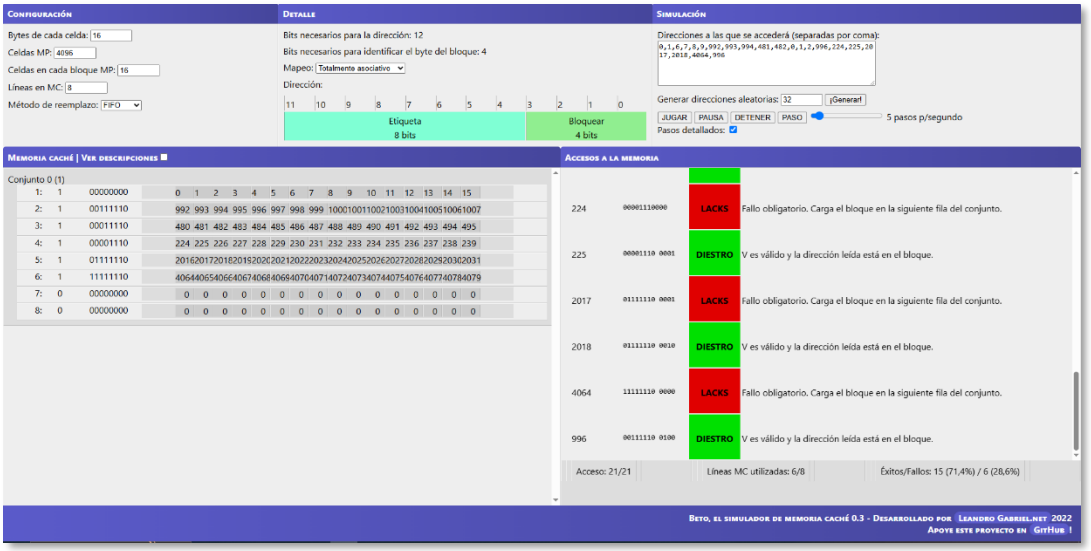
- hit: 75% = 0,75 entonces miss = 100 - 0,75 = 0,25

$AMAT = 1 + 0,25 \cdot 100 = 1 + 25 = 26ns$

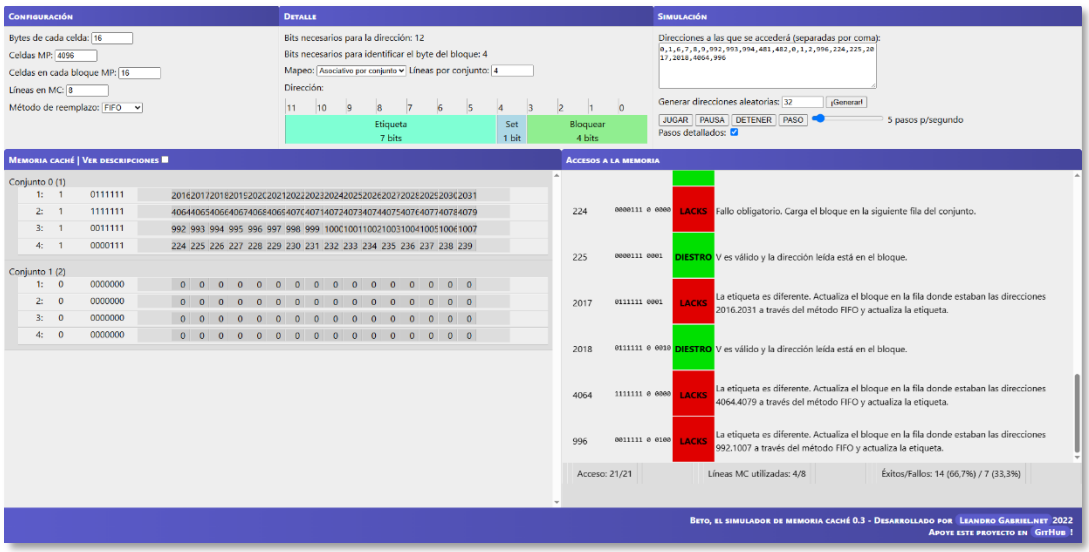
5.1.2. Mapeo Directo



5.1.3. Mapeo Totalmente Asociativo



5.1.4. Mapeo Asociativo por Conjunto:



Usando valores realistas de Tc = 1ns y Tm = 50 ns

Configuración	Hit	Miss	AMAT	~~Comentario~~ Hit rate
Directo/straight	13	8	20,05	61,9%
Totalmente Asociativo	15	6	13,8	71,4%
Asociativo por conjunto	14	7	17,65	66,7%

Configuración para todas las pruebas, lo único que cambia es el tipo de Mapeo

Bytes de cada celda: 16

Celdas MP: 4096

Celdas en cada bloque MP: 16

Líneas en MC: 8

Método de reemplazo: fifo

AMAT Directo: 1 + (0,38 · 50) = 1 + 19,05 = 20,05 ns

AMAT Totalmente Asociativo: 1 + (0,286 · 50) = 1 + 12,8 = 13,8 ns

AMAT Asociativo por conjunto: 1 + (0,333 · 50) = 1 + 16,68 = 17,65

Visualmente se puede comprobar que el mapeo totalmente asociativo.

Δ = 20,05 − 13,8 = 6,25 El Totalmente Asociativo ahorra 6,25 ns en comparación con Directo

Mejor porcentual entre TA y Directo = 6,25 / 20,05 ≈ 0,311 7 = 31,2%,

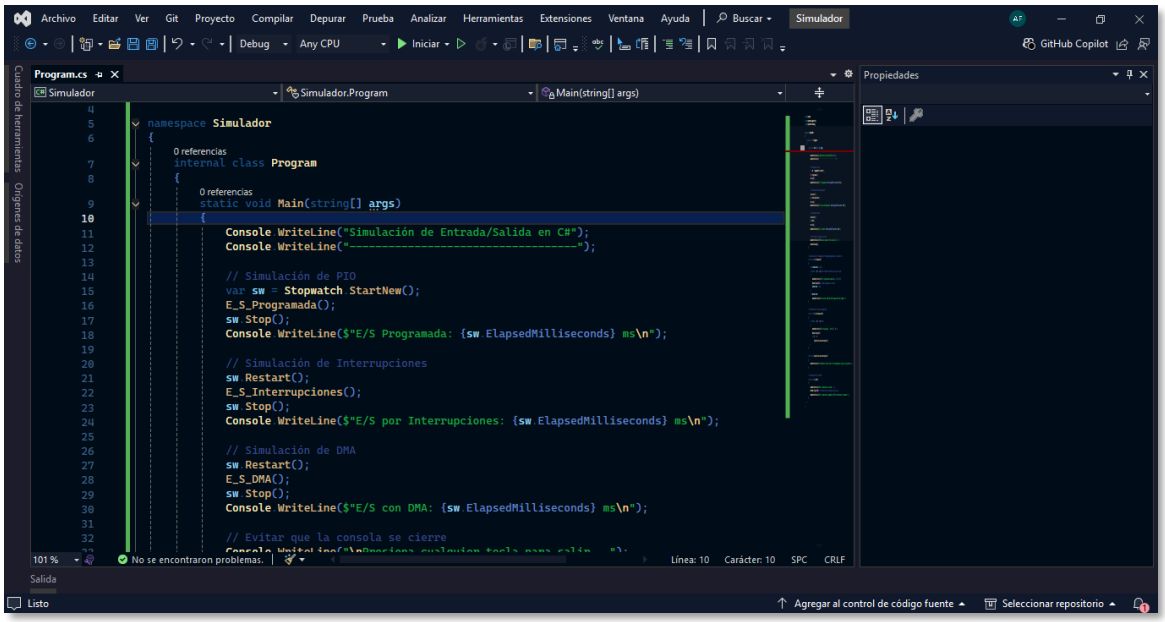
Δ = 17,65 − 13,8 = 3,85 El TA ahorra 3,85 ns en comparación con Asociativo por conjunto

Mejora Porcentual entre TA y Asociativo por conjunto = 3,85 / 17,65 ≈ 0,218 1 = 21,81%

Entonces el Mapeo Totalmente Asociativo es 31,2% más eficiente que el Mapeo Directo y 21,81% más eficiente que el mapeo Asociativo por conjunto

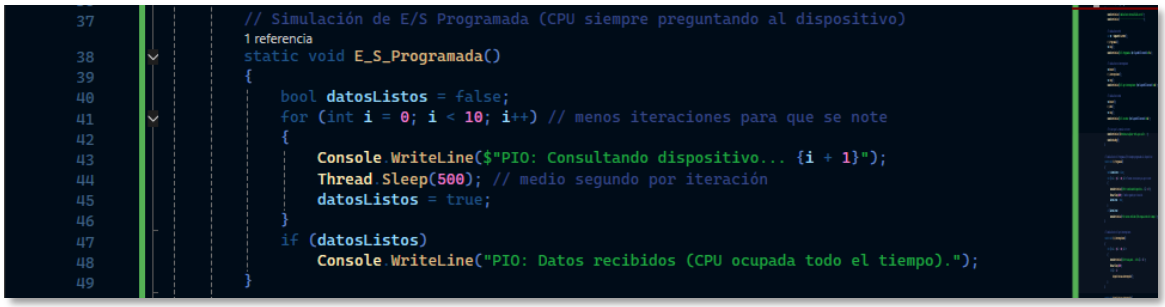
5.2. Operación de Entrada/Salida:

5.2.1. Simulación de Entrada/Salida en C#



En esta imagen se presenta el código de la simulación, en el apartado de anexos en el GitHub se encuentra el código junto a un video de la ejecución de este.

5.2.2. Entrada/Salida Programada (PIO)

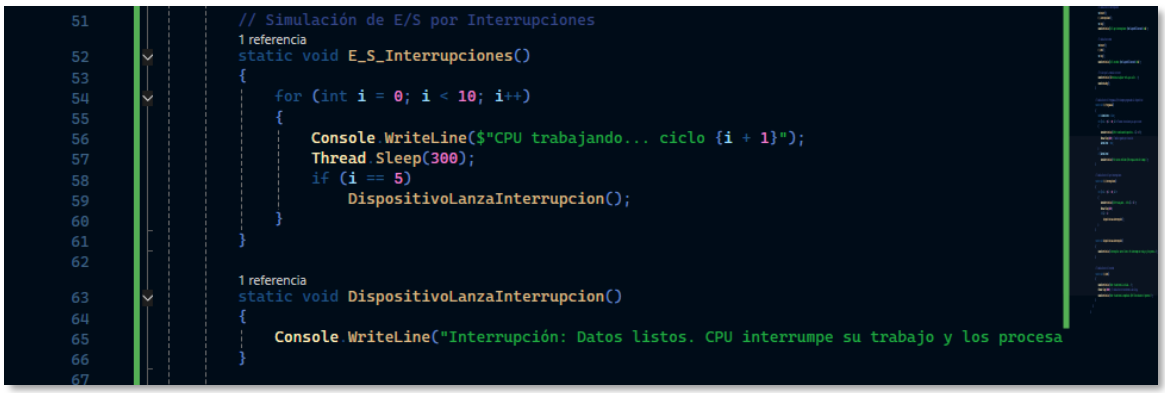


Lo que se simula en este apartado es la CPU, consulta repetidamente al dispositivo si los datos están preparados (*polling*).

Se ejecuta:

- Un bucle **`For`** de 10 iteraciones simula que la CPU pregunta repetidamente 10 veces.
- **`Theread.Sleep(500)`** pausa medio segundo cada vez, simulando la espera.
- **`datosListos`** se ponen en **`true`** para simular que eventualmente le llegan los datos.
- Al final, se imprime que recibe los datos.

5.2.3. Entrada/Salida por Interrupciones



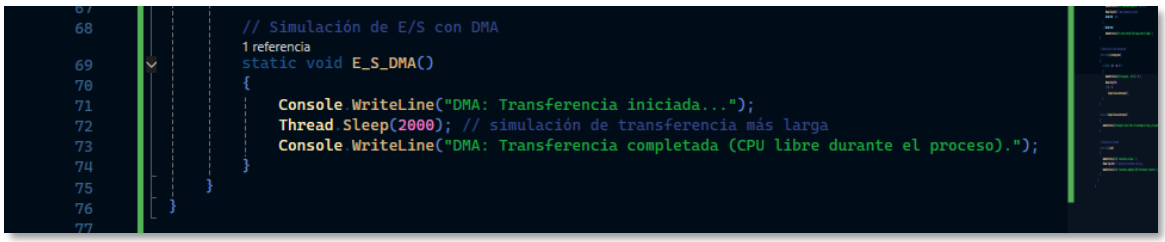
```
51      // Simulación de E/S por Interrupciones
        1 referencia
52      static void E_S_Interrupciones()
53      {
54          for (int i = 0; i < 10; i++)
55          {
56              Console.WriteLine($"CPU trabajando... ciclo {i + 1}");
57              Thread.Sleep(300);
58              if (i == 5)
59                  DispositivoLanzaInterrupcion();
60          }
61      }
62
        1 referencia
63      static void DispositivoLanzaInterrupcion()
64      {
65          Console.WriteLine("Interrupción: Datos listos. CPU interrumpe su trabajo y los procesa
66      }
67
```

Aquí la CPU hace otras tareas y solo se detiene cuando el dispositivo le “avisa” que los datos están listos.

Se ejecuta:

- El bucle que la CPU trabaja en otras cosas (**`Theread.Sleep(300)`** simula tiempo de trabajo).
- En el ciclo 6 (**`i== 5`**), se llama **`DispositivoLanzamietnointerrupcion()`**, que muestra un mensaje de que llegaron los datos.

5.2.4. E/S con DMA



```
68      // Simulación de E/S con DMA
        1 referencia
69      static void E_S_DMA()
70      {
71          Console.WriteLine("DMA: Transferencia iniciada...");
72          Thread.Sleep(2000); // simulación de transferencia más larga
73          Console.WriteLine("DMA: Transferencia completada (CPU libre durante el proceso).");
74      }
75  }
76  }
77
```

En este apartado simula un controlador DMA que transfiere datos directamente entre memoria y el dispositivo, sin la intervención del CPU.

Se ejecuta:

- La impresión de transferencia
- **`Theread.Sleep(2000)`** que simula que el proceso tarda 2 segundos.
- La transferencia que termina y que la CPU estuvo libre mientras tanto.

5.3. Diseño de Buses

5.3.1. Realizar diagramas que representen la estructura y función de un bus.

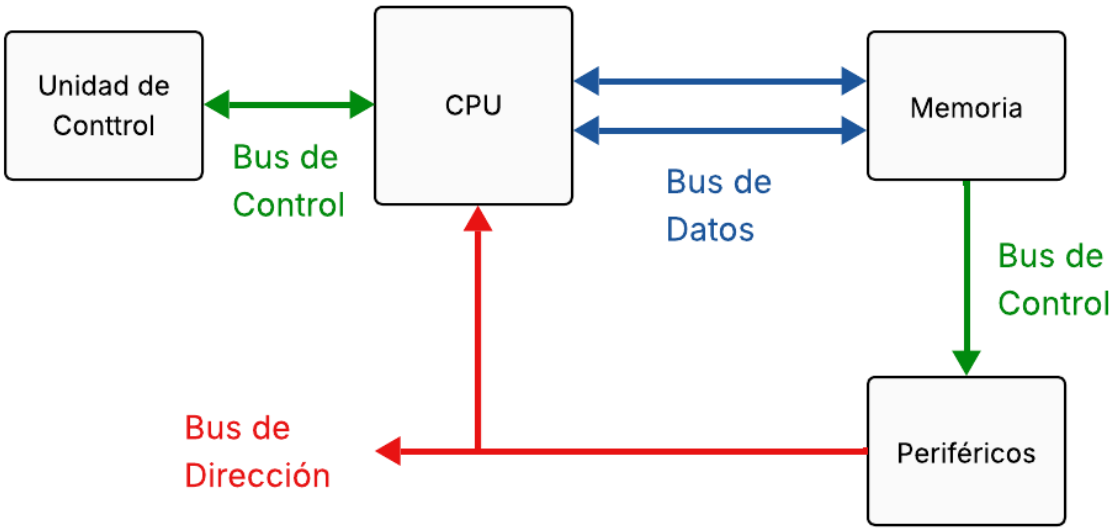


Figure 8. Estructura y función de un bus

Bus de Datos: Transfiere información entre CPU, memoria y periféricos. Es bidireccional.

Bus de Dirección: Indica la dirección de memoria o del periférico al que acceder. Es unidireccional desde la CPU.

Bus de Control: Coordina las operaciones mediante señales como lectura, escritura, interrupción y reloj.

La función de un bus es permitir la comunicación ordenada de la CPU y la memoria.

5.3.2. Identificar y describir los diferentes tipos de buses y su impacto en la velocidad de transferencia.

Tabla 9: Tipos de buses

Tipo de bus	Descripción	Impacto en velocidad
Interno	Une partes dentro de la CPU.	Muy rápido.
Externo	Une CPU con memoria y periféricos.	Más lento que el interno.
Datos	Lleva la información.	Más líneas = más bits a la vez.
Direcciones	Indica dónde leer o escribir.	Más líneas = más lugares posibles.
Control	Envía órdenes y señales.	Afecta coordinación y eficiencia.
Paralelo	Varios bits al mismo tiempo.	Rápido en corto alcance, problemas en largas distancias.
Serie	Bits uno por uno.	Más lento, pero estable en largas distancias.

REFERENCIAS

[1] B. Ryabko and A. Rakitskiy, “Investigation of the Processors Evolution Using the Computer Capacity,” in *2018 International Symposium on Information Theory and Its Applications (ISITA)*, IEEE, Oct. 2018, pp. 404–408. doi: 10.23919/ISITA.2018.8664322.

[2] V. Worlanyo Gbedawo, G. Agyeman Owusu, C. Komla Ankah, and M. Ibrahim Daabo, “An Overview of Computer Memory Systems and Emerging Trends,” *American Journal of Electrical and Computer Engineering*, Oct. 2023, doi: 10.11648/j.ajece.20230702.11.

[3] P. Fantini, “Memory technology enabling future computing systems,” *APL Machine Learning*, vol. 3, no. 2, Jun. 2025, doi: 10.1063/5.0253063.

[4] G. Molas and E. Nowak, “Advances in Emerging Memory Technologies: From Data Storage to Artificial Intelligence,” *Applied Sciences*, vol. 11, no. 23, p. 11254, Nov. 2021, doi: 10.3390/app112311254.

[5] D. Ielmini and G. Pedretti, “Device and Circuit Architectures for In-Memory Computing,” *Advanced Intelligent Systems*, vol. 2, no. 7, Jul. 2020, doi: 10.1002/aisy.202000040.

[6] A. Alsharef, S. Garg, S. Zahra, P. Jain, and G. Gupta, *Cache Memory: An Analysis on Performance Issues*. 2021. doi: 10.1109/INDIACom51348.2021.00033.

[7] F. Mahling, M. Weisgut, and T. Rabl, “Fetch Me If You Can: Evaluating CPU Cache Prefetching and Its Reliability on High Latency Memory,” in *Proceedings of the 21st International Workshop on Data Management on New Hardware*, New York, NY, USA: ACM, Jun. 2025, pp. 1–9. doi: 10.1145/3736227.3736231.

[8] S. Li, S. Wang, F. Yang, S. Hu, F. Saremi, and T. Abdelzaher, “Proteus: Power Proportional Memory Cache Cluster in Data Centers,” in *2013 IEEE 33rd International Conference on Distributed Computing Systems*, IEEE, Jul. 2013, pp. 73–82. doi: 10.1109/ICDCS.2013.50.

[9] Y. Chen *et al.*, “YOLoC,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, New York, NY, USA: ACM, Jul. 2022, pp. 1093–1098. doi: 10.1145/3489517.3530576.

[10] M. He *et al.*, “Newton: A DRAM-maker’s Accelerator-in-Memory (AiM) Architecture for Machine Learning,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, Oct. 2020, pp. 372–385. doi: 10.1109/MICRO50266.2020.00040.

[11] Y. Chen *et al.*, “Hidden-ROM,” in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, New York, NY, USA: ACM, Oct. 2022, pp. 1–9. doi: 10.1145/3508352.3549335.

[12] S. Yuvaraj, D. Padmanaban, G. PraveenKumar, S. Sahu, M. Umida, and R. Yokeshwaran, “Performance Analysis Of SRAM and Dram in Low Power Application,” *E3S Web of Conferences*, vol. 399, p. 01014, Jul. 2023, doi: 10.1051/e3sconf/202339901014.

[13] S. Neuwirth, H. Devarajan, C. Wang, and J. Lofstead, “XIO: Toward eXplainable I/O for HPC Systems,” in *Proceedings of the 37th International Conference on Scalable Scientific Data Management*, New York, NY, USA: ACM, Jun. 2025, pp. 1–6. doi: 10.1145/3733723.3733741.

[14] Q. Xu, Q. Jiang, and C. Wang, “Caiti: I/O transit caching for persistent memory-based block device,” *Journal of Systems Architecture*, vol. 150, p. 103109, May 2024, doi: 10.1016/j.sysarc.2024.103109.

[15] G. Lee, S. Shin, and J. Jeong, “Efficient hybrid polling for ultra-low latency storage devices,” *Journal of Systems Architecture*, vol. 122, p. 102338, Jan. 2022, doi: 10.1016/j.sysarc.2021.102338.

[16] D. Seo, Y. Joo, and N. Dutt, “Improving Virtualized I/O Performance by Expanding the Polled I/O Path of Linux,” in *Proceedings of the 16th ACM Workshop on Hot Topics in*

Storage and File Systems, New York, NY, USA: ACM, Jul. 2024, pp. 31–37. doi: 10.1145/3655038.3665944.

[17] J. Li *et al.*, "Fastmove: A Comprehensive Study of On-Chip DMA and its Demonstration for Accelerating Data Movement in NVM-based Storage Systems," *ACM Transactions on Storage*, vol. 20, no. 3, pp. 1–30, Aug. 2024, doi: 10.1145/3656477.

[18] A. Ruzhanskaia, P. Xu, D. Cock, and T. Roscoe, "Rethinking Programmed I/O for Fast Devices, Cheap Cores, and Coherent Interconnects," Apr. 2025.

[19] M. Wang, M. Xu, and J. Wu, "Understanding I/O Direct Cache Access Performance for End Host Networking," *ACM SIGMETRICS Performance Evaluation Review*, vol. 50, no. 1, pp. 5–6, Jun. 2022, doi: 10.1145/3547353.3522662.

[20] K. Guo *et al.*, "VPRI: Efficient I/O Page Fault Handling via Software-Hardware Co-Design for IaaS Clouds," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, New York, NY, USA: ACM, Nov. 2024, pp. 541–557. doi: 10.1145/3694715.3695957.

[21] A. Mera, Y. H. Chen, R. Sun, E. Kirda, and L. Lu, "D-Box: DMA-enabled Compartmentalization for Embedded Applications," Jan. 2022, doi: 10.14722/ndss.2022.24053.

[22] Y. Zhu, W. Jiang, and G. Alonso, "Multi-Tenant SmartNICs for In-Network Preprocessing of Recommender Systems," Jan. 2025.

[23] M. Z. Farooqi, M. Hemmatpour, and T. H. Larsen, "In-Network Memory Access: Bridging SmartNIC and Host Memory," Jul. 2025.

[24] G. F. Oliveira *et al.*, "DaPPA: A Data-Parallel Programming Framework for Processing-in-Memory Architectures," Apr. 2025.

[25] L. Shan, S. Luo, Z. Zhu, Y. Yuan, and Y. Wu, "Cognitive Memory in Large Language Models," Apr. 2025.

[26] Y. Wu *et al.*, "From Human Memory to AI Memory: A Survey on Memory Mechanisms in the Era of LLMs," Apr. 2025.

[27] A. Das, M. Palesi, J. Kim, and P. Pratim Pande, "Chip and Package-Scale Interconnects for General-Purpose, Domain-Specific, and Quantum Computing Systems—Overview, Challenges, and Opportunities," *IEEE J Emerg Sel Top Circuits Syst*, vol. 14, no. 3, pp. 354–370, Sep. 2024, doi: 10.1109/JETCAS.2024.3445829.

[28] H. Weerasena and P. Mishra, "Security of Electrical, Optical and Wireless On-Chip Interconnects: A Survey," Sep. 2023.

[29] L. Kekely, J. Cabal, V. Pus, and J. Korenek, "Multi Buses: Theory and Practical Considerations of Data Bus Width Scaling in FPGAs," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, IEEE, Aug. 2020, pp. 49–56. doi: 10.1109/DSD51259.2020.00020.

[30] J. Freund, A. Pirker, and W. Dür, "Flexible quantum data bus for quantum networks," *Phys Rev Res*, vol. 6, no. 3, p. 033267, Sep. 2024, doi: 10.1103/PhysRevResearch.6.033267.

[31] Krishnakanth Katteri Mahadeva Murthy, "DESIGN AND VERIFICATION OF APB BRIDGE," *International Research Journal of Modernization in Engineering Technology and Science*, Sep. 2022, doi: 10.56726/IRJMETS30019.

[32] H. CAO, Q. WANG, F. LIU, and Z. HUO, "A Novel Plane-Based Control Bus Design with Distributed Registers in 3D NAND Flash Memories," *Chinese Journal of Electronics*, vol. 31, no. 4, pp. 647–651, Jul. 2022, doi: 10.1049/cje.2021.00.283.

[33] Putri Andini Maulana, Muhammad Yusuf Habibi, Agus Gilang Hermawan, and Didik Aribowo, "Analisis Sistem Bus USB Dan PCI Pada Organisasi Arsitektur Komputer," *Jurnal Kendali Teknik dan Sains*, vol. 1, no. 4, pp. 115–122, Oct. 2023, doi: 10.59581/jkts-widyakarya.v1i4.1463.

[34] W. S. Liew, M. A. Rahaman, J. McMahon, R. Stutsman, and V. Nagarajan, "Stop Taking the Scenic Route: the Shortest Distance Between the CPU and the NIC is MMIO," in

Proceedings of the Workshop on Hot Topics in Operating Systems, New York, NY, USA: ACM, May 2025, pp. 144–150. doi: 10.1145/3713082.3730389.

[35] S. Mazumdar, A. Scionti, S. Zuckerman, and A. Portero, "NoC-based hardware software co-design framework for dataflow thread management," *J Supercomput*, vol. 79, no. 16, pp. 17983–18020, Nov. 2023, doi: 10.1007/s11227-023-05335-8.

6. ANEXOS

ENLACE DE GITHUB:
https://github.com/AnderJM/ArqComp-Grupo-D-Unid-4