UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

CIENCIAS DE LA INGENIERÍA

INGENIERÍA EN SOFTWARE

ECUADOR

TEMA:

COMPONENTES DEL COMPUTADOR

AUTORES:

ALMAGRO INTRIAGO LAINER PATRICIO

FUERTES ARRAES EDSON DANIEL

PEREZ RUIZ CARLOS ANDRES

ROCHINA ROCHINA FREDDY DAVID

DOCENTES:

ING. GUERRERO ULLOA GLEISTON CICERON

CURSO/PARALELO:

SEGUNDO SOFTWARE “B”

SPA 2025 – 2026

1 OBJETIVO

Conocer y analizar los principales componentes de un computador, incluyendo diferentes tipos de memoria, dispositivos de
entrada/salida y la interconexión mediante buses, para comprender su función y comportamiento en el sistema computacional.

2 COMPONENTES DEL COMPUTADOR

3 TEMA 1: MEMORIA– CONCEPTOS BÁSICOS

3.1 Definición y jerarquía de memoria:

Según *(Vaithianathan, 2025)* la memoria en computación es un componente fundamental que permite guardar datos e instrucciones que el procesador necesita para realizar sus tareas diarias, facilitando así el funcionamiento eficiente del sistema. Es importante entender que su función principal es almacenar información temporalmente para que pueda ser accedida rápidamente y optimizar el procesamiento ya que sin la memoria la ejecución de programas sería mucho más lenta y complicada.[1]

A diferencia del almacenamiento y del disco duro, que guarda datos de forma permanente, la memoria RAM brinda una capacidad limitada, pero de acceso veloz mientras que el disco duro o SSD sirven para guardar la información a largo plazo incluso cuando la máquina está apagada. Por esta razón en la arquitectura de los sistemas de cómputo y es muy esencial distinguir entre la memoria y el que se usa para almacenamiento temporal durante la operación y el almacenamiento además que se encarga de mantener los datos de forma permanente y seguros.

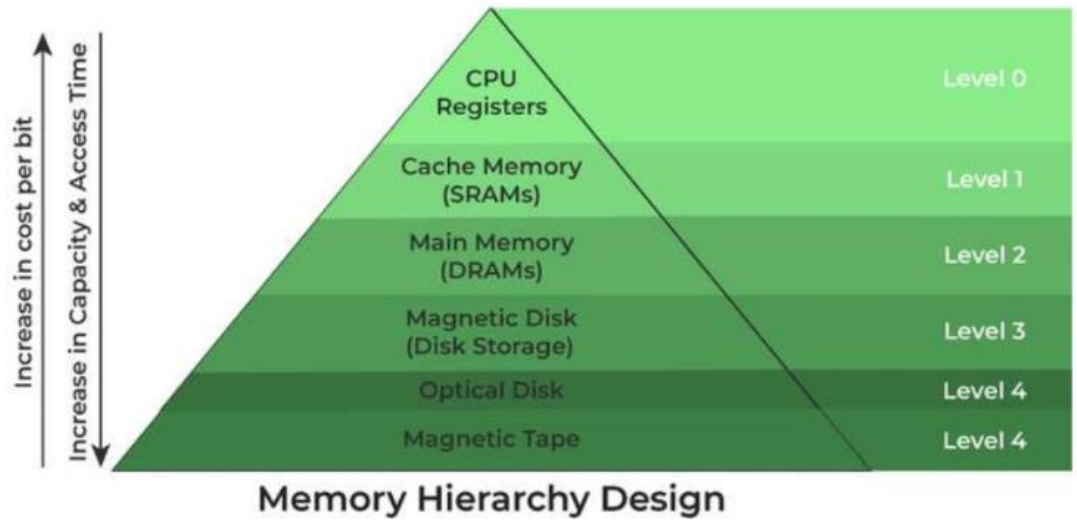


Figura 1. Pirámide de diseño de la jerarquía de memoria

3.2 Tipos de memoria (RAM, ROM, volátil vs no volátil):

Según ***(Liu et al. 2021)*** las memorias pueden ser volátiles o no volátiles, donde las RAM (Memoria de Acceso Aleatorio) y la RAM dinámica (DRAM) son ejemplos de memorias volátiles, las cuales requieren energía constante para mantener la información. En cambio, las memorias no volátiles, como las tecnologías NVMM (Memoria Principal No Volátil), conservan los datos incluso cuando se apaga el equipo, ofreciendo ventajas significativas en términos de persistencia y eficiencia energética. Estas memorias no solo permiten reducir los tiempos de recuperación ante fallos, sino que también facilitan una gestión más efectiva en arquitecturas híbridas [2].

Además, las NVMM son más rápidas que los discos duros tradicionales y tienen un menor consumo energético en comparación con las memorias Flash, lo que las convierte en una opción prometedora para aplicaciones de Big data, seguridad y sistemas de almacenamiento avanzado. En definitiva, las NVMM están revolucionando el campo de la memoria, permitiendo nuevas funciones y diseños que mejoran el rendimiento y la sostenibilidad de los sistemas informáticos.

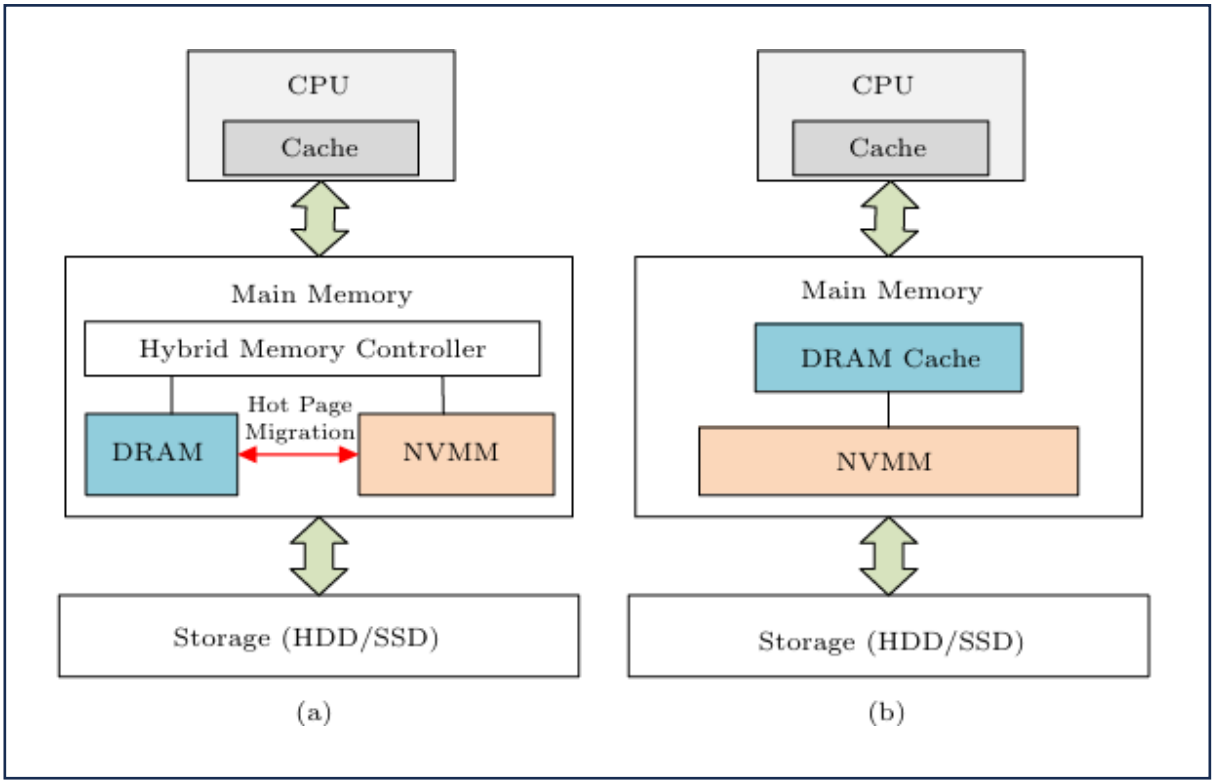


Fig.2. Hybrid memory architectures. (a) Horizontal at-addressable hybrid memory architecture. (b) Hierarchical hybrid memory architecture.

1. **Clasificación de la memoria.**
    - Memoria principal (RAM, ROM).
    - Memoria secundaria (disco duro, SSD).
    - Memoria volátil vs no volátil.
2. **Características principals.**
    - Capacidad (medida en bytes, KB, MB, GB, TB).
    - Velocidad de accesso.
    - Costo por bit almacenado.
3. **Tipos de memoria.**
    - RAM (DRAM, SRAM).
    - ROM (PROM, EPROM, EEPROM).
    - Memoria virtual.
4. **Funcionamiento básico.**
    - Cómo la CPU usa la memoria para ejecutar programas.
    - Ciclo de lectura y escritura.

4 TEMA 2: MEMORIA CACHÉ

4.1 Definición de memoria cache:

La memoria caché según *(Oluwatosin Abayomi et al. 2020*) es una memoria ultrarrápida que se encuentra entre la CPU y la RAM y su función principal es acelerar el acceso a los datos que la CPU necesita constantemente. La existencia de la caché surge porque acceder directamente a la RAM sería mucho más lento, así que la caché guarda las instrucciones y datos más utilizados para que la CPU pueda acceder a ellos rápidamente [3].

Estos datos almacenados en la caché pueden estar en diferentes niveles, como L1, L2 y L3, siendo L1 la más rápida y cercana a la CPU, pero con menor capacidad mientras que L3 es más grande pero un poco más lenta. Cuando la CPU busca un dato primero consulta la caché si lo encuentra, eso se llama un hit y el acceso es muy rápido, pero si no se produce una miss y se busca en la memoria más lenta lo que demora más y reduce el rendimiento.

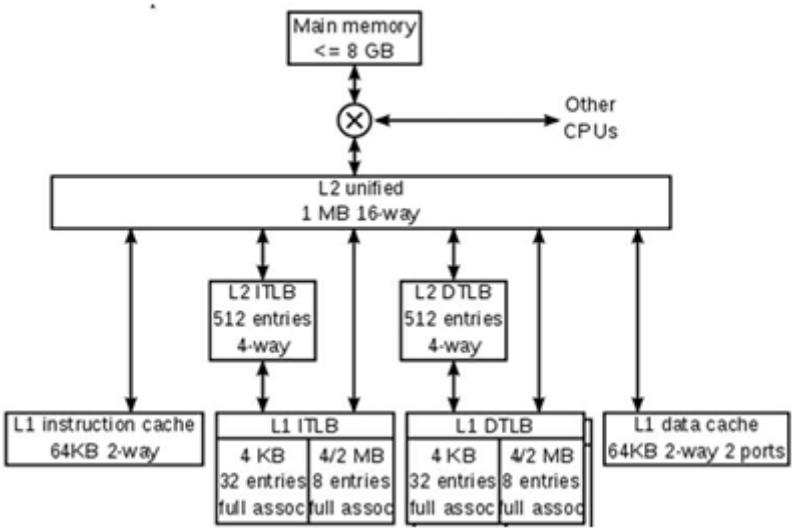


Figura 3. Jerarquía de caché del núcleo K8 de la CPU AMD Athlon 64.

1. **Ubicación y funcionamiento**
    - Dónde está (integrada en la CPU o en la placa base).
    - Cómo guarda datos que la CPU usa con frecuencia.
2. **Tipos de caché**
    - Nivel 1 (L1), Nivel 2 (L2) y Nivel 3 (L3).
    - Diferencias de velocidad, tamaño y función.
3. **Importancia y beneficios**
    - Mejora del rendimiento del sistema.
    - Ejemplos prácticos (cargar más rápido un programa usado).
4. **Relación con la memoria principal**
    - Cómo decide la CPU qué datos guardar en caché.
    - Qué pasa cuando el dato no está en la caché (fallo de caché).

La memoria caché actúa como un puente eficiente que reduce el tiempo de espera en el procesamiento de datos al mantener almacenados los datos más usados cerca del procesador.

4.2 Optimización y políticas de caché:

Según *(Jamet et al. 2020)* la diferencia en las latencias entre la caché de último nivel (LLC) y la memoria principal ha hecho necesario desarrollar políticas eficientes para gestionar la caché, ya que así se puede mejorar el rendimiento del sistema. Aunque

muchos estudios se han enfocado en perfeccionar el método de sustitución LRU, estas investigaciones generalmente evalúan solo ciertas cargas de trabajo, como las de las pruebas SPEC CPU 2006 y 2017, que no representan toda la variedad de trabajos actuales en HPC.[4]

En este artículo, se analizan diferentes cargas de trabajo, incluyendo procesamiento de grafos, ciencia e industria, en políticas modernas de sustitución LLC como MPPP, Glider, Hawkeye, SHiP, DRRIP y SRRIP, y se observa que las políticas superan a LRU en las pruebas tradicionales, no logran captar los patrones complejos de acceso en los trabajos de HPC y Big Data. Por eso, además, presentan dos nuevas políticas derivadas de MPPP, como MS-MPPPB, que emplea múltiples muestreadores para mejorar aún más el rendimiento y mejorar el desempeño general del sistema.

4.3 Reducción del movimiento de datos y nuevas arquitecturas:

Las jerarquías tradicionales según el artículo *(ASPLOS XXV: Twenty-Fifth Intel 2020)* de memoria obligan a realizar cálculos en lugares alejados de los datos necesarios, lo que genera ineficiencias en el procesamiento. Además, los enfoques convencionales no aprovechan bien la localidad de los datos, limitando la eficiencia. Por eso, proponemos Memory Services, un modelo de programación flexible que permite centrar los cálculos en los datos a lo largo de toda la jerarquía de memoria, optimizando su uso [5].

En este contexto se ha diseñamos y evaluamos Livia, una arquitectura que programa dinámicamente las tareas y datos en la ubicación ideal para reducir el movimiento de datos y mejorar el rendimiento. Gracias a Livia, las cargas de trabajo irregulares pueden acelerarse hasta 2,4 veces y reducir el consumo energético entre 1,2 y 4,7 veces, con una mínima sobrecarga de área en sistemas multinúcleo.

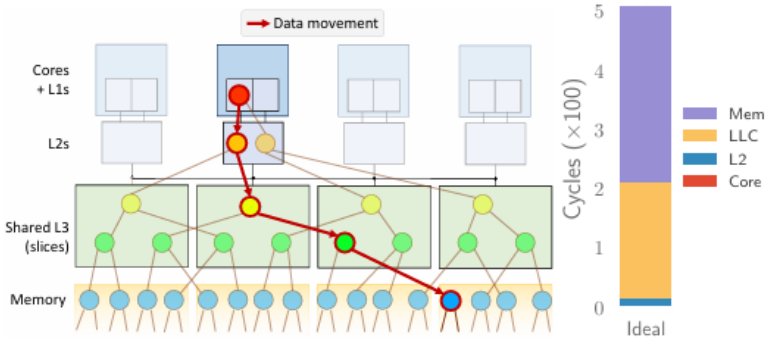


Fig. 4. Movimiento ideal de datos: el árbol recorre la jerarquía de la memoria, desde la raíz en L1 hasta la hoja en la memoria.

4.4 Memoria Interna

La memoria interna de un ordenador está formada por los componentes de almacenamiento más rápidos y cercanos al procesador, como los registros de la CPU y la memoria caché, estos niveles intermedios, que incluyen cachés L1, L2 y L3, permiten acceder a los datos en fracciones de nanosegundo, garantizando que el procesador no se detenga esperando información.[6]

En esta categoría también se encuentra la memoria principal, normalmente DRAM, que, aunque es más lenta que la caché, sigue siendo considerablemente más veloz que la memoria externa, su función es almacenar temporalmente los programas y datos que están siendo utilizados, manteniendo un equilibrio entre velocidad y capacidad.[6]

La jerarquía de la memoria interna busca reducir el tiempo de acceso mediante el uso de diferentes niveles con velocidades y tamaños distintos.[6]

A medida que se avanza hacia la CPU, el almacenamiento es más rápido, pero más costoso y de menor capacidad, mientras que hacia abajo se sacrifica velocidad a cambio de espacio [6].

La memoria interna es la parte principal donde se realiza toda la computación. En general, los sistemas operativos no permiten controlar directamente el movimiento de bloques de datos en la memoria principal, lo que dificulta que los programadores implementen eficientemente algoritmos que minimicen las operaciones de E/S.[7]

TPIE ofrece una gestión automática de la memoria interna, controlando la cantidad de memoria utilizada y distribuyéndola entre los diferentes componentes de un programa de forma eficiente. Esto permite a los desarrolladores no preocuparse por el manejo manual de memoria, a diferencia de STXXL, donde el programador debe asignar memoria explícitamente a cada parte.[7]

Además, TPIE soporta la ejecución paralela en CPUs multinúcleo, lo que mejora el rendimiento en tareas complejas como la ordenación multi-fusión. Esta capacidad facilita que distintas partes del algoritmo se ejecuten simultáneamente y aprovechen mejor los recursos disponibles.[7]

4.5 Memoria Interna en Linux

La memoria interna en Linux corresponde principalmente a la memoria RAM disponible para los procesos y al espacio usado para optimizar el acceso a datos mediante cachés y buffers. Estos elementos permiten que el sistema reduzca la frecuencia de accesos al disco, mejorando así el rendimiento general y la velocidad de respuesta.[8]

Dentro de esta memoria, la porción denominada *MemFree* indica la cantidad de RAM libre que el sistema tiene disponible en un momento dado. Aunque este valor puede parecer bajo, Linux aprovecha la RAM no utilizada para almacenamiento temporal de datos en caché, liberándola cuando un proceso la necesita.[8]

La *memoria en búfer* (*buffers*) se emplea para mantener temporalmente información relacionada con operaciones de escritura y lectura en dispositivos de almacenamiento. Por otro lado, la *memoria caché* (*cached*) guarda datos y programas recientemente utilizados, permitiendo que su acceso sea mucho más rápido que volver a leerlos desde el disco.[8]

En algunas arquitecturas, Linux distingue entre *memoria alta* (*High Memory*) y *memoria baja* (*Low Memory*). La memoria alta es aquella que el kernel accede de forma indirecta y que no tiene un mapeo permanente, mientras que la memoria baja sí está mapeada de manera directa, lo que facilita el acceso a los procesos.[8]

Además, el sistema gestiona otras áreas internas como *Slab Memory*, utilizada por la caché del kernel para almacenar estructuras de datos internas; *KernelStack*, dedicada a las pilas de ejecución del núcleo; y *PageTables*, que contienen las tablas de páginas para la traducción de direcciones de memoria virtual a física. Todas estas partes trabajan en conjunto para asegurar un uso eficiente y ordenado de la memoria interna.[8]

4.6 Memoria Externa

La memoria externa hace referencia a dispositivos de almacenamiento de gran capacidad, pero menor velocidad en comparación con la memoria interna.[6]

Ejemplos comunes incluyen los discos duros magnéticos, las unidades ópticas y las cintas, empleadas principalmente para archivo y respaldo de información.[6]

En estos medios, el acceso a los datos puede tardar varios milisegundos debido al movimiento físico de los cabezales y a la latencia de rotación, lo que los hace millones de

veces más lentos que la caché o los registros del procesador. Por esta razón, se transfieren bloques grandes de datos de forma secuencial para optimizar el tiempo de espera.[6]

Los sistemas de almacenamiento modernos, como RAID, permiten acceder a varios discos en paralelo para aumentar el ancho de banda y reducir la brecha de velocidad frente a la memoria interna. Aun así, la diferencia de rendimiento persiste, por lo que los algoritmos deben aprovechar la localidad de referencia para mejorar la eficiencia de las operaciones de entrada y salida.[6]

La memoria externa, como los discos, se utiliza para almacenar grandes volúmenes de datos que no caben en la memoria principal, el traslado de información entre la memoria interna y el disco suele ser el factor que más limita el rendimiento cuando se trabaja con cantidades masivas de datos.[7]

El acceso a disco es mucho más lento que el acceso a la memoria interna, por lo que los datos se transfieren en bloques contiguos grandes para optimizar las operaciones de entrada y salida (E/S), los algoritmos en este contexto miden su eficiencia principalmente por la cantidad de operaciones de E/S que requieren para procesar N elementos.[7]

Para hacer frente a estas limitaciones, se han desarrollado algoritmos especialmente diseñados para minimizar la cantidad de accesos al disco. El modelo I/O, creado por Aggarwal y Vitter, ayuda a analizar estos algoritmos considerando una memoria interna limitada y una memoria externa de gran tamaño donde se deben manejar los datos.[7]

Además, existen bibliotecas en C++ como TPIE y STXXL que facilitan la implementación de algoritmos eficientes en E/S. Mientras STXXL expone al programador detalles del hardware para maximizar el rendimiento, TPIE busca simplificar la programación ocultando estos detalles y automatizando la gestión de memoria y E/S.[7]

4.7 Memoria externa en (DNC)

La memoria externa en un Differentiable Neural Computer (DNC) es una matriz que permite a la red almacenar y procesar secuencias complejas de datos. El DNC usa cabezas de lectura y escritura para interactuar con esta memoria de forma diferenciable, lo que facilita su entrenamiento mediante descenso de gradiente.[9]

Para mejorar el uso de la memoria, el DNC implementa mecanismos como la vinculación temporal, que mantiene el orden de escritura, y la asignación dinámica, que reutiliza

posiciones libres. Durante la escritura y lectura, combina atención basada en contenido con estas técnicas para acceder eficazmente a la información.[9]

Se propuso una mejora que divide la memoria en pares clave-valor, reduciendo el tiempo de cómputo y aumentando la cantidad de información capturada. Otra mejora usa una red neuronal para la lectura, que mejora la precisión, pero aumenta el tiempo de ejecución, por lo que su combinación con la memoria clave-valor no siempre es conveniente.[9]

4.8 ENTRADA Y SALIDA

La entrada/salida (E/S) es la parte del computador que permite que la CPU hable con cosas externas: teclados, discos, pantallas, impresoras, redes, etc. Estas interacciones se resuelven mediante módulos o controladores que "traducen" entre el lenguaje interno del procesador (buses, direcciones, registros) y los detalles físicos del periférico (señales eléctricas, tiempos, formatos). En la práctica estos módulos ocultan la complejidad del dispositivo y presentan a la CPU operaciones sencillas de leer/escribir; eso facilita que programas y sistemas operativos trabajen sin conocer cada detalle eléctrico del periférico [10].

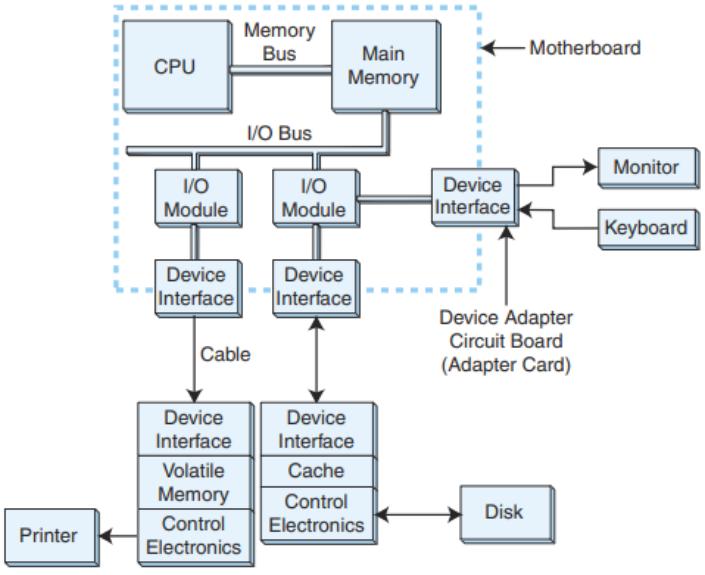


Figura 5 Una configuración modelo de E/S [11].

Los dispositivos externos varían mucho: unos son lentos (teclados, impresoras), otros muy rápidos (discos, tarjetas de red) y cada uno tiene su propia lógica de control y a menudo una memoria intermedia (buffer). Es decir, el periférico no suele conectarse

directamente al bus del sistema; en su lugar va a través de una interfaz que maneja el posicionamiento, la temporización y errores, y que asegura que la información llegue completa y en el formato correcto. Esta heterogeneidad es la razón principal por la que existen los módulos de E/S [11].

Un módulo de E/S (o controlador) es básicamente una pequeña "caja" con registros de datos, registros de estado y lógica de control que se conecta al bus y al dispositivo. El procesador le manda órdenes por el bus de control, el módulo usa su registro de datos para intercambiar bytes y ofrece señales de estado para decir si está listo, ocupado o con error. Además, los módulos suelen tener buffers para compensar la diferencia de velocidad entre la CPU/memoria y el periférico, evitando que la memoria principal quede ocupada todo el tiempo [10].

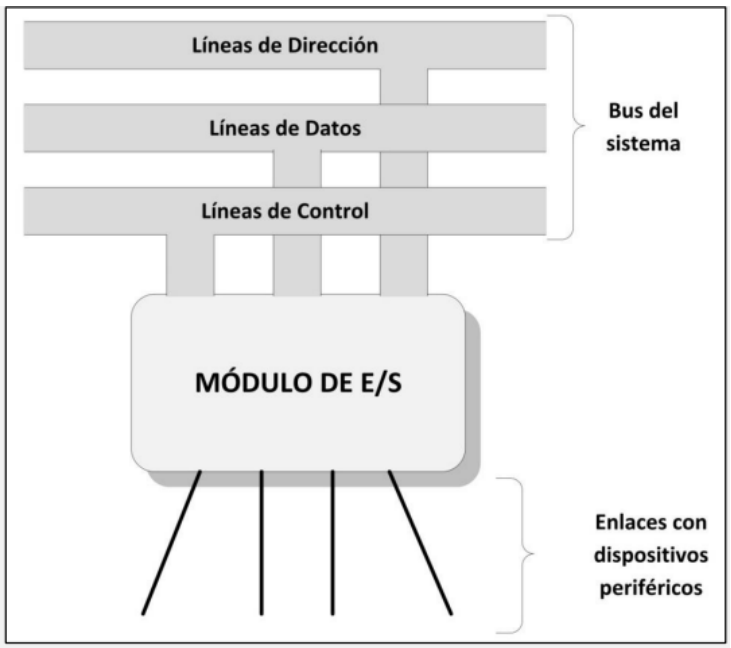


Figura 6 Módulo de E/S [10].

La E/S programada (o polled I/O) consiste en que la CPU controla todo: envía la orden y se queda "preguntando" al módulo si hay datos o si terminó la operación. Es fácil de implementar y de entender, pero ineficiente, porque la CPU entra en un bucle de espera y no hace trabajo útil mientras tanto. Por eso se usa cuando el sistema es muy simple o cuando el número de periféricos es pequeño y el coste de esperar es tolerable [11].

La E/S mediante interrupciones mejora esto: el procesador inicia la operación y sigue con otras tareas; cuando el módulo termina avisa con una interrupción y la CPU salta a una

rutina de servicio para completar la transferencia. Este método necesita soporte de hardware y de software (vectores de interrupción, guardar/restaurar contexto), pero reduce el tiempo de CPU desperdiciado y es mucho más práctico para sistemas multitarea [11].

El acceso directo a memoria (DMA) es la técnica que más libera a la CPU: un controlador DMA se encarga de mover bloques de memoria entre el periférico y la RAM sin que la CPU copie cada byte. La CPU solo configura la transferencia (dirección, tamaño) y el DMA "roba" ciclos de memoria para transferir los datos (cycle stealing) o toma el control del bus por ráfagas. Esto es ideal para discos o redes donde se manejan bloques grandes; en sistemas grandes incluso se usan procesadores de E/S (IOP) o canales que ejecutan programas de E/S independiente [11][12].

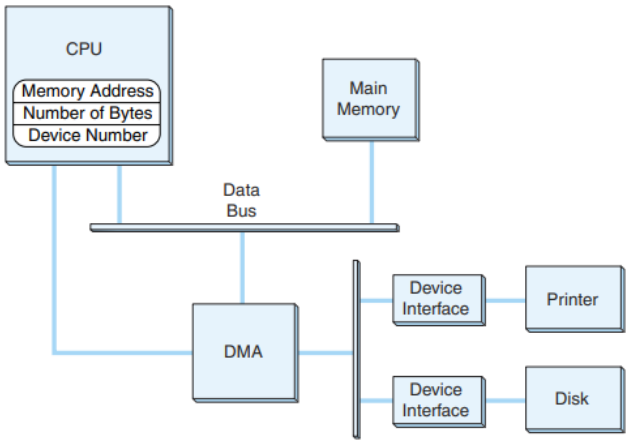


Figura 7 Un ejemplo de configuración DMA [11].

Por dentro, la interfaz interna de un módulo muestra claramente las líneas que todos conocemos: bus de datos (bidireccional), líneas de dirección, y líneas de control/estado. Un ejemplo clásico es el chip programable 82C55A (un ejemplo didáctico) que ofrece puertos de 8 bits, registros de control y señales que permiten configurar modos y sincronizar teclado/pantalla. En resumen: la interfaz interna coordina la selección del puerto, el formato de los datos y las señales de control/handshaking para que la comunicación sea fiable [10].

Para tareas pequeñas o sistemas embebidos sencillos la E/S programada es más fácil de programar; para sistemas interactivos y multitarea las interrupciones son la norma; y para transferencias grandes (discos, tarjetas de red) siempre que sea posible se usa DMA o canales para no "secuestrar" la CPU. Además, diseñar o seleccionar un módulo de E/S

implica pensar en buffers, priorización de interrupciones y en cómo se mapean las direcciones de E/S en el sistema [12].

5 ARQUITECTURA DE BUS PARA PROTECCIÓN CONTRA HARDWARE TROJAN EN CHIPS DE SEGURIDAD

Según Changlong et al. (2011), en el artículo "A System-On-Chip bus architecture for hardware Trojan protection in security chips"[13], se propone una arquitectura de bus Segura que protegerá al usuario de Hardware Troyano, además nos indica lo siguiente:

Los circuitos integrados actuales, dependen aun más de núcleos IP de terceros y de desarrollos externos para su desarrollo. Esta es la razón que los vuelve muy vulnerables a modificaciones maliciosas, conocidas como "Hardware Trojans", las cuales pueden ser el motivo de las filtraciones de datos o fallos extremos en el sistema.

Los Hardware Troyanos se clasifican dependiendo de sus características físicas, método de activación y accionamiento. Sin embargo, existen métodos para detectarnos antes de que logren hacer algún daño, aunque siguen siendo muy peligrosos.

En el área de chips de seguridad, si un virus troyano contamina el diseño o fabricación de este, puede evitar casi todas las medidas de seguridad, y extraer información como usuarios y contraseñas, y a veces sin ser detectados.

Antes de que el árbitro del bus permita la comunicación entre un maestro y un esclavo, se genera un número aleatorio mediante un Generador de Números Aleatorios (RNG). Este número se envía simultáneamente a ambos dispositivos para sincronizar la transmisión. Los datos reales se combinan con pseudo-datos, de forma que solo el receptor autorizado, usando el mismo número aleatorio, pueda extraer la información auténtica.

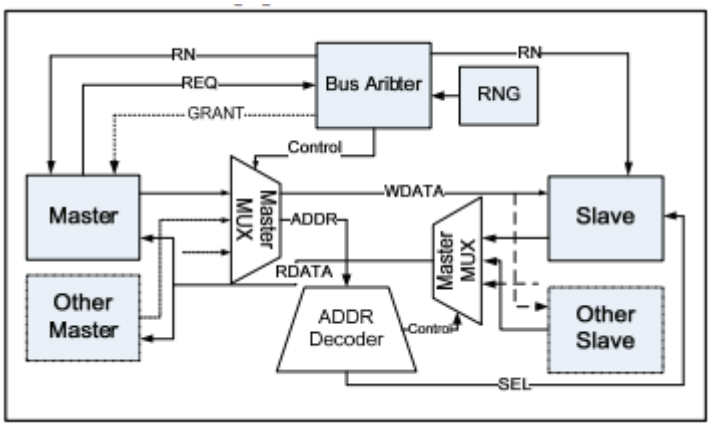


Figura 8 La estructura de la arquitectura de bus propuesta.

El RNG tiene una estructura híbrida compuesta por una Fuente Aleatoria Analógica (ARS), que emplea osciladores de alta frecuencia no correlacionados y un muestreo con reloj principal de baja frecuencia con jitter. También incluye un Registro de Corrimiento con Retroalimentación Lineal (LFSR), que utiliza un polinomio característico específico y posee un periodo de $2^{128} - 1$.

Esta arquitectura busca impedir que un Troyano pueda interceptar y filtrar la comunicación interna del chip.

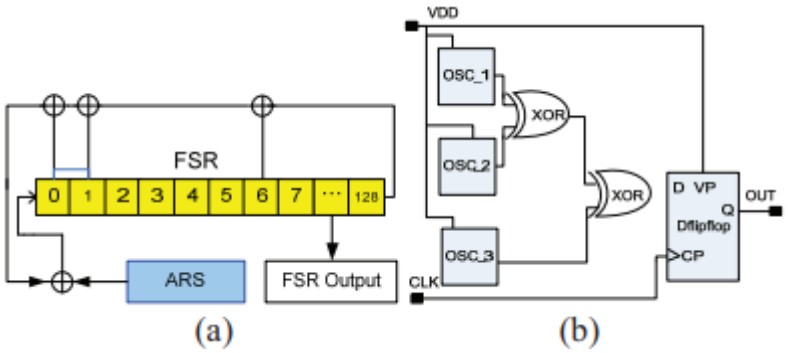


Figura 9 (a) La estructura del RNG-(b) El diagrama de bloques detallado del ARS.

6 BUS DE COMPONENTES

En la conferencia dada por M. Yasuda et al. (2002). Titulada "A top-down hardware/software co-simulation method for embedded systems based upon a component logical bus architecture" [14], nos indica lo siguiente:

La arquitectura lógica de bus de componentes se conforma por tres tipos de señales. Señales del bus del sistema, señales de puerto y señales de interrupción. El objetivo de esta arquitectura es servir de interfaz entre componentes de software y hardware.

6.1 Señales del bus del sistema

Incluyen dirección, datos y control. Los datos se leen o escriben según la señal de control y la dirección. El espacio de direcciones se divide en espacio de memoria y espacio de entrada/salida (I/O). La memoria de los componentes de software se asigna en el espacio de memoria, mientras que los puertos de software y los registros de hardware están en el espacio de I/O mapeado en memoria.

6.2 Señales de puerto

Son señales específicas que permiten la comunicación directa entre componentes de software y hardware. Los puertos del software también se asignan en el espacio de I/O.

6.3 Señales de interrupción

Se usan para notificar a un componente de software cuando un componente de hardware o software ha completado una operación o ha detectado un error.

El intercambio de datos ocurre mediante la escritura en puertos y registros del hardware por parte del software para iniciar operaciones y transferir datos. El hardware notifica la finalización mediante interrupciones o estableciendo una bandera de estado. El software lee los puertos y registros para obtener datos e información de estado.

7 BUS PUENTE

Según T. Wang et al. (2008) en el artículo "A Hardware Implement of Bus Bridge Based on Single CPU and Dual Bus Architecture" [15], describe la función y estructura del bus Puente.

Las arquitecturas tradicionales utilizan una sola CPU y un solo bus, como la arquitectura de Von Neumann. Esta arquitectura es escasa en los ordenadores de red los cuales son exigentes en los métodos de mantener seguros sus datos. Además, la red y el almacenamiento local están conectados al mismo bus, esto representa un peligro, ya que con software malintencionado los datos de ese almacenamiento podrían ser robados y llegan a controlar el bus del sistema.

7.1 Arquitectura mínima del sistema de hardware

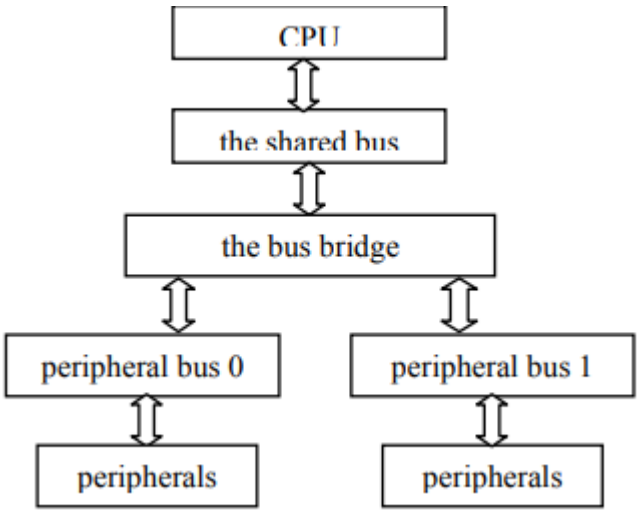


Figura 10 Arquitectura mínima del sistema de hardware

Como se puede observar en la figura 3 el sistema consta de cuatro partes: CPU, buses, el puente de bus y los periféricos. El bus puente se conecta a la CPU a través del bus compartido. Los buses periféricos se conectan a los puertos maestros del bus puente.

Además, los dos buses periféricos están aislados y solo uno de ellos contiene el módulo del CPU y puede conectarse a los buses compartidos.

7.2 Función del bus puente

La función del bus puente es conectar el bus compartido con uno de los buses periféricos. El bus bridge debe asegurar que el bus compartido pueda conectarse en cualquier momento a uno de los buses periféricos. Además, debe garantizar que solo un bus periférico esté conectado al bus compartido simultáneamente. Cuando la CPU necesita cambiar de bus periférico, el bus bridge corta la conexión actual y conecta el bus compartido con el otro bus periférico.

Antes del diseño del bus puente, se definen dos términos importantes. El área de trabajo indica cuál bus periférico está conectado actualmente al bus compartido. Si el bus periférico 0 está conectado, el sistema trabaja en el área 0; si es el bus periférico 1, entonces trabaja en el área 1. Las señales correspondientes son aquellas con la misma definición en el bus compartido y en los buses periféricos, pero con nombres distintos para diferenciarlas. Por ejemplo, la señal write_n en el bus compartido corresponde a ava0_write_n en el bus periférico 0 y ava1_write_n en el bus periférico 1.

7.3 Estructura del bus puente

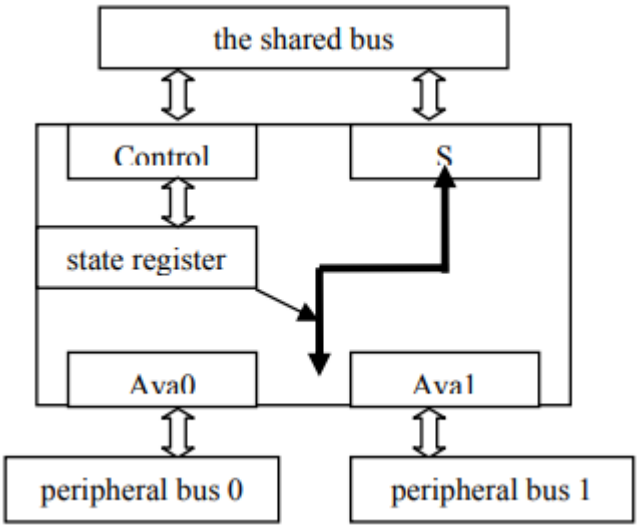


Figura 11 Estructura del bus puente

La estructura del bus bridge incluye dos puertos esclavos llamados Control y S, dos puertos maestros denominados Ava0 y Ava1, y un registro de estado de cinco bits. El puerto Control recibe instrucciones del CPU para el bus bridge y envía los datos del registro de estado al CPU. El puerto S recibe y transmite todas las señales Avalon del bus

compartido. Estas señales se conectan con sus señales correspondientes en los puertos Ava0 o Ava1, que están vinculados a los buses periféricos.

El registro de estado tiene cinco bits: los cuatro bits inferiores almacenan los modos de trabajo, y el bit superior, llamado bit bandera, indica qué bus periférico está conectado actualmente al bus compartido a través del bus bridge. Cuando el bit bandera está en 0, todas las señales del puerto S se conectan con las del puerto Ava0, por lo que el sistema opera en el área 0. Si el bit bandera está en 1, el sistema opera en el área 1 conectando el bus compartido con el bus periférico 1.

8 BUS DE DIRECCIONES DDR4

Según K. Pandey (2018), en el artículo: "Signal and power integrity analysis of DDR4 address bus of onboard memory module" [16], explica que para garantizar la integridad de señal y potencia en el bus de direcciones DDR4, es fundamental usar una topología "fly-by" y terminaciones adecuadas para reducir ruido y reflexiones, asegurando una transmisión confiable a altas velocidades.

En el diseño de memorias DDR4, el bus de direcciones es fundamental para transmitir la información con precisión a altas velocidades, que pueden llegar hasta 3.2 Gbps. Para manejar estas velocidades, se utiliza una topología llamada "fly-by", donde la señal de dirección pasa en cadena desde el controlador hacia cada módulo de memoria, lo que ayuda a reducir problemas como las reflexiones y el ruido. Además, en el extremo final del bus se coloca una terminación pull-up de 40 ohmios que mejora la integridad de la señal.

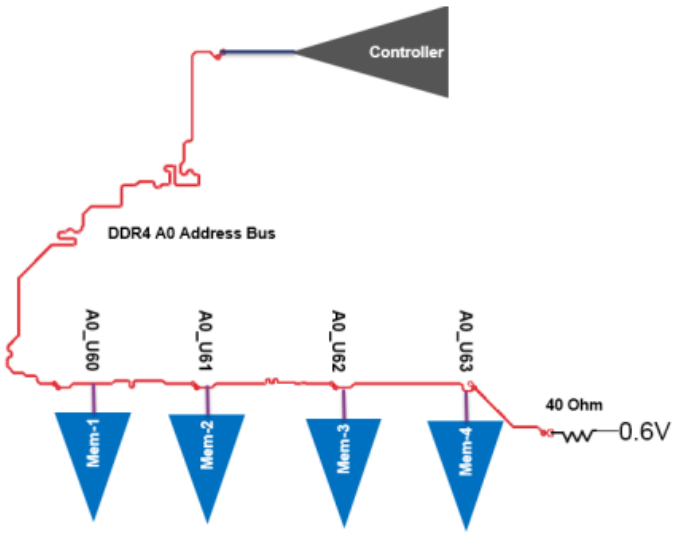


Figura 12 Topología de buses de dirección en memoria DDR4

Cada línea en el bus de direcciones representa un bit, por lo que la cantidad de líneas determina la capacidad máxima de memoria que puede direccionarse directamente. El diseño del enrutamiento es crucial; se recomienda que las señales de reloj diferencial, dirección, control y comando estén en capas cercanas para minimizar interferencias y garantizar una buena calidad de señal.

El ruido generado por la conmutación simultánea de varias líneas, conocido como ruido SSN, puede afectar negativamente la señal, por lo que se emplean terminaciones pasivas, como resistencias en serie y paralelo, para controlar estas interferencias. Además, el plano de potencia debe diseñarse cuidadosamente para mantener la impedancia dentro de límites adecuados y reducir el ruido eléctrico.

Para evaluar el desempeño del bus de direcciones, se usan simulaciones con modelos IBIS que permiten analizar el comportamiento dinámico de la señal y generan diagramas de ojo que muestran la calidad en diferentes módulos de memoria. Estos análisis ayudan a identificar el punto más débil en la cadena de memoria, asegurando que el sistema funcione de forma estable y eficiente.

9 BUSES DE SERIE UNIVERSAL

Según L. Ramadoss y J. Y. Hung (2008), en el artículo: “A study on universal serial bus latency in a real-time control system” [17], nos explica lo siguiente acerca del bus de serie universal.

Cuando se conecta un bus de serie universal (USB) al computador, el sistema operativo automáticamente lo detecta y se encarga de configurarlo y el controlador USB detecta las características del dispositivo a través de un dispositivo descriptor.

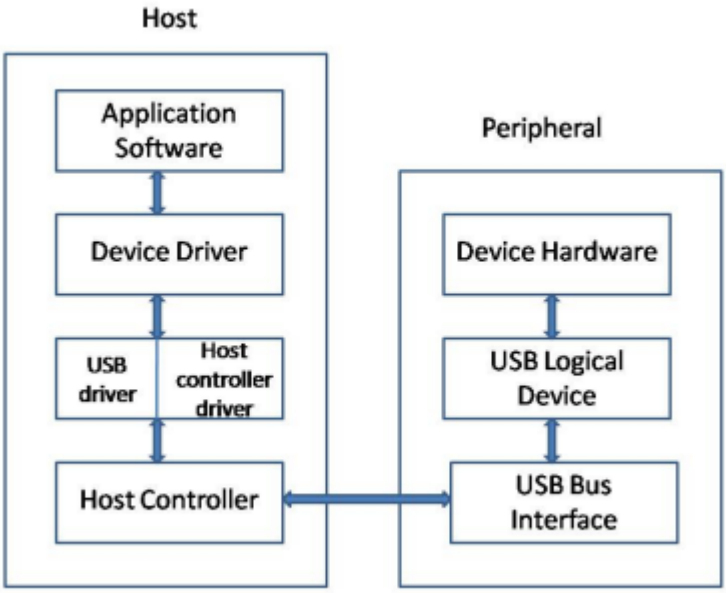


Figura 13 Comunicación USB

Un dispositivo descriptor es una estructura de datos la cual contiene información sobre las propiedades de los dispositivos conectados. Inmediatamente el controlador USB localiza el driver del dispositivo y lo carga o actualiza. Una vez realizado esto establece una conexión lógica llamada canalización entre el driver y el punto final. El punto final es el registro por donde entran y salen datos del dispositivo.

9.1 Comunicación del bus de serie universal

La comunicación USB se basa en una relación maestro-esclavo, donde la computadora actúa como maestro y los dispositivos USB como esclavos. Solo puede haber un maestro en el sistema y los dispositivos USB no se comunican directamente entre sí, sino siempre a través del host.

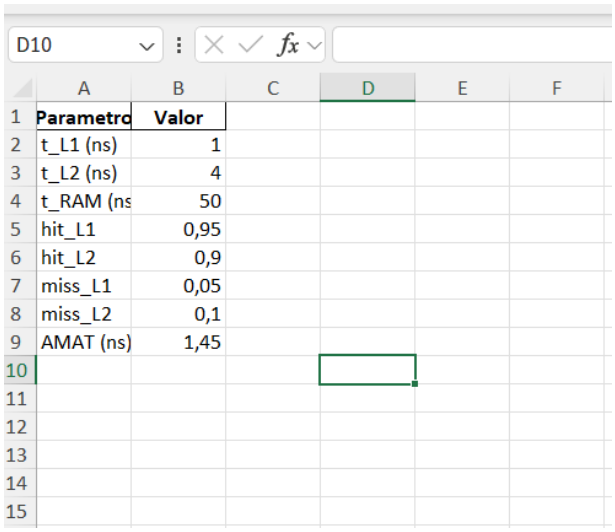
El software del sistema USB incluye dos componentes principales: el controlador USB y el controlador del host. Cuando un dispositivo USB necesita enviar o recibir datos, el controlador del dispositivo envía una solicitud de transferencia al controlador USB, llamada paquete de solicitud de entrada/salida (IRP).

Luego, el controlador del dispositivo proporciona un área de memoria para almacenar los datos durante la transferencia. El controlador USB divide esa solicitud en varias transacciones pequeñas que se ejecutan en intervalos de 1 ms. Estas transacciones se organizan según las necesidades del dispositivo y la capacidad del USB.

10 PROCEDIMIENTOS

10.1 Procedimiento # 1

10.1.1 Realizar ejercicios prácticos para calcular tiempos de acceso y capacidad en diferentes niveles de memoria.



	A	B	C	D	E	F
1	Parametro	Valor				
2	t_L1 (ns)	1				
3	t_L2 (ns)	4				
4	t_RAM (ns	50				
5	hit_L1	0,95				
6	hit_L2	0,9				
7	miss_L1	0,05				
8	miss_L2	0,1				
9	AMAT (ns)	1,45				
10						
11						
12						
13						
14						
15						

Figura 14 - AMAT (ejemplo fijo)

Archivo 1 - Capacidades De Caché

	Nivel	Sets	Asociatividad	Tamaño_bloque_Bytes	Capacidad_KB
1	L1	128	4	64	32.0
2	L2	1024	8	64	512.0

Figura 15 - Capacidades de caché

10.1.2 Análisis Del Rendimiento De La Memoria Caché Mediante Ejemplos Teóricos Y Simulaciones.

A1 | hit_L1

	A	B	C	D	E	F	G	H
1	hit_L1	hit_L2	t_L1_ns	t_L2_ns	t_RAM_ns	AMAT_ns		
2	0,8	0,9	1	4	50	2,8		
3	0,81	0,9	1	4	50	2,71		
4	0,82	0,9	1	4	50	2,62		
5	0,83	0,9	1	4	50	2,53		
6	0,84	0,9	1	4	50	2,44		
7	0,85	0,9	1	4	50	2,35		
8	0,86	0,9	1	4	50	2,26		
9	0,87	0,9	1	4	50	2,17		
10	0,88	0,9	1	4	50	2,08		
11	0,89	0,9	1	4	50	1,99		
12	0,9	0,9	1	4	50	1,9		
13	0,91	0,9	1	4	50	1,81		
14	0,92	0,9	1	4	50	1,72		
15	0,93	0,9	1	4	50	1,63		
16	0,94	0,9	1	4	50	1,54		
17	0,95	0,9	1	4	50	1,45		
18	0,96	0,9	1	4	50	1,36		
19	0,97	0,9	1	4	50	1,27		
20	0,98	0,9	1	4	50	1,18		
21	0,99	0,9	1	4	50	1,09		
22								
23								

Figura 16 - Simulación AMAT (L1 variable)

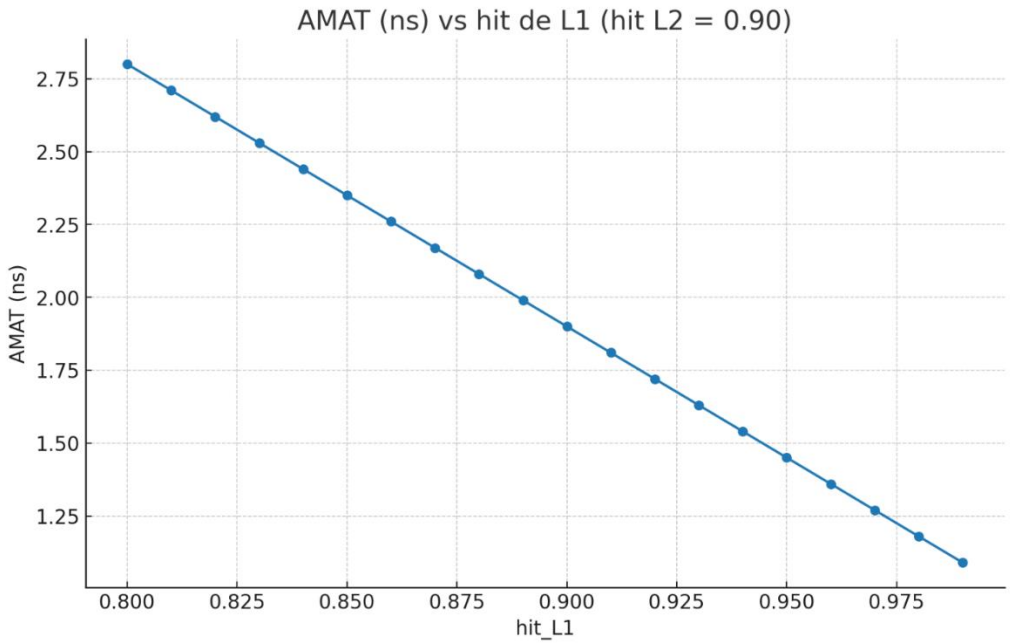


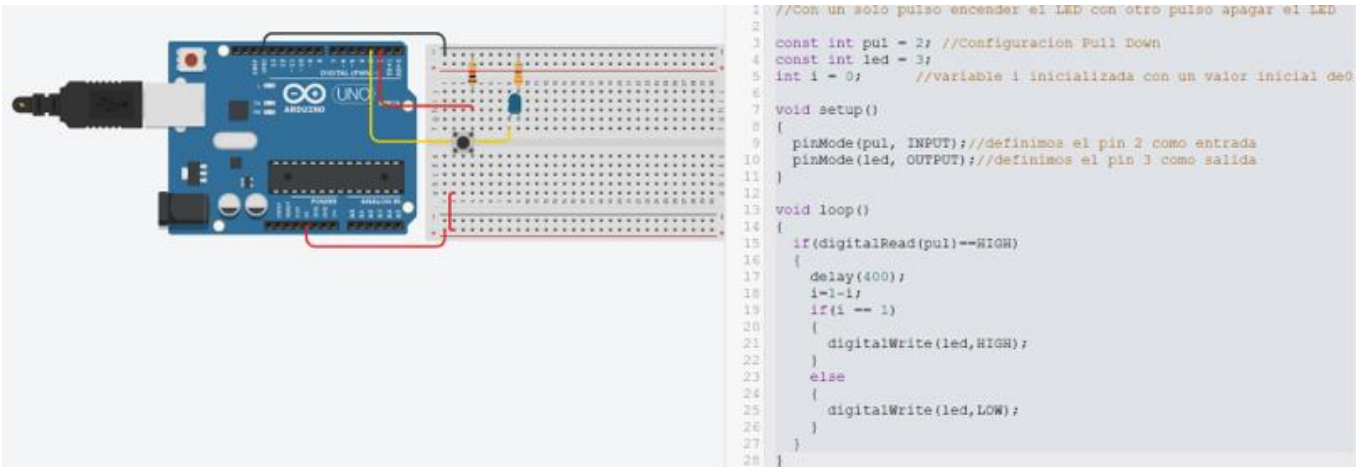
Figura 17 - AMAT (ns) vs hit de L1 (hit L2 = 0.90)

10.2 Procedimiento # 2

Operación de Entrada/Salida:

• Implementar ejemplos de E/S programada y E/S mediante interrupciones usando simuladores.

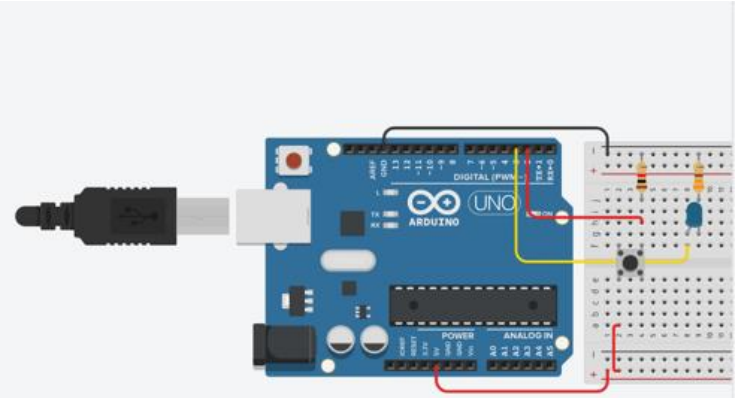
• Analizar el flujo de datos mediante DMA con un ejemplo práctico



```
//Con un solo pulso encender el LED con otro pulso apagar el LED

const int pul = 2; //Configuracion Pull Down
const int led = 3;
int i = 0;       //variable i inicializada con un valor inicial de0

void setup()
{
  pinMode(pul, INPUT);//definimos el pin 2 como entrada
  pinMode(led, OUTPUT);//definimos el pin 3 como salida
}

void loop()
{
  if(digitalRead(pul)==HIGH)
  {
    delay(400);
    i=1-i;
    if(i == 1)
    {
      digitalWrite(led,HIGH);
    }
    else
    {
      digitalWrite(led,LOW);
    }
  }
}
```

Figura 18: Implementación de E/S programada para encender y apagar un LED con cada pulsación de un botón usando digitalRead *en el ciclo* loop ().

10.2.1 E/S Con Interrupciones



```arduino
// Con interrupción: encender y apagar el LED con un botón
const int led = 3;
volatile int estado = 0;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(2, INPUT);  // Botón en pin 2

  // Interrupción al detectar flanco de subida (botón presionado)
  attachInterrupt(digitalPinToInterrupt(2), toggleLed, RISING);
}

void loop() {
  digitalWrite(led, estado);
}

// Función de interrupción
void toggleLed() {
  estado = !estado;  // Cambia entre 0 y 1
}
```

Figura 19: Uso de interrupciones para controlar un LED con un botón, permitiendo una respuesta inmediata al detectar un flanco de subida.

10.2.2 Flujo de Datos Mediante DMA

```csharp
using System;
using System.Threading;

0 referencias
class Program
{

    static int[] sourceData = new int[100];
    static int[] destinationData = new int[100];
    static bool dmaCompleted = false;

    0 referencias
    static void Main(string[] args)
    {

        for (int i = 0; i < sourceData.Length; i++)
            sourceData[i] = i + 1;


        Thread dmaThread = new Thread(() => DmaTransfer(0, 0, sourceData.Length));
        dmaThread.Start();


        while (!dmaCompleted)
        {
            Console.WriteLine("CPU haciendo otras tareas...");
            Thread.Sleep(200);
        }


        Console.WriteLine("DMA terminó la transferencia. Datos en destino:");
        for (int i = 0; i < destinationData.Length; i++)
            Console.Write(destinationData[i] + " ");

        Console.WriteLine("\nProceso finalizado.");


    1 referencia
    static void DmaTransfer(int sourceIndex, int destIndex, int size)
    {
        Console.WriteLine("DMA iniciando transferencia...");
        for (int i = 0; i < size; i++)
        {
            destinationData[destIndex + i] = sourceData[sourceIndex + i];
            Thread.Sleep(50);
        }
        dmaCompleted = true;
        Console.WriteLine("DMA transferencia completada.");
    }
}
```

Figura 20: Simulación realizada en C# aplicación de consola.

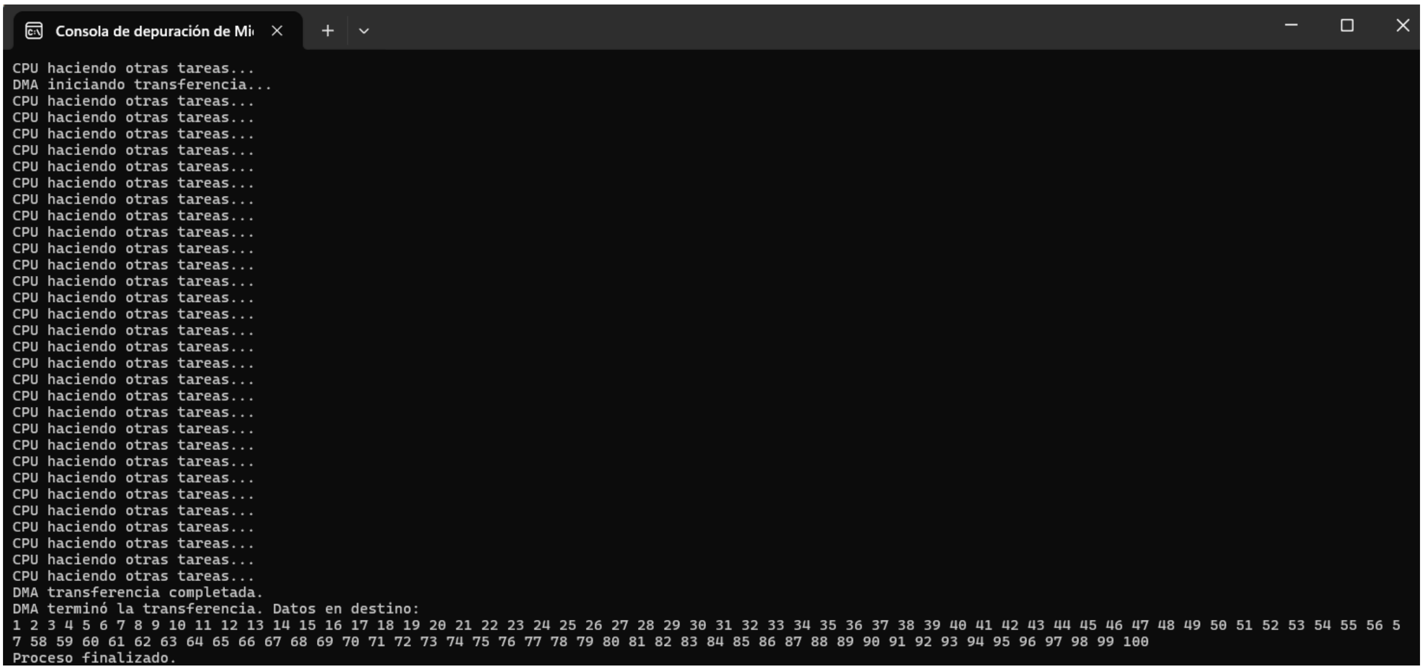


Figura 21: Ejecución de la simulación del programa.

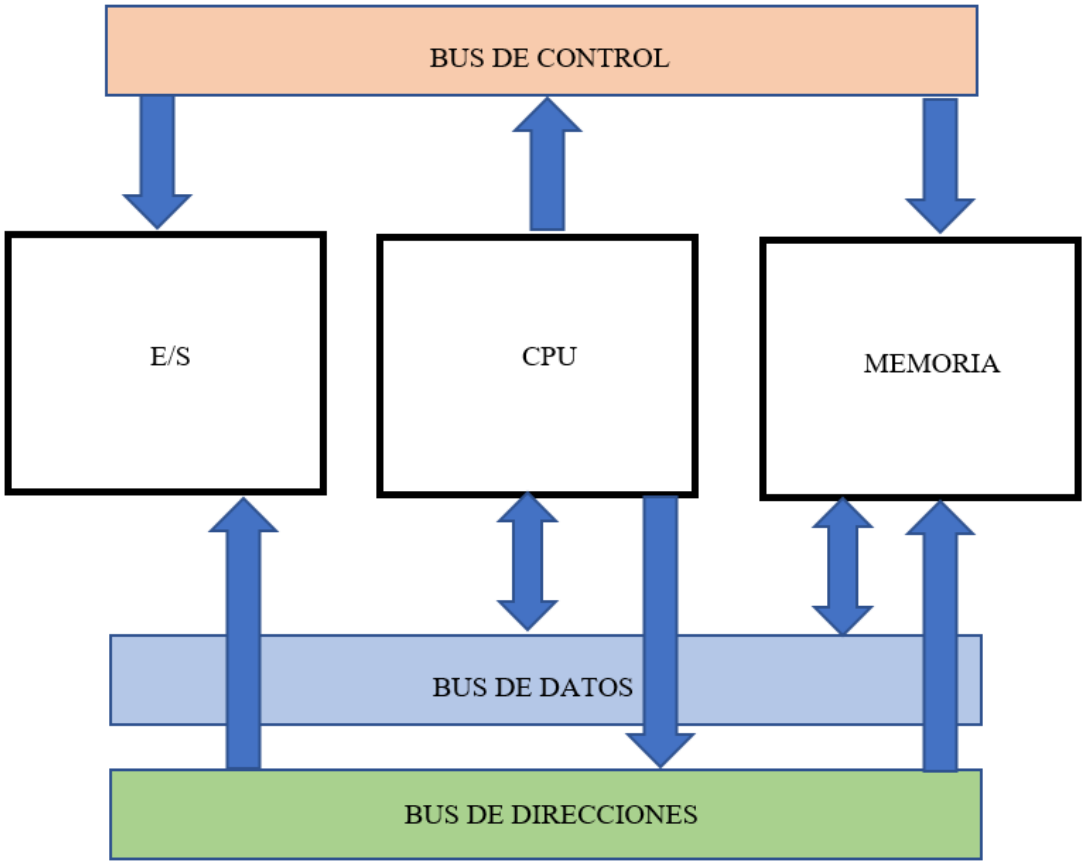
En la figura 3 y 4 se presenta la simulación del funcionamiento del DMA (acceso directo a memoria) usando C#. Se crean dos arreglos: uno con los datos fuente y otro vacío como destino. Se utilizó un hilo separado para representar al DMA, que copia los datos de forma paralela mientras la CPU continúa ejecutando otras tareas, esto se refleja en la consola con el mensaje “CPU haciendo otras tareas...”. Cuando el DMA termina la transferencia, se imprime el mensaje de finalización junto con los datos copiados, este ejemplo demuestra cómo el DMA permite liberar a la CPU de tareas de transferencia, mejorando la eficiencia del sistema.

10.3 Procedimiento #3

10.3.1 Realizar Diagramas Que Representen La Estructura Y Función De Un Bus.

Un bus es un conjunto de líneas o canales de comunicación que trasmiten información, direcciones y señales trasportándola mediante estos canales entre los componentes del sistema como la CPU, memoria, periféricos, entre otros.

10.3.2 Diagrama de buses en arquitectura de computadoras



Si el CPU quisiera leer un dato en la memoria se daría el siguiente proceso:

1. La CPU coloca la dirección a leer en el Bus de Direcciones.
2. La CPU envía la señal de lectura por el Bus de Control.
3. La memoria reconoce la dirección y la señal de lectura.
4. La memoria pone el dato solicitado en el Bus de Datos.
5. La CPU lee el dato desde el Bus de Datos.

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

A continuación, se muestra un cuadro comparativo con los tipos de buses, su descripción, ventajas y desventajas y como impacta a la velocidad de transferencia.

Tipo de Bus	Descripción	Ventajas	Desventajas	Impacto en la Velocidad de Transferencia
Bus de Datos	Transporta los datos entre componentes.	Mayor ancho permite transferir más bits a la vez.	Mayor complejidad con más líneas.	Ancho mayor = mayor cantidad de datos transferidos simultáneamente, aumenta la velocidad.
Bus de Direcciones	Transporta las direcciones de memoria o dispositivos.	Permite direccionar gran cantidad de memoria.	No afecta directamente la velocidad de datos.	Limita la cantidad máxima de memoria direccionable, no impacta velocidad directa.
Bus de Control	Señales para controlar y sincronizar la transferencia.	Controla la coordinación y sincronización.	Requiere buena gestión para evitar retrasos.	Afecta la sincronización y timing, impactando indirectamente la velocidad.
Bus Paralelo	Varias líneas transportan bits simultáneamente.	Transferencia rápida, envía muchos bits a la vez.	Limitado a distancias cortas; interferencias y costos elevados.	Alta velocidad por transferencia simultánea, pero menos eficiente a largas distancias.
Bus Serie	Bits enviados uno tras otro por pocas líneas.	Menor interferencia, permite distancias largas.	Velocidad menor para grandes volúmenes de datos.	Velocidad menor comparado con bus paralelo en volumen, pero más estable y confiable en distancias.

11 BIBLIOGRAFÍA

[1] M. Vaithianathan, "Memory Hierarchy Optimization Strategies for HighPerformance Computing Architectures," *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 6, pp. 24–35, 2025, doi: 10.63282/3050-9246.ijetcsit-v6i1p103.

[2] H.-K. Liu *et al.*, "A survey of non-volatile main memory technologies: State-of-the-arts, practices, and future directions," *J Comput Sci Technol*, vol. 36, no. 1, pp. 4–32, 2021, doi: 10.1007/s11390.

[3] A. Oluwatosin Abayomi, A. Abayomi Olukayode, and G. Oluwole Olakunle, "An Overview of Cache Memory in Memory Management," *Automation, Control and Intelligent Systems*, vol. 8, no. 3, p. 24, 2020, doi: 10.11648/j.acis.20200803.11.

[4] A. V. Jamet, L. Alvarez, D. A. Jimenez, and M. Casas, "Characterizing the impact of last-level cache replacement policies on big-data workloads," in *Proceedings - 2020 IEEE International Symposium on Workload Characterization, IISWC 2020*, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 134–144. doi: 10.1109/IISWC50251.2020.00022.

[5] *ASPLOS XXV: Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems: March 16-20, 2020, Lausanne, Switzerland*. The Association for Computing Machinery, 2020.

[6] J. S. Vitter, "Algorithms and data structures for external memory," *Foundations and Trends in Theoretical Computer Science*, vol. 2, no. 4, pp. 305–474, 2006, doi: 10.1561/0400000014.

[7] L. Arge, M. Rav, S. C. Svendsen, and J. Truelsen, "External memory pipelining made easy with TPIE," in *2017 IEEE International Conference on Big Data (Big Data)*, IEEE, Dec. 2017, pp. 319–324. doi: 10.1109/BigData.2017.8257940.

[8] D. Kayande and U. Shrawankar, "Performance Analysis for Improved RAM Utilization for Android Applications."

[9] A. Yadav and K. Pasupa, "Augmenting Differentiable Neural Computer with Read Network and Key-Value Memory," in *2021 25th International Computer Science and Engineering Conference (ICSEC)*, IEEE, Nov. 2021, pp. 262–266. doi: 10.1109/ICSEC53205.2021.9684629.

[10] Daniel M. Argüello, Santiago C. Pérez, and Higinio A. Facchini, "Arquitectura de Computadoras," 2022.

[11] R. M. . Merrill, *Statistical methods in epidemiologic research*. Jones & Bartlett Learning, 2016.

[12] S. Harris, "Digital Design and Computer Architecture," 2007.

[13] L. Changlong, Z. Yiqiang, S. Yafeng, and G. Xingbo, "A System-On-Chip bus architecture for hardware Trojan protection in security chips," in *2011 IEEE International Conference of Electron Devices and Solid-State Circuits*, IEEE, Nov. 2011, pp. 1–2. doi: 10.1109/EDSSC.2011.6117727.

[14] M. Yasuda, K. Seo, H. Koizumi, B. Shackleford, and F. Suzuki, "A top-down hardware/software co-simulation method for embedded systems based upon a component logical bus architecture," in *Proceedings of 1998 Asia and South*

Pacific Design Automation Conference, IEEE, pp. 169–175. doi: 10.1109/ASPDAC.1998.669438.

1. [15] T. Wang, F. Shao, R. Sun, and H. Huang, “A Hardware Implement of Bus Bridge Based on Single CPU and Dual Bus Architecture,” in *2008 International Symposium on Computer Science and Computational Technology*, IEEE, 2008, pp. 17–20. doi: 10.1109/ISCSCT.2008.106.

[16] A. K. Pandey, “Signal and power integrity analysis of DDR4 address bus of onboard memory module,” in *2018 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, IEEE, Dec. 2018, pp. 1–3. doi: 10.1109/EDAPS.2018.8680896.

[17] L. Ramadoss and J. Y. Hung, “A study on universal serial bus latency in a real-time control system,” in *2008 34th Annual Conference of IEEE Industrial Electronics*, IEEE, Nov. 2008, pp. 67–72. doi: 10.1109/IECON.2008.4757930.

12 ANEXOS

12.1 Git-Hub

https://github.com/MiloSaurio4kHD/ARQ_GRC_COMPONENTES-DEL-COMPUTADOR