PROTOTIPO DE APOYO PARA EMULAR EL FUNCIONAMIENTO DEL PROTOCOLO MESI

PROTOTYPE OF SUPPORT FOR EMULATING MESI PROTOCOL OPERATION

Rodrigo Vázquez López
Instituto Politécnico Nacional
rodrigo_em2@hotmail.com

Esther Viridiana Vázquez Carmona
Instituto Politécnico Nacional
ev.vazquezc@gmail.com

Juan Carlos Herrera Lozada
Instituto Politécnico Nacional
jlozada@ipn.mx

Miguel Hernández Bolaños
Instituto Politécnico Nacional
mbolanos@ipn.mx

Magdalena Marciano Melchor
Instituto Politécnico Nacional
mmarciano@ipn.mx

Resumen

En los sistemas multiprocesador los datos pueden residir tanto en distintos niveles de caché como en la memoria principal. Mantener la coherencia de los datos entre los diferentes cachés y la memoria principal se conoce como el problema de coherencia de caché, el cual se puede solventar con el uso de protocolos como MESI (Modified-Exclusive-Shared-Invalid). En este trabajo se presenta el desarrollo de un prototipo cuyo objetivo es explicar el funcionamiento del protocolo antes mencionado en cursos de arquitecturas avanzadas de computadoras vistos a nivel ingeniería. El prototipo emula el funcionamiento de un sistema multiprocesador de dos procesadores (cada uno con un único nivel de caché) y la memoria principal utilizando dos tarjetas Arduino. El sistema interactúa con el usuario por medio de botones para leer y modificar datos de la memoria, así

como pantallas que despliegan el contenido de las cachés y el estado en el que se encuentran.

Palabras Claves: Arduino, arquitecturas avanzadas de computadoras, coherencia de caché, protocolo MESI.

Abstract

In the multiprocessor systems data can reside in different cache memory levels as in the main memory. Maintaining data coherence between different caches and main memory is known as the cache coherence problem, which can be solved with the use of protocols such as MESI (Modified-Exclusive-Shared-Invalid). This paper presents the development of a prototype whose objective is the explanation of the aforementioned protocol in courses of advanced computer architectures seen at the engineering level. The prototype emulates the operation of a multiprocessor system with two processors and the main memory using two Arduino boards. The system interacts with the user through means of buttons to read and modify data from the memory, as well as the screens that display the contents of the caches and the state in which they are located.

Keywords: *Advanced computer architectures, Arduino, cache coherence, MESI protocol.*

1. Introducción

El desarrollo de los sistemas de cómputo ha crecido de manera exponencial en los últimos años de manera que han surgido nuevas arquitecturas multiprocesador o multinúcleo que permiten a los programadores aprovechar los recursos para paralelizar los diferentes procesos o tareas que requiere realizar un microprocesador. El cómputo paralelo surge para realizar tareas simultáneas de procesamiento de datos con el objetivo de aumentar la velocidad computacional [Tanenbaum, 2000]. En ocasiones, se utiliza una memoria especial de muy alta velocidad, llamada caché la cual se emplea en los sistemas de computadora para compensar la diferencia de velocidad entre el tiempo de acceso a la memoria principal y la lógica de procesador [Mano, 1994].

Uno de los inconvenientes que surgen cuando se trabaja con dos o más procesadores es el problema de la coherencia de caché, el cual ocurre cuando dos o más procesos ven a la memoria compartida a través de diferentes cachés, por lo cual existe el peligro de que una caché contenga datos actualizados, mientras que otra contenga datos erróneos [Stallings, 2006].

Para atacar el problema de coherencia de cachés, existen diferentes soluciones como los protocolos de sondeo, protocolo de invalidación de tres estados (MSI), el protocolo de invalidación de 4 estados (MESI), entre otros [Arnau, 2012].

El protocolo MESI, es un protocolo de coherencia de caché y coherencia de memoria que también se le conoce como protocolo Illinois por haber sido publicado por investigadores de la misma Universidad en 1984 [Papamarcos, 1998] variantes de este protocolo se usan en microprocesadores modernos tales como la familia Pentium de Intel [INTEL, 2018]. Dicho protocolo trabaja con 4 estados los cuales son:

- Modificado (Modified M): Indica que el valor en la caché ha sido modificado, por lo tanto, es necesario que los datos sean escritos otra vez en la memoria principal antes de realizar una lectura.
- Exclusivo (Exclusive E): Significa que una caché tiene una copia del bloque y no ha sido modificada (es decir la memoria principal está actualizada).
- Compartido (Shared S): Significa que potencialmente dos o más procesadores tienen este bloque en su caché en un estado no modificado.
- Inválido (Invalid I): Indica que esta línea de caché ya no es válida.

El seguimiento del protocolo MESI se torna complicado si solo se consulta el diagrama de estados, por lo que se han desarrollado algunos simuladores que permiten la visualización de los cambios de estado del protocolo MESI. En [Kehagias, 2016] se presenta un simulador interactivo de propósito educativo para explicar el funcionamiento del protocolo MESI por medio de animaciones interactivas, representando un sistema de cómputo con tres núcleos. El simulador se desarrolló en Visual Studio utilizando Unity y scripts escritos en lenguaje C#, mientras que [Mallya, 2015] presenta un estudio similar al anterior acerca del

rendimiento de los protocolos de coherencia de caché utilizando gem5, el cual es un simulador para la investigación de sistemas de arquitectura de computadoras.
El trabajo reportado por [Gómez, 2009] presenta un simulador del protocolo MESI que se utiliza para mostrar el problema de coherencia de caché y explicar el proceso de la ubicación de la memoria caché en sistemas de memoria multinivel. Los resultados experimentales corresponden a un entorno real de enseñanza. [Laguéns, 2011] presenta un simulador de protocolos de coherencia de caché en flash que muestra animaciones interactivas los cambios. La herramienta es interactiva, lo cual facilita a alumnos observar los cambios de estado y transiciones. Ya que los autores consideran que otras herramientas son complejas y no permiten visualizar el funcionamiento de los protocolos. Por otra parte [Pimentel, 2013] centra su atención en el desarrollo de una herramienta que sirve para la comprensión de la administración de la información dentro de un sistema jerarquizado de memoria como sucede en los sistemas con procesadores de propósito general y particularmente en la memoria cache. Finalmente [Jones, 2018] emula el funcionamiento del protocolo MESI para una mejor comprensión a través de una aplicación web basada en Java Script, la cual consiste en un entorno con tres procesadores, una memoria, un bus de datos y un bus de direcciones que simulan el comportamiento de sistema multiprocesador.
Este trabajo aborda el desarrollo de un prototipo en el que se implementa el protocolo MESI con el propósito de ayudar a comprender el funcionamiento de dicho protocolo a través del uso de herramientas de desarrollo como la tarjeta Arduino. El prototipo representa una arquitectura de dos procesadores con un nivel de caché, conectados al sistema de memoria.
El desarrollo del trabajo presenta el diseño del sistema, los algoritmos implementados y finalmente las pruebas realizadas para la validación del prototipo.

2. Métodos

La figura 1 presenta el esquema básico del prototipo, el cual está integrado por dos tarjetas Arduino MEGA, cada una de éstas representa un procesador con sus respectivas operaciones de lectura y escritura que se disparan por medio de dos

botones, adicionalmente cuentan con un switch para elegir las direcciones de memoria (se pueden seleccionar 4 direcciones). El sistema de memoria (RAM estática) se representa a través de una tarjeta Arduino UNO.

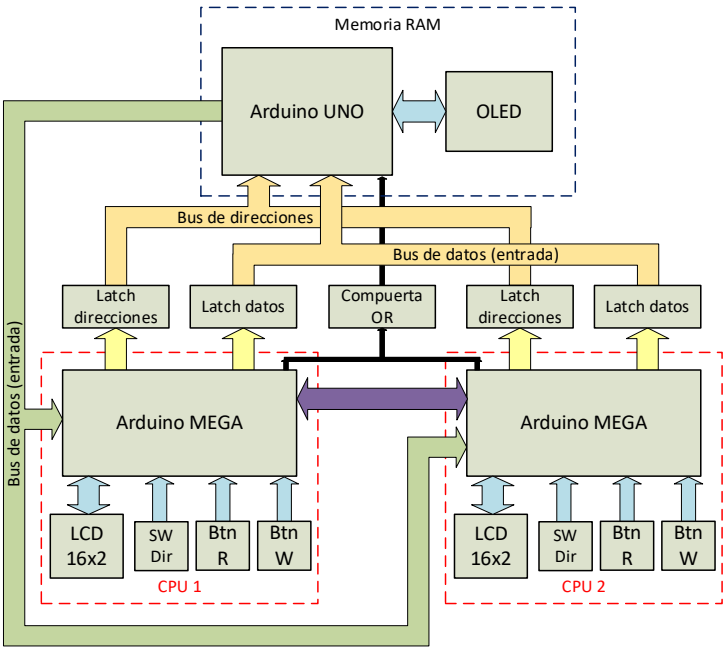

Figura 1 Esquema básico del prototipo.

Las pantallas LCD (2x16) muestran en tiempo real, el estado y valor interno de la caché de los procesadores. La pantalla OLED (128x64) despliega los valores internos de las cuatro direcciones de la memoria RAM. El resto de los bloques son circuitos de control adicionales, entre ellos los latches cuyo objetivo es evitar que los procesadores quieran escribir al mismo tiempo en los buses. Adicionalmente se requiere de una compuerta OR que sirva como control de acceso a la memoria. La elección de las tarjetas Arduino, se realizó con base en las características más apropiadas de cada tarjeta, por ejemplo, para implementar el diseño de los procesadores se requiere un número importante de conexiones de entrada/salida por lo que se seleccionó la placa Arduino MEGA debido a que cuenta con 54 entradas digitales. Para el diseño de la memoria basto con las 14 entradas digitales que ofrece la placa Arduino UNO. En la tabla 1 se presentan las principales características de las tarjetas utilizadas [Arduino, 2018].

Tabla 1 Características de las tarjetas Arduino utilizadas.

Característica	Arduino UNO	Arduino MEGA
Microcontrolador	ATmega328P	ATmega2560
E/S digitales	14	54
Salidas PWM	6	15
Entradas analógicas	6	16
Memoria Flash	32 kb	256 kb
SRAM	2 kb	8 kb
Frecuencia de reloj	16 MHz	16 MHz

Funcionamiento de la caché en el prototipo

El caché del procesador está representado como una variable dentro de Arduino la cual almacena los datos de la dirección de memoria (dos bits), el estado en el que se encuentra la caché (dos bits) y los datos de lectura (cuatro bits). Se decidió trabajar con un nivel de caché que permitiera almacenar la información de una dirección de memoria a la vez. Los valores contenidos en la caché están disponibles para el otro procesador por medio de la conexión entre los dos Arduinos. La figura 2 muestra de forma detallada las principales conexiones al bloque.

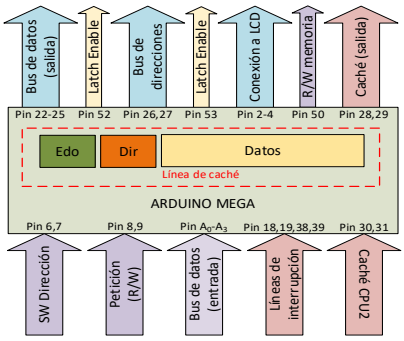

Figura 2 Conexiones principales de cada procesador.

Diseño de la memoria

El diseño de la memoria se basa en el funcionamiento de una memoria RAM estática, la cual es utilizada especialmente para buffers de alta velocidad, cachés y registros [Dunning, 2018]. Cuenta con un espacio de 4 direcciones con un ancho de palabra de 4 bits los cuales están representados como una matriz de enteros dentro de Arduino. Las conexiones necesarias para el funcionamiento son: la señal

de habilitación del chip, una señal que indique la operación que debe realizar (lectura o escritura), conexión para recibir los datos de entrada y salida, así como la dirección de memoria, tal y como se observa en la figura 3.

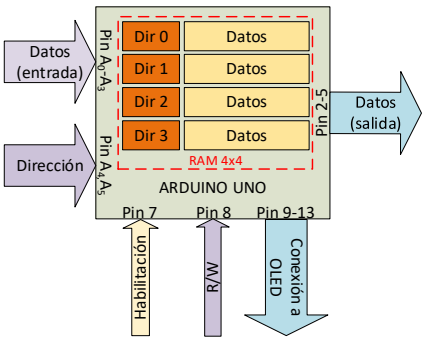

Figura 3 Conexiones principales de la memoria.

Ciclos de lectura y escritura a memoria

En el ciclo de lectura el procesador coloca la dirección previamente leída por los switches en el bus de direcciones y habilita el latch asociado, a la memoria se le indica que se hará una operación de lectura y se activa la señal de habilitación. Por su parte la memoria realiza las operaciones que se observan en el diagrama de flujo de la figura 4. Una vez finalizado el proceso, el procesador lee los datos del bus de entrada y los coloca en la línea de caché.

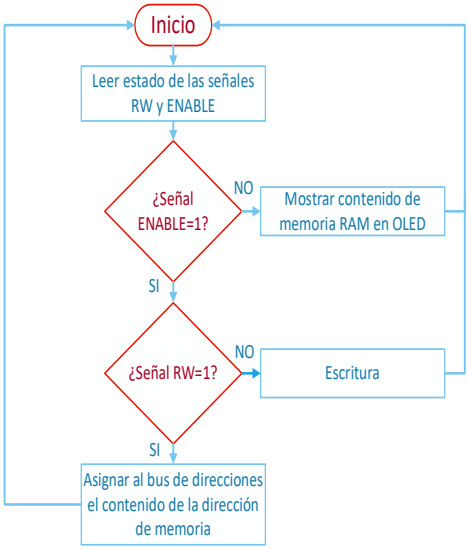

Figura 4 Ciclo de lectura de la memoria.

En el ciclo de escritura, el procesador coloca desde la caché el valor de la dirección y los datos en los respectivos buses, habilitando los latches. Inmediatamente, se le indica a la memoria que se hará una operación de escritura y se activa la señal de habilitación para realizar las operaciones que se describen en el diagrama de flujo de la figura 5.

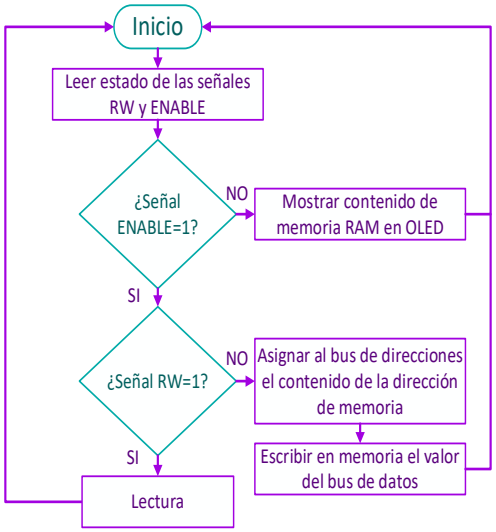

Figura 5 Ciclo de escritura de la memoria.

Algoritmo del protocolo MESI

Al realizarse una petición de lectura, el algoritmo del protocolo para dos procesadores tiene un funcionamiento que se describe de la siguiente forma:

- Si el procesador que activa la lectura se encuentra en el estado ‘I’ y el otro procesador se encuentra en el estado ‘I’, se lee directamente el dato desde la memoria y el procesador que activó la lectura cambia su estado a ‘E’.
- Si el procesador que activa la lectura se encuentra en el estado ‘E’ o ‘M’ y el otro procesador se encuentra en el estado ‘I’ se lee directamente el valor desde la caché y se mantiene en el mismo estado. El proceso anterior se repite si ambos procesadores se encuentran en el estado ‘S’.
- Cuando el procesador que activa la lectura se encuentra en estado ‘I’ y el otro procesador se encuentra en el estado ‘E’, ambos pasan al estado ‘S’ y el procesador que activa la lectura lee los datos directamente de la caché del otro procesador.

- Cuando el procesador que activa la lectura se encuentra en estado ‘I’ y el otro procesador se encuentra en estado ‘M’, este último debe cambiar su estado a ‘S’, actualizar el dato en memoria y una vez que concluye, el primer procesador lee directamente desde la memoria y cambia su estado a ’S’.

La figura 6 presenta el diagrama de flujo que representa el proceso de petición de lectura.

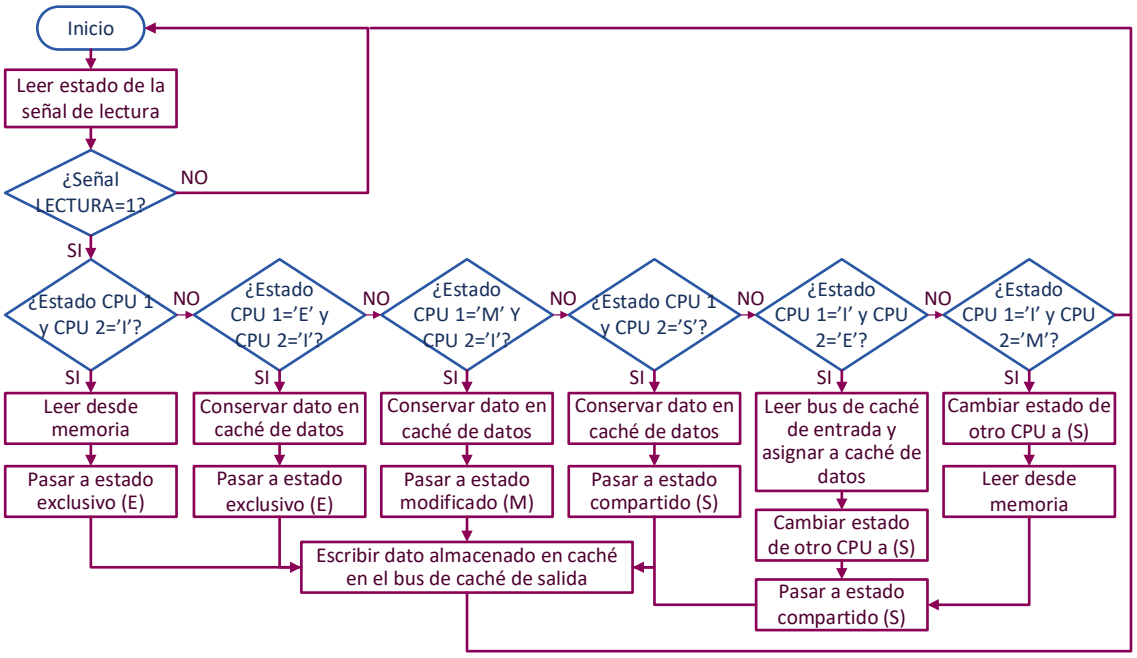

Figura 6 Diagrama de flujo que representa la petición de lectura.

Cuando la petición realizada es de escritura, el protocolo realiza lo siguiente:

- Si ambos procesadores se encuentran en el estado ‘I’, el procesador que activa la escritura lee el dato directamente desde memoria y una vez que lo tiene en su línea de caché lo modifica y cambia su estado en ‘M’.
- Cuando el procesador que activa la escritura se encuentra en el estado ‘E’ o ‘M’ y el otro procesador se encuentra en el estado ‘I’, se modifica el valor directamente en la línea de caché y se mantienen en el mismo estado.
- Si ambos procesadores se encuentran en el estado ‘S’ además de realizar el proceso del punto anterior, el procesador que activa la escritura actualiza el

valor en memoria, éste cambia su estado a ‘E’ y cambia el estado del otro procesador a ‘I’.

- Si el procesador que activa la escritura se encuentra en el estado ‘I’ y el otro procesador en el estado ‘E’ se debe leer el dato directamente desde la línea de caché, modificar el valor, actualizarlo en memoria y cambiar el estado a ‘E’, mientras que el otro procesador pasa al estado ‘I’.
- Si el procesador que activa la escritura se encuentra en estado ‘I’ y el otro procesador se encuentra en estado ‘M’, este último debe actualizar el dato en memoria, el primer procesador lee directamente de la línea de caché y el resto de la secuencia del punto anterior se mantiene.

La figura 7 presenta el diagrama de flujo que representa la petición de escritura.

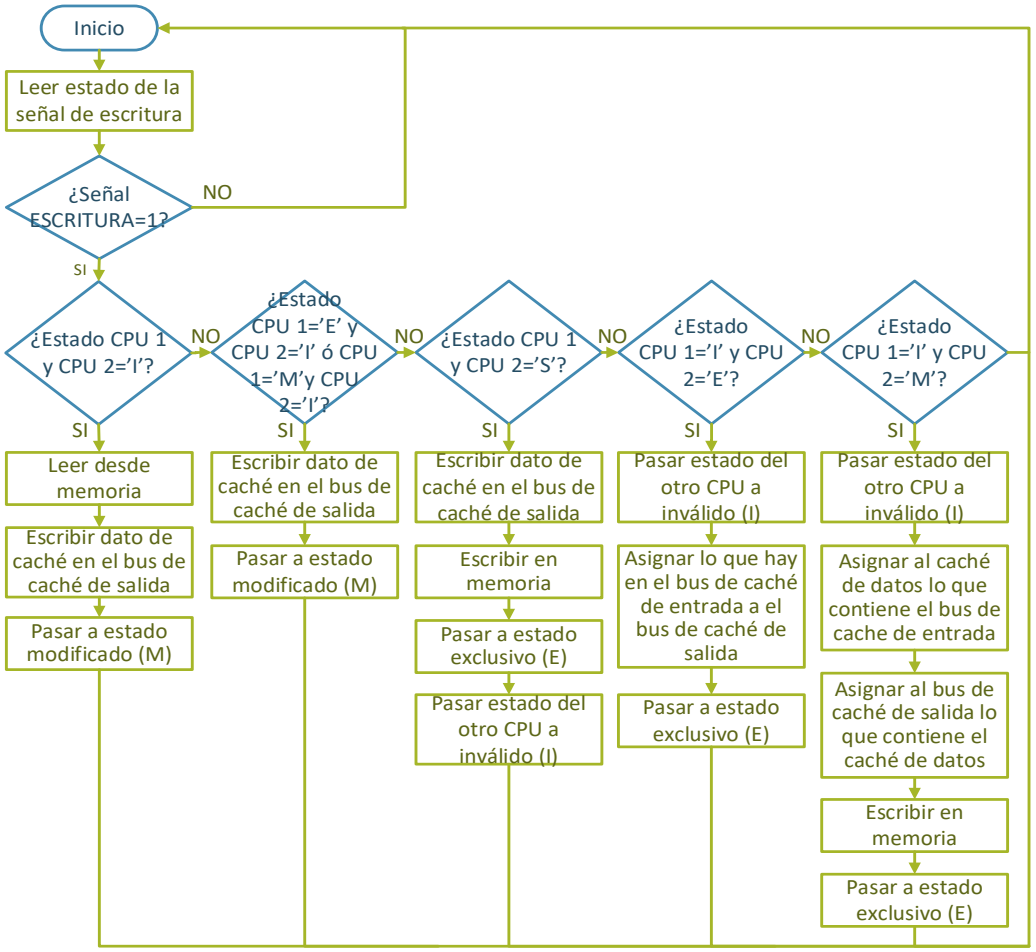

Figura 7 Diagrama de flujo que representa la petición de escritura.

3. Resultados

A continuación, se muestran los resultados de la implementación del protocolo MESI. En la figura 8 se observa cómo el prototipo muestra en las pantallas LCD los resultados de las operaciones de lectura y escritura que se realizan en la caché, así como el contenido de la memoria que se despliega en la pantalla OLED.

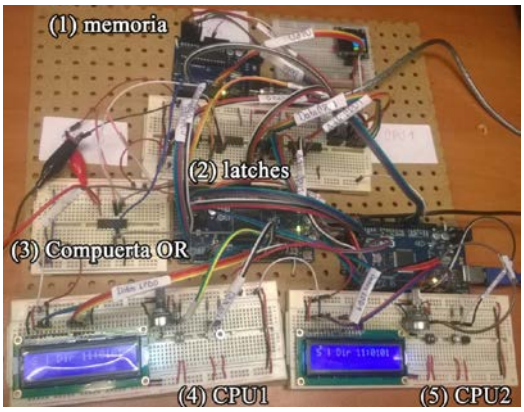

Figura 8 Prototipo ensamblado.

Para validar el funcionamiento de los algoritmos implementados, se realizaron pruebas aplicando las operaciones de lectura y escritura en ambos procesadores basados en ejemplos de la literatura utilizada [Arnau, 2012].

La primera prueba consistió en realizar las siguientes operaciones:

- EL CPU 2 realiza una solicitud de lectura a una dirección de memoria, por lo tanto, al iniciar ambos cachés en estado ‘I’ el CPU 2 debe leer directamente el valor de memoria y cambiar el estado a ‘E’.
- El CPU 2 realiza una solicitud de escritura, por lo tanto, modifica el dato en caché que pasa al estado ‘M’.
- El CPU 1 realiza una petición de lectura a la misma dirección, por lo cual los datos son proporcionados por la caché del CPU 2, pasando al estado ‘S’ el caché de ambos procesadores.
- El CPU 1 realiza la petición de escritura directamente en su caché, cambiando el estado a ‘E’ y pasando a ‘I’ el caché del procesador 2.
- El CPU 2 realiza la petición de lectura a la misma dirección, por lo que ambos cachés pasan al estado ‘S’.

Se utilizaron las cuatro direcciones de memoria, además la secuencia se repitió en numerosas ocasiones, esto con la finalidad de verificar que los resultados obtenidos son correctos, dichos resultados se muestran en la tabla 2. La figura 9 presenta algunas imágenes de las pruebas.

Tabla 2 Resultados de las pruebas.

Operación	Origen de los datos	Caché CPU 1	Caché CPU 2	Datos	Dirección
P2:Read	Memoria	I	E	0000	11
P2:Write	Caché P2	I	M	0011	11
P1:Read	Caché P2	S	S	0011	11
P1:Write	Caché P1	E	I	0101	11
P2:Read	Caché P1	S	S	0101	11

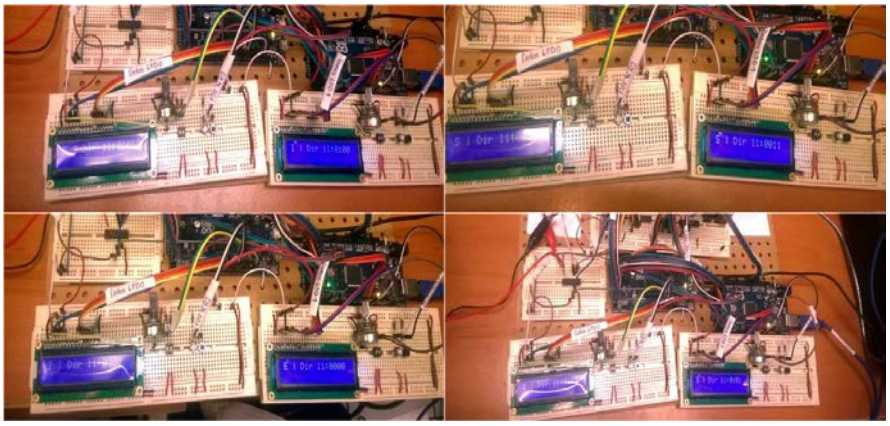

Figura 9 Imágenes de los resultados de las pruebas.

4. Discusión

Durante el desarrollo del prototipo se presentaron diferentes problemas debido a la sincronización de la memoria con los procesadores. La falta de experiencia en el diseño provocó retraso en el desarrollo que se fue subsanando, por ejemplo, se recurrió al uso del sistema de interrupciones para que el funcionamiento de uno de los procesadores atendiera las peticiones donde se requiriera invalidar el estado de los demás CPU´s, de tal forma que la cantidad de conexiones entre dispositivos aumentó.

Arduino es la base del desarrollo del prototipo, debido a que facilitó el desarrollo y permitió integrar varios componentes de manera sencilla. El objetivo era visualizar

el funcionamiento del protocolo MESI, observando en todo momento las transiciones, los estados en el que se encuentran las caches y los valores que hay en la memoria.

Como se observó en la sección anterior, las pruebas realizadas cumplen con los resultados que muestra la literatura. Por lo anterior se puede afirmar que el objetivo principal se alcanzó, ya que el prototipo muestra el funcionamiento del protocolo MESI. El prototipo puede ser un recurso de apoyo para cursos de arquitectura de computadoras a nivel licenciatura y maestría, debido al grado de sencillez con la que trabaja.

5. Conclusiones

El protocolo MESI es una de las soluciones que existen para atacar el problema de la coherencia de caché. La dificultad de aprendizaje del protocolo se debe a la necesidad de recursos visuales que permitan al estudiante observar de forma visual los cambios de estado que ocurren, cuando uno o más procesadores requieren acceder a una misma dirección de memoria.

El desarrollo del prototipo permitió enfrentarse en un escenario real al problema de coherencia de caché y sincronización de multiprocesadores, ya que, cada Arduino que simula el funcionamiento de un microprocesador que trabaja de forma independiente, hace que ningún dispositivo tenga la certeza de las acciones que van a ocurrir, por lo que el algoritmo debe prever las posibles situaciones que puedan desencadenarse.

El desarrollo del prototipo implementado en placas Arduino disminuyó gran parte del trabajo, ya que debido a la gran cantidad de recursos que contiene y la facilidad para el manejo de dispositivos como las pantallas y los botones permitieron enfocar el trabajo en el desarrollo de la programación.

Como trabajo futuro se propone mejorar el diseño del prototipo de forma que se requieran menos conexiones, así como la integración de más de dos procesadores trabajando en conjunto y dos niveles de caché, además de considerar circuitos impresos para una estética más profesional.

6. Bibliografía y Referencias

[1] Arduino. Arduino Products: https://www.arduino.cc/en/Main/Products.

[2] Arnau, L. V. Organización de Computadores, Capítulo 5 Multiprocesadores, Universidad de Valencia, Marzo, 2012.

[3] Dunning D. Tipos de RAM: estática y dinámica: https://techlandia.com/tipos-ram-estatica-dinamica-info_290309/.

[4] Gómez Luna, J., Herruzo Gómez, E., Benavides Benítez, J. I. MESI Cache Coherence Simulator for Teaching Purposes. CLEI, 2009.

[5] INTEL, Intel 64 and IA-32 architectures software developer's manual. Volume 1: Basic Architecture, Mayo 2018.

[6] Jones, J. MESI cache coherency animation, Computer Architecture Animations. https://www.scss.tcd.ie/Jeremy.Jones/vivio/caches/MESIHelp.htm, 2018.

[7] Kehagias, D., Raptis, I. An Interactive MESI Cache Coherence Simulator for Educational Purposes. Proceedings of the 20th Pan-Hellenic Conference on Informatics, 61, November, 2016.

[8] Laguéns, A. A., Mir, S. B., Quintana Orti, E. S. An Interactive Animation for Learning How Cache Coherence Protocols Work. Proceedings of INTED2011 Conference, 7-9, March, 2011.

[9] Mallya, N. B., Patil, G., Raveendran, B. Simulation based Performance Study of Cache Coherence Protocols. Nanoelectronic and Information Systems (iNIS), 2015 IEEE International Symposium, 125-130, December, 2015.

[10] Papamarcos, M. S., and Patel, J. H. A low-overhead coherence solution for multiprocessors with private cache memories. 25 years of the international symposia on Computer architecture, 284-290, 1998.

[11] Pimentel, C. J., Pérez, R. P, Rivera, Z. I. Programa de simulación del protocolo de coherencia MESI, Boletín UPIITA, Núm. 35, Enero, 2013.

[12] Stallings, W. Computer organization and architecture: designing for performance, 8th Edition, Pearson Education, India, 2006.

[13] Tanenbaum, A. S. Organización de computadoras: un enfoque estructurado, 4ta edición, Pearson education, México, 2000.