AOC2 Proyecto 2 - Jerarquía de memoria de datos

Curso 2023-2024

Samuel Corpas Puerto (875301)
Daniel Salas Sayas (875308)

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Universidad
Zaragoza

Índice

1. Resumen del proyecto

Este proyecto se enfoca en el desarrollo de una estructura de memoria más avanzada que la utilizada en proyectos previos, incorporando varios componentes que se comunican a través de un bus semi-síncrono. Se compone de una memoria caché (MC), una memoria de datos (MD) con una velocidad reducida comparada con la del proyecto anterior, y una segunda memoria de datos (MD Scratch), considerablemente más rápida que la MD y cuyos datos no se almacenan en la MC.

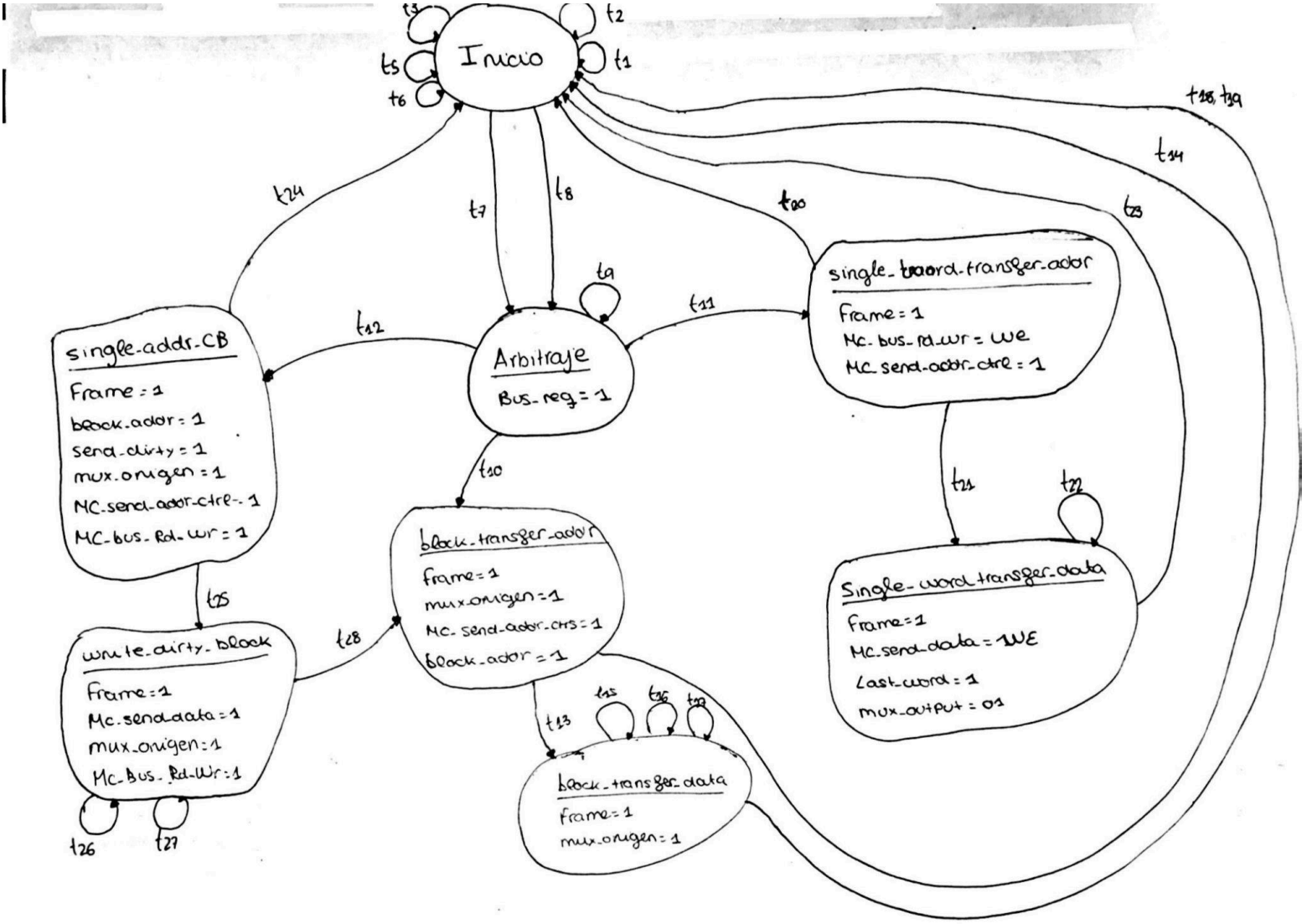
El propósito central es crear un controlador para la MC que procese de manera eficiente las solicitudes del procesador, coordinando las transferencias necesarias a través del bus. Este proyecto introduce contadores de desempeño adicionales para monitorear la operatividad de la MC.

Respecto a la arquitectura del sistema, se ha añadido un dispositivo de entrada/salida (IO Master) que funciona como maestro del bus junto con el controlador de la MC. Se ha incorporado un sistema de arbitraje para mitigar conflictos entre los maestros y asignar prioridades de manera dinámica para garantizar un reparto equitativo del bus.

La MC diseñada tiene una capacidad de 128 bytes y es asociativa por conjuntos con dos vías. Implementa políticas como la actualización diferida en las escrituras y un método de reemplazo FIFO. En paralelo, la MD Scratch se maneja directamente desde el controlador de la MC, funcionando con baja latencia y sin interacciones con la MC.

El controlador de la MC además cuenta con un sistema de gestión de estados para tratar los errores de memoria, activando una señal de error cuando detecta direcciones no reconocidas o accesos no alineados, entre otras incidencias.

2. Diagrama de estados de la unidad de control



Transición	Condición	Señales
t1	RE = 0 AND WE = 0	ready = 1
t2	(RE = 1 OR WE = 1) AND unaligned = 1	ready = 1, next_error_state = memory_error, load_addr_error = 1
t3	RE = 1 AND internal_addr = 1	ready = 1, max_output = 10, next_error_state = no_error
t4	WE =1 AND internal_addr = 1	ready = 1, next_error_state = memory_error, load_addr_error = 1
t5	RE = 1 AND hit = 1	ready = 1, inc_r = 1, mux_output = 00

t6	WE = 1 AND hit = 1	ready = 1, inc_w = 1,MC_WE1 = hit1, MC_WE0 = hit0, mux_origen = 0, update_dirty = 1
t7	addr_non_cacheable = 1	Bus_req = 1, one_word = 1
t8	(RE = 1 or WE = 1) AND hit = 0 AND addr_non_cacheable = 0	Bus_req = 1, inc_m = 1
t9	Bus_grant = 0	
t10	Bus_grant = 1 AND dirty_bit = 0 AND addr_non_cacheable = 0	
t11	Bus_grant = 1 AND addr_non_cacheable = 1	
t12	Bus_grant = 1 AND dirty_bit = 1 AND addr_non_cacheable = 0	
t13	Bus_DevSel = 1	
t14	Bus_DevSel = 0	ready = 1, next_error_state = memory_error, load_addr_error = 1
t15	Bus_TRDY = 0	
t16	Bus_TRDY = 1 AND via_2_rpl = 0 AND last_word_block = 0	MC_WE0 = 1, inc_w = 1, count_enable = 1
t17	Bus_TRDY = 1 AND via_2_rpl = 1 AND last_word_block = 0	MC_WE1 = 1, inc_w = 1, count_enable = 1
t18	Bus_TRDY = 1 AND via_2_rpl = 0 AND last_word_block = 1	MC_WE0 = 1, inc_w = 1, MC_tags_WE = 1, last_word = 1, count_enable = 1
t19	Bus_TRDY = 1 AND via_2_rpl = 1 AND last_word_block = 1	MC_WE1 = 1, inc_w = 1, MC_tags_WE = 1, last_word = 1, count_enable = 1
t20	Bus_DevSel = 0	ready = 1, next_error_state = memory_error, load_addr_error = 1
t21	Bus_DevSel = 1	
t22	Bus_TRDY = 0	
t23	Bus_TRDY = 1	ready = 1
t24	Bus_DevSel = 0	ready = 1, next_error_state = memory_error, load_addr_error = 1

t25	Bus_DevSel = 1	
t26	Bus_TRDY = 0	
t27	Bus_TRDY = 1 AND last_wors_block = 0	count_enable = 1
t28	Bus_TRDY = 1 AND last_wors_block = 1	count_enable = 1, block_copied_back = 1, update_dirty = 1, last_word = 1, inc_cb = 1

A la hora de afrontar el diseño de nuestra máquina de estados diferenciamos primeramente entre 3 casos, los casos relacionados con la memoria Scratch, los casos de acierto, ya definido previamente, y el caso de fallo, donde se abren distintos caminos.

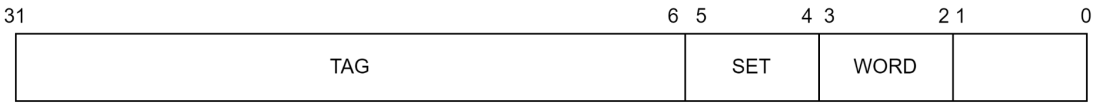
- Primero para los casos relacionados con la Scratch, una vez detectamos que la dirección pertenece a la Scratch necesitamos solicitar el bus, poniendo Bus_Req a 1, una vez el árbitro nos conceda el bus, con Bus_Grant tenemos que activar Frame, que indica que el bus está siendo utilizado, y tenemos que enviar primero la dirección sobre la que queremos actuar en Scratch, esperando hasta que la Scratch acepte esta dirección, con bus_devSel, y a continuación pasamos a mandar el dato de la palabra,con Frame activado y activando last_word, debido a que es una única palabra. Una vez la memoria sea capaz de realizar la operación en ese ciclo, que lo sabremos con la señal bus_TRDY, volvemos a inicio permitiendo que siga avanzando el procesador.
- Los aciertos en MC estaban ya definidos.
- En los fallos de MC se abren dos posibilidades, fallo sucio y fallo limpio. Los fallos limpios se tratan en reemplazar el bloque de caché por el que nos interesa de MD. Para eso primero tenemos que solicitar el bus de la misma forma que en la memoria Scratch, y una vez nos lo dan activamos Frame y tenemos que pasarle la dirección, siempre y cuando la dirección sea correcta y la acepte MD, de la primera palabra del bloque, esto nos lo permite la señal block_addr. Una vez la dirección ha sido enviada hay que enviar los datos, manteniendo Frame y esperando a que la memoria sea capaz de hacer la operación. Por cada palabra que se mande se activará count_enable para detectar cual es la última en enviar y al llegar a esta escribir el nuevo tag y volver a inicio para dar hit.
- En cuanto a los fallos sucios, para detectarlos tenemos que comprobar el dirty_bit, a continuación solicitamos el bus y una vez lo tengamos activamos Frame. De la misma forma que se ha realizado en el fallo limpio y en la Scratch comprobamos que la dirección sea aceptada por la memoria y mandamos la dirección de la primera palabra del bloque a través del bus activando las señales necesarias, una vez se ha pasado la dirección,

pasamos los datos manteniendo Frame activado y siempre que la memoria sea capaz de hacer la operación, escribimos la palabra en su dirección de memoria correspondiente aumentando el count_enable hasta llegar a last_word_block que nos indica que se ha copiado todo el bloque en memoria. Una vez se ha copiado el bloque, hay que realizar el nuevo reemplazo de bloque pero ahora en caché. Esto se podía hacer volviendo al ciclo inicial, lo que conlleva otro miss y solicitar otra vez el bus, o pasar directamente a la transferencia de dirección de bloque de MD a MC, de esta forma mantenemos Frame activado y el bus lo empleamos nosotros en todo momento. Nosotros hemos decidido implementar la segunda opción ya que consideramos que al ser todo la misma operación es preferible mantener en nuestra posesión el bus.

Aparte de esta máquina de estados que controla la MC, también hemos sacado otra relacionada con los fallos de memoria aunque mucho más simple. A la hora de atacar esta máquina de estados vamos a detectar 3 posibles fallos, un dato mal alineado, la intención de escribir en un registro que es solo de lectura, y que la memoria no acepte la dirección que se le quiere mandar.
Este autómata es de 2 estados, No_error y Memory_error y partimos del primero. En caso de que un dato esté mal alineado se detectará en el estado del autómata anterior denominado como Inicio comprobando que la señal unaligned esté activada. El segundo caso lo detectamos también en Inicio y comprobaremos que no estén activadas WE y internal_addr, que es la señal que indica que se quiere realizar la acción sobre el registro de MC. Por último, el tercer caso nos lo podemos encontrar siempre que se envíe una dirección por el bus y la memoria de destino no reconozca esta dirección, es decir, siempre y cuando la señal bus_devSel esté desactivada en estos estados correspondientes. En estos casos se volverá al estado de inicio. Siempre que ocurra una de estas tres opciones, se pasará al estado memory_error y se activará la señal load_addr_error que permitirá cargar en el registro interno de MC, que es el registro de error, la dirección errónea, y activaremos a su vez memready para que el procesador siga avanzando.
Esto cambiará cuando se lea el registro de error, es decir, cuando estén activadas las señales RE y internarl_addr, cuando esto ocurra se pasará al estado no_error.

3. Descomposición de la instrucción

Como se ha mencionado en el resumen del proyecto, se tiene una memoria caché asociativa con 2 vías, es decir, por cada set habrá 2 bloques de palabras. Cada bloque, está conformado por 4 palabras. Con toda esta información, además de la separación de bits que se hace en el archivo MC_datos_2024.vhd obtenemos la siguiente descomposición de bits para el direccionamiento a la MC:



4. Análisis de latencias

Para el cálculo de estos datos, empleamos la simulación en GTKWave para contabilizar el número de ciclos de cada caso. Para el arbitraje, como no siempre es un valor constante, hemos cogido un valor de 1,5 que es el que se nos recomienda en el guión de la práctica.

Evento	**Dirección**	**Datos**	**Total**
CrB (MD)	1	6 + 3 * 2 = 12	13
CwB (MD)	1	12	13
CrW (MDScratch)	1	2	3
CwW (MDScratch)	1	1	2
CrW o CwW (IO_REG)		1	1

5. Cálculo de ciclos efectivos

A partir de los datos calculados en el apartado anterior, podemos realizar la expresión de cálculo de los ciclos efectivos. Puesto que el número de ciclos que consume el arbitraje no es siempre constante, se ha tomado como valor medio 1,5 que es el que se recomienda en el guión del trabajo.

La expresión es la siguiente:

Ceff = 1 + ((1 * num_rh) + (1 * num_wh) + ((1,5 + CrB +1) * num_fallos_limpios) + ((1,5 + CwB) * num_fallos_sucios) + ((1 + CrB) * num_fallos_sucios) + ((1,5 + CrW) * num_rw_scratch) + ((1,5 + CwW) * num_ww_scratch) + (1 * num_accesos_IOREG_&_errorreg)) / num_refs

Fraccionando la fórmula por partes, primero tenemos el ciclo de acceso con valor 1, a continuación le siguen los distintos casos que nos podemos encontrar a la hora de analizar una dirección. Los primeros dos casos son tanto el acierto en escritura como en lectura, siendo cada acierto un ciclo y multiplicándose por el número de ocasiones en lo que ocurra determinada acción, le siguen los fallos limpios, iguales para escritura y lectura, para los cuales tenemos que contar los ciclos de arbitraje y los ciclos en leer un bloque de MD, además de sumarle previamente un ciclo debido al ciclo de acierto una vez se ha traído ya el bloque a caché y multiplicar todo esto por las instrucciones en las que haya fallo limpio. A continuación tenemos los fallos sucios, en los cuales se cuentan los ciclos de arbitraje, los ciclos que tardas en escribir un bloque en MD y se le suma también los costes del reemplazo de bloque limpio sin pasar por el arbitraje y multiplicar todo esto por las instrucciones en las que haya fallo sucio. Seguido a esto nos encontramos los tiempos relacionados con la scratch, en los cuales se suma el arbitraje y el tiempo de lectura en caso de esta o escritura en el caso contrario, multiplicándose por su determinado número de instrucciones. Por último los accesos a registros, tanto el registro interno de MC como los IO_reg tienen un coste 1 para lectura y escritura multiplicándose por el número de accesos a estos registros. Todo esto excepto el ciclo de acceso partido por el número de instrucciones.

6. Programas de pruebas

6.1. Explicación de los tests

Para comprobar el correcto funcionamiento de nuestro proyecto hemos decidido implementar distintos tests para cada una de las distintas implementaciones. Estos test tienen comentado su correspondiente código en ensamblador y la información acerca de sus loads y stores antes de su contenido de la memoria de instrucciones.

- En primer lugar, el test que hemos utilizado para comprobar lectura con acierto y fallo limpio es el que se proporciona como material del proyecto.
- Seguido a este, el test 2, lo implementamos para comprobar el correcto funcionamiento de las escrituras, tanto aciertos, como y fallos limpios y fallos sucios. El funcionamiento de este test es bastante similar al proporcionado de lecturas, buscando que realice un fallo limpio y un fallo sucio en todos los sets en ambas vías. Para comprobarlo primero hacemos dos stores para activar los bits de sucio del set 0, con distinto tags para ocupar las ambas vías y a continuación, otros dos stores con tags distintas que provoquen el copy-back. Seguido a esto entramos en un bucle en el que lo mismo, 4 stores, los dos primeros que activan los bits de sucio, y los dos siguientes que realicen el copy-back, después de esto se suma el desplazamiento para cambiar de conjunto y volvemos a la primera instrucción del bucle. En las vías se puede ver como en los dos primeros stores del bucle se escriben, en la primera palabra de los bloque de la vía 0 y en la segunda palabra de los bloques de la vía 1, los datos que se suman de 10 en 10 (16 en decimal) cada vez que se cambia de set, mientras que los segundos stores borran este dato y en caso de la vía 0 se escribe un 8 en la segunda palabra y en la vía 1 se escribe el dato que había en la palabra 2, en la palabra 1, y en la 2 se escribe un 8 también. En cuanto a la MD en las primeras palabras de los bloques al hacer copy-back en tercer store se escribe el número que había en la vía y el cuarto store lo hace en la segunda palabra de los bloques.
- El test 3 lo hemos utilizado para comprobar el correcto funcionamiento de la scratch, para ello debíamos de comprobar que se escribía y se leía correctamente. Para ello hemos decidido que este test lea y escriba en cada una de las direcciones de la scratch, de forma que el primer dato que se encuentra en la dirección 2 de la scratch, debido a que la 1 la modifica el IO_master. Una vez leemos el dato inicialmente con valor 1, le sumamos el dato y la dirección y lo escribimos en su dirección correspondiente.
- En cuanto al test 4 lo hemos empleado para el correcto funcionamiento de la gestión del data_abort. Para ello hemos querido comprobar los 3 tipos de errores posibles. En primer lugar queremos leer sobre una dirección acabada en 1, es decir mal alineada, de forma que salta el unaligned y cambiamos al estado de error. El siguiente tipo de error es que la memoria no reconozca la

dirección que se le pretende enviar, es decir, que cuando se pretenda enviar la dirección, bus_devsel sea 0. En este momento se pasará al estado de error. Por último teníamos que comprobar que si se pretende escribir en un registro solo de lectura saltar data_abort. Para ello probamos a realizar una escritura en el registro interno de la MC y se comprueba que se cambia al estado de error.
A su vez este test comprueba también que si leemos el registro de error, volvemos a pasar al estado de no error. Esto se comprueba con la subrutina que gestiona el abort en el programa proporcionado, el cual realiza un lectura de este tipo y una vez se ha leido el registro vuelve a la siguiente instrucción después del error. Esto es posible gracias a que siempre que salte uno de los errores nombrados anteriormente, no se pare el procesador y se le permita seguir avanzando.

- Por último hemos decidido plantear un test, en base a los creados anteriormente, en el que se compruebe todo en conjunto. En este test se realiza en primar un load de una dirección mal alineada, una vez vuelve se realizan operaciones para obtener datos guardados en la scratch y realizamos una escritura para comprobar que funciona correctamente. Una vez tenemos los datos que nos interesan realizamos un load y un store, los cuales completan el set 0 de las dos vías, siendo fallos limpios, a continuación sumamos un desplazamiento cambiando así de conjunto y entramos en un bucle en el que realizamos dos stores, uno en la vía 0 y otro en la vía 1, y de esta forma se ponen los bits de sucio. Ahora realizamos un load con tag distinto a los dos escritos anteriores, para que de fallo, y salte a hacer un copy-back con un load, y le sigue un store con otro tag para hacer un copy-back con un store. Una vez realizado el store, sumamos otra vez desplazamiento y volvemos a la primera instrucción del bucle con el siguiente set. De esta forma los resultados del test serian, en las primeras palabras de los bloques de MD a partir de la dirección 4 segundo bloque, se van sumando de 16 en 16, 10 en hexadecimal, de la misma forma ocurre a partir de la dirección 68 pero en este caso en la segunda palabra del bloque. En cuanto a los tags van cambiando en las distintas vías. Por último, en cuanto a contenido de las vías, en la vía 1, en el primer store del bucle, se escribe en la primera palabra del bloque el dato 10 (16 decimal) y se suma de 16 en 16 cada vez que se cambia de conjunto, ocurriendo de la misma forma en la vía 2, pero en este caso la segunda palabra del bloque se cambiará también a 8. Seguido a estos stores llega un load el cual realiza copy-back y cambia el contenido de la vía 0 y de la misma forma actúa el siguiente store pero en la vía 1

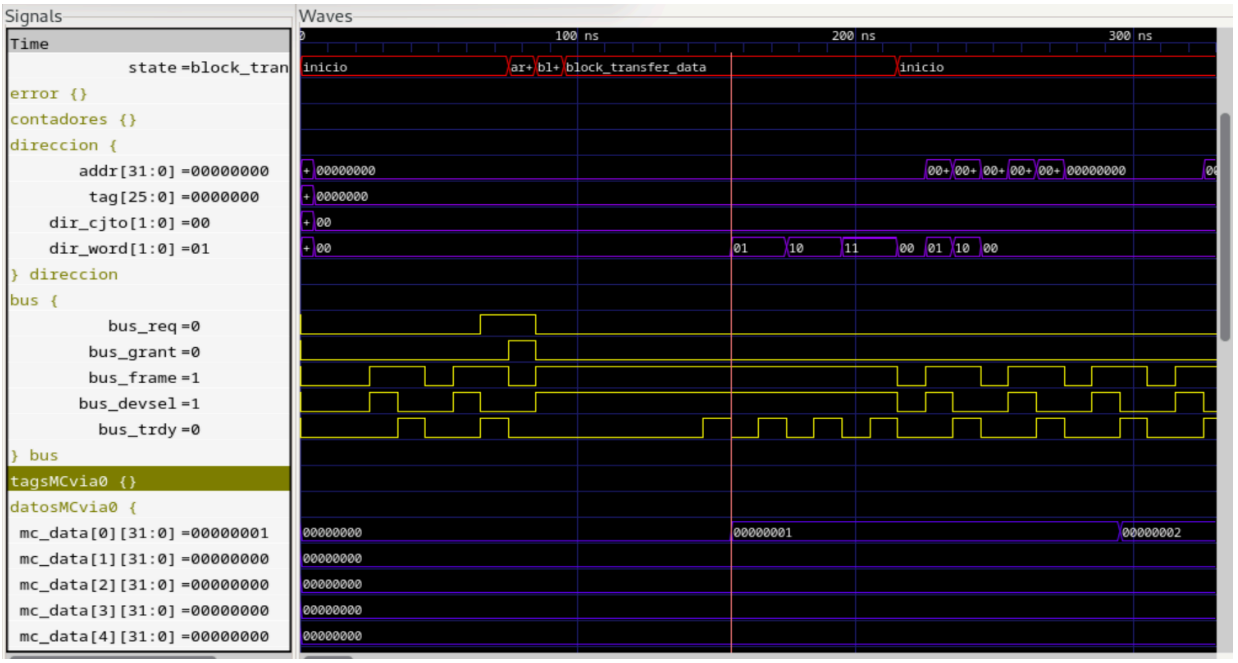
6.2 Puntos de interés

En la siguiente tabla se van a mostrar ejemplos de puntos de interés de los distintos programas

EVENTO	TEST	TAG	SET	EJEMPLO
Fallo limpio en lectura	1	0	0	Foto 1
Fallo limpio en lectura	1	3	2	Foto 2
Acierto en lectura	1	2	1	Foto 3
Fallo limpio en escritura	2	0	1	Foto 4
Acierto en escritura	2	2	2	Foto 5
Fallo sucio escritura	2	2	1	Foto 6
Fallo sucio en escritura	2	0	3	Foto 7
Fallo sucio en lectura	5	2	1	Foto 8
Lectura y escritura en Scratch	3	—	—	Foto 9
Error unaligned	4	—	—	Foto 10
Error addr_in_range	4	—	—	Foto 11
Error escritura reg solo lectura	4	—	—	Foto 12
Leer reg error	4	—	—	Foto 13
Escritura y	4	—	—	Foto 14

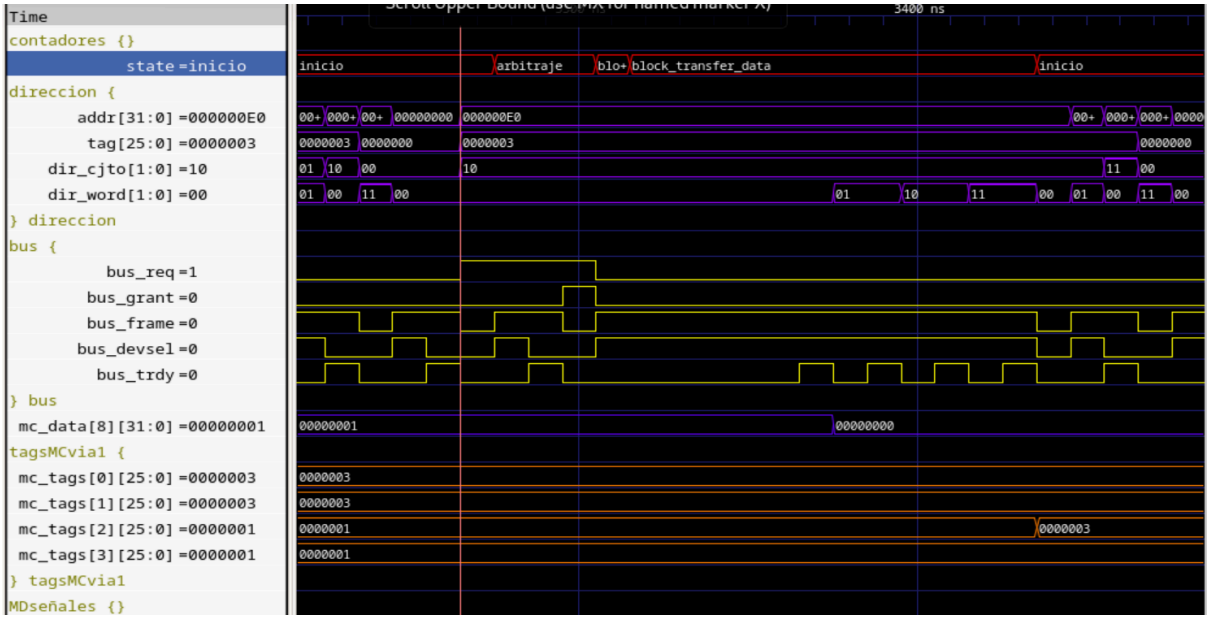
lectura en IO_reg				

Foto 1:



En este ejemplo podemos observar como se trae un bloque de MD a MC. Vemos un load con tag = 0 y set = 0 y la caché está vacía, luego hay un fallo y al estar el dirty_bit a 0 es un fallo limpio. Podemos observar como al pasar a arbitraje se solicita el bus con bus_req = 1 y cuando bus_grant = 1 pasamos a mandar la dirección activando el bus_frame. Como bus_devsel = 1 avanzamos a mandar los datos, mandando las palabras por ráfagas siempre que bus_TRDY sea 1, hasta llegar a last_word, que será la cuarta palabra. En este ejemplo podemos ver que solo se modifica la primera palabra del conjunto, y el tag no varía porque es 0.

Foto 2:

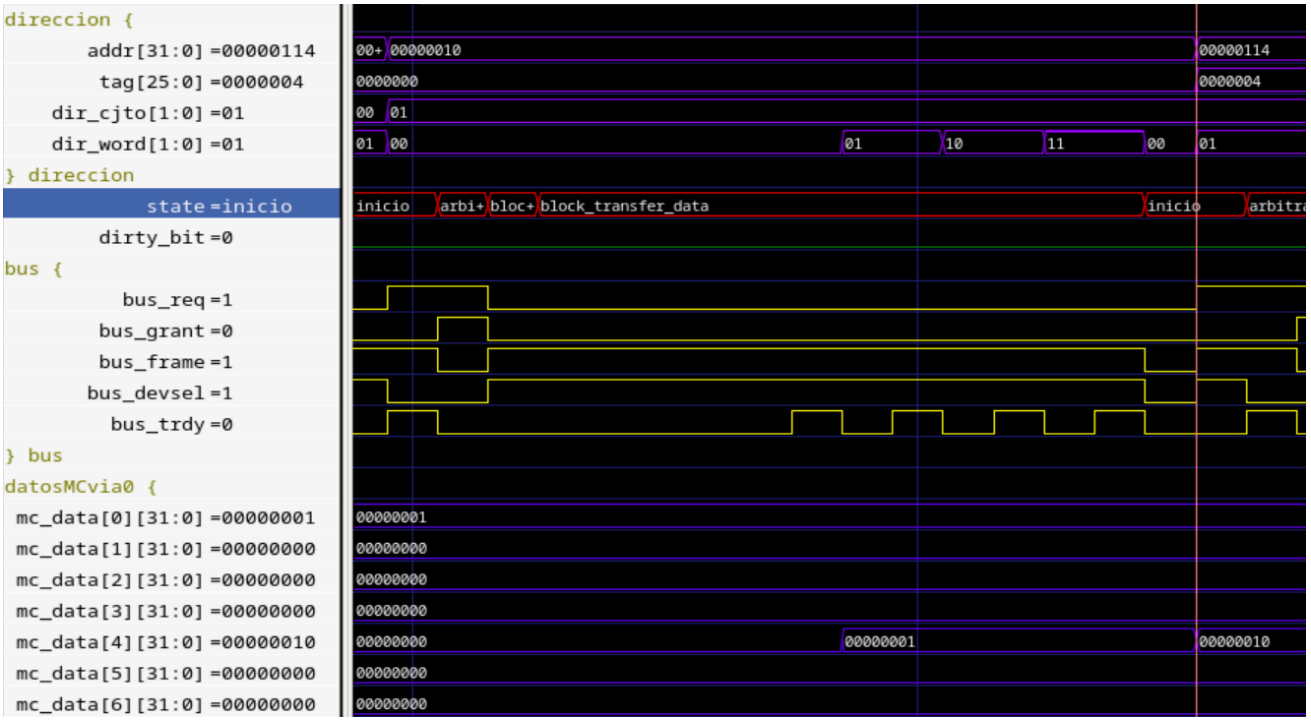


En esta imagen podemos ver cómo ocurre lo mismo que en el anterior pero en este caso con set 2 y tag 3. En este caso el bloque se reemplaza en la vía 1, en la cual en el conjunto 2 se encuentra tag1. Se puede observar el comportamiento del bus descrito anteriormente y los cambios de valor del tag al escribir la última palabra y el cambio en la primera palabra ya que el resto son iguales y no se aprecia el cambio de onda.

Foto 3:

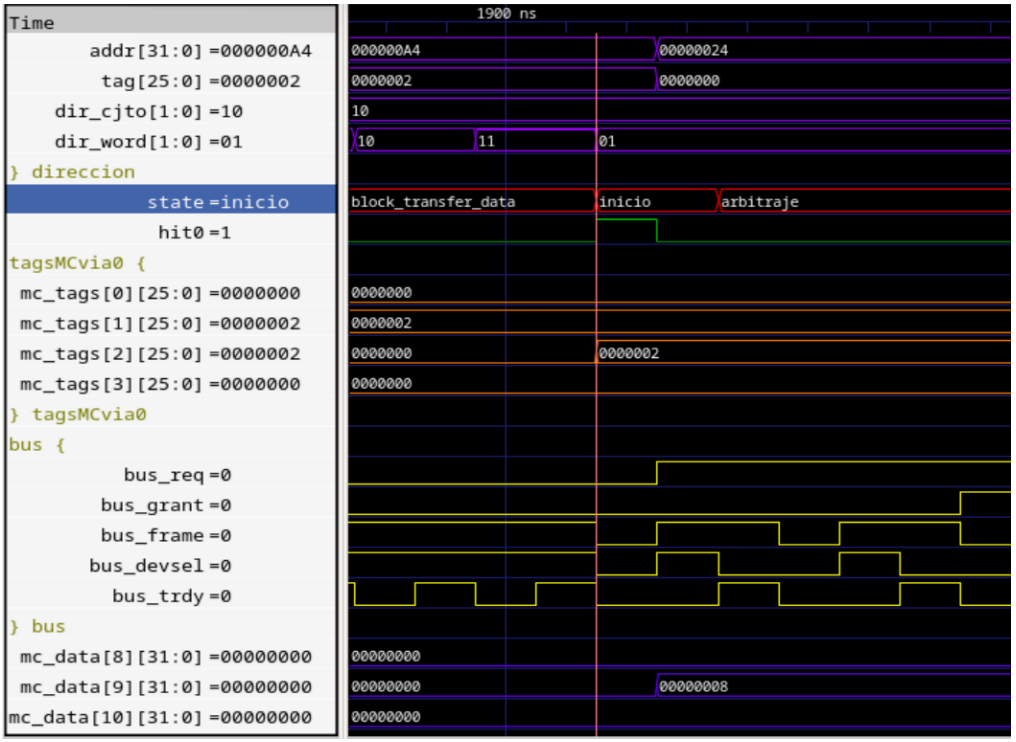
En esta imagen se está ejecutando instrucción posterior a un fallo limpio, con dirección 90, en el que se ha traído a memoria caché a la vía 0, en el set 1, el bloque compuesto por las palabras 90, 94, 98, 9C y con tag 2, lo que provoca que al querer leer la segunda palabra de este bloque, es decir, la dirección 94, haya un hit en vía 0.

Foto 4:



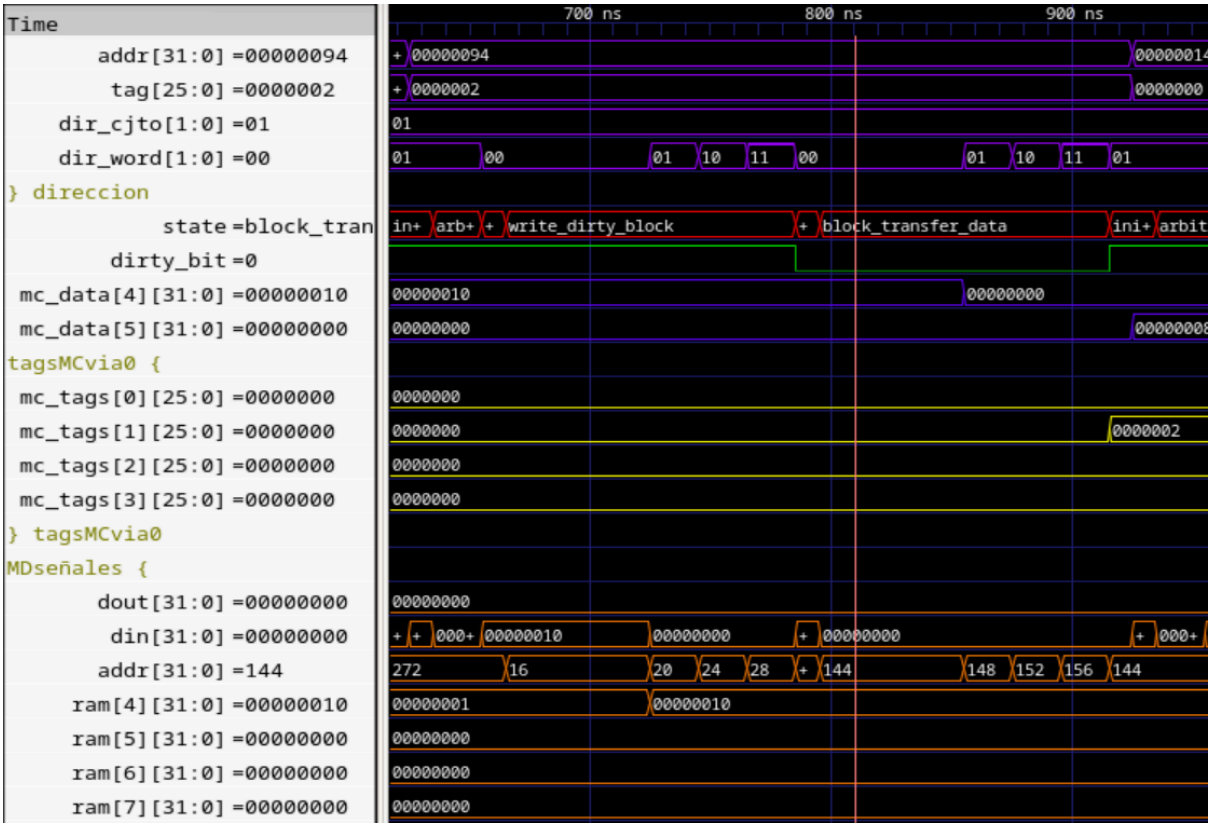
En esta imagen se puede apreciar un comportamiento bastante similar a las imágenes 1 y 2, aunque con una pequeña diferencia. El comportamiento del reemplazo del bloque es el mismo que en las dos imágenes anteriores, ahora actuando en vía 0, con set 1 y tag 4. En la imagen se puede ver el nuevo bloque, aunque solo se aprecia el cambio de la primera palabra, de 0 a 1. Como es un st, después de realizar el reemplazo de bloque se tiene que escribir sobre la palabra en este caso 1 de este bloque, de ahí que aparezca el 10 al final.

Foto 5:



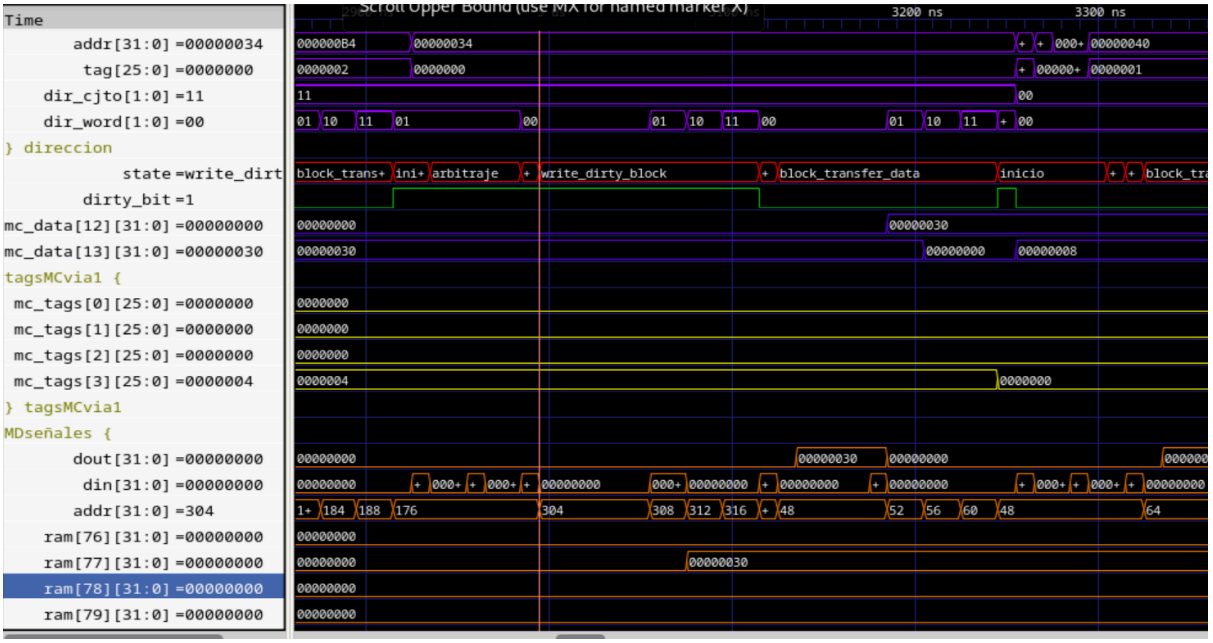
En esta imagen se puede observar cómo ocurre un acierto de escritura. Vemos como previamente se ha traído un bloque de MD a MC a la vía 0 con tag 2 y en el conjunto 2, de forma que cuando se quiere hacer una escritura sobre la segunda palabra de este bloque, los tags coincidan y salte hit0, y se escriba sobre la palabra correspondiente, en este caso se escribe un 8 en la dirección 9, que es la segunda palabra del tercer conjunto, es decir, conjunto 2.

Foto 6:



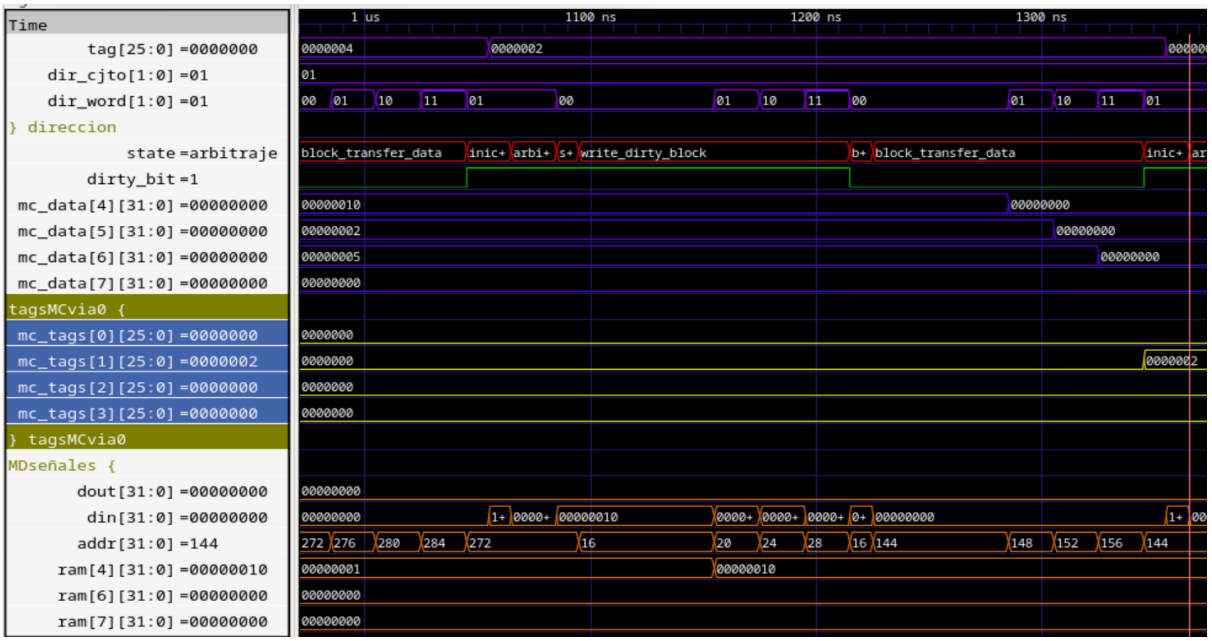
Tanto en esta imagen como en las dos siguientes se puede observar el funcionamiento del copy-back. En este caso en concreto aparece un fallo sucio en la vía 0, con set 1 y tag 2. Esto se debe a que previamente en este test hacemos una escritura en vía 0 y set 1 con un tag distinto de 2, además de que el tag de la vía 1, set 1 es distinto a 2 también. Al analizar la dirección y comprobar que es fallo, miramos el dirty_bit y vemos que está activado, entonces empezaremos el copy-back. Primero tenemos que mandar la dirección de MC a MD con el mismo comportamiento del bus descrito en la primera foto. Una vez se envíe la dirección se comienzan a escribir las palabras hasta la última, en este caso solo se modifica la primera palabra y se puede ver como cambia de 1 a 10. Cuando ya se ha realizado la copia del bloque en MD, ahora hay que traer el nuevo bloque de manera inversa, como se ha realizado en fallo limpio, en este caso sólo varían la 1 primera palabra del bloque y al ser un st en la segunda palabra, al acabar el reemplazo de bloque, se modifica el contenido de esta y se vuelve activar el bit de sucio.

Foto 7:



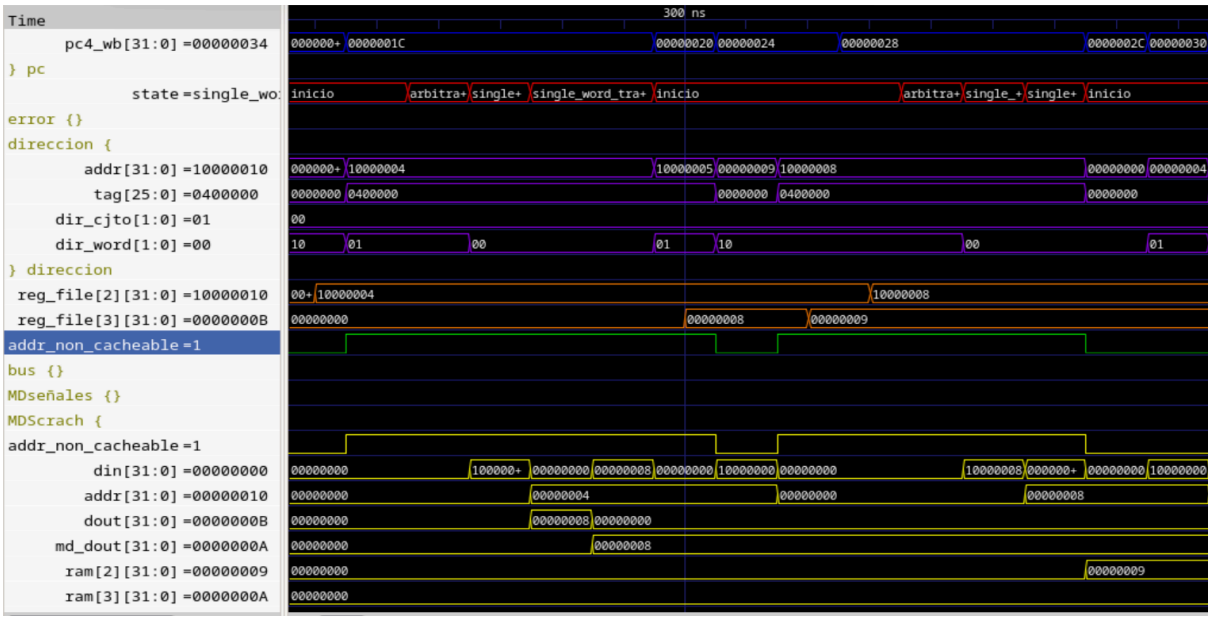
Esta imagen actúa de igual forma que la anterior sin embargo se modifica tanto la vía, que pasa a ser la 1, el set, que pasa a ser 3 y el tag, que pasa a ser 0. A su vez ocurren las condiciones previas de contenido de caché nombradas en el caso anterior, y el desarrollo es el mismo. Los cambios los podemos apreciar en la dirección 77 de la ram que es la única dirección que cambia su contenido al realizar el cambio de bloque y a su vez el reemplazo en el que sólo varían la primera y segunda palabra del boque en caché, siendo la segunda la que será últimamente modificada debido a ser un st en esta palabra y activándose otra vez el bit de sucio.

Foto 8:



De la misma forma que los dos ejemplos anteriores, en esta imagen se aprecia el recorrido completo de una instrucción que realiza un copy-back. En este caso en concreto concuerdan con la vía, el set y el tag del primer ejemplo de copy-back, aunque aquí podemos observar únicamente la modificación de la primera palabra del bloque llevado a la MD, ya que el resto tendrá el mismo valor, sin embargo al llevar el nuevo bloque a la caché, varían todas las palabras excepto la última, sin que se modifique el contenido de ninguna en este caso, ya que es un load.

Foto 9:

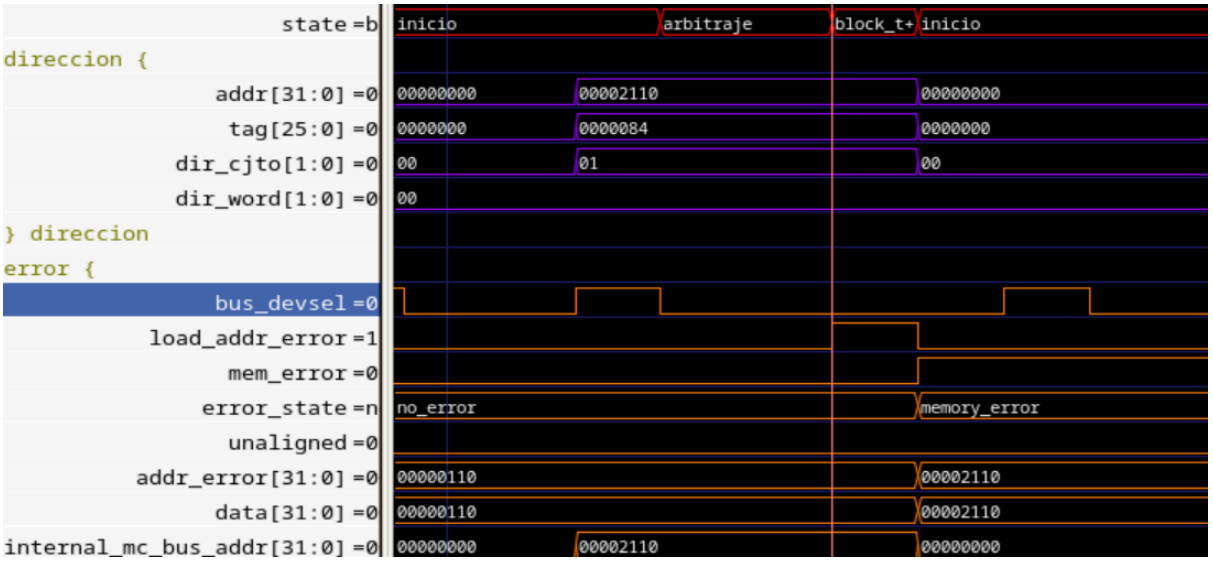


En esta imagen podemos observar el correcto funcionamiento tanto de lectura como de escritura en la memoria Scratch.
Primero podemos ver como se quiere realizar un load sobre la dirección 10000004, es decir, la segunda dirección de Scratch. Para ello comprobamos cómo se activa addr_non_cacheable que nos indica que pertenece a la scratch y pasamos a solicitar el bus, el cual seguirá el mismo comportamiento descrito en el primer caso. Una vez enviada la dirección a la scratch, vemos como hay un 8 en esta dirección y este es el dato que saca por su salida y el que le enviamos al procesador y lo escribimos en el registro 3.
Siguiendo el 3, se le suma 1 dirección a la caché y 1 al contigo de r3, pasando así a la escritura, que actúa de la misma forma que la lectura lo único que cambia es que ahora a la scratch también hay que enviarle el dato que queremos escribir a parte de la dirección, que en este caso son 9 y la dirección es la tercera palabra de la scratch. En la imagen se puede ver como se escribe el contenido en la dirección que le corresponde.

Foto 10:

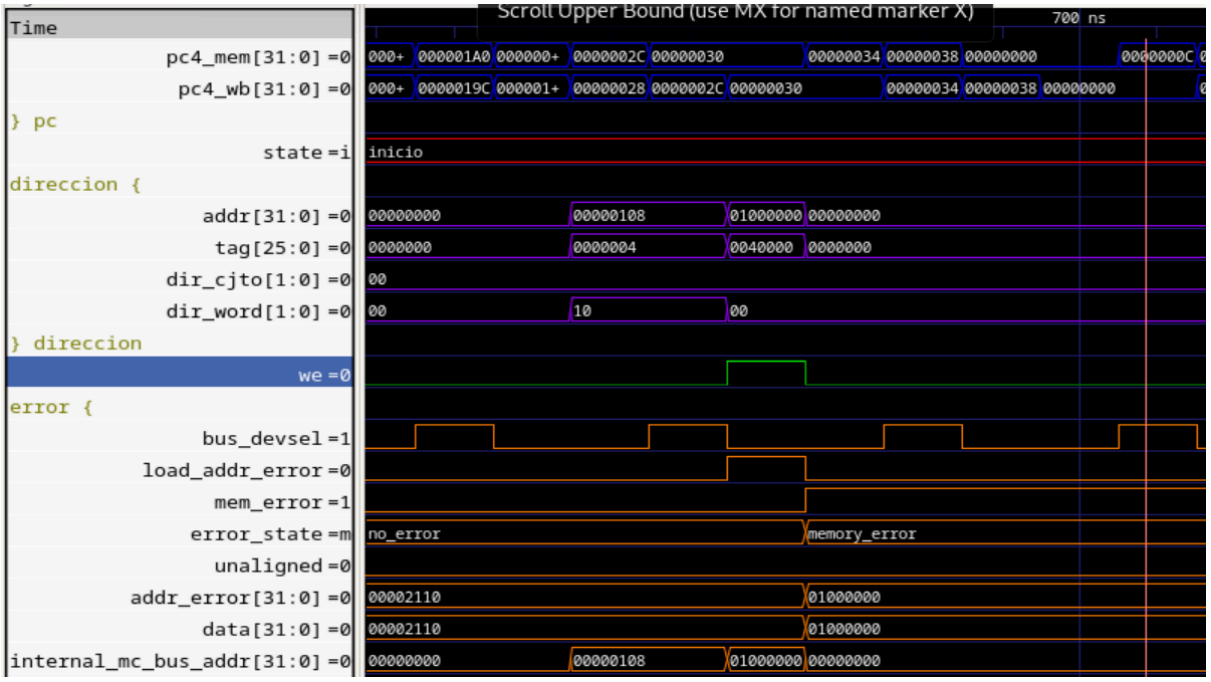
En este test se va a comprobar el correcto funcionamiento de la gestión del data abort. En primer lugar se va a comprobar los errores por una dirección mal alineada. En este ejemplo aparece la dirección 111, la cual está mal alineada, en este momento se activa la señal unaligned, que a su vez provoca que cambiemos de estado en el autómata de error, de no_error a memory_error que a su vez provoca la activación de la señal mem_error que será el data_abort del procesador. Al ocurrir todo esto también se carga el registro interno de la MC con la dirección errónea poniendo a 0 los dos últimos bits de la dirección. Esto se mantendrá hasta que este registro se lea.

Foto 11:



El segundo caso en el que debe activarse data_abort es cuando la memoria no reconoce la dirección. En este caso la dirección 2110 está fuera de rango tanto de la scratch como de la MD, luego bus_devsel nunca va a activarse con esta dirección, lo que provoca todos los cambios nombrados en la imagen anterior.

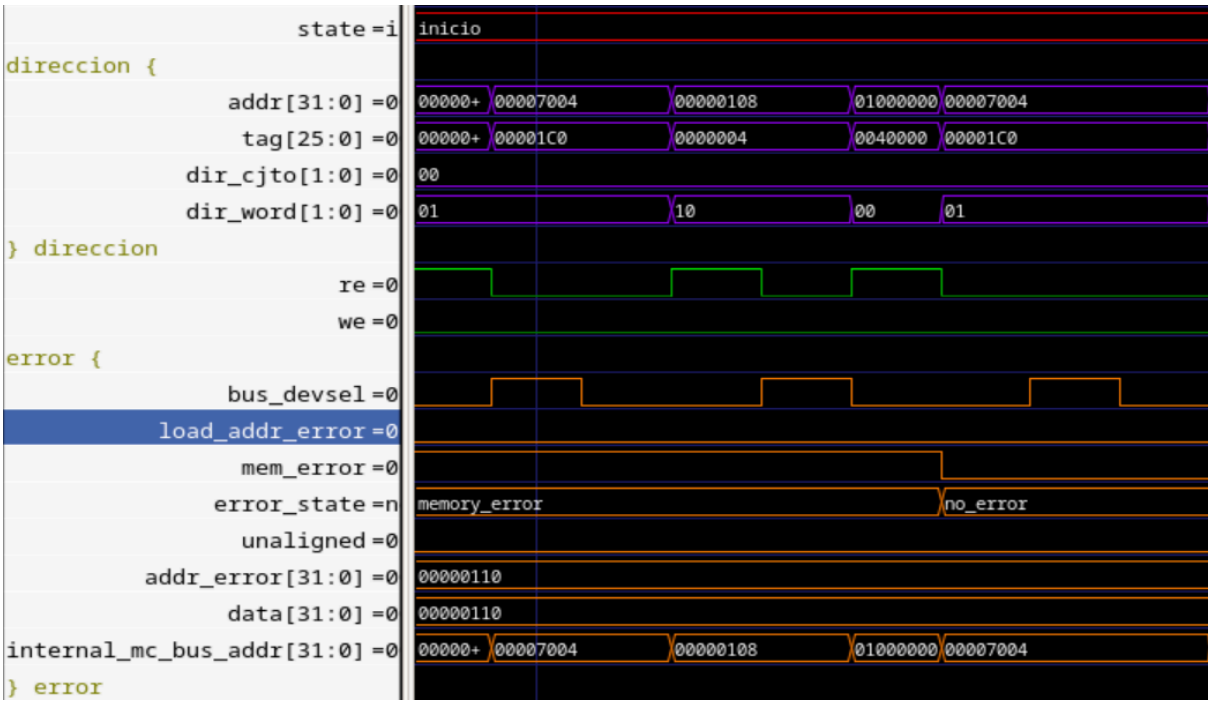
Foto 12:



Por último aparece el error que ocurre cuando se pretende escribir en un registro solo de lectura, como es el caso de registro interno de MC. En este caso vemos

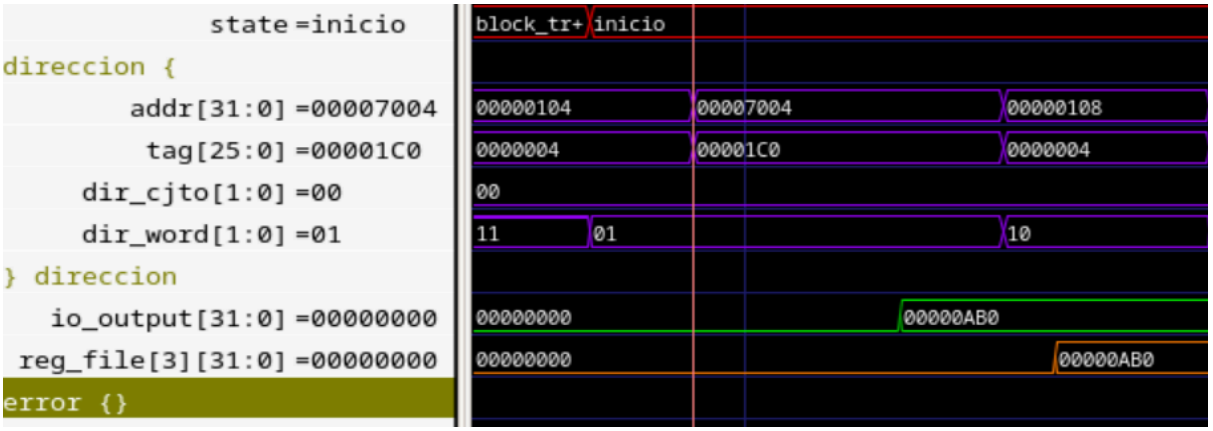
como WE está activado y la dirección de destino es 01000000, que es la dirección en la que está mapeado en registro interno de MC, en este momento se activan las señales de error de la misma forma que en los anteriores casos, pasando al registro de error la dirección de el mismo en este caso, para que el usuario sepa que ha intentado escribir en él cuando no es posible.

Foto 13:



Para salir del estado de error en el autómata de error, necesitamos leer el registro de error, es decir, el registro interno de MC, mapeado en la dirección 01000000. En esta imagen podemos ver su correcto funcionamiento, debido a que RE está activado y la dirección es la del registro. En ese momento se pasa al estado de no_error y podemos mostrarle al usuario la dirección en la que se ha cometido el error.

Foto 14:



Por último en esta imagen podemos ver como se escribe y se lee del registro externo, el registro IO. Primero escribimos el dato AB0 en la dirección 7004, dirección en la que está mapeado este registro, y a continuación lo leemos en r3, y vemos como r3 tiene el mismo contenido. En esta imagen la dirección 7004 aparece seguida ya que se han realizado las dos operaciones seguidas.

7. Cálculo del speedup

Esta jerarquía de memoria aporta, sobre todo, una mejora en el rendimiento en los casos que existan muchos aciertos en escritura y lectura, dado que el número de acceso a memoria es de 1 ciclo. Para poder analizar esta mejora, se proporciona un ejemplo de código en el que se explota estas ventajas comparando un modelo con MC y MD Scratch, respecto a otro que solo tiene una memoria de datos normal. Teniendo en cuenta las simulaciones realizadas, se ha considerado que el número de ciclos de acceso a memoria en cualquier instrucción lw o sw, en un sistema sin MC ni MD Scratch, sería de alrededor de 8,5 ciclos, por lo que se ha tomado un valor de 9 ciclos.

Para calcular el speedup, deberemos calcular el tiempo de ejecución del programa en ambos modelos:

speedup = Tex' / Tex

El programa que se va a utilizar para este cálculo es el siguiente:

```
lw r1, 0(r0)
lw r2, 4(r1)
lw r3, 8(r1)
lw r4, 12(r2)
lw r5, 20(r0)
lw r6, 24(r5)
lw r7, 28(r5)
```

El primer y quinto load serán fallos en lectura, mientras que el resto serán aciertos por lo que el tiempo de acceso será mucho menor, en caso de tener una MC. Sin embargo, los fallos en lectura serán más costosos respecto a un sistema sin MC puesto que habrá que traer el bloque entero de 4 palabras a MC.

speedup = Tex' / Tex = (I' * CPI' * Tc') / (I * CPI * Tc) = (7 * (63 / 7) * 10(ns)) / (7 * (31 / 7) * 10(ns)) = 2,03

En este caso, obtenemos que nuestro sistema con MC y MD Scratch es 2,03 veces más rápido respecto de otro sistema con una MD normal.

Puesto que en el análisis de latencias, se nos pedía que obviamos el coste adicional del arbitraje, a la hora de realizar el speedup hemos tenido en cuenta esta consideración, por lo que no se ha añadido el coste adicional de arbitraje.

8. Suma total de horas

TAREA	HORAS DEDICADAS	
	Samuel Corpas	**Daniel Salas**
Lectura del enunciado y estudio de los archivos fuentes	2	3
Diseño de la UC	8	10
Implementación código VHDL	2	2
Análisis y depuración	15	20
Realización de la memoria	5	5
TOTAL	32	40

9. Autoevaluación

• ¿Crees que has cumplido los objetivos de la asignatura? (Daniel Salas)

Desde mi punto de vista si que he cumplido con los objetivos de la asignatura. El haber tenido que realizar un problema cada semana conllevaba tener que prestar atención los viernes en clase para luego poder realizar el problema con algo de sentido. A su vez, la propia corrección de los problemas ayudaba más que incluso el hacer los problemas, ya que te quedan las cosas más claras al ponerlas en común con otros compañeros y además te das cuenta de que errores has cometido en tu planteamiento y porque. A su vez las prácticas han servido mucho de apoyo a la hora de realizar los proyectos, ya que por ejemplo la práctica 3 asentaba mucho las bases del MIPS segmentado y permitía realizar el proyecto con menos problemas, y algo similar con la práctica 4 y el proyecto 2 aunque algo más diferente. Por último los proyectos eran las pruebas finales que definen si los conceptos están claros o no, y desde mi punto de vista los dos grandes bloques que tratan los proyectos quedan muy asentados.

• ¿Qué nota te pondrías si te tuvieses que calificar a ti mismo? (Daniel Salas)

La nota estaría entre un 8 debido a que tanto los problemas como las prácticas las he seguido sin falta, si que es verdad que algún que otro problema no estaba bien resuelto, y en cuanto a los proyectos el 1 no tuvimos ningún error y este en un principio parece estar bien.

• ¿Crees que has cumplido los objetivos de la asignatura? (Samuel Corpas)

En mi opinión, sí que estaría de acuerdo con que he cumplido los objetivos de la asignatura. Gracias a las clases teóricas semanales y la obligación de tener que realizar un problema cada semana, me ha permitido no dejar la asignatura de lado y llevarla al día. Con esto, ha resultado mucho más sencillo la comprensión tanto de las prácticas como de los proyectos de evaluación continúa. Otro punto a destacar, es que la corrección de los ejercicios semanales en clase, me ha permitido también entender con mucha más facilidad los conceptos de la asignatura. En resumen, diría que sí he cumplido los objetivos de la asignatura, puesto que me han quedado claros todos los bloques de la y he conseguido realizar satisfactoriamente todas las prácticas y trabajos de la asignatura.

• ¿Qué nota te pondrías si te tuvieses que calificar a ti mismo? (Daniel Salas)

Si tuviese que darme una nota a mi mismo, sería entre un 7 y un 8, dado que en ciertas ocasiones no disponía del tiempo suficiente para realizar los problemas semanales correctamente y para el proyecto final de la asignatura, he tenido que hacer un esfuerzo extra por tratar de entender el completo funcionamiento del sistema. Aun así estoy muy contento con mi trabajo a lo largo del semestre y pienso que esta nota sería apropiada.