Protocolo Snooping

Relatório de Laboratório de Arquitetura e Organização de Computadores II

Autores

Daniel Santana
Isaque Fernando

Centro Federal de Educação Tecnológica de Minas Gerais

2018

Sumário

1 Introdução

Este relatório surgiu com a necessidade de se documentar os detalhes de projeto da atividade prática 4 da disciplina de Laboratório de Arquitetura e Organização de Computadores 2. A atividade em questão demandava uma simulação, em linguagem Verilog, da execução de diversas instruções em três processadores, cada um deles com uma cache e uma memória compartilhada, bem como uma máquina de estados do protocolo MESI Snooping, usado para solucionar o problema de coerência de caches que acontece atualmente.

Nos processadores atuais, o pararelismo tem sido notavelmente aplicado, e ele é o responsável pelo poder de processamento que temos hoje. Quanto melhor um processador consegue paralelizar suas instruções, mais rápido seu usuário final terá seus resultados esperados. Porém, nesse contexto de paralelismo, e compartilhamento de memórias cache (especialmente comum em nível L2) para obtenção mais rápida de dados, existe um grande problema a ser enfrentado, que é a coerência de dados entre as memórias cache. Um dado, quando requisitado ou alterado em uma cache, não pode estar inconsistente em outra, já que isso geraria problemas de confiabilidade, por causa do paralelismo explorado entre os processadores de uma máquina.

Para solucionar este problema, o protocolo de Snooping sugere o bloqueio de escrita, garantindo que um processador tenha acesso exclusivo a um item de dados antes que escreva neste item. Neste protocolo, o barramento é utilizado como uma forma de controle: Uma escrita não pode ser completada até se ter o acesso ao barramento. Enquanto isso, todos os outros processadores ficam "bisbilhotando"este barramento, ou seja, ficam em modo de escuta, para avaliar se elas possuem cópia do bloco solicitado. Para a serialização, o processador envia ao bus o endereço a ser invalidado, e todos os processadores que estão "bisbilhotando"o bus que têm este endereço, logo o invalidam.

Este trabalho contempla a implementação de três CPU's diferentes. Cada uma dessas CPU's possui uma cache própria com 4 posições de 1 byte, carregada individualmente. Foi implementada também uma memória principal, com 8 posições de 6 bits cada, que é compartilhada entre as unidades de processamento. A comunicação dos processadores entre si, e com a memória, é feito através de um barramento. Ao final da prática, o grupo não conseguiu simular com êxito todas as instruções passadas no código de testes sugerido na instrução do trabalho. Apenas duas instruções funcionaram perfeitamente.

2 Implementação

Foi utilizado o softwate *Quartus 2*, 64-bit, ver.13.0sp1 para a escrita do código do projeto. Para as imagens de simulação geradas e apresentadas

abaixo, foi usado o software *ModelSim*, uma vez que os alunos possuem familiaridade com o uso destes dois programas, e já que estes possuem uma integração nativa e de fácil utilização.

2.1 Máquinas de Estados MESI

As decisões de projeto que concernem às máquinas de estados são as seguintes:

- O sinal da CPU possui 4 bits. o MSB indica se a máquina em questão é a de escuta (0), ou a de escrita (1). O LSB indica se o bloco é compartilhado ou não por outras caches;
- Os significados dos sinais da CPU, excluindo o MSB e o LSB (portanto, os dois bits do meio), são: read miss - 00, read hit - 01, write miss - 10, e write hit - 11;
- O sinal do BUS possui dois bits, e os seus significados são: nada - 00, read miss - 01, write miss - 10, invalidate - 11.
- O sinal que vai para a memória possui 2 bits, sendo eles: nada - 00, abort mem access + write back - 01, e write back - 10;
- Os estados das máquinas são representados, respectivamente, pelos bits: 00 - Invalid, 01 - Shared, 10 - Exclusive, 11 - Modified.

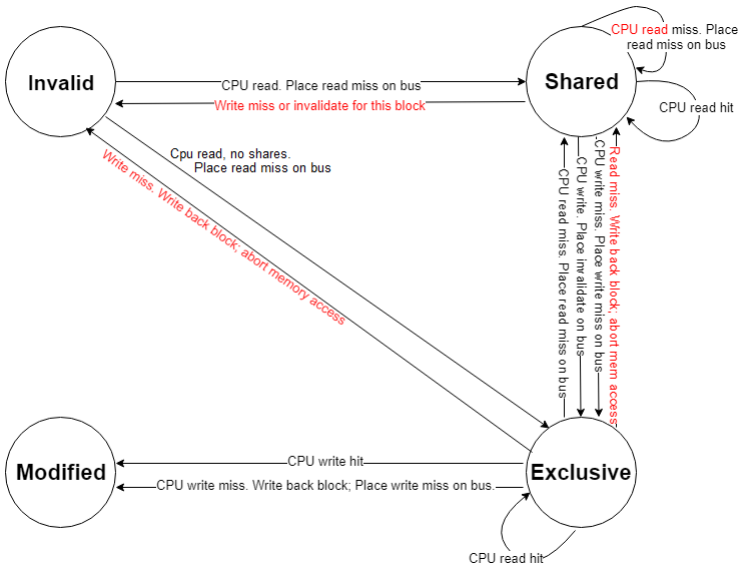

Figura 1: Máquina de estados do protocolo MESI de coerência de dados.

O protocolo MESI possui duas máquinas de estados: uma baseada nas informações enviadas pela CPU (descrita pelas setas com legenda preta), e

outra baseada nas informações do BUS (descritas pelas setas com legenda vermelha). Para fins de praticidade, o grupo uniu as duas máquinas em uma única imagem. A descrição de cada transição se encontra a seguir.

2.1.1 Máquina de Estados para Escrita

Esta é a máquina de estados que funciona com base nas requisições da CPU. Seus estados, e respectivas transições são as seguintes.

- Estado Invalid:

  Este estado indica que quaisquer dados contidos nesta cache estão inválidos caso a CPU indique a requisição de leitura, e o bloco requisitado exista em outra cache, o estado futuro desta máquina sera Shared, e uma mensagem de read miss será enviada ao bus. No mesmo caso de requisição de leitura, se o bloco desejado não existir em outra cache, o estado futuro da máquina sera Exclusive, e um read miss é colocado no bus. Caso a requisição seja de escrita, o estado futuro será Exclusive, um write miss será enviado ao barramento.

- Estado Shared:

  Este estado indica que os dados contidos nesta cache existem também em outras caches. Caso a CPU indique um read, o estado futuro será shared, mas o barramento só receberá algum sinal caso ocorra um miss. Neste caso, o sinal será um read miss. Caso a CPU indique um write, o estado futuro da máquina será Exclusive. Além disso, caso seja um hit, o bus receberá um sinal de invalidade, e caso seja um miss, ele receberá um sinal de write miss.

- Estado Exclusive:

  Este estado indica que os dados contidos nesta cache não estão válidos em nenhuma outra. Caso a CPU indique um read hit, o estado futuro não será alterado, e o barramento não receberá sinais. Caso haja um write hit, o estado futuro será Modified, e o barramento não receberá sinais. Caso haja um write miss, o estado futuro será Modified, e o sinal de write miss será enviado ao bus. De forma similar, caso ocorra um read miss, o estado futuro sera Shared, e um read miss será enviado ao bus.

- Estado Modified:

  Este estado indica que, além de o dado buscado ser válido somente nesta cache, ele foi sobrescrito. Na máquina de Escrita, este estado não possui transições para nenhum outro.

2.1.2 Máquina de Estados para Escuta

Esta é a máquina de estados que funciona com base nas informações do barramento. Seus estados, e respectivas transições são as seguintes.

- Estado Invalid:

  Este estado não possui transições para nenhum outro nesta máquina.

- Estado Shared:

  Caso haja um write miss, ou um invalidate para este bloco no barramento, o estado futuro será Invalid. Caso haja um read miss, o estado futuro será inalterado. Nenhum sinal para a memória é emitido.

- Estado Exclusive:

  Caso "escute"um read miss neste bloco, o estado futuro será Shared. Se o sinal "escutado"for um invalidade, ou um write miss, o sinal futuro será Invalid. São enviados para a memória sinais para abortar o acesso de outros blocos à memória, para que ocorra o write-back deste bloco.

- Estado Modified:

  Este estado não possui transições para nenhum outro nesta máquina.

2.2 Central Processing Units

A especificação do trabalho demandava a implementação de três processadores diferentes. É importante ressaltar a decisão de criar três módulos diferentes para cada um deles. Como cada CPU teria sua própria cache instanciada como uma matriz de registradores, e cada umas dessas caches seria iniciada com valores distintos, o grupo tomou esta decisão para facilitar o carregamento de cada cache separadamente. A opção de instanciar uma matriz de registradores dentro de cada processador foi tomada para evitar atrasos em ciclos de clock para acessar esta memória dentro de cada unidade. Estes atrasos ocorreriam caso fossem instanciados módulos separados para cache. É importante salientar que, ao receber uma tag, o mapeamento desta nas caches ocorre apartir da operação *mod 4*. Por exemplo, a tag 100 seria mapeada para a posição (100 % 4) 0 da cache.

Cada CPU possui um sinal recebido do top level, que indica se esta irá trabalhar em modo de leitura ou de escrita. Em cada um destes modos, a unidade segue uma sequência de passos, até a finalização da instrução, onde ocorre o reset dos passos.

2.2.1 CPU em estado de Escrita

Quando o processador é alvo direto da instrução ele seguirá a sequência de passos chamados de *estado de escrita*.

1. No passo zero, a tag enviada pela instrução é mapeada para alguma posição na cache através da operação *mod 4*.

2. No primeiro passo, o processador decodifica a instrução. Uma cadeia *if-else* externa verifica a existência da tag válida requisitada na instrução. Então, uma cadeia *if-else* interna verifica se o que ocorreu foi um read ou write.

3. O segundo passo é um ciclo de stall, para que a máquina de estados possa operar e retornar o estado futuro do bloco.

4. No terceiro passo, a ação tomada depende diretamente de ter ocorrido um read, ou write, hit ou miss. De acordo com isto, a mensagem para o barramento é enviada, através da concatenação das mensagens que voltam da máquina de estados para o bus, para a memória, e dos 6 bits finais da instrução, que indicam a tag e o dado operados. Neste passo também ocorre a atualização da cache com os dados provindos das operações além dos valores retornados pela máquina de estados.

5. No quarto passo, há um ciclo de stall.

6. No passo 5, a cache é atualizada caso a instrução tenha sido um read miss e outra cache possuía o bloco requisitado válido. Então este dado é envidado para a saída.

7. No sexto e último passo, caso tenha havido um read miss e nenhuma outra cache possuía o bloco requisitado válido, este dado vem da memória e atualiza a cache.

2.2.2 CPU em estado de escuta

Quando a CPU não é alvo direto da instrução recebida, ela observa constantemente o barramento, em busca de mensagens que afetem indiretamente a sua cache. Esta unidade então, segue a sequência de passos chamados de *estado de escuta.*

1. Nos passo zero e um, o processador informa ao bus se possui a tag requisitada pela instrução de outra CPU.

2. O passo dois é um ciclo de stall para a ação da máquina de estados.

3. No último passo, as CPU's em modo de escrita que possuem a tag válida requisitada, a passam para o barramento, para que seja compartilhada.

2.3 Memórias Cache

Cada memória cache é uma matriz de registradores dentro da sua CPU. Esta matriz possui 4 posições de 1 byte(8bits) cada, distribuídos da seguinte maneira: XXYYYZZZ, sendo:

- XX - Estado do bloco;
- YYY - Tag mapeada para o bloco;
- ZZZ - Dado correspondente para a tag.

2.4 Instruções

O formato padrão de instruções adotado foi XXYYZZZWWW, 10 bits sendo: XX - processador; YY - instrução:00:nada;01:read;10:write; ZZZ - tag (para mapear as 8 posições de memória); WWW - dado a ser escrito, no caso de instrução write.

Por exemplo, a instrução **P0: write 000 <− 01**, que visa escrever o dado 01 na tag 000 do primeiro processador, fica: 00_10_000_001.

Todas as instruções usadas na implementação foram carregadas no código, segundo o código de testes disponibilizado na instrução do trabalho. Foi utilizada uma memória de instruções na forma de uma matriz de registradores no top-level, com 9 posições, para conter as 9 instruções do código de testes.

É importante salientar que o grupo decidiu por "traduzir"os valores de tag e dado fornecidos no código de teste, uma vez que seria necessário uma grande quantidade de bits para reproduzir os valores deste, causando um certo desperdício, já que seriam usados poucos valores e poucas tags. A adaptação feita foi a seguinte:

- Para tags:
  - Valor original: 100(dec). Valor adaptado: 000(bin).
  - Valor original: 108(dec). Valor adaptado: 001(bin).
  - Valor original: 110(dec). Valor adaptado: 010(bin).
  - Valor original: 118(dec). Valor adaptado: 011(bin).
  - Valor original: 120(dec). Valor adaptado: 100(bin).
  - Valor original: 128(dec). Valor adaptado: 101(bin).
  - Valor original: 130(dec). Valor adaptado: 110(bin).
- Para dados:
  - Valor original: 20(dec). Valor adaptado: 000(bin).
  - Valor original: 10(dec). Valor adaptado: 001(bin).

- Valor original: 8(dec). Valor adaptado: 010(bin).
- Valor original: 30(dec). Valor adaptado: 011(bin).
- Valor original: 68(dec). Valor adaptado: 100(bin).
- Valor original: 18(dec). Valor adaptado: 101(bin).
- Valor original: 40(dec). Valor adaptado: 110(bin).
- Valor original: 60(dec). Valor adaptado: 111(bin).

2.5 Memória de Dados

Para evitar excessivos ciclos de stall na execução, foi instanciada uma matriz de 8 registradores de 6 bits, dispostos da seguinte maneira: XXXYYY, sendo XXX - tag, YYY - valor.

Os valores desta memória são carregados no início da execução, de acordo com o código de testes disponibilizado, e com a adaptação de valores descrita no item anterior.

2.6 BUS

O barramento, chamado de *bus*, possui 10 bits: XXYYZZZWWW, sendo XX - mensagem BUS; YY - mensagem MEM; ZZZ - tag; WWW - valores.

O grupo definiu o valor 0011 nos quatro bits mais significativos como sendo uma espécie de mensagem única, que, quando decodificada e identificada pelos processadores, indica que estes devem retornar um bit denominado *shared*, que informa se aquele processador possui a tag ZZZ em algum estado válido, sinalizando que o bloco da instrução é compartilhado, antes mesmo de se iniciar a execução da instrução.

2.7 Bus Arbiter

Módulo responsavel por comandar a escrita no bus. O top level decide a entrada e o bloco insere na saída valor passado na entrada desde que o sinal de habilita esteja em nível alto.

2.8 Top-Level

Decodifica a instrução provinda da memoria de instruções e, através dos dois bits mais significativos identifica qual processador deverá realizar a operação. Neste momento, cada processador recebe por parâmetro o bit que indica se este operará em modo de escrita ou leitura.

Este módulo possui uma sequência de passos própria, que comanda todo o funcionamento dos processadores, e da comunicação entre estes e o barramento. A memória de instruções e de dados são instanciadas neste módulo como matrizes de registradores, mas, apesar disto, a memória de dados não

se comunica diretamente com os processadores. As operações de write back e passagem de dados ocorrem apenas através do BUS.

Antes de iniciar toda a sequência de passos que rege a lógica do programa, as memórias de instrução e dados são carregadas de acordo com o PDF do código para testes, disponibilizado na instrução da prática, e de acordo com a adaptação de valores já mencionada acima. Além disso, um sinal inicial de *clear* é necessário para dar o valor inicial zero às variáveis de controle do código.

As variáveis de controle são *hab_cpu1*, *hab_cpu2* e *hab_cpu3* (que servem para habilitar a ação e contagem de passos dos processadores), *controleP1*, *controleP2* e *controleP3* (servem para indicar qual processador está em modo de escrita na instrução atual), *pc* (tem a função de program counter), *passos* (utilizada para contar os passos), *hab_bus* (tem a funçao de habilitar a escrita no barramento) e *shared* (que serve para informar se o dado em questão está compartilhado entre mais de um processador).

Após este carregamento inicial, os passos seguidos pelo top-level são os seguintes:

1. Identificar através do *program counter*, qual é a instrução que vai ser realizada, fazendo com que, ao obter essa instrução, o *pc* exponha qual será a próxima a ser executada ao retornar a este passo. A instrução que será executada é armazenada em um registrador de instruções.

2. É definido, através dos dois MSB (bits mais significativos) da instrução qual processador será usado em modo de escrita, indicando para as três CPU's a sua função.

3. As CPU's em modo de escuta informam se o bloco buscado existe em sua cache em estado válido ou não.

4. Stall para recebimento do sinal *shared*.

5. Stall para recebimento do sinal *shared*. Ao final deste passo a variável *shared* já pode receber a informação de compartilhamento do bloco.

6. Habilita-se a ação do processador de escrita, para a execução da instrução (já com a informação de compartilhamento do bloco) e consequente contagem de passos.

7. Stall necessário para a ação da CPU de escrita.

8. Stall necessário para a ação da CPU de escrita.

9. Stall necessário para a ação da CPU de escrita.

10. Stall necessário para a ação da CPU de escrita.

11. Após ação do processador em modo de escrita, habilita-se a contagem de passos nos processadores em modo de escuta, para que verifiquem possíveis alterações(ou invalidações) de blocos em sua cache, de acordo com a ação da instrução na CPU alvo. Neste passo também é desabilitada a ação e contagem de passos nesta CPU, para que aguarde a atualização das demais.

12. A escrita no bus é desabilitada ($hab\_bus = 0$) para evitar possíveis erros de escrita equivocada no barramento.

13. Caso algum processador em modo de escuta envie sinal de write-back+abort memory access, este sinal é enviado ao bus, juntamente com o dado para ser escrito na memória.

14. Ocorre a escrita na memória caso tenha haja um sinal write back + abort memory access no barramento.

15. O processador utilizado volta a ser habilitado, pois caso a instruçao tenha sido um read miss, ele aguarda a verificação se o dado existe em algum dos outros processadores. Caso exista, é feita a escrita na memória (passo anterior) para que a CPU que precisava de ler esse dado, pegue-o da memória e conclua seu objetivo.

16. Stall para ação final da CPU de escrita.

17. Desabilita-se a CPU de escrita para reinício do ciclo, com uma nova instrução, no passo 0.

3 Simulações

As simulações a seguir foram feitas com base no código de teste passado pela professora Daniela Cascini através do portal Moodle(AVA).

3.1 Bus Arbiter

Figura 2: Simulação do Bus Arbiter

A marcação em vermelho demonstra a ação do sinal de reset, setando o valor zero na saída do bus. A marcação laranja exibe que a escrita no bus ocorre no momento em que o clock se encontra na borda de subida, desde que o sinal de *hab_bus* esteja em nível alto. A marcação amarela e verde demonstram a escrita assim que a borda de subida é detectada. Em linhas gerais o módulo serve como um registrador e cabe ao toplevel escolher quem irá escrever neste BUS.

3.2 CPU em Estado de Emissão

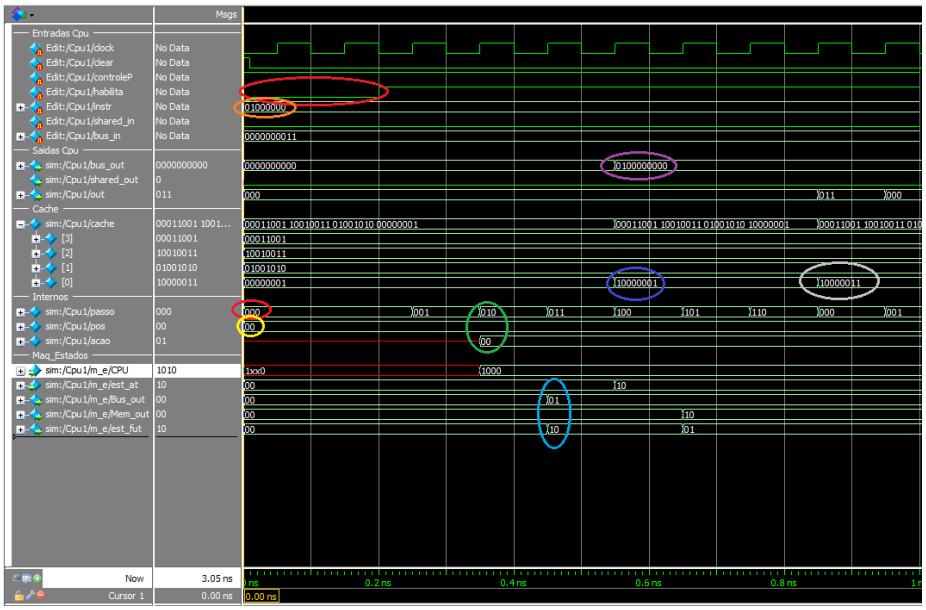

Figura 3: Simulação da CPU em modo de emissão

As marcações em vermelho permitem verificar que os passos somente são avançados quando o sinal de habilita está em nível alto. A marcação laranja exibe a instrução que é recebida para execução Px: read 100 no shares, sendo que shares é verificado pelo sinal shared_in. A marcação amarela exibe o valor cálculado para o registrador pos no caso B0. Acima destes sinais percebemos a cache inicializada através de uma declaração *initial*. Na marcação verde percebemos a ação final recebida na cache (read miss) já que a posição em questão (B0) esté inválida. Com estes resultados a marcação azul demonstram o estado futuro e as mensagens produzidas pela máquina de estados (Mesi). Estes resultados são setados na posição da cache mapeada (marcação azul) e colocados a disposição para escrita no BUS (marcação roxa). Como último passo o valor final é disponibilizado no BUS e escrito na cache no passo 6 (marcação cinza).

3.3 CPU em Estado de Escuta

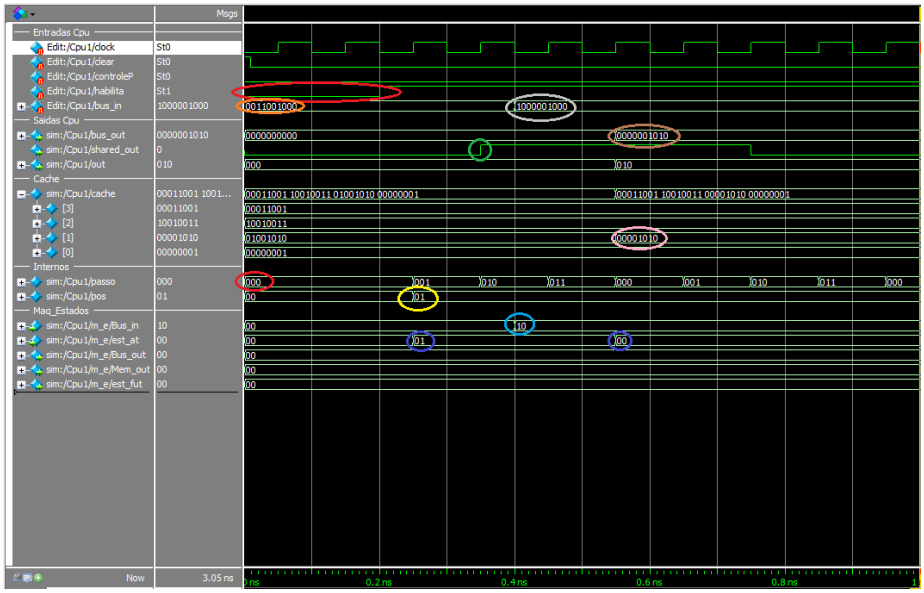

Figura 4: Simulação da CPU em modo de escuta

Neste modo também foi definido o funcionamento da CPU somente quando o sinal de habilita estiver em nível alto, sendo assim o passo não é incrementado (marcação vermelha). Assim que é habilitado o sinal a CPU escuta o BUS e verifica se há a solicitação do sinal shares (sinal laranja). Neste caso há a solicitação se a tag 001 está presente e válida na cache da CPU. Como este sinal está presente (marcação amarela) o passo 01 é executado e o sinal shares enviado (marcação verde). Após a verificação do shares simulamos um resultado de outra CPU sendo recebido através do BUS (marcação cinza). O estado no momento da escuta é shared (marcação azul escura) e o estado futuro com a audição do write miss (10 MSBs do BUS marcação azul clara) é 00 (inválido marcação azul escura a direita). Os valores são inseridos na cache (marcação rosa) e também colocados a disposição do toplevel para escrita no bus através do bus_out (marcação marrom).

3.4 Snooping Primeira Instrução

Neste subtópico teremos a adição de duas imagens para cada instrução, já que a simulação ultrapassou o tamanho disponível na tela do computador utilizado durante as simulações.

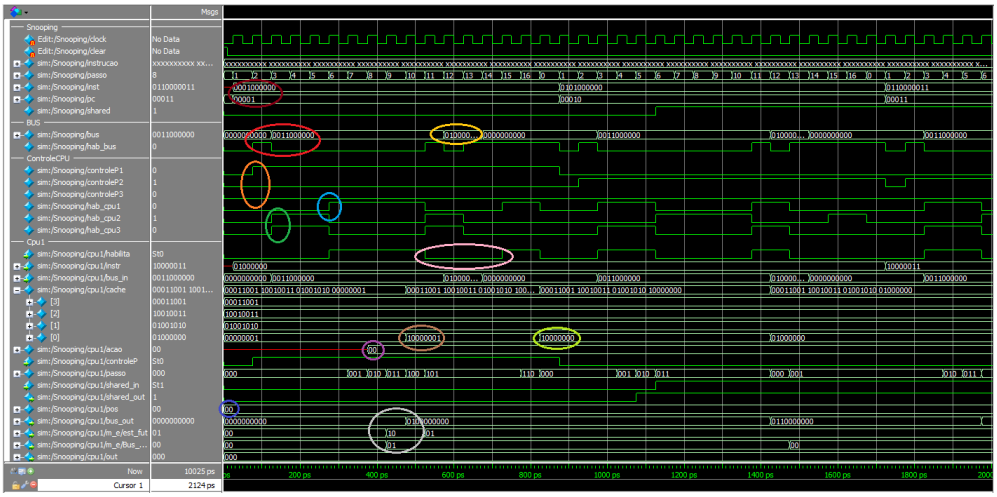

Figura 5: Primeira parte da simulação do Protocolo Snooping com a primeira instrução.

Iniciamos com o carregamento da instrução e a adição do endereço do registrador PC (marcação vermelha escura). Através da instrução são definidos os sinais que indentificam quais processadores serão utilizados para escuta e qual processador será o emissor (marcação laranja). Para a obtenção do sinal shares o toplevel coloca no BUS um valor questionando a existência da tag nos processadores ouvintes (marcação vermelha). Para que as CPUs de escuta executem o passo shares são setados os sinais de *hab_cpu* de acordo com a necessidade (marcação verde). Após esse passo e a obtenção do sinal correto a CPU de emissão é colocada em ação (marcação azul claro). De acordo com a solicitação a posição B0 é mapeada e colocado no registrador pos (marcação azul escura). De acordo com o estado inicial da cache a ação de read miss é definida (marcação roxa). Com a ação definida a máquina de estados produz o estado futuro (exclusivo 10) e as mensagens necessárias (marcação cinza). Esses valores são carregados na cache (marcação marrom) e a cpu desabilitada momentaneamente (marcação rosa). A mensagem produzida pela cache é colocada disponível no BUS (marcação laranja claro) para que as CPUs de esuta executem a sua ação. Após essas ações o valor exato disponibilizado pela memória através do BUS é colocado na cache (marcação amarela).

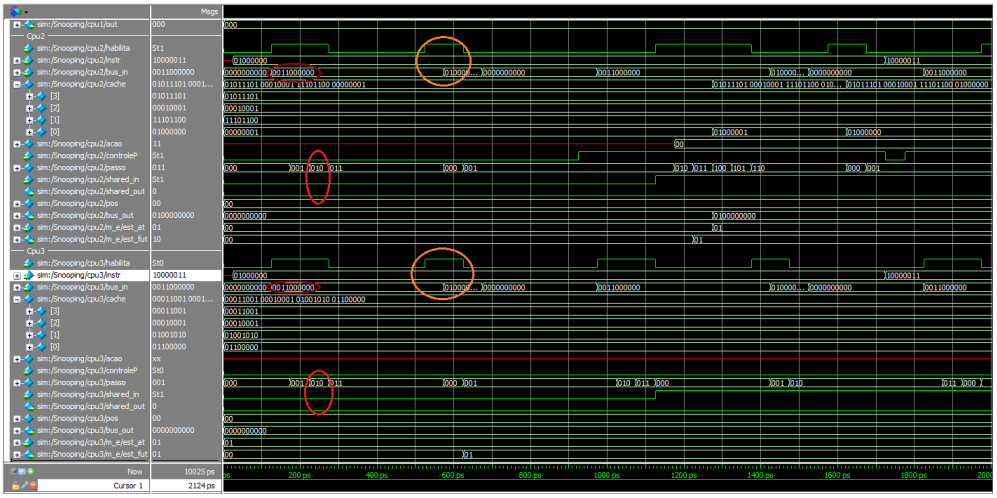

Figura 6: Segunda parte da simulação do Protocolo Snooping com a primeira instrução.

Nesta imagem as CPUs 2 e 3 serão utilizadas como CPUs de escuta. Primeiramente assim que há o nível alto do sinal *habilita* as CPUs ouvem a solicitação do BUS quanto ao sinal shares perguntando a existência da tag utilizada na instrução (marcação vermelha escura). No passo 001 esse valor é produzido segundo o estado inicial da cache e o passo é passado para o estado futuro (marcação vermelha). Após esta pesquisa as CPUs são desabilitadas para a execução da CPU de emissão e são colocadas em ação assim que a CPU de emissão já produziu sua mensagem e o toplevel já inseriu essa mensagem no BUS (marcação laranja), como os processadores não contavam com a tag ou a mesma não estava válida eles não modificaram o estado dos blocos na cache.

3.5 Snooping Segunda Instrução

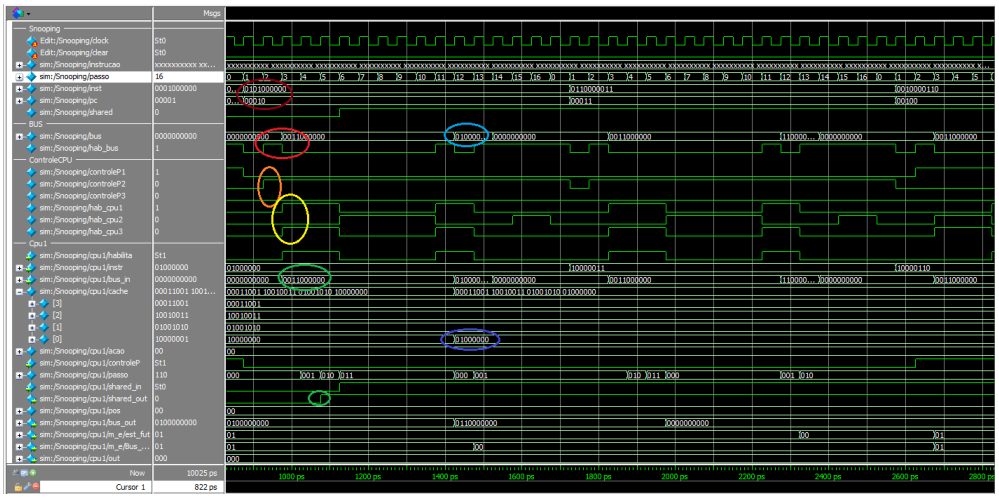

Figura 7: Primeira parte da simulação do Protocolo Snooping com a segunda instrução.

Nesta instrução vemos o carregamento do registrador de instrução e também a adição do PC (marcação vermelha). A escrita no BUS é habilitada e é solicitado o sinal shares da tag utilizada na instrução para os demais processadores (marcação vermelha). Nesse ponto os papeis de cada CPU já foram escolhidos (marcação laranja) e as CPUs de escuta habilidatas para execução do sinal shares (marcação amarela). A CPU1 assim que ouve a solicitação através do BUS (marcação verde) faz a pesquisa na cache e produz o seu sinal share através da saída shared_out (marcação verde). Na segunda imagem veremos que a CPU2, responsável pela execução da instrução produz o sinal de read miss (marcação azul clara) e com este sinal o protocolo implementado na máquina de estados Mesi faz com que a cache vá para o estado shared 01 (marcação azul escura). Podemos perceber que na saída *bus_out* da CPU1 as mensagens produzidas pela máquina de estado são de um sinal de write back e abort memory access o que fará com que a CPU2 no seu read miss utilize o valor passado no bus como valor de atualização da cache.

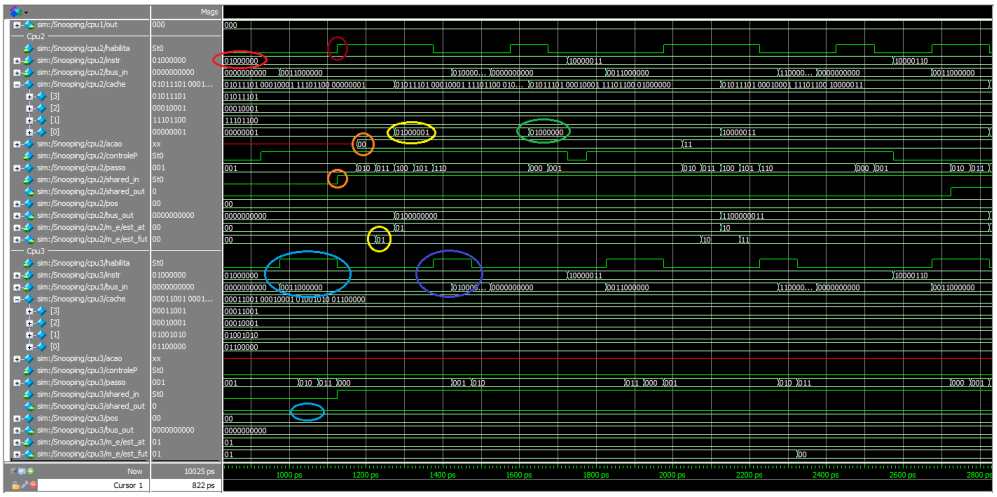

Figura 8: Segunda parte da simulação do Protocolo Snooping com a segunda instrução.

Percebemos que já no primeiro passo a instrução é colocada a disposição da CPU2 (marcação vermelha) porém a CPU só é liberada para execução em ciclos posteriores (marcação vermelha escura). Notamos que a CPU3 assim que habilitada ouve a mensagem da operação shares e no ciclo seguinte produz o seu resultado (marcações azuis). A CPU2 percebendo o sinal *shared_in* e a ação (marcação laranja) assim que habilitada (marcação vermelha escura) produz o sinal shared resultante do estado anterior e ação realizada e o grava na cache na posição mapeada (marcação amarela). A saída *bus_out* é carregada com as mensagens enviadas pela máquina de estados e assim que ouvida nas demais CPUs produz o efeito necessário de acordo com o protocolo de coerência. Para a CPU3 não é modificado (marcação azul escura) porém para a CPU1 a cache é modificada como foi exibido na simulação acima.

3.6 Snooping Terceira Instrução

Nesta simulação foi verificado um erro de sincronização demonstrada em uma simulação posterior.

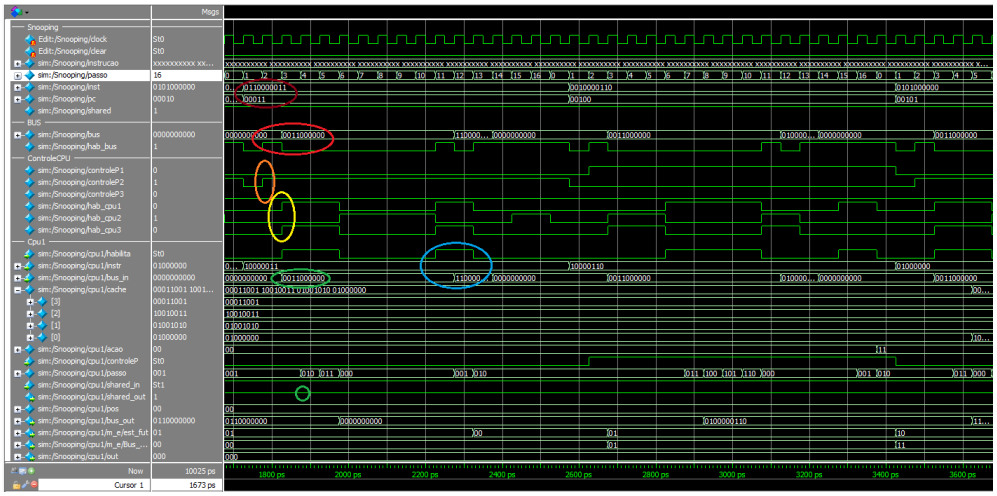

Figura 9: Primeira parte da simulação do Protocolo Snooping com a terceira instrução.

Aqui vemos o carregamento do registrador de instrução com a instrução correta juntamente com o incremento do PC (marcação vermelha escura). Novamente assim que a instrução é carregada é solicitado o sinal shares (marcação vermelha). Neste ponto é definido qual processador irá executar a instrução e quais serão ouvintes (marcação laranja). Assim que isto é definido os sinais de *hab_cpu* são setados para que as CPUs de escuta ouçam a solicitação do sinal shares (marcação amarela) mesmo que este não seja utilizado nesta instrução. Percebemos pelas marcações verdes que no caso da CPU1 o sinal de shared é mantido em nível alto porém este não será utilizado futuramente. Na marcação azul é exibido o erro em que após uma operação de write hit da CPU2 , vista na simulação abaixo, o bloco mapeado pelo registrador pos (BO) não é invalidado mesmo com a mensagem de invalidate colocada no BUS (exibida na mesma marcação azul). Tal erro atrapalhou a conclusão exata dessa operação e das demais como será exibido em simulações futuras.

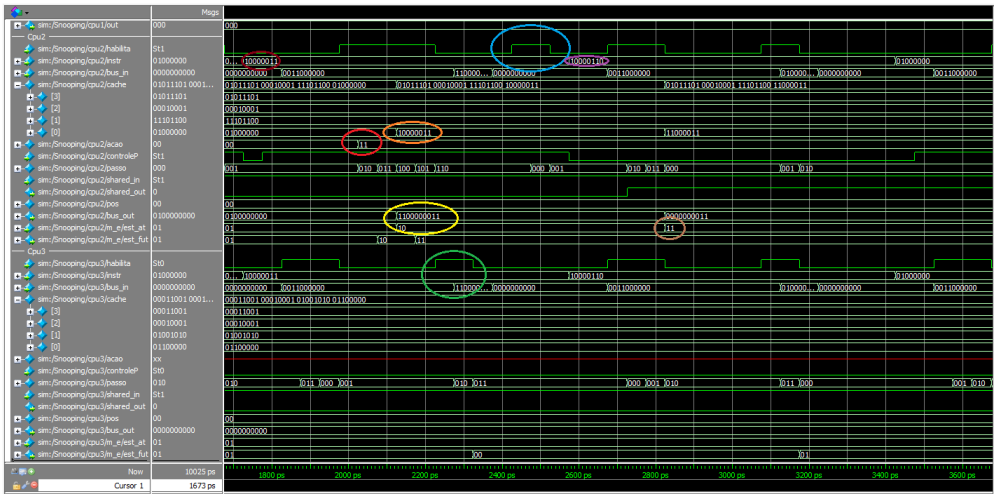

Figura 10: Segunda parte da simulação do Protocolo Snooping com a terceira instrução.

A instrução é carregada no processador através do registrador *instr* já sem os bits iniciais que demonstram qual processador irá executar a instrução (marcação vermelha escura). Como supracitado nesta intrução de escrita não é analisado o valor do sinal shared, portanto percebemos que de acordo com estado atual e a ação (write hit marcação vermelha) o estado do bloco na cache é modificado e o valor gravado (marcação laranja, estado futuro exclusive) de acordo com a saída dada pela máquina de estados (marcação amarela), tendo também disponível na saída *bus_out* as mensagens específicas sobre estas movimentações em nosso casso o write hit. Notamos que após estes valores serem processador as CPUs de escutas são ativadas e deveriam produzir os sinais necessários (marcação verde). No processodor CPU2 a instrução é terminada com sucesso porém na CPU1 os valores não foram atualizados o que gerou um erro nas demais instruções.

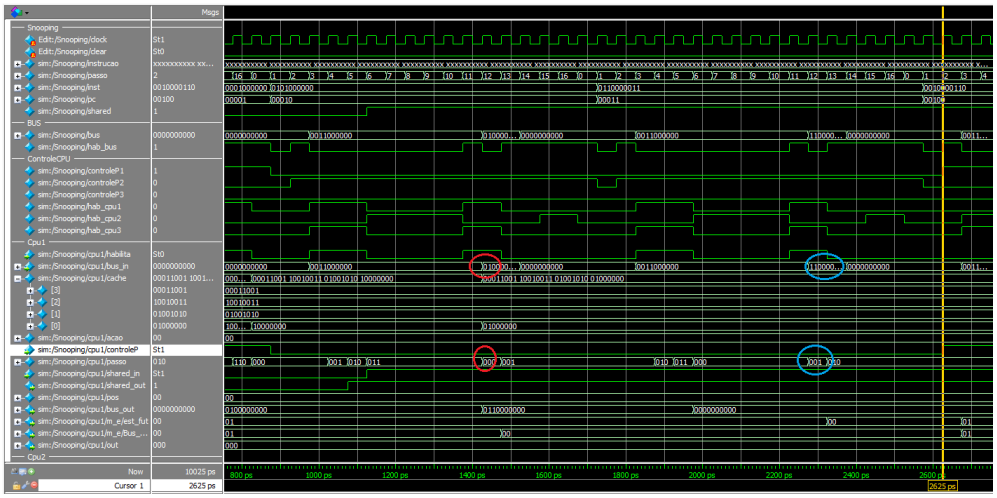

Figura 11: Erro percebido durante as simulações

Entre a segunda e a terceira instrução foi percebido um erro de sincronização que provavelmente produziu todo efeito negativo no comportamento do protocolo. Percebemos que na segunda instrução a ação de escuta no BUS no processador de escuta é realizada no passo 011 (marcação vermelha) já na terceira instrução esta escuta ocorre no passo 000 (marcação azul) o que não deve acontecer já que a lógica de escuta está posicionada no passo 011.

3.7 Observações

Como a memória de dados foi implementada utilizando uma matriz de registradores durante a simulação pode não ter ficado claro o passo de *Write Back*, porém o mesmo foi implementado e pode ser observado no código do toplevel no passo 13 (01101).

4 Considerações Finais

A realização deste trabalho foi um grande desafio, porém, apesar da frustração de não haver conseguido o sucesso total, o sentimento foi de vitória, por haver chegado, neste trabalho, bem mais perto do sucesso, do que nos outros. O início da realização desta prática se iniciou após a semana de provas finais do grupo, de forma que, durante a semana de entrega, os integrantes se dedicaram quase que exclusivamente à execução desta tarefa. Esta dedicação se mostrou eficiente, pois rapidamente o grupo alcançou resultados positivos, podendo ver o funcionamento de algumas instruções passadas como teste. Após chegar neste ponto, o grupo recorreu ao monitor, a colegas e professores, numa tentativa de encontrar e corrigir quaisquer erros que

estivessem impedindo o sucesso completo da execução do código de testes, porém sem sucesso, o que gerou um característico sentimento de frustração.

No entanto, após a conclusão da atividade, o grupo conseguiu entender de maneira mais concreta sobre o funcionamento do protocolo Snooping, bem como sobre a importância da dedicação de ambos os integrantes para a realização de atividades complexas como esta. O grupo pôde perceber que não repetiu os mesmos erros cometidos em práticas anteriores, o que possibilitou um crescimento pessoal maior e desenvolvimento melhor da atividade passada.

Obs: As simulações podem ser vistas a parte caso não estejam visíveis no relatório.