

1. Introdução

Este trabalho apresenta e relata a construção do protocolo MSI Snooping de coerência de cache em Verilog. Foram implementados os módulos necessários para simular a mudança de etiquetas entre as caches L1 de três processadores distintos com quatro blocos de memória diretamente mapeados, cada um contendo estado, tag e dados. As instruções para teste foram explicitadas diretamente no código.

2.1 Desenvolvimento

Para este projeto optamos por utilizar instruções de 9 bits, sendo elas:

2 bits	1 bit	3 bits	3 bits
Proc.	Op (R/W)	Tag	Dados

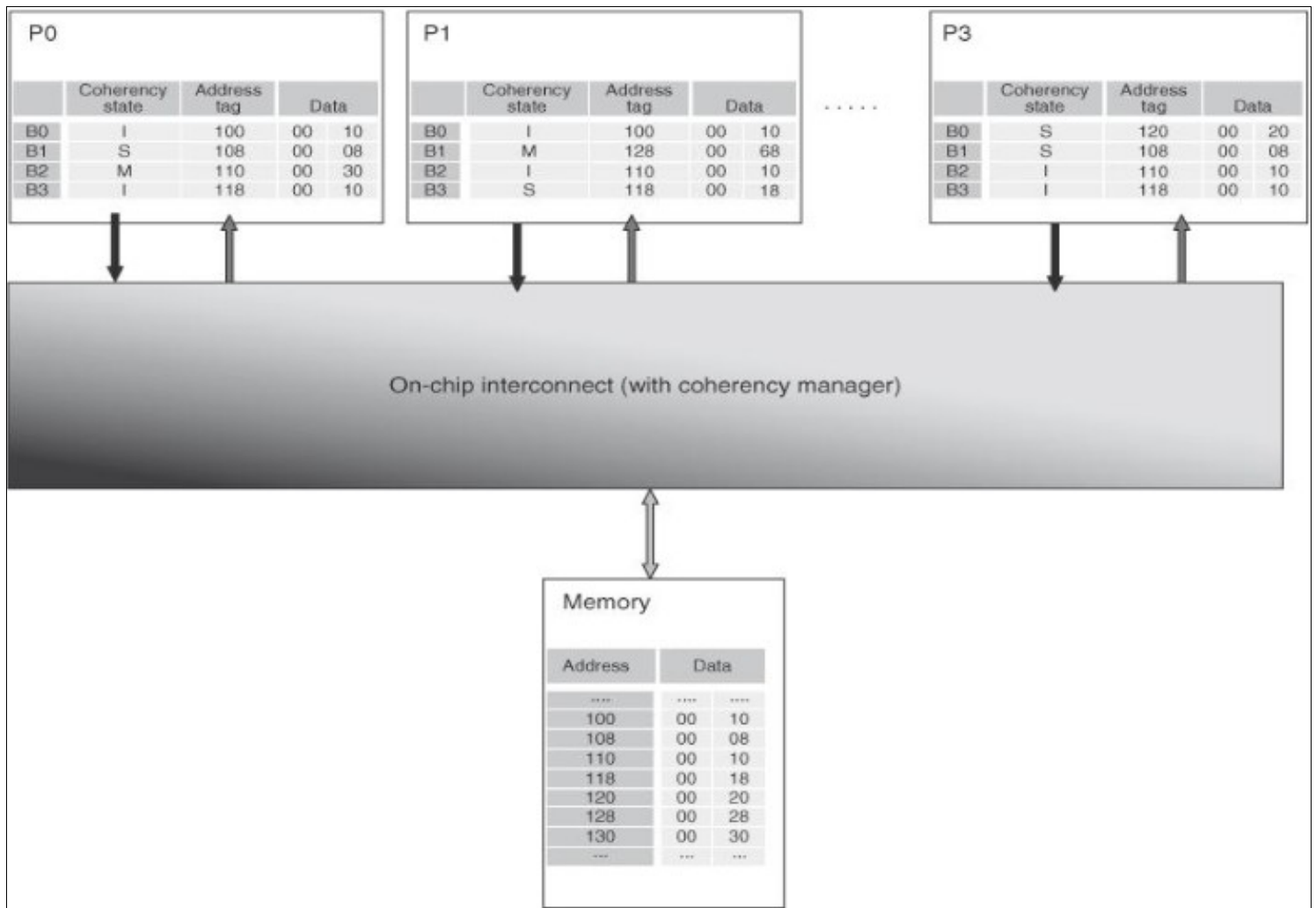
(Sendo a Read $\rightarrow 0$ e Write $\rightarrow 1$)

Além disso, tivemos que alterar os valores previsto pelo código de teste para adaptar às condições do projeto.

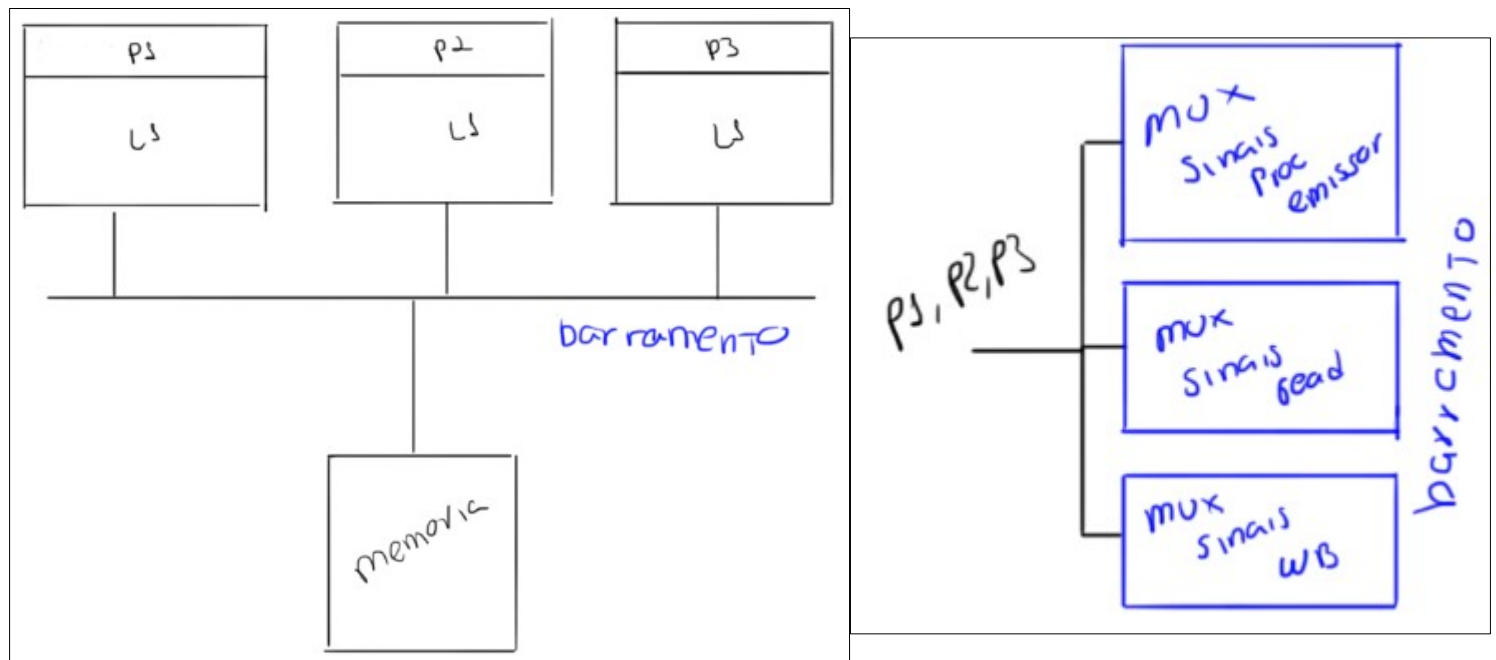
Blocos	
#0	100 \rightarrow B0 \rightarrow 000
#1	108 \rightarrow B1 \rightarrow 001
#2	110 \rightarrow B2 \rightarrow 010
#3	118 \rightarrow B3 \rightarrow 011
#4	120 \rightarrow B0 \rightarrow 100
#5	128 \rightarrow B1 \rightarrow 101
#6	130 \rightarrow B2 \rightarrow 110
Valores	
#0	08/68 \rightarrow 000
#1	10 \rightarrow 001
#2	18 \rightarrow 010
#3	20/28 \rightarrow 011
#4	30 \rightarrow 100
#5	48 \rightarrow 101
#6	78 \rightarrow 110
#7	80 \rightarrow 111

2.2 Módulos

Foi criado um diagrama a partir do esquema do algoritmo disponibilizado no livro da disciplina (Referência 1), além do arquivo do código teste.

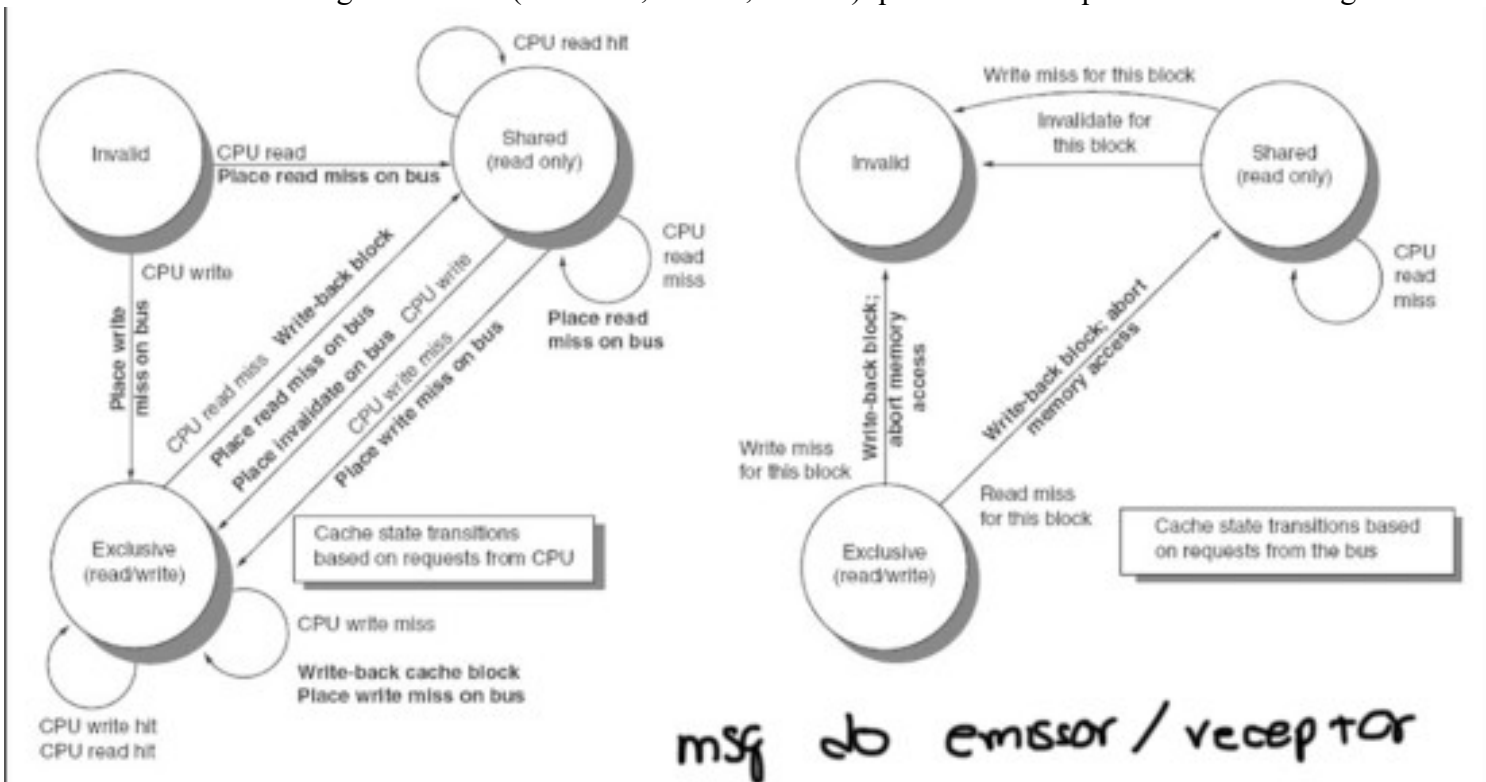


A partir do esquema, foram definidos os seguinte diagramas



- Processadores:
Realiza operações de *read* e *write*, armazena os blocos da cache e manipula as máquinas de estado.
- Máquinas de estado (Emissor e receptor)
A partir de valores recebidos, retorna as mudanças necessárias em um bloco de cache para preservar a coerência entre as caches e a memória.
- Memória
Armazena os valores à depender de suas tags.
- Barramento (Bus)
Escolhe o processador emissor e distribui suas saídas.
- Mux de sinais (WB/Read)
Escolhe o processador que vai utilizar a memória.

Foi utilizado o algoritmo MSI (Modified, Shared, Invalid) que utiliza a máquina de estados a seguir



2.3 Funcionamento

Enviamos manualmente uma instrução ao programa, em seguida, o processador escolhido “estuda” a instrução e define se ela é *read* ou *write*. Caso encontre a tag em sua cache, ele verifica a validade do bloco e realiza a operação requisitada. A seguir, a máquina emissora é chamada e são realizadas as operações de transição de estados e o resultado é distribuído para o resto dos módulos. Os receptores dos sinais verificam se existe a tag desejada e realiza as operações a partir da saída da máquina receptora.

3.1 Máquina de estados

A seguir estão as simulações que demonstram o correto funcionamento da máquina de estados. As imagens estão separadas por estado inicial. Segue também a equivalência entre os valores binários e o que representam.

Máquina

Emissor → 0

Receptor → 1

Estados

Invalido → 00

Modificado → 01

Compartilhado → 10

Operações

Read hit → 00

Read miss → 01

Write hit → 10

Write miss → 11

Mensagens

Invalidar → 00

Read miss → 01

Write miss → 10

Sem mensagem → 11

Máquina emissora

Inválido

maquina	emissor			
estadoAtual	00 invalido			
op	01 read miss	11 write miss		
novoEstado	10 compartilhado	01 modificado		
saidaMaquina	01 msg read miss	10 msg write miss		

Modificado







maquina	emissor				
estadoAtual	01 modificado				
op	00 read hit	01 read miss	10 write hit	11 write miss	
novoEstado	modificado	10 compartilhado	01 modificado	modificado	
saidaMaquina	11 sem msg	01 msg read miss	11 sem msg	10 msg write miss	

Compartilhado

maquina	emissor				
estadoAtual	10 compartilhado				
op	00 read hit	01 read miss	10 write hit	11 write miss	
novoEstado	10 compartilhado	compartilhado	01 modificado	modificado	
saidaMaquina	11 sem msg	01 msg read miss	00 msg invalidar	10 msg write miss	

Máquina receptora

Modificado

 maquina	receptor			
 estadoAtual	01 modificado			
 entradaMaquina	01 msg read miss	10 msg write miss		
 novoEstado	10 compartilhado	00 invalido		
 writeBack	verdadeiro			
 abortAccessMemory	verdadeiro			

Compartilhado

 maquina	receptor					
 estadoAtual	10 compartilhado					
 entradaMaquina	00	01	10			
 novoEstado	invalido	10 compartilhado	00 invalido			
 writeBack	falso					
 abortAccessMemory	falso					

3.2 Código teste

Utilizamos a sequência de instruções disponibilizadas pela professora.

A primeira linha de cada item representa a instrução e as linhas seguinte são o resultado esperado.

As mudanças significativas estão marcadas entre bordas vermelhas.

Clock → borda de descida, 200 ps, 50 duty

a) P0: read 120
P1.0(2, 4, 3)

instrução		
000100000		
barramento		
	01	read miss
write-back		
tag	4	
dado	3	
P1		
hit	início	0
estado	0 1 2 0	0 1 2 2
tag	3 2 1 0	3 2 1 4
dado	1 4 0 1	1 4 0 3
P2		
hit		0
estado	1 0 2 0	
tag	3 2 5 0	
dado	0 1 0 1	
P3		
hit		1
estado	0 0 2 2	
tag	3 2 1 4	
dado	1 1 0 3	
outP		
	0	
	0	
	0	

P1 → Read miss no barramento, busca o bloco 120 (#4) com valor 20 (#3) da memória e aloca no bloco B0, o estado passa a ser compartilhado (#2)

b) P0: write 120 <- - 80
P1.0(1, 4, 7)
P3.0(0, 4, 3)

instrução		
001100111		
barramento		
	00	invalidar
write-back		
tag		
dado		
P1		
hit		1
estado	0 1 2 1	
tag	4	
dado	1 4 0 7	
P2		
hit		
estado		
tag		
dado		
P3		
hit		
estado	0 0 2 0	
tag	4	
dado	3	
outP		

P1 → Write hit, invalidar no barramento, escreve no B0 a tag 120 (#4) e o valor 80 (#7), o estado passa a ser modificado (#1)

P3 → Recebe invalidar, o bloco B0 de tag 120 (#4) passa a ser invalido (#0)

c) P3: write 120 <- -80
P1.0(0, 4, 7)
P3.0(1, 4, 7)
M(4, 7)

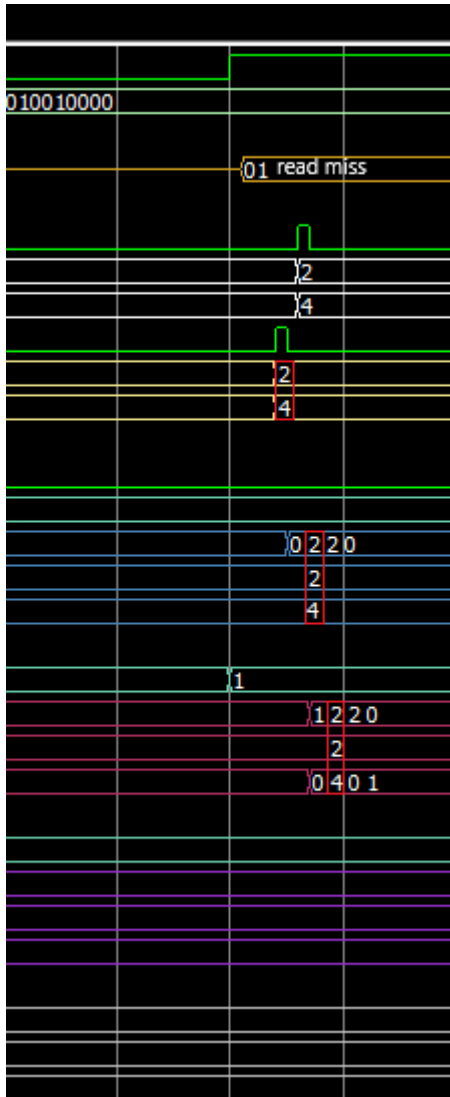
instrução		
101100111		
barramento		
	10	write miss
write-back		
tag	4	
dado	7	
P1		
hit		
estado	0 1 2 0	
tag	4	
dado		
P2		
hit		
estado		
tag		
dado		
P3		
hit		
estado	0 0 2 1	
tag	4	
dado	1 1 0 7	
outP		

P3 → Write miss no barramento, busca a tag 120 (#4) na cache de P1, aloca no bloco B0 e escreve o valor 80 (#7), o estado passa a ser modificado (#1)

P1 → Recebe invalidar e write-back, o bloco B0 de tag 120 (#4) passa a ser invalido (#0)

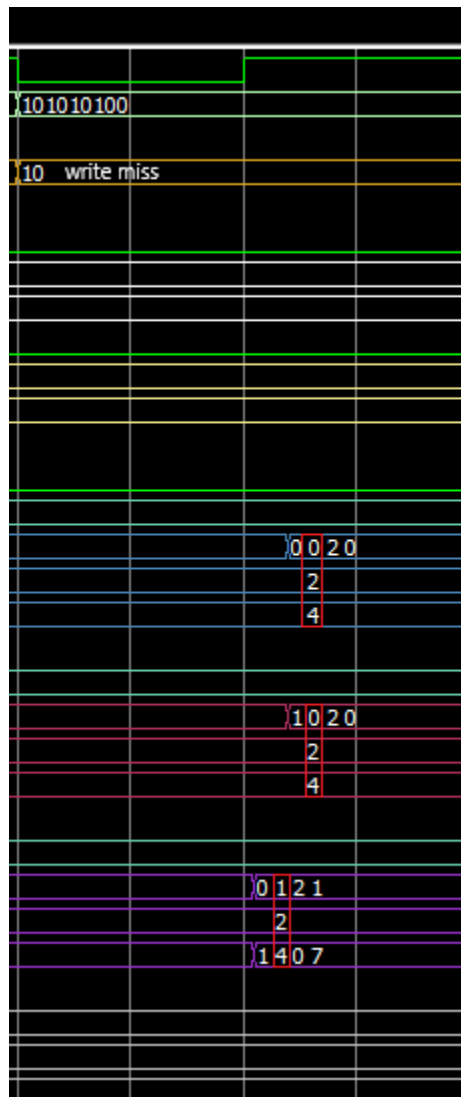
M → Sofre write-back na tag 120 (#4) com o valor 80 (#7)

d) P1: read 110
P1.2(2, 2, 4)
P2.2(2, 2, 4)
M(2, 4)



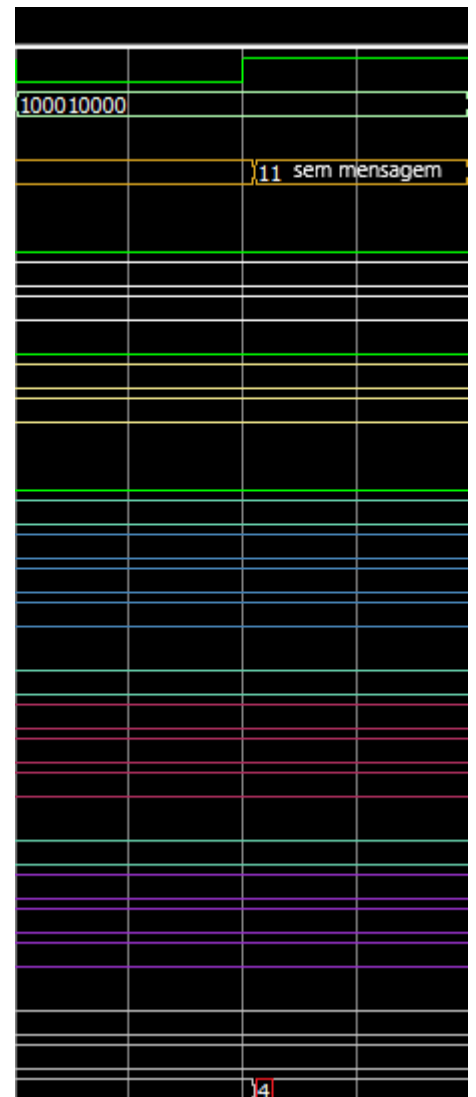
P2 → Read miss no barramento, busca a tag 110 (#2) na cache de P1 e aloca no bloco B2 com o valor 30 (#4), o estado passa a ser compartilhado (#2)
P1 → Recebe read miss e write-back, o estado do bloco B2 de tag 110 (#2) passa a ser compartilhado (#2)
M → Sofre write-back na tag 110 (#2) com o valor 30 (#4)

e) P3: write 110 <-- 30
P1.2(0, 2, 4)
P2.2(0, 2, 4)
P3.2(1, 2, 4)



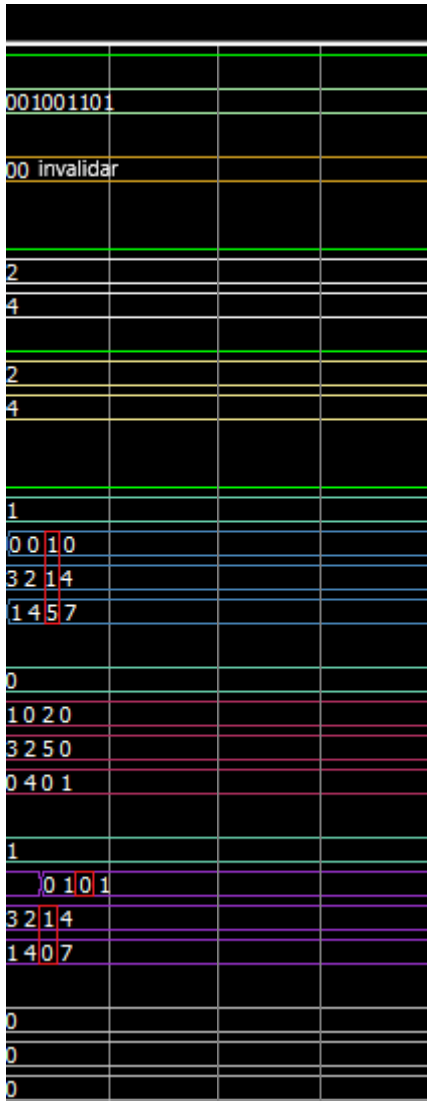
P3 → Write miss no barramento, busca a tag 110 (#2) na memória, aloca no bloco B2 e escreve o valor 30 (#4), o estado passa a ser modificado (#1)
P1 → Recebe invalidar, o bloco B2 de tag 110 (#2) passa a ser invalido (#0)
P2 → Recebe invalidar, o bloco B2 de tag 110 (#2) passa a ser invalido (#0)

f) P3: read 110
outp3(4)



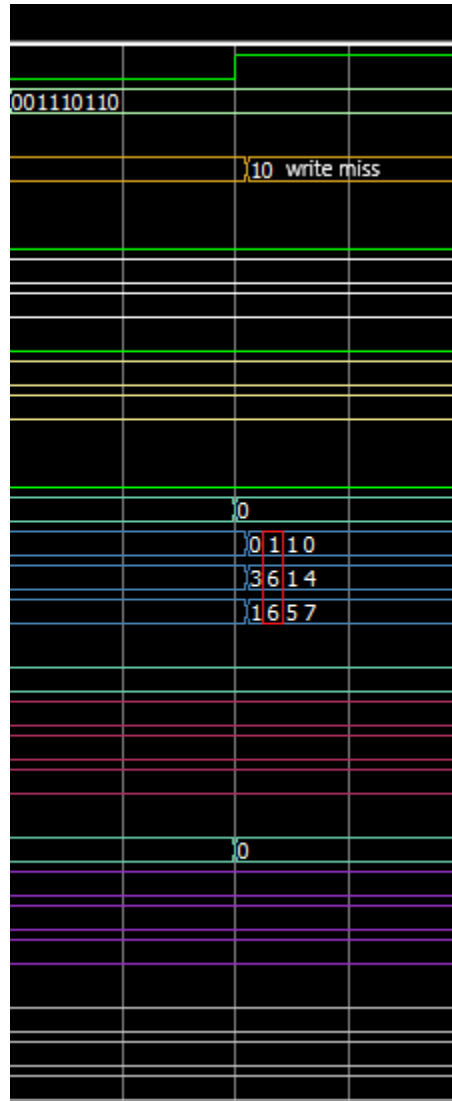
P3 → Read hit, retorna o valor 30 (#4) para o processador

g) P0: write 108 <-- 48
P1.1(1, 1, 5)
P3.1(0, 1, 0)



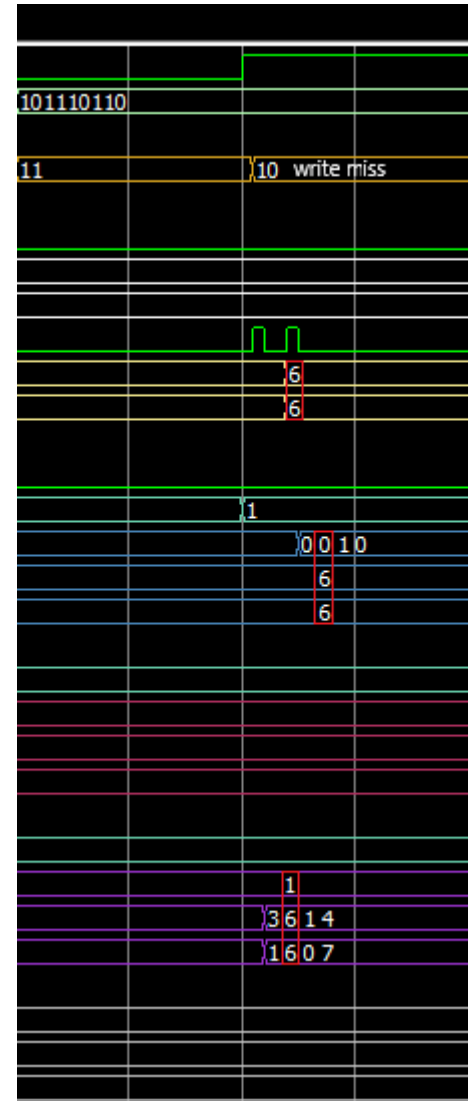
P0 → Write hit, escreve no bloco B1 de tag 108 (#1) o valor 48 (#5)
P3 → Recebe invalidar, o bloco B1 de tag 108 (#1) passa a ser invalido (#0)

h) P0: write 130 <--78
P1.2(1, 6, 6)



P0 → Write miss no barramento, busca a tag 130 (#6) na memória, aloca no bloco B2 e escreve o valor 78 (#6), o estado passa a ser modificado (#1)

i) P3: write 130 <--78
M(6, 6)
P1.2(0, 6, 6)
P3.2(1, 6, 6)
M(2, 1)



P3 → Write miss no barramento, busca a tag 130 (#6) na cache de P0, aloca no bloco B2 e escreve o valor 78 (#6), o estado passa a ser modificado (#1)
P0 → Recebe invalidar e write-back, o bloco B2 de tag 130 (#6) passa a ser invalido (#0)
M → Sofre write-back na tag 110 (#2) com o valor 10 (#1) (**Vindo da cache de P3**), então, sofre write-back na tag 130 (#6) com o valor 78 (#6) (**Vindo da cache de P0**)

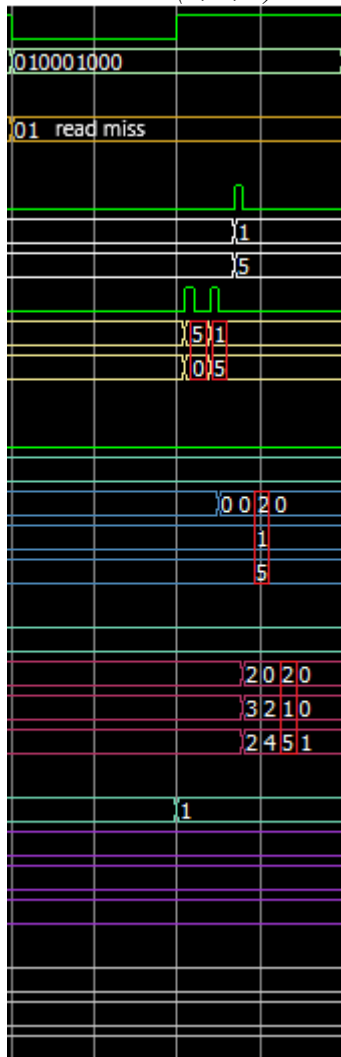
j) P1: read 108

$M(5, 0)$

$P1.2(2, 1, 5)$

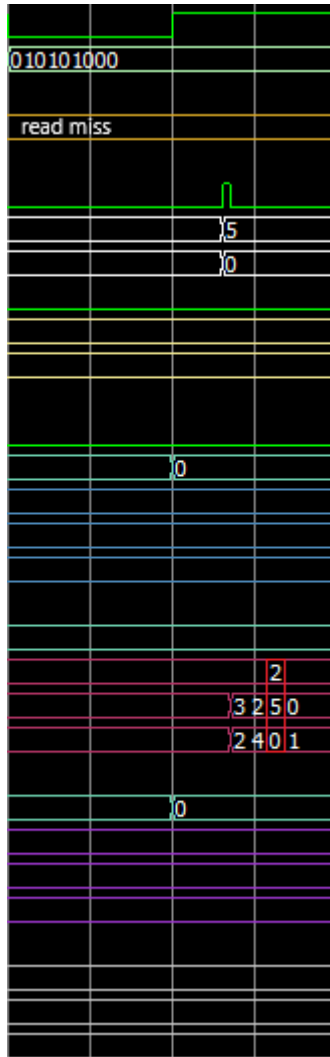
$M(1, 5)$

$P2.2(2, 1, 5)$



k) P1: read 128

$P2.2(2, 5, 0)$



P1 → Read miss no barramento, busca a tag 108 (#1) na cache de P1, aloca no bloco B1 e escreve o valor 48 (#5), o estado passa a ser compartilhado (#2)

P0 → Recebe read miss e write-back, o estado do bloco B1 de tag 108 (#1) passa a ser compartilhado (#2)

M → Sofre write-back na tag 128 (#5) com o valor 68 (#0) (**Vindo da cache de P1**), então, sofre write-back na tag 108 (#1) com o valor 48 (#5) (**Vindo da cache de P0**)

P1 → Read miss no barramento, busca a tag 128 (#5) na memória e aloca no bloco B1 com o valor 68 (#0), o estado passa a ser compartilhado (#2)

4. Dificuldades

Inicialmente planejamos estabelecer apenas um módulo representante de ambas máquinas de estados, porém, devido à um *bug* não identificado decidimos separar o emissor do receptor. Além disso, tivemos dificuldades para definir delays adequados e enviar sinais corretos aos processadores (write-back, sinal de máquina, etc...). Para este último, resolvemos usando *muxes* que controlam a entrada de valores na memória principal.

5. Sugestões

No geral, acreditamos que a maior das complicações encontradas nas práticas é o uso do Quartus e do ModelSim como principais ferramentas de trabalho, assim como o entendimento da lógica necessária para o uso do Verilog. Uma sugestão seria a procura de um novo conjunto de softwares capazes de simular projetos em linguagem de descrição de hardware e algumas aulas dedicadas à linguagem Verilog em si.

6. Comentários adicionais

Apesar dos desafios e a dificuldade de implementação, este projeto, em conjunto com os estudos do livro, nos auxiliou a compreender melhor o algoritmo de coerência de cache por *snooping*.