Cache 2-vias

Guimarães, João Guilherme M.
joaog95@live.com

9 de outubro de 2019

1 Introdução

A cache associativa de 2-vias é um *trade-off* entre a cache diretamente mapeada e a totalmente associativa. Seu funcionamento consiste em dividir a cache em *sets* de dimensão 2, o que possibilita uma maior taxa de acerto quando comparada a diretamente mapeada e uma redução do custo energético em relação a totalmente associativa.

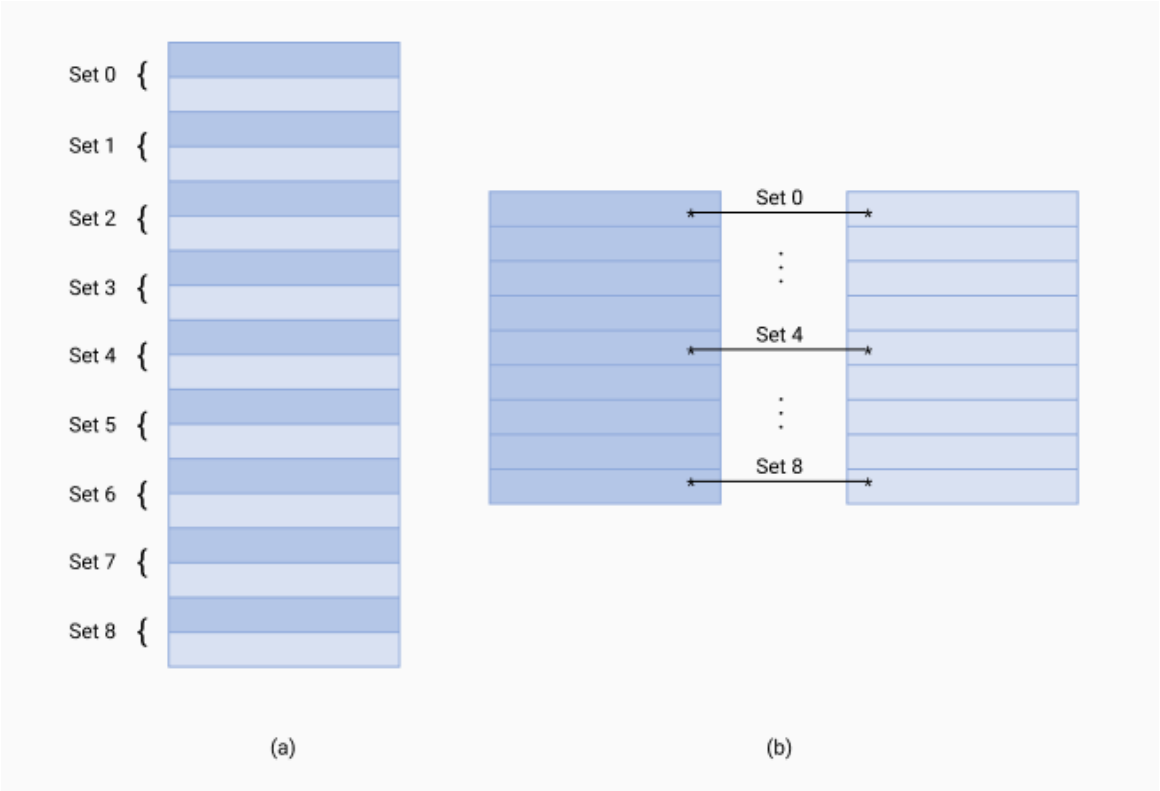

Figura 1: Cache 2-vias

Na figura 1, é representado as possíveis abstrações de uma cache de 2-vias, sendo que em *a* é utilizado um único módulo de memória, enquanto em *b*, são utilizados dois módulos. Neste projeto foi utilizado a abstração *a*.

2 Objetivos

Implementação de uma cache associativa de 2-vias, com utilização do arquivo MIF e realizar a leitura e escrita utilizando o display de 7-segmentos.

3 Material

Para realização desta prática, foi utilizado os seguintes equipamentos e softwares:

- ModelSim 10.1d;
- Quartus 13.0sp1;
- FPGA EP2C35F672C6 e
- Kernel Linux / SO Deepin 15.11.

4 Desenvolvimento

O primeiro passo para o desenvolvimento do projeto, foi a detalhação das especificações da cache, que são:

- Associatividade: 2-vias;
- Tamanho da palavra: 16 bits;
- Tamanho do bloco: 2 palavras e
- Tamanho da memória: 16 palavras.

A partir do detalhamento acima, foi proposto a abstração *a* da figura 1, pois o desenvolvimento do código seria facilitado, devido a necessidade da comunicação entre as caches. O próximo passo, foi a divisão dos campos do endereçamento, dado por:

- TAG: 8 - Index - Block Offset = 5
- Index: $ln(8/2) = ln(2^2) = 2$ bits
- Block Offset: ln(Palavras por bloco) = 1 bit

Com a obtenção do tamanho da TAG, foi definido as seguintes divisões para a cache:

- Valid: 1 bit;
- Dirty: 1 bit;
- Tag: 5 bits;
- Bloco: 16 bits e
- LRU: 1 bit

Para facilitar o entendimento da estrutura acima, foi criado o arquivo MIF.xlsx que é exibido na figura a seguir.

	Valid [1]	Dirty [1]	TAG [5]	Data [16]		LRU [1]		Output [24]
0	1	0	00101 [05]	11110001 [F1]	00010101 [15]	0		100010111110001000101010
	0	1	10001 [11]	00100011 [23]	11111010 [FA]	1		011000100100011111110101
1	1	1	00010 [02]	00010100 [14]	00010101 [15]	1		110001000010100000101011
	1	0	00111 [07]	00110010 [32]	01000101 [45]	0		100011100110010010001010
2	0	1	00111 [07]	00101111 [2F]	00111010 [3A]	1		010011100101111001110101
	0	0	10001 [11]	00000001 [01]	00100000 [20]	0		001000100000001001000000
3	0	0	00101 [05]	00001001 [09]	00000101 [05]	1		000010100001001000001011
	1	0	00001 [01]	00000010 [02]	00000011 [03]	0		100000100000010000000110

Figura 2: Estrutura da Cache

5 Simulação

Como discutido em sala, a simulação poderia ser simplificada devido a correta execução do código na placa da Altera, sendo assim, foi simulado a leitura com sucesso do endereço 13_{16} (Set 1, Bloco 0), a escrita no mesmo endereço utilizando o dado 51_{16} e posteriormente a leitura do endereço $3A_{16}$ (Set 1, Bloco 1).

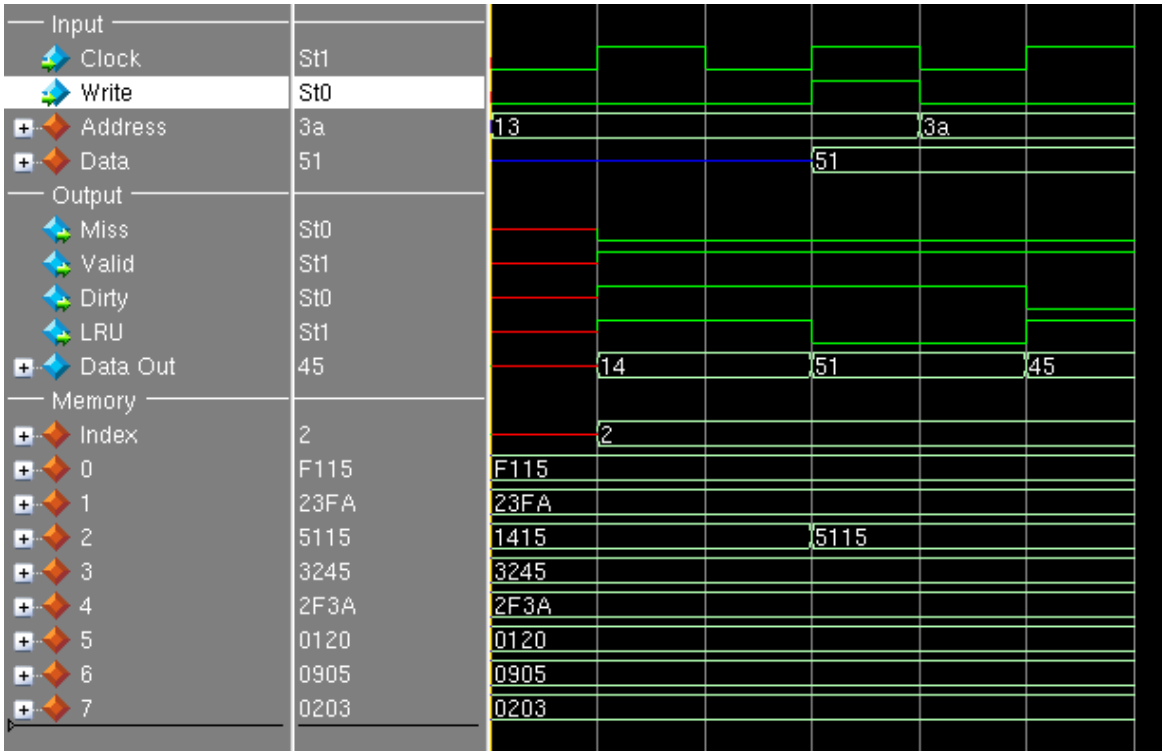

Figura 3: Leitura e Escrita na Cache

Obs.: como o módulo para a exibição no display de 7-segmentos foi reaproveitado da prática anterior, sua simulação não foi refeita.

6 Conclusão

Com a execução desta prática, foi possível entender melhor o funcionamento das políticas de pesquisa e inserção em uma cache de 2-vias e um entendimento dos problemas de assincronicidade do Clock em Verilog.