

LAOCII - PRÁTICA I

Tarcísio B. Prates (turma 2)

Francisco A. Gonçalves (turma 1)

Parte 1

Na primeira parte da prática, foi proposto a implementação de uma memória RAM utilizando a biblioteca LPM.

ESTRUTURA DA RAM

A memória foi estruturada com o endereçamento (*address [7:0]*) e tamanho de 8 bits.

Em relação às suas funcionalidades, a partir de um sinal de escrita/leitura (*wren*) é possível definir a operação de leitura (*wren = 0*), a qual, por meio do endereço informado na entrada, lê o seu dado armazenado na memória. Para a escrita (*wren = 1*), escreve o dado (*data_in[7:0]*) no endereço informado na memória.

Para ambas as operações, é mostrado o dado lido e escrito na saída (*data_out[7:0]*).

Segue abaixo, o código do módulo principal da memória RAM.

Módulo principal de acesso ao módulo de memória RAM

```
module memory(clock,data_out);

input clock;
output [7:0] data_out;

reg wren;
reg [7:0] address;
reg [7:0] data_in;

initial
begin

    // Escrita
    wren = 1;
    address = 8'b00001111;
    data_in = 8'b00011110; //30 número de chamada
    #50;

    wren = 1;
    address = 8'b11110000;
    data_in = 8'b00000011; // 3 número de chamada
    #100;

    // Leitura
    wren = 0;
    address = 8'b00001111;
    #150;

    wren = 0;
    address = 8'b11110000;
    #200;
```

```
end
```

```
// Módulo de acesso à memória  
ram_memory _RAM_ (address,clock,data_in,wren,data_out);  
// data_out recebe o valor lido na memória
```

```
endmodule
```

Simulação no ModelSim

Para fins de teste do funcionamento da memória RAM, foram passados os seguintes valores de entrada:

Ciclo 1:

```
wren = 1 // Escrita  
address = 00001111  
data_in = 00011110
```

Ciclo 2:

```
wren = 1 // Escrita  
address = 11110000  
data_in = 00000011
```

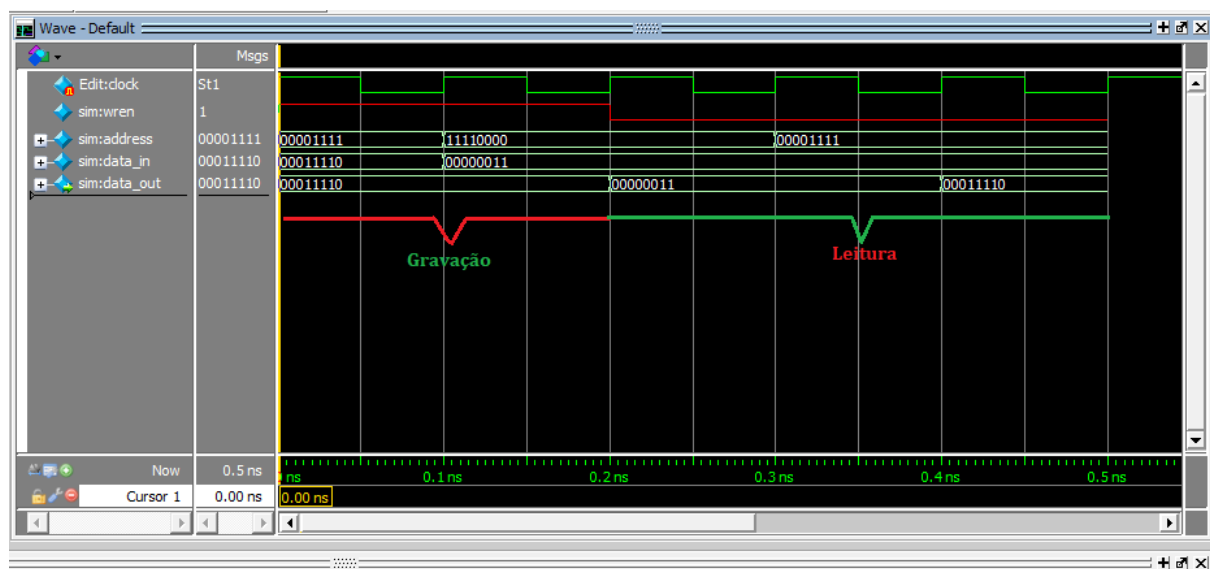
Ciclo 3:

```
wren = 0 // Leitura  
address = 11110000  
data_out = 00000011 (saída esperada)
```

Ciclo 4:

```
wren = 0 // Leitura  
address = 00001111  
data_out = 00011110 (saída esperada)
```

Esses valores foram testados no ModelSim, como mostra a imagem abaixo:



Observe que no instante **0.2 ns** o estado do registrador wren muda de alto para baixo, indicando que o circuito passou a não mais gravar na memória, e sim ler os dados conforme o endereço passado.

Parte 2

Na segunda parte da prática, foi proposto a utilização de um arquivo .mif para inicializar a memória RAM utilizando a biblioteca LPM implementada na parte 1 da prática.

ESTRUTURA DO ARQUIVO

O arquivo .mif foi utilizado para fins de inicialização de valores na memória RAM. Para as duas primeiras posições, foram inseridos os números de chamada da dupla e valores em sequência após o número 30, para as posições subsequentes. Um print do arquivo pode ser visto logo abaixo:

memory_MIF.mif									
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	3	30	31	32	33	34	35	36	... !"#
8	37	38	39	40	41	42	43	44	%&'()*+,
16	45	46	47	48	49	50	51	52	-.01234
24	53	54	55	56	57	58	59	60	56789;:<

ESTRUTURA DA RAM

A estrutura da memória RAM utilizada na parte 2 é a mesma da parte 1, como mostra o código abaixo:

```

module memory(clock,data_out);

input clock;
output [7:0] data_out;

reg wren;
reg [7:0] address;
reg [7:0] data_in;

initial
begin
    wren = 0;
    address = 8'b00000000;
    #100;
end

// Contador para percorrer todas as posições da memória
always @(posedge clock)
begin
    address = address + 1;
end

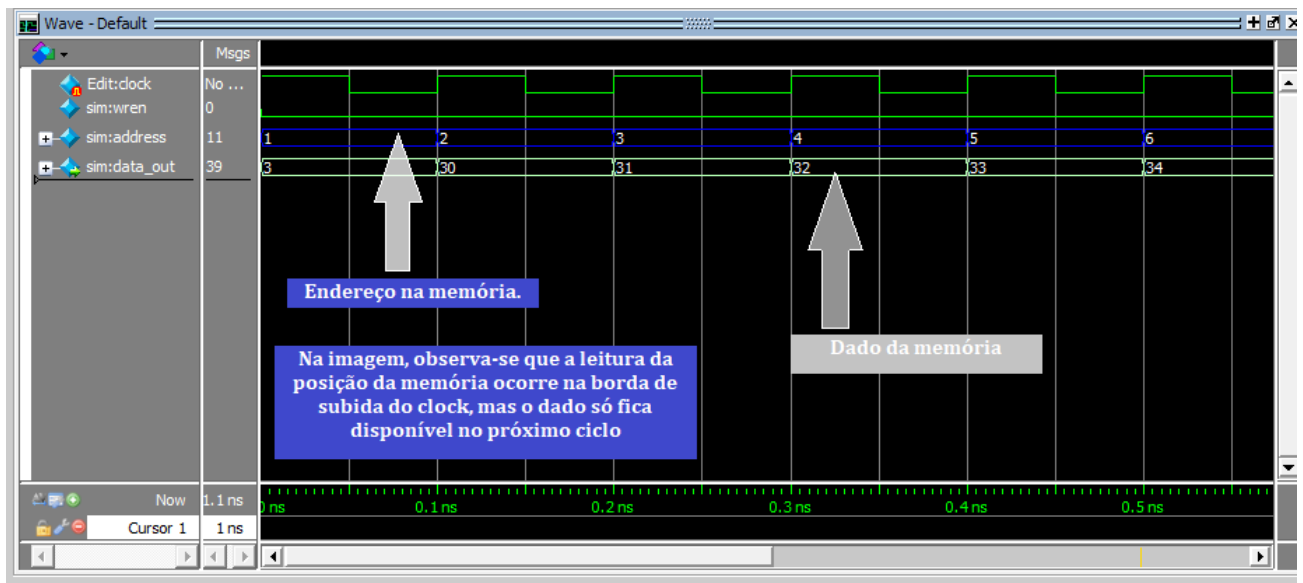
// Módulo LPM
ram_memory _RAM_ (address,clock,data_in,wren,data_out);

endmodule

```

Simulação no ModelSim

Para a simulação, iniciou-se o endereço (*address [7:0]*) na posição 0 e, a cada ciclo de clock, incrementava-se 1 no valor. Desse modo, com o *wren = 0*, leu-se todos os valores salvos na memória, a partir do arquivo .mif. Na imagem abaixo, os valores são impressos em sequência pelo *data_out [7:0]*.



Na imagem, observa-se que a leitura da posição da memória ocorre na borda de subida do clock, mas o dado só fica disponível no próximo ciclo

Parte 3

Na parte 3, foi proposto a criação de uma memória cache associativa por conjunto de 2 vias. Formato da instrução

```
module main (clock, HEX0) ;

input clock;
output [6:0] HEX0;

reg [8:0] instruction[7:0];
reg [2:0] index;

wire [2:0] data_out;
wire [6:0] display;

initial begin
    index = 0;
    instruction[0] = 9'b000100zzz;
    instruction[1] = 9'b000101zzz;
    instruction[2] = 9'b000100zzz;
    instruction[3] = 9'b101000111;
```

```

        instruction[4] = 9'b110111010;
        instruction[5] = 9'b110110011;
        instruction[6] = 9'b010001zzz;

end

always @(posedge clock)
begin
    index = index + 1;
end

conf_7seg_out_display_(clock,{1'b0, data_out},display);

cache cache_memory(clock, instruction[index], miss, wback, data_out);

assign HEX0 = display;

endmodule

```

ESTRUTURA DA CACHE

A cache implementada foi uma memória associativa de 2 vias com profundidade de 2 bits. Cada bloco possui 9 bits de tamanho , sendo este com o seguinte formato:

[8:8]	[7:7]	[6:6]	[5:3]	[2:0]] <- Índice
[valid]	[dirty]	[LRU]	[TAG]	[DATA]] <- Função
[1 bit]	[1 bit]	[1 bit]	[3 bits]	[3 bits]] <- Tamanho

valid: Informa se o bloco associado está válido.

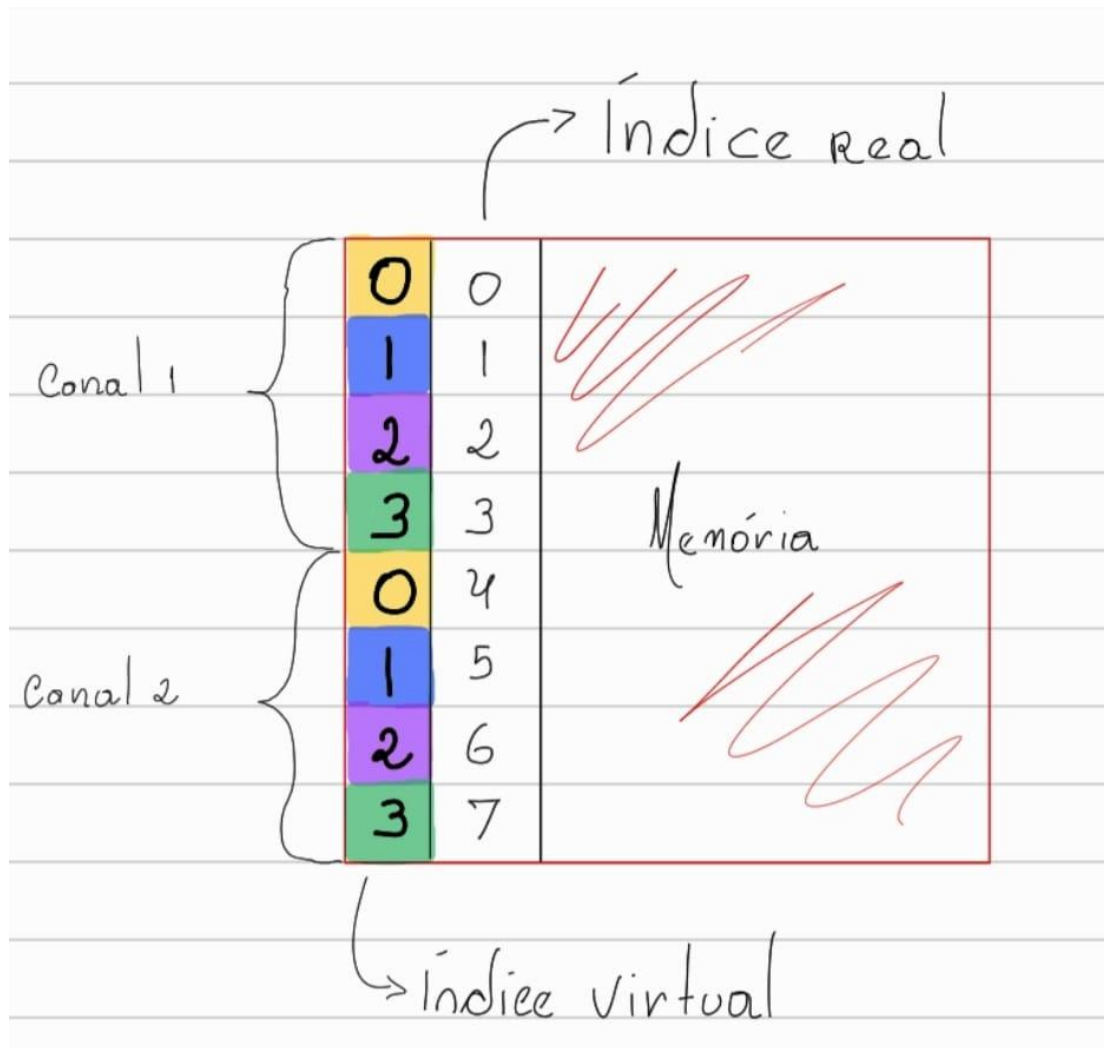
dirty: Informa se o bloco associado está sujo. Caso esteja antes de uma instrução ser tomada no bloco, ocorre o writeback do dado.

LRU: Informa se o bloco foi o mais recentemente usado entre as duas vias. Caso seja verdade, a escrita naquele índice será tomada no outro bloco.

TAG: Tag do bloco para uma cache associativa.

Data: Dado a ser escrito no bloco.

Por ser uma cache de 2 vias, os blocos da primeira via foram atribuídos para as 4 primeiras posições do vetor ([7:0] cache [8:0]) e os da segunda via para o 4 últimos, como mostra a representação abaixo:



ESTRUTURA DA INSTRUÇÃO

A memória cache recebeu como entrada uma instrução de 9 bits. A partir de sua especificação, os comandos foram executados. O formato da cache segue abaixo:

[8:8]	[7:6]	[5:3]	[2:0]] <- índice
[wr]	[index]	[TAG]	[DATA]] <- Função
[1 bit]	[2 bits]	[3 bits]	[3 bits]] <- Tamanho

wr: Indica se o comando é de escrita ($wr = 1$) ou leitura ($wr = 0$).

index: Indica o index do bloco a ser executado a instrução.

TAG: Indica a Tag do bloco a ser executada a instrução.

DATA: Indica o dado a ser escrito no bloco.

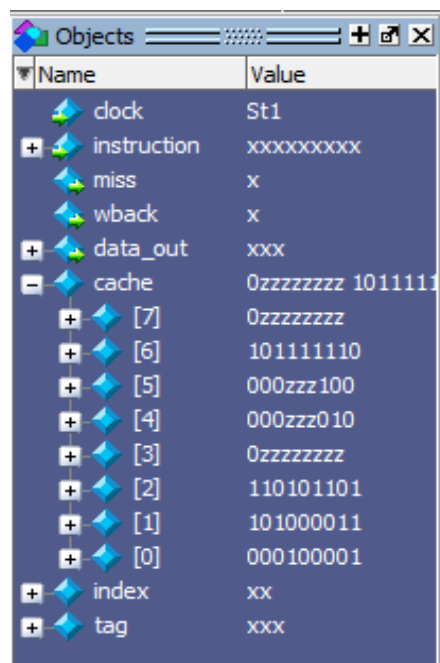
Simulação no ModelSim

As seguintes instruções foram passadas em sequência para objetivo de teste da memória cache:

Instruction 1 = 000100zzz;
Instruction 2 = 000101zzz;
Instruction 3 = 000100zzz;
Instruction 4 = 101000111
Instruction 5 = 110111010
Instruction 6 = 110110011
Instruction 7 = 010001zzz

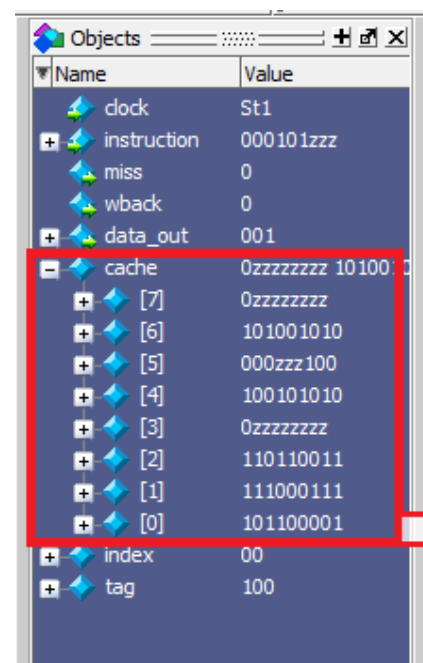
Na simulação foram observados os sinais de miss, write back e data out. O estado inicial e final da cache são mostrados abaixo.

ESTADO INICIAL



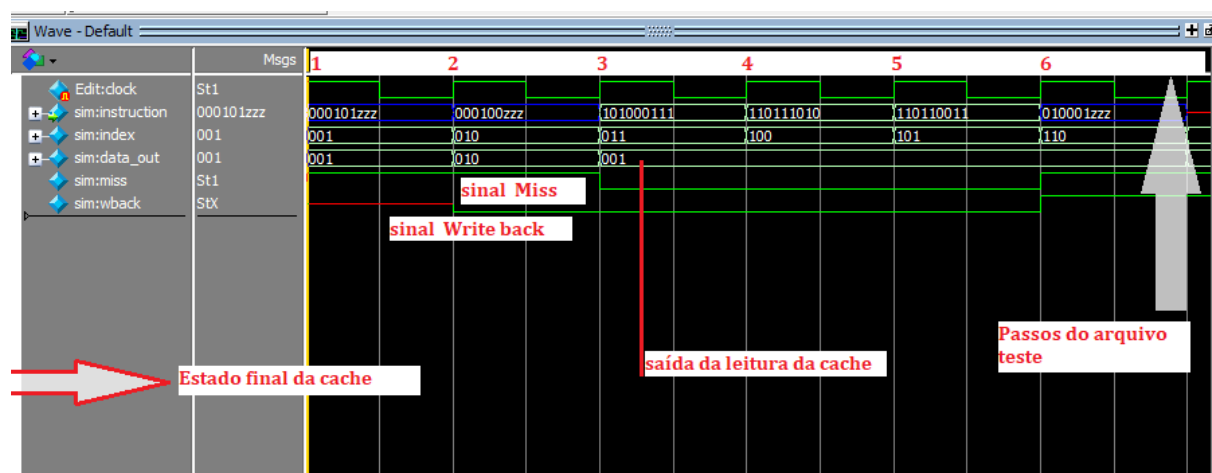
Name	Value
clock	St1
instruction	xxxxxxxx
miss	x
wback	x
data_out	xxx
cache	0zzzzzzzz 1011111
[7]	0zzzzzzzz
[6]	10111110
[5]	000zzz100
[4]	000zzz010
[3]	0zzzzzzzz
[2]	110101101
[1]	101000011
[0]	000100001
index	xx
tag	xxx

ESTADO FINAL

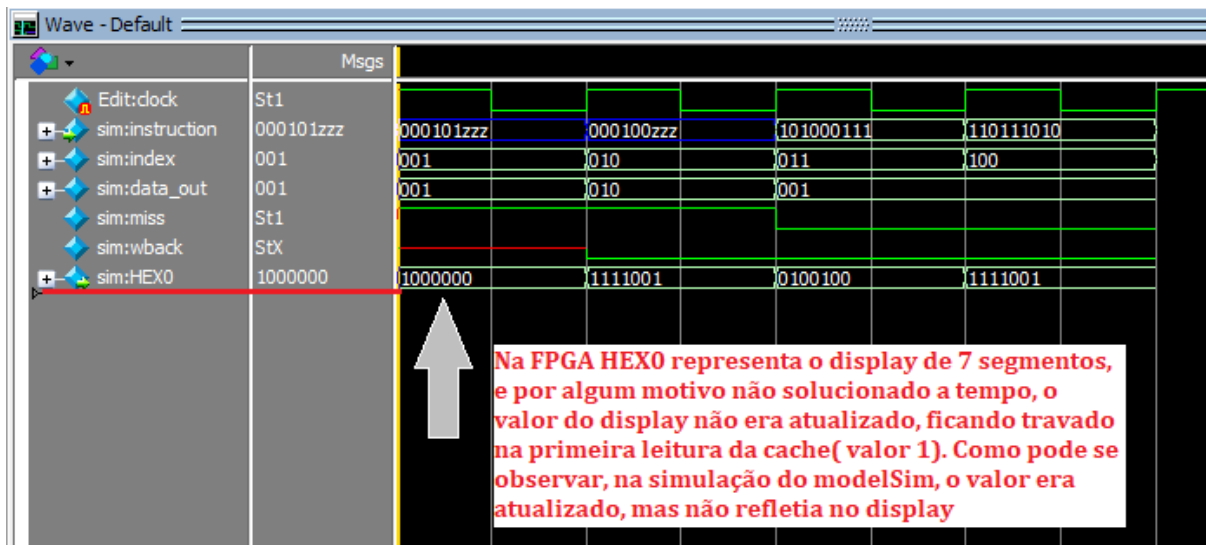


Name	Value
clock	St1
instruction	000101zzz
miss	0
wback	0
data_out	001
cache	0zzzzzzzz 010001zzz
[7]	0zzzzzzzz
[6]	101001010
[5]	000zzz100
[4]	100101010
[3]	0zzzzzzzz
[2]	110110011
[1]	111000111
[0]	101100001
index	00
tag	100

Segue abaixo o teste de ondas no ModelSim



Falha na simulação na FPGA



Na FPGA HEX0 representa o display de 7 segmentos, e por algum motivo não solucionado a tempo, o valor do display não era atualizado, ficando travado na primeira leitura da cache(valor 1). Como pode se observar, na simulação do modelSim, o valor era atualizado, mas não refletia no display