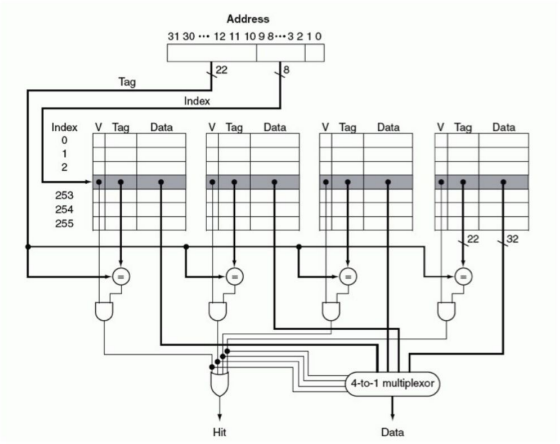
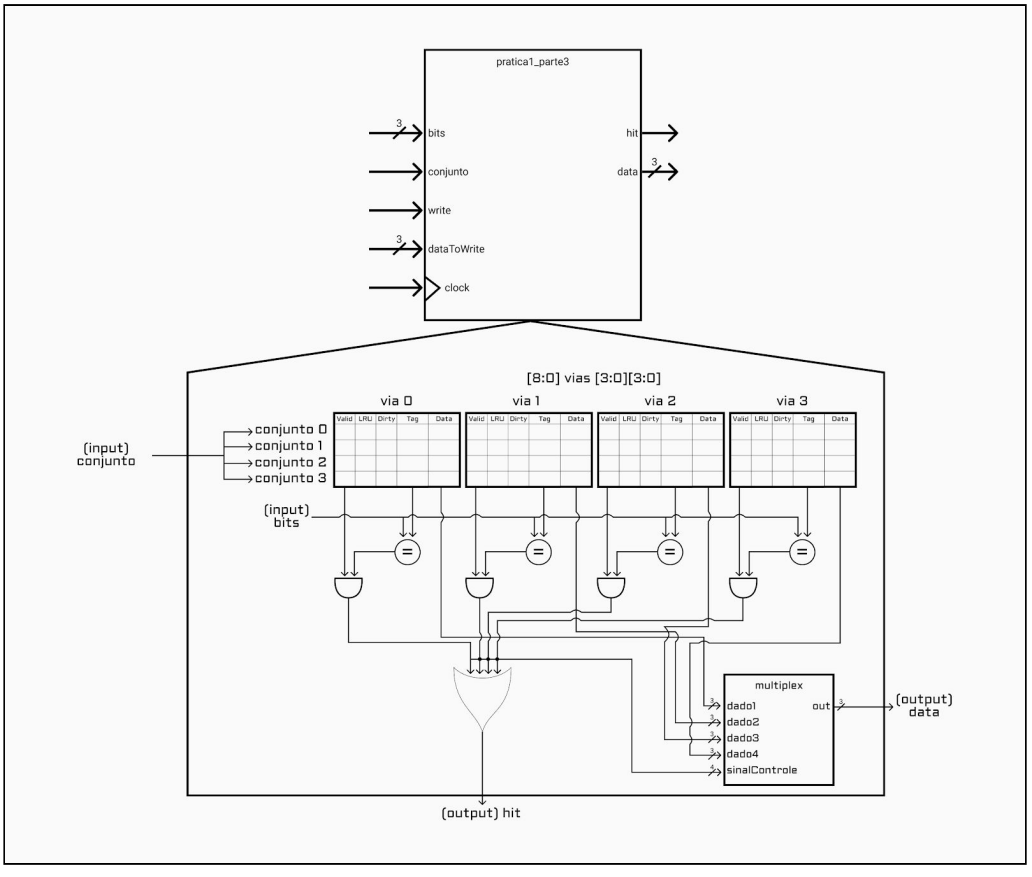
RELATÓRIO LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES 2 - Prática 1 Parte III

Alunos:
Julio Cesar Rocha
Iagor de Sousa

INTRODUÇÃO:

Para a terceira parte da prática 1 tivemos de implementar uma cache associativa de 4 vias, considerando os casos de acertos e falhas, além dos bits de: Validade, LRU e Dirty. Inicialmente a dupla se reuniu para debater sobre como seria realizada a implementação da cache, analisando quais seriam as entradas, saídas e registradores internos que usariamos.





DESENVOLVIMENTO:

Começamos pela implementação do Multiplexador 4-pra-1, que de certa forma foi simples, criamos um módulo que receberia os dados de determinado conjunto e como sinal de controle do Multiplexador foi escolhido como sendo o resultado da lógica booleana “Bit de Validade = 1 AND TAG procurada ser igual a TAG do elemento”. Código do módulo multiplexador:

```
module multiplex(dado1, dado2, dado3, dado4, sinalControle,out);
      input [2:0] dado1, dado2, dado3, dado4;
      input [3:0] sinalControle;
      output reg [2:0] out;
      integer i,j;

      always @(dado1, dado2, dado3, dado4, sinalControle) begin
            j = 1;

            for(i = 0; i < 4 && j; i  = i + 1) begin
                  if(sinalControle[i] == 1)
                        j = 0;
            end
            i = i - 1;

            case (i)
                  0: out <= dado1;
                  1: out <= dado2;
                  2: out <= dado3;
                  3: out <= dado4;
            endcase
      end
endmodule
```

Feito o multiplexador, começamos a projetar a cache e suas vias/conjuntos, inicializando toda a memória com o valor 0 e posteriormente atribuindo os valores referentes ao arquivo “Teste_Prática1.pdf”:

```
initial begin
            for(i=0; i < 4; i = i + 1)begin
                  for(j=0; j < 4; j = j + 1) begin
                        vias[i][j] = 9'b0;
                  end
            end

            vias[0][0] = 9'b110001001;
            vias[0][1] = 9'b110000010;
            vias[1][1] = 9'b110010110;
```

```
            vias[2][1] = 9'b110001011;
            vias[3][1] = 9'b000011000;
            vias[0][2] = 9'b110000101;
            vias[0][3] = 9'b000100000;
end
```

Para encontrar a TAG correspondente começamos analisando os bits referentes à TAG de todo conjunto selecionado. Salvamos em um vetor booleano com base na via quais TAG's eram iguais a TAG procurada, além dos bits de Validade e LRU de elemento do conjunto. Caso não encontrasse nenhuma TAG no conjunto igual à TAG procurada fazemos a avaliação do bit LRU de cada elemento do conjunto, pegamos o primeiro elemento encontrado com o bit de LRU = 0 e salvamos sua Via/Conjunto para atribuição/atualização dos bits do elemento.

```
      integer firstHitTagIndex;
      integer encontrou;
      integer lastLRU;


always@(posedge clock) begin
            //Numero arbitrario maior que o indice de hit
            firstHitTagIndex = 4;
            encontrou = 0;
            break = 1;
            for(i = 0; i < 4 && break; i = i + 1) begin
                  tagEquals[i] = vias[i][conjunto][5:3] == bits;
                  validBit[i] = vias[i][conjunto][8];
                  LRU[i] = vias[i][conjunto][7];
                  $display("LRU[%d] = %d", i, LRU[i]);

                  hits[i] = validBit[i] & tagEquals[i];
                  datas[i] = vias[i][conjunto][2:0];
                  // Encontrou a primeira correspondÃªncia de TAG
                  if (tagEquals[i] && encontrou == 0) begin
                        firstHitTagIndex = i;
                        lastLRU = i;
                        encontrou = 1;
                  end
            end

            // NÃ£o encontrou correspondÃªncia de TAG
            break = 1;
            if(firstHitTagIndex == 4)
                  for(i=0; i<4 && break; i = i + 1)
                        if(LRU[i] == 0)begin
                              lastLRU = i;
                              firstHitTagIndex = i;
                              break = 0;
                        end
```

Depois de encontrar a TAG correspondente e a Via do elemento a ser atualizado analisamos os bits “Hit” do conjunto, caso algum deles esteja em 1, então seus 3 bits de dados são passados para ser a saída “Data” do multiplexador, caso não há nenhum elemento com Hit em estado lógico alto, então os 3 bits de dados serão os 3 bits de TAG. Quando é necessário realizar uma escrita/Write e o Hit está em estado lógico alto, os 3 bits de dados serão os 3 bits da variável de entrada “dataToWrite”.

```
hit = |hits;

if(hit) dataWrite = datas[firstHitTagIndex];
else dataWrite = bits;

if(hit && write)begin dirty = 1; dataWrite = dataToWrite; end
else dirty = 0;

vias[firstHitTagIndex][conjunto] = {1'b1, 1'b1, dirty, bits, dataWrite};
```

Por fim fazemos a verificação dos bits de LRU do conjunto, caso todos estejam em 1, então atualizamos todos para 0 (menos o LRU do último elemento atualizado).

```
LRU[firstHitTagIndex] = 1;

if(&LRU)begin
        for(i = 0; i < 4; i = i +1)
                if(i != lastLRU) vias[i][conjunto][7] = 0;
end
end
```

SIMULAÇÃO:

Para simulação utilizamos um Clock com as seguintes propriedades:
Offset: 0
Duty: 50
Period: 100
First Edge: Rising

O código final do projeto:

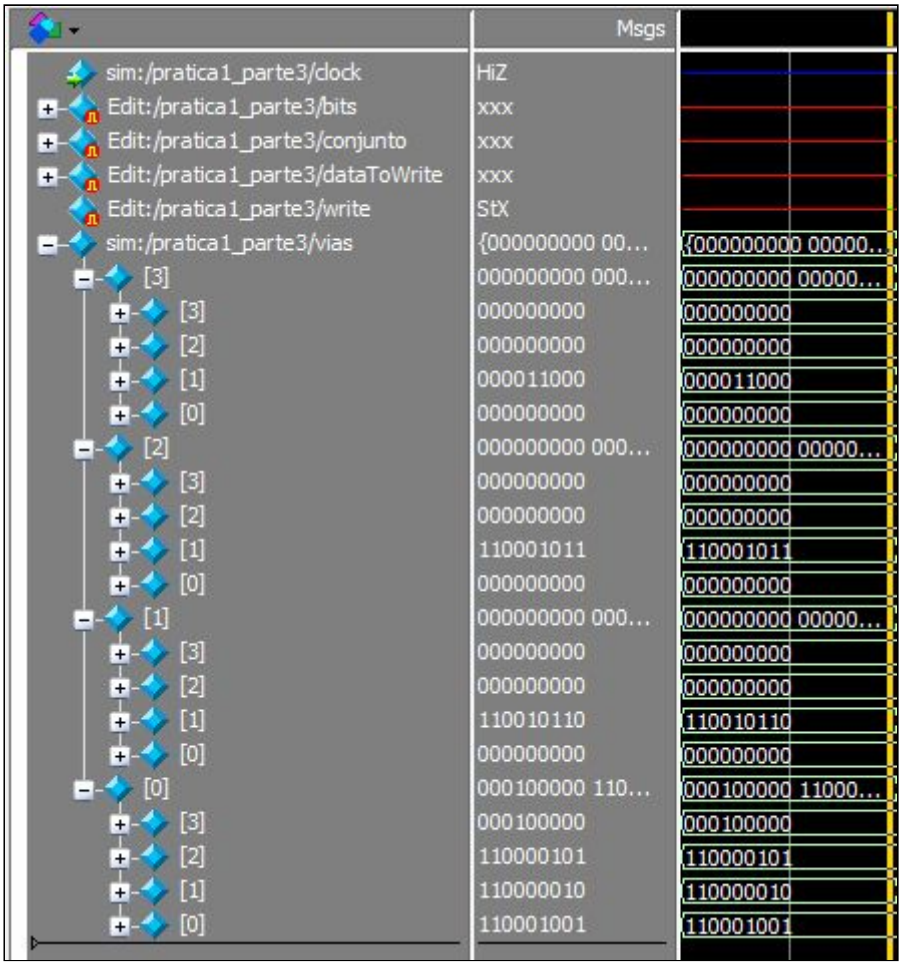
```
module multiplex(dado1, dado2, dado3, dado4, sinalControle,out);
      input [2:0] dado1, dado2, dado3, dado4;
      input [3:0] sinalControle;
      output reg [2:0] out;
      integer i,j;

      always @(dado1, dado2, dado3, dado4, sinalControle) begin
            j = 1;

            for(i = 0; i < 4 && j; i  = i + 1) begin
                  if(sinalControle[i] == 1)
                        j = 0;
```

```
            end
            i = i - 1;

            case (i)
                  0: out <= dado1;
                  1: out <= dado2;
                  2: out <= dado3;
                  3: out <= dado4;
            endcase
      end
endmodule

module pratica1_parte3(
      input clock,
      input [2:0]bits,
      input [2:0]conjunto,
      input write,
      input [2:0]dataToWrite,
      output reg hit,
      output [2:0]data);

      reg [8:0] vias [3:0][3:0]; // 4 vias de 8 espaÃ§os de 9
bits
      reg [3:0] hits;
      reg [2:0] datas [3:0]; // dados de 3 bits vindos de 4 vias
      reg tagEquals [3:0];
      reg validBit [3:0];
      reg [3:0] LRU;
      reg dirty;
      reg [2:0] dataWrite;
      integer i,j, break;

      initial begin
            for(i = 0; i < 4; i = i + 1)begin
                  for(j = 0; j < 4; j = j + 1) begin
                        vias[i][j] = 9'b0;
                  end
            end

            vias[0][0] = 9'b110001001;
            vias[0][1] = 9'b110000010;
            vias[1][1] = 9'b110010110;
            vias[2][1] = 9'b110001011;
            vias[3][1] = 9'b000011000;
            vias[0][2] = 9'b110000101;
            vias[0][3] = 9'b000100000;

            //Teste 1
            //Conjunto = 1
            //Bits = 011
            //Write = 0

            //Teste 2
            //Conjunto = 3
            //Bits = 100
```

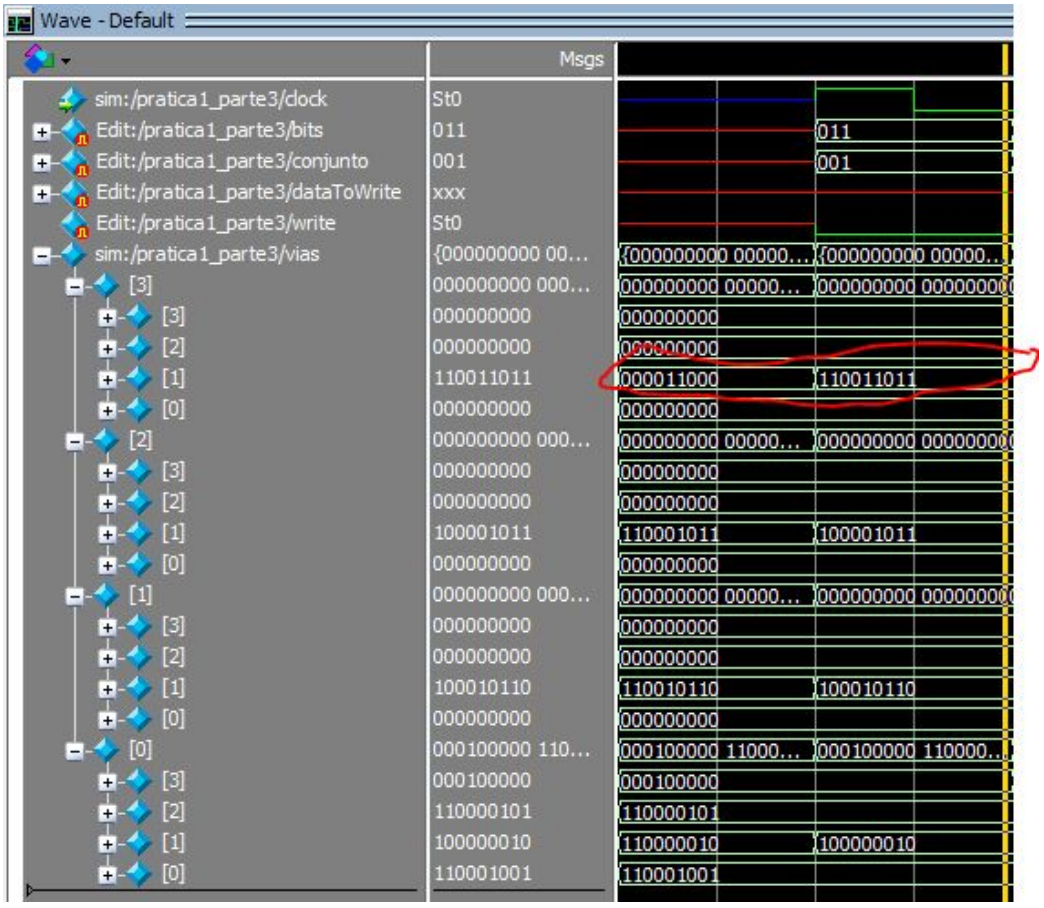
```
        //Write = 0

        //Teste 3
        //Conjunto = 1
        //Bits = 000
        //Write = 1
        //dataToWrite = 111

        //Teste 4
        //Conjunto = 1
        //Bits = 100
        //Write = 0

        //Teste 5
        //Conjunto = 1
        //Bits = 001
        //Write = 0

        //Teste 6
        //Conjunto = 3
        //Bits = 000
        //Write = 0

        //Teste 7
        //Conjunto = 1
        //Bits = 101
        //Write = 0


    end

    integer firstHitTagIndex;
    integer encontrou;
    integer lastLRU;

    always@(posedge clock) begin
        //Numero arbitrario maior que o indice de hit
        firstHitTagIndex = 4;
        encontrou = 0;
        break = 1;
        for(i = 0; i < 4 && break; i = i + 1) begin
            tagEquals[i] = vias[i][conjunto][5:3] == bits;
            validBit[i] = vias[i][conjunto][8];
            LRU[i] = vias[i][conjunto][7];
            $display("LRU[%d] = %d", i, LRU[i]);

            hits[i] = validBit[i] & tagEquals[i];
            datas[i] = vias[i][conjunto][2:0];
            // Encontrou a primeira correspondÃªncia de TAG
            if (tagEquals[i] && encontrou == 0) begin
                firstHitTagIndex = i;
                lastLRU = i;
                encontrou = 1;
            end
        end
```

```
          // NÃ£o encontrou correspondÃªncia de TAG
          break = 1;
          if(firstHitTagIndex == 4)
               for(i=0; i<4 && break; i = i + 1)
                    if(LRU[i] == 0)begin
                         lastLRU = i;
                         firstHitTagIndex = i;
                         break = 0;
                    end

          hit = |hits;

          if(hit) dataWrite = datas[firstHitTagIndex];
          else dataWrite = bits;

          if(hit && write)begin dirty = 1; dataWrite =
dataToWrite; end
          else dirty = 0;

          vias[firstHitTagIndex][conjunto] = {1'b1, 1'b1, dirty,
bits, dataWrite};

          LRU[firstHitTagIndex] = 1;

          if(&LRU)begin
               for(i = 0; i < 4; i = i +1)
                    if(i != lastLRU) vias[i][conjunto][7] = 0;
          end
     end

     multiplex
mult(datas[0],datas[1],datas[2],datas[3],hits,data);

endmodule
```

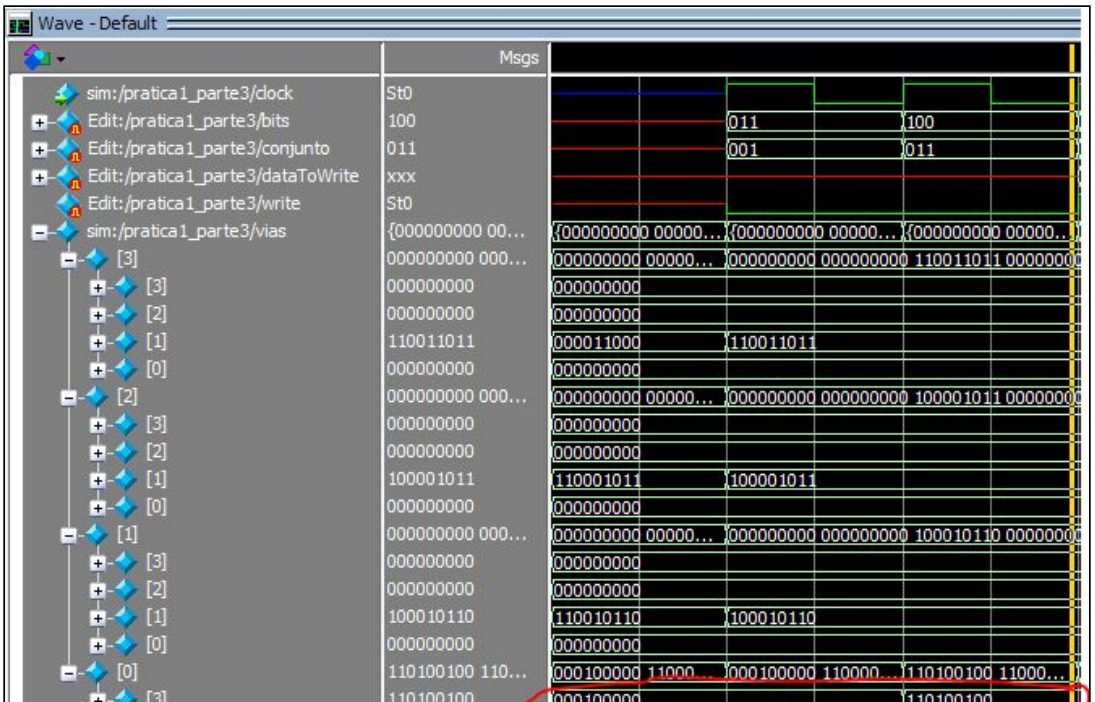
- Inicialmente é feita a inicialização da memória com os valores de teste:
  Tempo: 0ns ~ 0.1ns



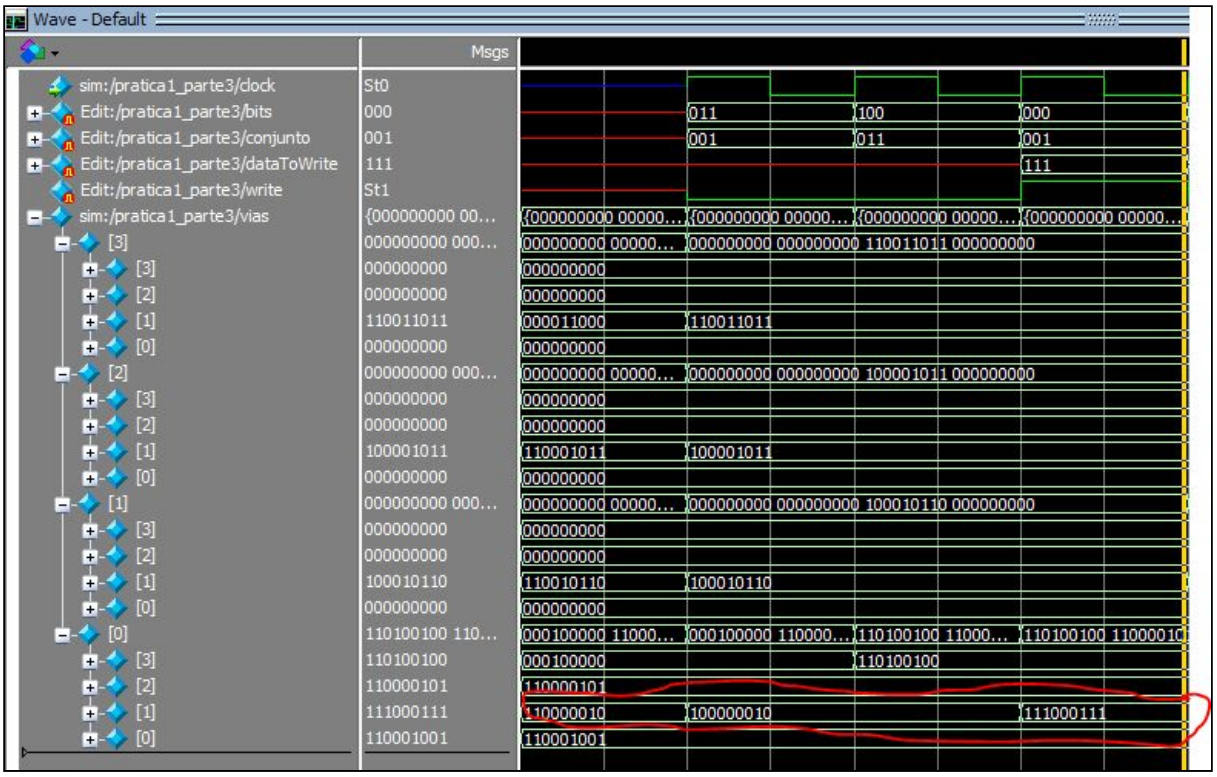
- Após a inicialização da memória, o primeiro caso de teste (Teste 1) é feito, obtendo o valor esperado para o elemento esperado (Via 3 - Conjunto 3):
  Tempo: 0.1ns ~ 0.2ns



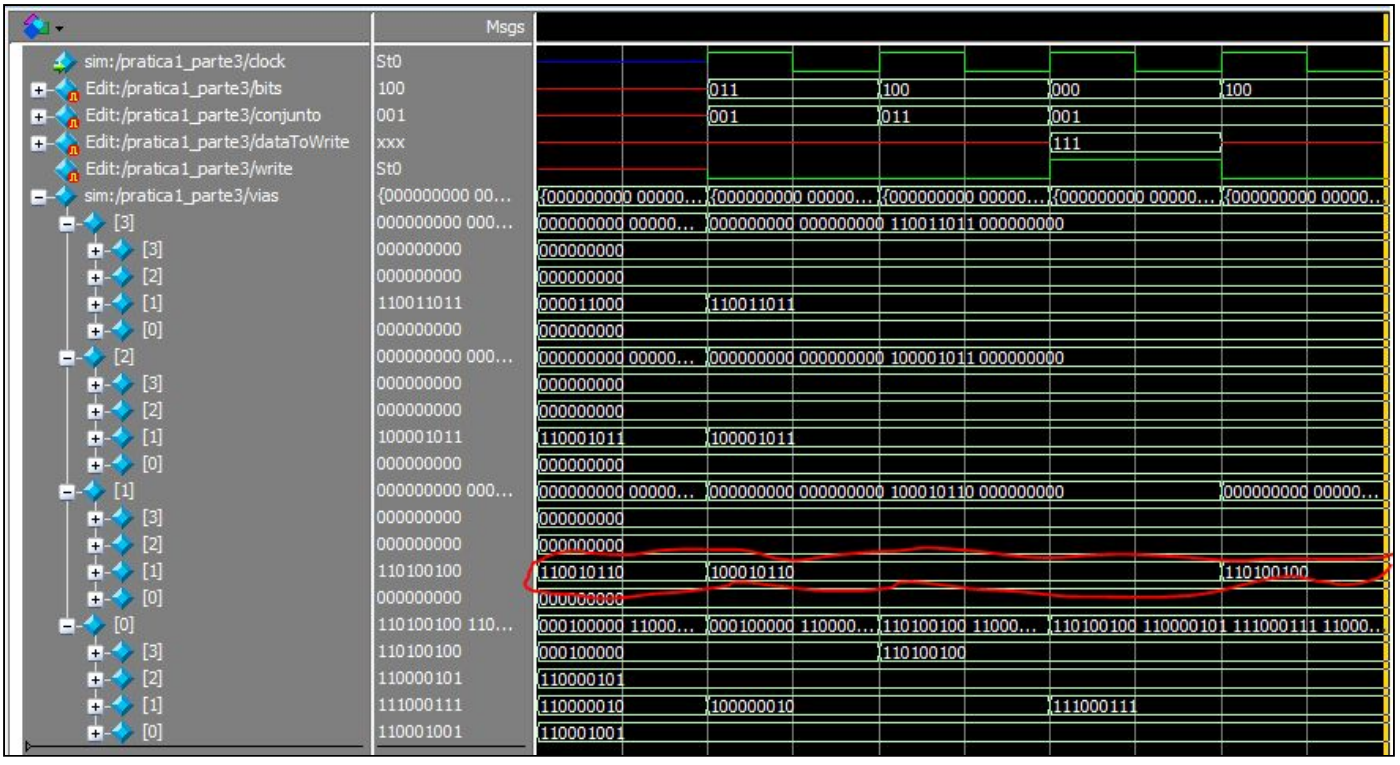
- Para o 2o caso de teste é atualizado a Via 0 Conjunto 3 para 110 100 100:
  Tempo: 0.2ns ~ 0.3ns



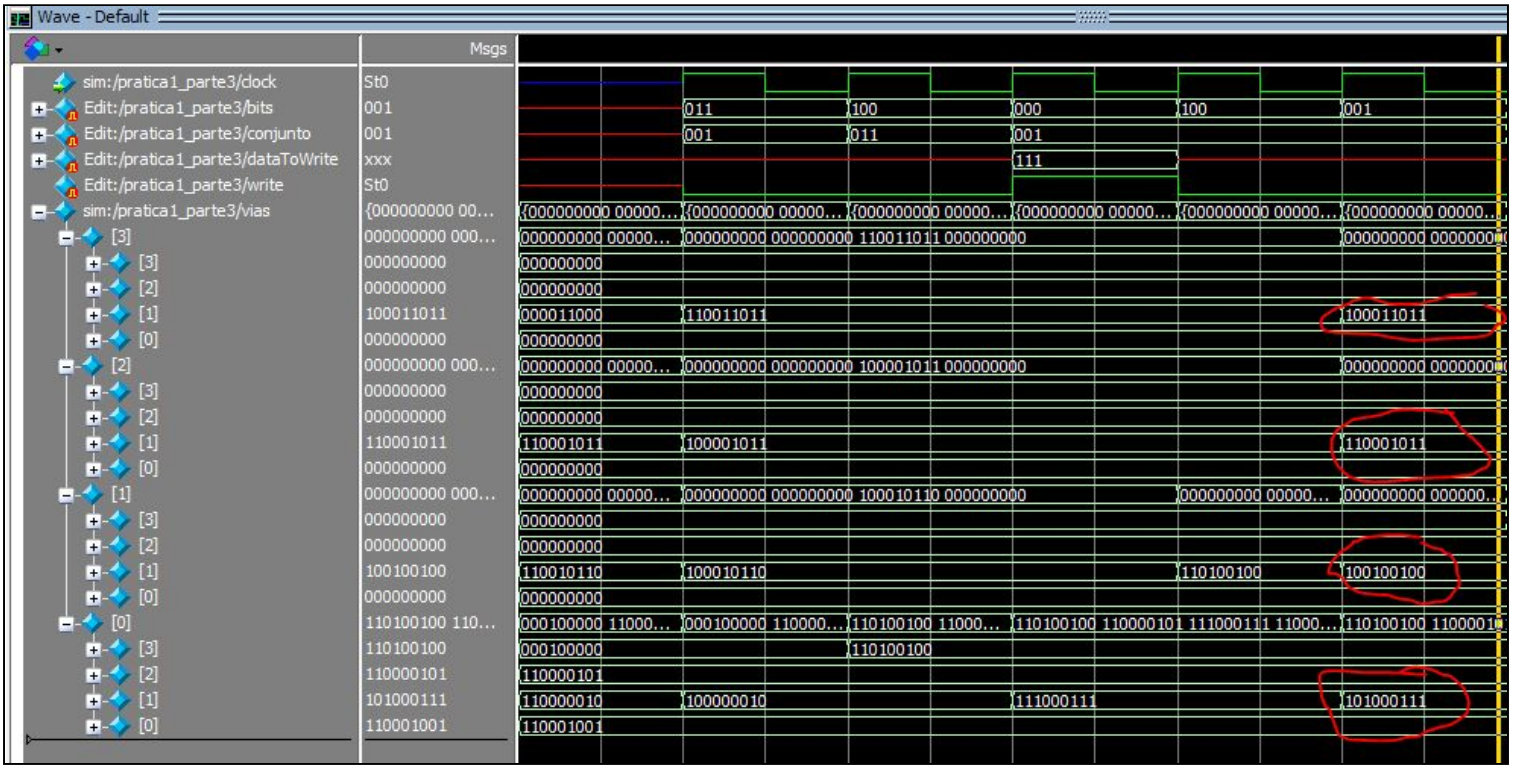
- Para o 3o caso de teste é atualizado a Via 0 Conjunto 1 para 111 000 111:
  Tempo: 0.3ns ~ 0.4ns



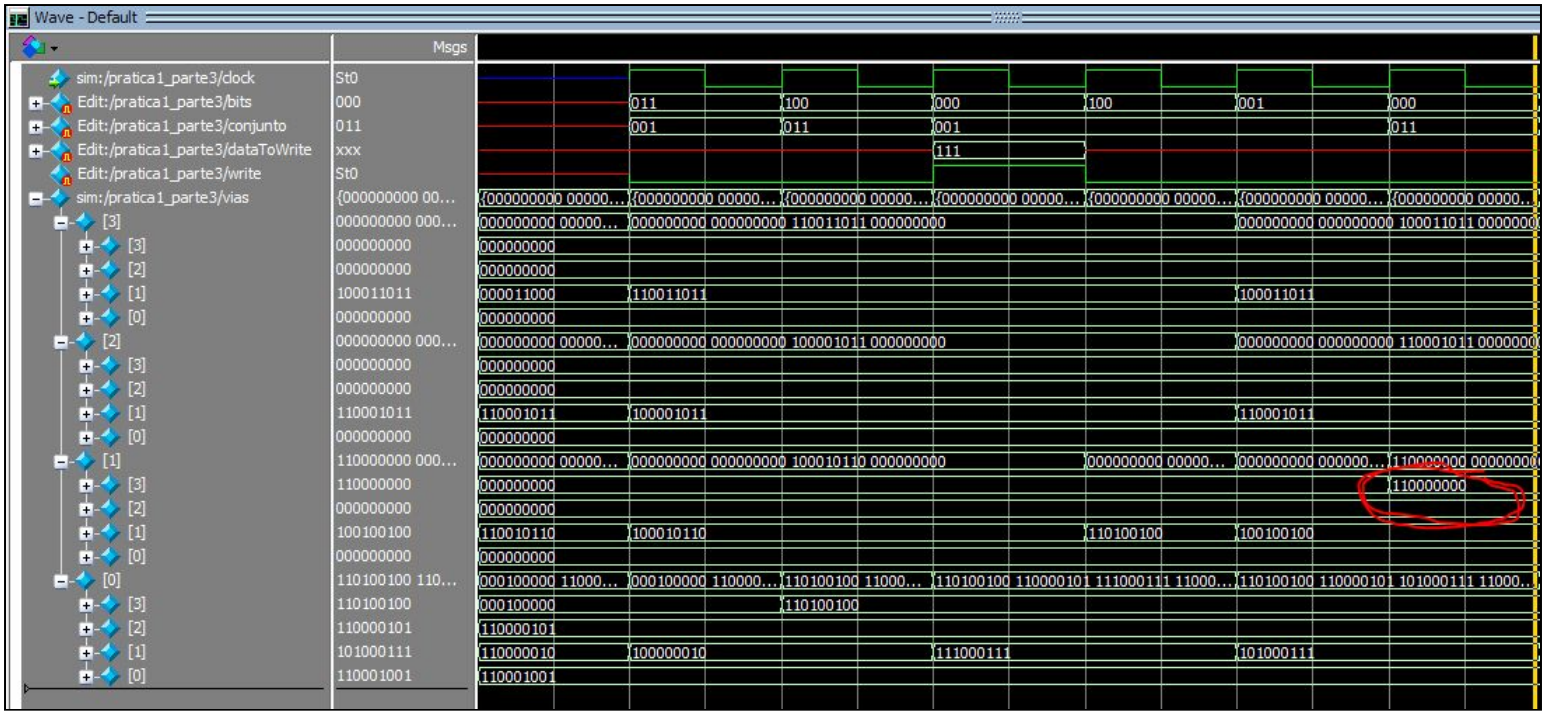
- Para o 4o caso de teste é atualizado a Via 1 Conjunto 1 para 110 100 100:
  Tempo: 0.4ns ~ 0.5ns



- Para o 5o caso de teste o bit de LRU dos elementos do conjunto 1 é atualizado:
  Tempo: 0.5ns ~ 0.6ns



- Para o 6o caso de teste é atualizado a Via 1 Conjunto 3 para 110 000 000:
  Tempo: 0.6ns ~ 0.7ns



- Para o 7o caso de teste é atualizado a Via 0 Conjunto 1 para 110 101 101:
  Tempo: 0.7ns ~ 0.8ns