UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

DISCIPLINA: Organização e Arquitetura de Computadores

DOCENTES: Monica Magalhães Pereira

GABRIEL VITOR DE ASSIS AZEVEDO

MARCEL LUIZ DE OLIVEIRA MENDONÇA

THAYRONE DAYVID DOS SANTOS

RELATÓRIO DE DESENVOLVIMENTO DE PROJETO

MECANISMO DE ASSOSSIATIVIDADE ADAPTATIVA

NATAL – RN

2014

GABRIEL VITOR DE ASSIS AZEVEDO

MARCEL LUIZ DE OLIVEIRA MENDONÇA

THAYRONE DAYVID DOS SANTOS

RELATÓRIO DE DESENVOLVIMENTO DE PROJETO MECANISMO DE ASSOSSIATIVIDADE ADAPTATIVA

Trabalho apresentado como requisito para obtenção de nota referente à terceira unidade da disciplina de Organização e Arquitetura de Computadores, do Bacharelado em Tecnologia da Informação, da Universidade do Rio Grande do Norte.

NATAL – RN

2014

RESUMO

Este relatório tem a finalidade de apresentar todo o processo de desenvolvimento de um projeto que simule determinados mecanismos de associatividade em uma memória cache. A linguagem utilizada para síntese e simulações foi a VHDL (VHSIC Hardware Description Language). Os métodos de associatividade definidos foram: *“Direct Mapping”, “4-way”, “8-way” e “Fully Associative”.* O objetivo do projeto é armazenar dados contidos em um banco de registradores, em um componente de memória menor -cache-, utilizando os 4 métodos descritos, alternando-os de acordo com uma variável representando a taxa de miss da cache.

Palavras-chave: Cache, VHDL, Associatividade, Adaptabilidade.

Sumário

INTRODUÇÃO

A composição do projeto de simulação de um *MAA* (mecanismo de associatividade adaptativa), além de aprofundar os conhecimentos na sintaxe de VHDL, permitiu pôr em prática os conhecimentos obtidos durante a disciplina referentes ao componente de memória Cache. Para o desenvolvimento, foi estudada toda sua estrutura, com o fim de manter o programa fiel a seu objetivo. O projeto foi modularizado, de modo que seus componentes sejam posteriormente reutilizados ou alterados.

A importância deste trabalho se dá no âmbito da pesquisa, uma vez que os processadores mais potentes costumam utilizar uma associatividade estática, definida durante o planejamento da arquitetura. A implementação de uma arquitetura cujo funcionamento se adapte à situação atual, pode acarretar em um aumento de sua eficiência, possivelmente superando o custo gerado pela complexidade de tal algoritmo. Com devidas pesquisas, à longo prazo, sua implementação pode se tornar viável, e até comum.

Neste relatório, é explanado como foi planejado o andamento do trabalho. Analisamos os problemas encontrados durante o desenvolvimento do código, ferramentas utilizadas, bem como as lógicas por trás de cada uma de suas seções. O objetivo principal é que fique claro como tudo foi concebido, e, consequentemente, permitir uma correção mais profunda por parte do(s) discente(s).

O PROJETO

Segue a descrição do projeto, conforme retratado no arquivo usado como base para o desenvolvimento:

“Implementar em hardware, um mecanismo que permite mudar a associatividade da memória de acordo com a taxa de miss. Exemplo: o sistema inicia a execução com uma memória com mapeamento direto. Se a taxa de miss aumentar, a associatividade é mudada para conjunto de 4 blocos, 8 blocos, ou para totalmente associativa.

Deve ter:

- *Memória*
- *Mecanismo para acesso à memória*
- *Mecanismo para mudança de associatividade*

Linguagem: VHDL”

Entende-se por associatividade o método utilizado para armazenar os dados em um componente de memória cache. Os métodos acima são discutidos posteriormente.

As ferramentas utilizadas foram o sintetizador de hardware Quartus, e a ferramenta de simulação ModelSim – Altera edition. Ambas pertencentes à Altera.

ANÁLISE DE OBJETIVOS

A fim de entender como seria trabalhado o projeto, e que dificuldades/eventualidades poderiam vir a ser encontradas, foi feito um planejamento quanto ao que seria necessário para que ele pudesse ser tido como bem-sucedido. A partir disso, pôde-se fazer um roteiro de produção, disposto em tópicos (em formato semelhante a uma “to-do-list”):

Objetivos primários:

- Definir entradas/saídas
- Modularizar o código
- Decidir tamanhos para a memória principal / Cache
- Criar um componente principal para associar os componentes de memória
- Criar uma FSM baseada no *clock* e taxa de *miss* no componente principal
- Garantir o funcionamento correto de cada associatividade
- Definir o método para a posição do dado dentro do bloco de memória
- Impedir que um componente consumisse mais que um ciclo de clock
- Gerar valores randômicos com um componente à parte
- Garantir o funcionamento de cada componente individualmente
- Testar todas as combinações de entradas e analisar o comportamento do programa
- Testar o fluxo de dados e garantir seu funcionamento

MÁQUINA DE ESTADOS FINITA

Conforme o planejamento, foi implementada uma máquina de estados, responsável por alterar todo o comportamento dos componentes relacionados à associatividade.

Primeiramente, o que define a mudança de estado é a entrada "*Miss Ratio".* Ela representa a taxa de miss da memória cache. Seu objetivo é "dizer" se os dados estão sendo encontrados no módulo de memória ou não. Em um processador real, essa taxa é calculada constantemente. Nesse projeto, essa função é abstraída, sendo uma entrada do programa, definida pelo usuário.

A variável acima possui 7 bits, podendo representar valores entre 0 e 128. Quando ela atinge aproximadamente 70% de seu valor máximo (~90_{10} ou 1011010_2), é forçada uma mudança de estado.

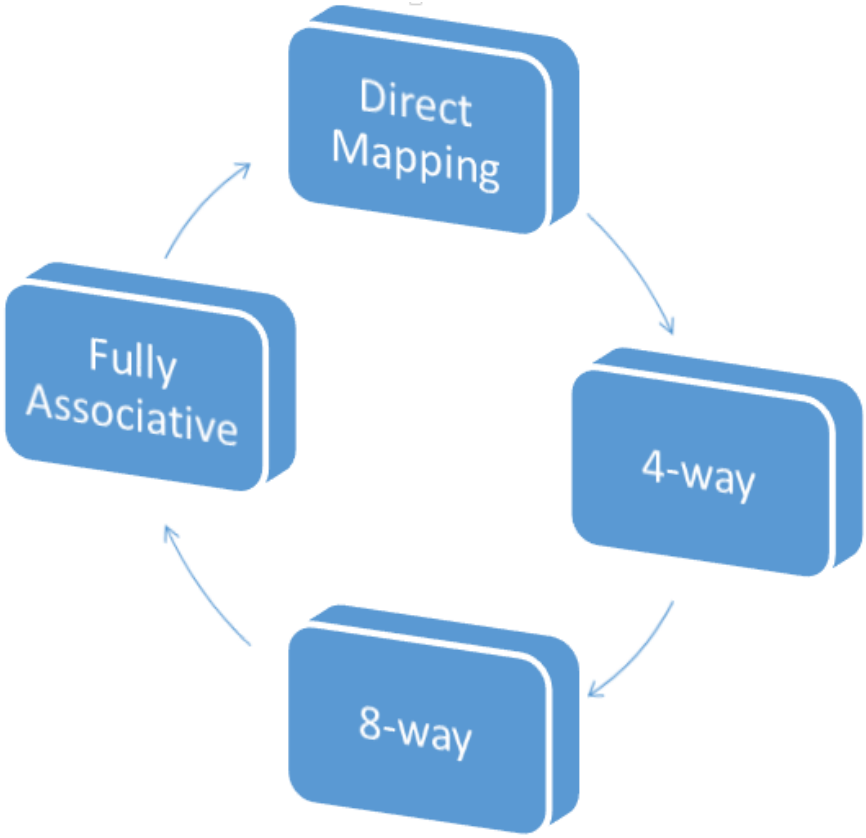


Figura 1: FSM

O estado inicial é o de mapeamento direto. Nele, a posição de um dado na memória cache é definida dividindo o endereço do registrador pelo tamanho do cache. O resto da divisão é a posição final do valor.

O estado seguinte, 4-way, funciona dividindo toda a memória em blocos de tamanho 4. O bloco é decidido dividindo o endereço do registrador pela quarta

parte do tamanho do cache (¼ de seu tamanho original). Uma vez realizada a operação, resta escolher em qual posição do bloco o valor irá. Em nossa arquitetura, essa posição é randômica.

O terceiro estado, 8-way, funciona de modo semelhante ao anterior. A diferença é que a cache é dividida em blocos de tamanho 8. Logo, a divisão é pela $\frac{1}{8}$ parte do tamanho original da cache.

O último estado, totalmente associativo, não possui as restrições anteriores quanto à posição/bloco. Qualquer registrador pode assumir qualquer posição do cache. Por essa razão, é utilizado um gerador de valores randômicos de 0 ao tamanho do módulo.

COMPONENTES

Os componentes do projeto são independentes, e realizam funções respectivas a seus nomes. Conforme os padrões de programação, há uma entidade Main, que contém todos os "*port maps"* para as entidades subsequentes, e realiza a interação entrada/saída. Segue a lista de cada componente, e suas funções:

1. **Main.vhd**
   - Controlador da arquitetura
   - Possui e controla a FSM (Máquina de Estados Finita) responsável por definir a associatividade atual.
   - Recebe o número do registrador cujo dado está sendo requisitado
   - Recebe a taxa de miss e trata a FSM de acordo.
   - Recebe e reproduz o "*clock*" a seus componentes

2. **Registers.vhd**
   - O componente é, na realidade, um banco de registradores, que recebe e envia dados conforme requisitados pela entidade principal.
   - Doravante denominada memória principal, possui 32 registradores de 32 bits.

3. **Cache.vhd**
   - Serve como a memória cache.
   - Possui 16 registradores de 32 bits
   - Tem funções semelhantes à do banco de registradores, mas seu comportamento ao lidar com os dados varia de acordo com o estado de associatividade.

4. **Operator.vhd**
   - A entidade mais importante para o MAA.
   - Realiza as operações responsáveis por definir, para um dado registrador, a posição/bloco da cache ao qual seu dado será posteriormente encaminhado.
   - Sensível ao estado atual.

5. **Randomizer.vhd**

   - Responsável por gerar valores randômicos.

   - Utilizado devido a problemas com o método responsável por essa função da biblioteca da Altera.

   PS: Implementação de autoria externa (Explicitada na seção final, de *copyright*).

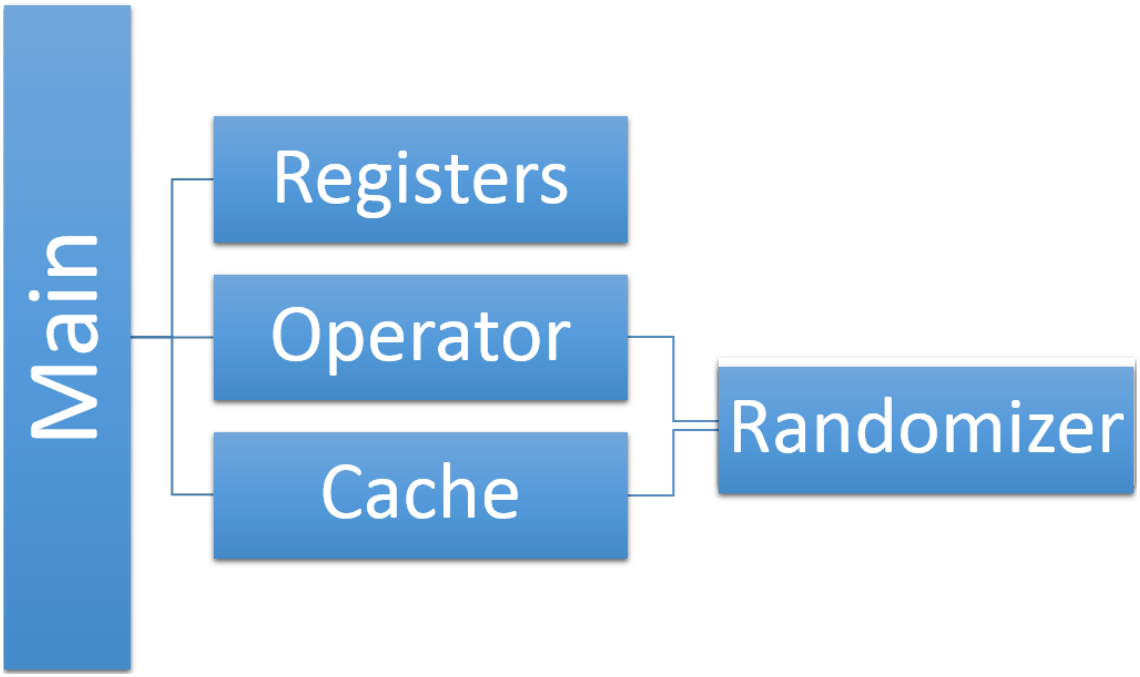


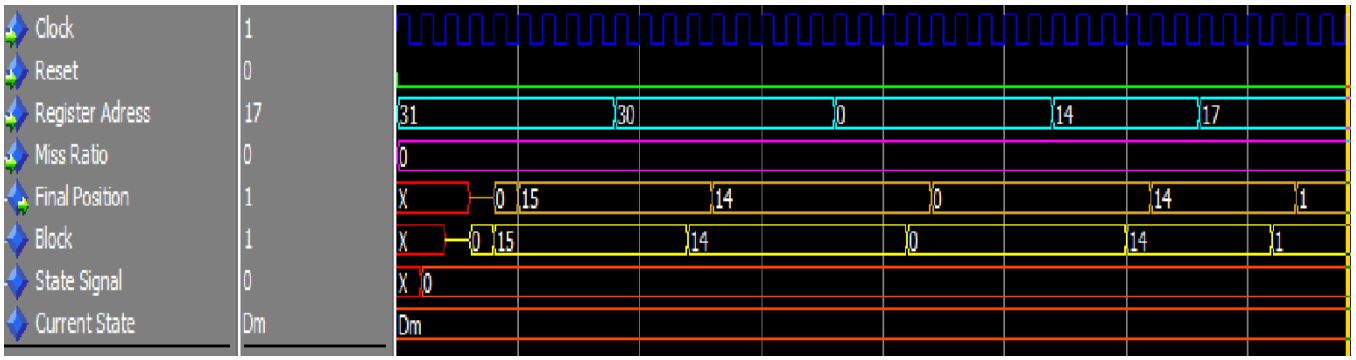
Figura 2: Fluxo de dados

SIMULAÇÕES DOS ESTADOS

(*A qualidade das imagens no Word não ficou boa. Por essa razão, estão sendo anexadas na pasta "Simulações").*

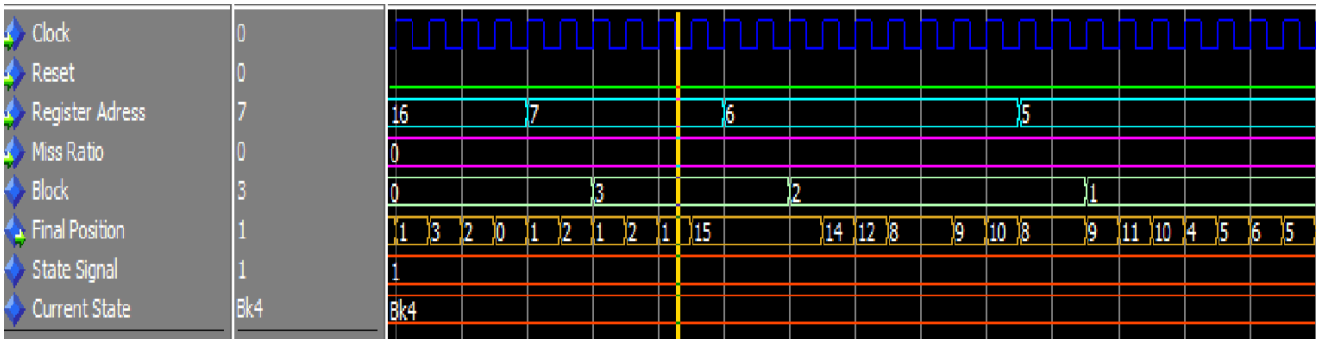
(A seção "Interpretação" abaixo é redundante caso já tenha sido lida a explicação na subdivisão que explica o funcionamento da FSM).

Mapeamento Direto:



Interpretação: O dado do registrador de endereço x é requisitado, e o operator retorna a posição y, obtida pela operação y = x mod CacheSize. Nesse caso,mod16.
Como no mapeamento direto, os blocos são de tamanho 1, a posição final e o bloco selecionado são iguais.

4-way (blocos de tamanho 4):



Interpretação: O dado do registrador de endereço x é requisitado, e o operator retorna o BLOCO y, obtida pela operação y = x mod CacheSize/4. Nesse caso, mod 4.

Em seguida, um valor randômico é selecionado no bloco. A relação das posições de cache contidas em cada bloco segue:

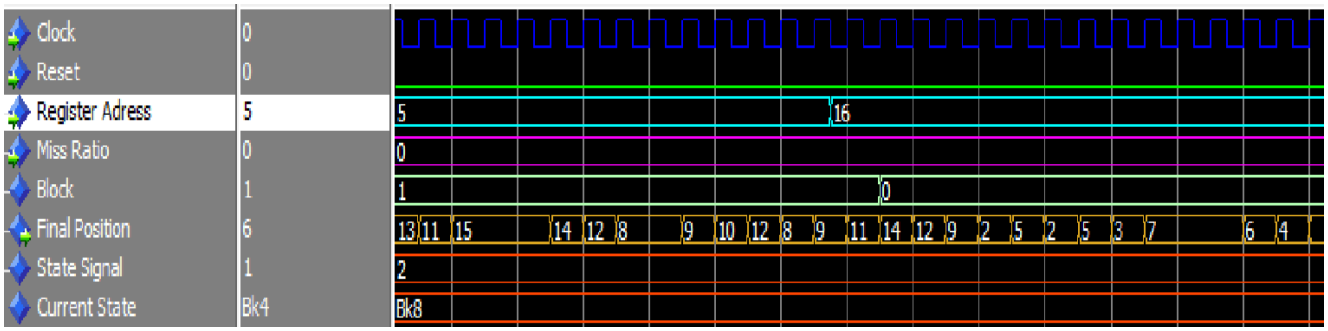
Bloco 0: 0,1,2,3

Bloco 1: 4,5,6,7

Bloco 2: 8,9,10,11

Bloco 3: 12,13,14,15

8-way (blocos de tamanho 8):



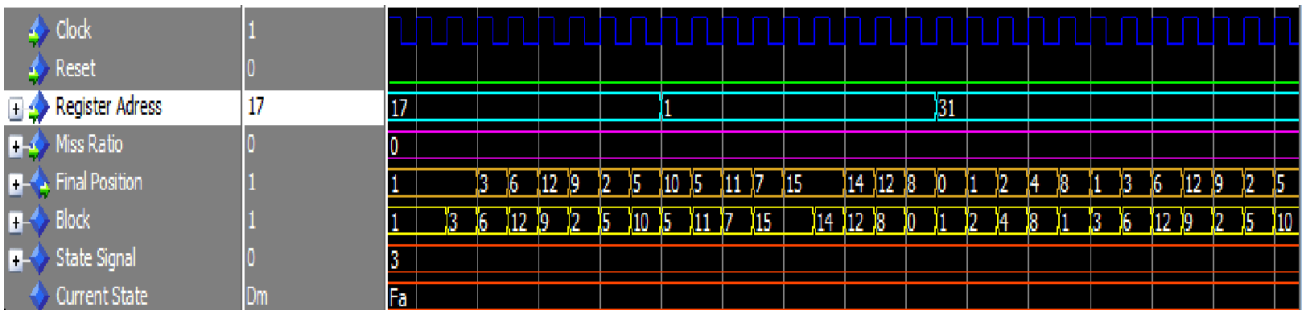
Interpretação: O dado do registrador de endereço x é requisitado, e o operator retorna o BLOCO y, obtida pela operação y = x mod CacheSize / 8. Nesse caso, mod 2.

Em seguida, um valor randômico é selecionado no bloco. A relação das posições de cache contidas em cada bloco segue:

Bloco 0: 0,1,2,3,4,5,6,7

Bloco 1: 8,9,10,11,12,13,14,15

Totalmente associativa:



Interpretação: O dado do registrador de endereço x é requisitado, e o operator retorna um valor randômico de 0 a 15, permitindo que qualquer registrador assuma qualquer posição da cache.

Nota: O atraso entrada/saída é consequência das operações realizadas em todos os componentes antes da chegada à cache. Como eles são sensíveis ao *RisingEdge* do clock, São executados na ordem: Registers -> Operator -> Cache, e a saída de um é a entrada do seguinte, logo, a saída final chega ao main após 3 clocks.

A única solução não testada foi a de uma variável que se alinha à saída, para mostrar qual entrada ela representa. Um estado de espera, e o uso de flags não solucionaram o problema da simulação.

Nota 2: Há mais saídas no projeto final, indicando qual o dado salvo na posição final, por exemplo. Mas são dados de importância menor, e que poluíam a imagem das simulações.

DIFICULDADES ENCONTRADAS

- Gerar valores realmente randômicos sem o uso da biblioteca Altera
- Alinhar as E/S
- Utilizar as funções originalmente criadas para variáveis do tipo Integer, em signals do tipo std_logic_vector.

ORGANIZAÇÃO DO CÓDIGO

Anexo a esse relatório, há uma pasta chamada “ProjetoOAC 2.0”. Ela contém a pasta Simulações, os componentes em. vhd avulsos, e pastas com o nome de cada componente, que possuem arquivos de projeto de cada componente individualmente.

A pasta principal é a Final Project, que inclui o arquivo de projeto Quartus. Os módulos chamados por esse projeto são os das pastas citadas no parágrafo acima, garantindo que qualquer alteração individual seja reproduzida ao projeto final.

Quaisquer outras pastas são geradas pelo Quartus, e não são necessariamente importantes à execução do programa.

GUIA DE COMPILAÇÃO/SIMULAÇÃO

Para compilar o projeto, é necessário abrir o arquivo de projeto situado na pasta Final Project, e apertar o botão “*Play”* ou o *“Synthesis”* logo ao lado. O projeto estará compilado, e pode ser pinado para funcionar em qualquer FPGA Altera Cyclone IV.

Para realizar a simulação, podem ser utilizados os arquivos de simulação já incluídos, e apenas executá-los. Para criar uma nova simulação, é de critério livre a escolha do simulador. Para realizar a simulação no Quartus, é necessário criar um arquivo *.vwf* através do comando *Ctrl+N*, setar as entradas, e executar o arquivo.

Para realizar a simulação no ModelSim-Altera, é necessário criar um projeto, e incluir nele, todos os arquivos .vhd associados ao projeto do Quartus. Uma vez feito isso, é feita a compilação, e a simulação pode ser realizada alterando-se os valores das entradas.

REFERÊNCIAS

- **Estudo de memória cache**
  http://en.wikipedia.org/wiki/Cache_(computing)

- **Associatividade**
  http://www.pcguide.com/ref/mbsys/cache/funcComparison-c.html

- **Referências de sintaxe:**
  http://www.quicknet.se/hdc/hdl/educaton/mux4_1/

  https://www.doulos.com/knowhow/vhdl_designers_guide/components_and_port_maps/

DADOS DE COPYRIGHT