MIPSFPGA - Um Simulador MIPS Incremental com Validação em FPGA

Jeronimo Costa Penha, Geraldo Fontes, Ricardo Ferreira
Universidade Federal de Viçosa, Viçosa - MG CEP 36570-900
Departamento de Infomática
jeronimopenha@gmail.com, gsfj03@gmail.com, ricardo@ufv.br

Abstract

Este artigo apresenta inovações no ensino da arquitetura do processador MIPS com apoio de um simulador gráfico. O ambiente proposto, denominado MIPSFPGA (Mips Incremental Processor Simulator and Fpga Prototyping), além da interface gráfica para visualização do caminho de dados das diversas implementações do MIPS com e sem pipeline, o MIPSFPGA inclui vários recursos adicionais. Primeiro, o projeto pode ser editado graficamente para explorar outras implementações. Todos os projetos do livro Organização de Computadores de Patterson e Hennessy são disponibilizados. O projeto pode ser exportado e prototipado em FPGA. As implementações podem ser depuradas durante a simulação e a prototipação. Finalmente, a metodologia é incremental, permitindo começar com uma simples apresentação dos exemplos até a personalização e derivação de novas implementações e extensões.

1. Introdução

O MIPS é um processador que representa um marco de inovação na introdução da arquitetura RISC e na consolidação de ideias simples e regulares: poucas instruções, decodificação direta, registradores gerais, pipeline, load/store como unidade de execução. O livro "*Computer Organization and Design*" [13] de David A. Patterson e John L. Hennessy é uma das referências mais usadas e está na lista dos 10 mais vendidos da Amazon na área de *Hardware*. Apesar de vários simuladores já terem sido propostos para apoio ao ensino com o MIPS [3], este trabalho apresenta uma nova abordagem para uma ferramenta de simulação e apoio ao ensino de arquiteturas RISC com foco no processador MIPS. A referência básica são os exemplos do livro de Patterson&Hennessy [13].

Dentre os diversos simuladores propostos para a arquitetura MIPS, pode-se destacar o SPIM [11] e o MARS [15]. O SPIM é um simulador para o conjunto de instruções MIPS32. É capaz de executar programas escritos em *assembly*, mostrar o conteúdo do banco de registradores e da memória. O SPIM é multiplataforma (Windows, Linux Mac-OS). Apesar de ser indicado pelo livro de Patterson&Hennessy [13] como material de apoio, este simulador não detalha a implementação do processador e não mostra o caminho de dados do processador [11]. O simulador MARS é o simulador MIPS mais popular, desenvolvido em Java, semelhante ao SPIM, o MARS trabalha no nível *assembly*. É bem documentado e pode ser estendido, como por exemplo uma versão recente apresentada [1] para a simulação do MIPS Vetorial. Assim como o simulador SPIM, o MARS não detalha a implementação [15].

Para compreender os detalhes de implementação do caminho de dados do MIPS, vários simuladores foram propostos como por exemplo o ViSiMIPS [10], RAVI [12], WEBMIPS [2] e DIMIPSS [5]. Estes simuladores mostram com uma interface gráfica simples e didática a visualização da execução de um trecho de código MIPS. O usuário pode acompanhar como os encaminhamentos foram executados, visualizar com cores as instruções passo a passo no pipeline. Porém nenhum deles permite que o estudante modifique a implementação como por exemplo adicionando o tratamento de uma nova instrução ou modificando a implementação com a inclusão de multiplexadores e registradores.

A abordagem proposta aqui também é baseada em um simulador com interface gráfica para visualização do caminho de dados. Além disso, nossa proposta segue a sequência didática do livro de Patterson&Hennessy [13], na qual os autores propõem o ensino MIPS em partes e de forma incremental. O ambiente proposto é o MIPSFPGA - *Mips Incremental Processor Simulator and Fpga Prototyping* - que é baseado no simulador Hades [9] como camada básica. Todas as implementações do livro estão disponibilizadas e o estudante pode fazer alterações no projeto no nível RTL (*Register Transpfer Level*). O MIPSFPGA contribui com novas possibilidades: 1) alteração do projeto para o desenvolvimento de novas versões ou a resolução de exercícios propostos pelo livro; 2) exportar o projeto para implementação física em FPGA; 3) Depuração do

conteúdo dos registradores e memória na simulação e na prototipação; 4) Desenvolvimento de novos componentes ampliando a biblioteca do simulador.

2. Simulador Incremental

O ambiente MIPSFPGA proposto aqui é composto por um conjunto de projetos e componentes que executam no simulador Hades. O Hades é implementado em Java para simulação de circuitos digitais em diversos níveis e foi selecionado pela sua portablidade, flexibilidade para a criação de novos componentes, extensão, múltiplas finalidades, além de possuir uma ampla biblioteca de componentes RTL [9].

2.1. HADES

Proposto em 1998 por Norman Hendrich, o simulador Hades [9] é um simulador de eventos com interface gráfica. Sua finalidade inicial é a simulação de circuitos digitais em vários níveis (portas, RTL, processadores, etc.). Todos os componentes e conexões (fios) são objetos e graças a uma definição simples, o ambiente é bem genérico e eficiente, podendo ser utilizado para outras áreas como processamento de imagens, computação paralela, sistemas embarcados, etc [8]. O Hades é um simulador dinâmico no qual a edição do projeto pode ser feita mesmo com circuito em modo de simulação. A biblioteca de componentes do Hades possui mais de 500 componentes. Além disso, o usuário pode desenvolver seus próprios componentes, fazendo reuso dos componentes da biblioteca. O desenvolvimento das extensões tem uma rápida curva de aprendizado. Várias extensões já foram propostas para o Hades como por exemplo, a implementação de fluxo de dados [7, 6].

Para a simulação e o ensino do MIPS, existe também o simulador RaVi [12] que é baseado no Hades. Porém o RaVi é voltado para a visualização, tem sua implementação encapsulada e não está disponível para extensões. Nossa proposta é criar projetos e componentes de forma colaborativa permitindo o reuso e extensões. As abordagens [14, 8] também são baseadas no ensino de conceitos onde os estudantes são estimulados a desenvolver seus próprios componentes.

2.2. Sequência de Implementações MIPS

O MIPSFPGA é formado por um conjunto de exemplos de projetos de MIPS seguindo a sequência do livro Patterson e Hennessy [13]. O livro apresenta os projetos de forma incremental. Para auxiliar o ensino, os projetos do MIPSFPGA são numerados seguindo a sequência de figuras do livro. Foram desenvolvidos novos componentes no Hades para que o projeto no MIPSFPGA fosse visualmente

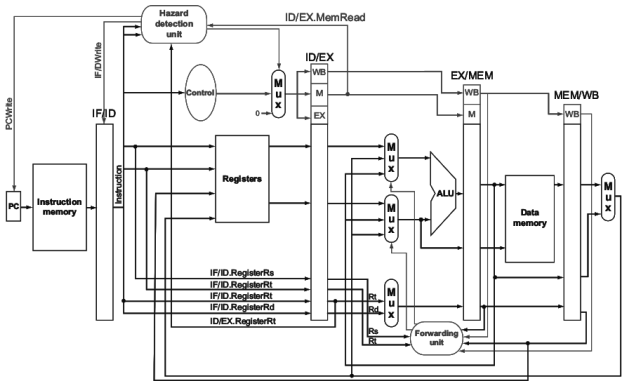

Figura 1. MIPS Pipeline Figura 4.60 Original do Livro [13]

o mais próximo da ilustração original do livro. Foram implementados os caminhos de dados do MIPS para 4^a e 5^a edições do livro texto [13].

Ao carregar o projeto no MIPSFPGA, o estudante tem acesso a uma figura interativa do livro. A cada passo da simulação (ciclo de clock), os fios e barramentos podem ter seus valores monitorados. A Figura 1 foi extraída do livro e ilustra o processador MIPS *Pipeline* com a *Forward Unit* e a *Hazard Unit*. A Figura 2 apresenta a tela do simulador para a mesma implementação, na qual pode-se observar a forte semelhança. Diferentemente dos outros simuladores ViSiMIPS [10], RAVI [12], WEBMIPS [3], DIMIPSS [5], a figura é dinâmica e pode ser editada. Assim como nos outros simuladores é possível ver o conteúdo dos registradores, memória, avançar ciclo à ciclo, ver detalhes do encaminhamento etc. Mas somente o MIPSFPGA permite ao estudante editar graficamente o projeto e derivar uma nova implementação com mais instruções. Algumas figuras do livro apresentam simplificações, nos projetos do MIPSFPGA pode-se também esconder alguns fios ou barramentos. A metodologia adotada foi colocar explicitamente todos os barramentos. Apenas alguns sinais como *clock*, reset e habilitações não são exibidos para simplificar a visualização.

3. Exercícios

Com o reuso do componente para questões múltipla escolha criado em [8], vários exercícios do livro foram remodelados para adquirirem o formato de múltipla escolha e estão disponibilizados com o MIPSFPGA. Novos exercícios foram propostos também. Este componente permite que os professores e/ou estudantes criem exercícios de múltipla escolha em um arquivo de texto simples e que estes exercícios sejam resolvidos pelo aluno durante a simulação através do componente. As questões são visualizadas dentro do simu-

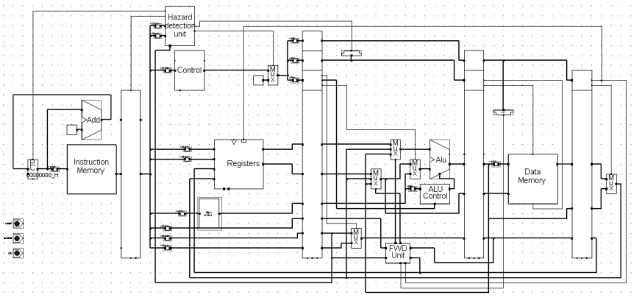

Figura 2. Projeto da Figura 4.60 do MIPS na interface do MIPSFPGA

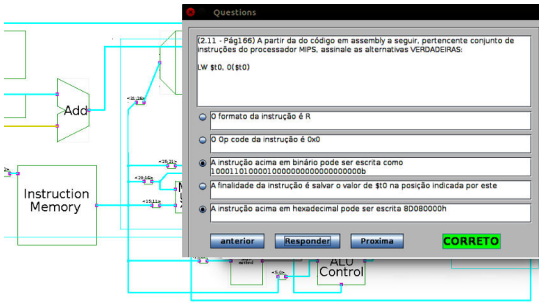

Figura 3. Componente para questões múltipla escolha.

lador na mesma janela. O aluno tem questões para refletir, solucionar e pode, ao mesmo tempo, simular o circuito para conferência das respostas.

O formato para criar questões é bem simples com marcadores. Cada enunciado de questão é incluído entre "<q>" e "</q>". Após estes marcadores, são inseridas alternativas de respostas com os marcadores "<a1>" e "</a1>" para alternativa 1, "<a2>" e "</a2>" para alternativa 2, etc. Para cada alternativa, seguem as respostas. As questões são carregadas uma a uma. O estudante pode navegar para questão anterior ou posterior e verificar se respondeu corretamente. Por se tratar de um componente desenvolvido em linguagem Java, novos componentes com outros recursos para elaboração e visualização de exercícios podem ser futuramente incluídos. As questões seguem um estilo "verdadeiro/falso". Pode haver mais de uma alternativa correta por exercício. Isto permite mostrar que podem existir várias soluções equivalentes para um mesmo exercício.

A Figura 3 mostra o simulador com o componente de questões acionado com uma questão a ser respondida e a Figura 4 exibe um exemplo de arquivo fonte no qual estão descritas as questões.

```
<q>(2.11 - Pág166) A partir da do código em assembly a seguir, pertencente conjunto de
instruções do processador MIPS, assinale as alternativas VERDADEIRAS:

LW $t0, 0($t0)</q>
<a1>O formato da instrução é R f</a1>
<a2>O Op code da instrução é 0x0 f</a2>
<a3>A instrução acima em binário pode ser escrita como
10001101000010000000000000000000b v</a3>
<a4>A finalidade da instrução é salvar o valor de $t0 na posição indicada por este f</a4>
<a5>A instrução acima em hexadecimal pode ser escrita 8D080000h v</a5>

<q>(2.11 - Pág166) Após a execução do algoritmo abaixo, assinale a(s) alternativa(s)
verdadeira(s):
                        ADDI R1,R4,4
                        ADD R2,R4, R0
                        SW R2, 0(R1)
                        LW R1, 0(R1)
                        ADD R3, R2, R1
</q>
<a1>A segunda instrução em hexadecimal é 801020h v</a1>
<a2>O valor final da posição 8 da memória RAM será de 4h f</a2>
<a3>O resgistrador R0 contém 10h no instante inicial f</a3>
<a4>Ao fim da execução, o registrador R3 terá o valor 8h armazenado v</a4>
<a5>A soma de R4 com o valor 4 faz com que a instrução LW leia a posição 8 da memória. f</a5>
```

Figura 4. Exemplo do arquivo de questões múltipla escolha.

4. Desenvolvimento de Novas Implementações

Vários recursos do ambiente utilizado podem ser explorados para a criação de novas implementações do circuito do MIPS. Como já mencionado, o MIPSFPGA oferece uma biblioteca adicional com componentes desenvolvidos para esta função além da ampla biblioteca de componentes do simulador Hades. Com uma rápida curva de aprendizado, é fácil criar novos componentes com reuso dos componentes já desenvolvidos.

4.1. Ampliando o ISA

O livro texto implementa apenas um subconjunto das instruções MIPS para fins didáticos. A ampliação do ISA (*Instruction Set Architecture*) com inclusão de novas instruções é tema de vários exercícios. Ao estender um projeto, uma ou mais partes podem ser alteradas. Como o MIPSFPGA é um editor/simulador, novos fios e componentes podem ser adicionados através da interface gráfica. Suponha que novos sinais são adicionados ao estágio de busca (*Fetch*) em uma implementação com pipeline como a ilustrada na Figura 2. Neste caso, novos sinais devem ser propagados pela barreira. A implementação da barreira é feita como um conjunto de registradores. Muitas vezes, os estudantes não sabem o que tem na caixa preta da barreira. Ao editar a barreira e acrescentar uma nova funcionalidade, o estudante passa a compreender melhor a implementação com estágios pipeline e como estes sinais são propagados e armazenados.

Nos projetos do MIPSFPGA, as barreiras foram implementadas como sub-projetos hierárquicos e podem ser facilmente editadas e estendidas. Ao editar a barreira, a parte interna é visualizada como ilustrado na Figura 5 para a barreira IF_ID (entre os estágios *Fetch* e *Decode*).

A primeira implementação do livro é o *Single* MIPS (sem *pipeline*). Neste projeto, a unidade de controle é

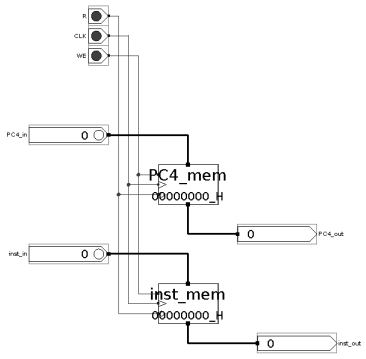

Figura 5. Detalhamento da barreira pipeline IF_ID da Figura 2 com a propagação dos registros IR e PC.

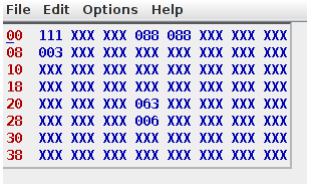

Figura 6. Edição da unidade de controle para o processador Single Mips.

```
...
//opcode 35 = LW
 case LWopcode:
    vector_Signals = new StdLogicVector(8,23); // Hex 0x23 = 0 00 1 0111
// Regdst RT=0, OP=00 +, ALUsrc = IM = 1, MW=0, MR=1, RegW=1, MemtoReg=1
    value_Branch = new StdLogic1164(False); // bit signal
    value_IFflush = new StdLogic1164(False);  // bit signal
    break;
//opcode 8 = ADDI
 case opcodeADDI:
   vector_Signals = new StdLogicVector(8,18); // Hex 0x18 = 0 00 1 0001
 // Regdst RT=0, OP=00 +, ALUsrc = IM = 1, MW=0, MR=0, RegW=1, MemtoReg=0
   value_Branch = new StdLogic1164(False);
   value_IFflush = new StdLogic1164(False);
   break;
....
```

Figura 7. Código parcial da Unidade de Controle em Java para as versões *Pipeline*

uma tabela com o mapeamento do código da instrução (*opcode*) para os sinais internos de controle do MIPS. A implementação do MIPSFPGA é baseada na tabela como um módulo de memória. Os módulos de memória do Hades já possuem interface gráfica com edição, leitura e escrita em arquivo,. Portanto, o projeto e suas alterações são simplificados. Suponha que deseja-se incluir uma nova instrução que não necessita modificar o caminho de dados com sinais extras. Neste caso, basta editar o componente memória ROM e programar os sinais de controle para o *opcode* desejado. A Figura 6 mostra a janela de edição da Unidade de Controle do processador *Single* Mips. Observe que apenas 6 *opcodes* estão programados para as instruções descritas no livro texto. Por exemplo, o *opcode* da instrução *load* é 35 (decimal) ou 23 (hexadecimal) e foi programando para gerar os sinais 063 (hexadecimal) ou 00 0110 0011 em binário para controle dos recursos internos (Alu, multiplexadores, banco de registros, memória). Na Figura 6, XX significa indefinido. É importante destacar que o MIPSFPGA não foi proposto para programar em *assembly* como os simuladores MARS [15] e SPIM [11].

Os componentes unidade de controle e memória de instruções permitem a edição em binário e hexadecimal. Portanto os estudantes são estimulados ao exercício de codificar que é um passo fundamental para se projetar processadores. Para verificação do funcionamento, os valores dos sinais internos, registradores e memória de dados são facilmente visualizados com interface gráfica.

Se não existir o componente pronto na biblioteca, pode-se derivar um novo componente de algum outro já existente. Os componentes são implementados em Java. Por exemplo, a unidade de controle das implementações pipeline tem seu comportamento descrito em Java. A Figura 7 ilustra uma parte do código da unidade de controle com destaque para a decodificação dos *opcodes* das instruções de *Load* e Addi. Pode-se observar que a programação está em alto nível com o mapeamento do *opcode* nos sinais de controle. Mesmo para estudantes que não estão habituados a Java, pequenas alterações são possíveis de ser realizadas para inclusão ou exclusão de instruções.

O componente pode ser recompilado e/ou um novo componente pode ser criado. O componente tem associado um arquivo "símbolo" com sua representação gráfica que será visualizada no editor. Pode ser uma simples caixa com os nomes dos sinais, ou então algo mais elaborado. O arquivo que contém o símbolo também é descrito com componentes básicos (linhas, formas, I/O) e sua descrição é simples e direta. A representação pode também ser gráfica com animações, como por exemplo, a máquina de estados que mostra uma animação durante a simulação.

4.2. Novas Funcionalidades

Similar aos demais simuladores de caminho de dados que mostram o código com mnemônicos em *assembly* (*load*, *store*,etc.). O MIPSFPGA inclui componentes de visualização que servem para facilitar a compreensão. No caso específico do *pipeline*, a visualização passo a passo da propagação das instruções através dos estágios está presente em muitos simuladores MIPS com interface gráfica. Através do tipo *String*, foram criados alguns componentes que decodificam os sinais binários do caminho de dados e exibem de forma legível com mnemônicos, por exemplo

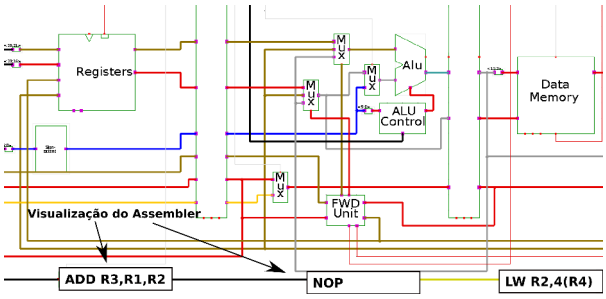

Figura 8. Exibição do deslocamento das instruções no MIPS *Pipeline*.

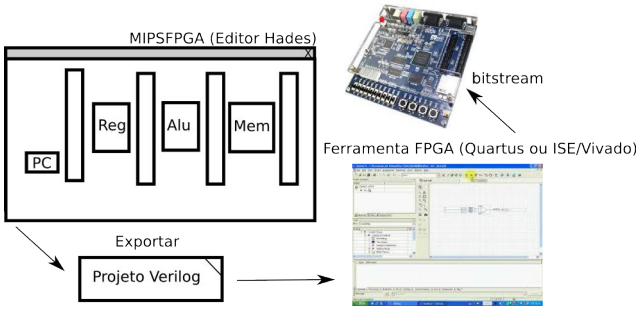

Figura 9. Fluxo para exportação da implementação para prototipação em FPGA

"add r1,r2,r3"ao invés de $0000\ldots10$. A Figura 8 mostra os componentes que exibem a instrução executada em cada *pipeline* no processador Mips *Pipeline* com *Forward Unit* e *Hazard Detection Unit*.

5. Síntese em Kits FPGA

Os programas de simulação são ferramentas importantes para auxílio no aprendizado, porém a implementação física do projeto com a prototipação em FPGA, permite ao estudante ter uma comprovação do funcionamento. Existem diversos Kits educacionais para ensino de circuitos digitais com FPGA e acessórios (chaves, leds, *displays*, etc). Entretanto, o ensino e uso de linguagem de descrição de *hardware*, como VHDL e Verilog, é mais complexo para a maior parte da comunidade de programadores que estão habituados aos paradigmas de linguagem de programação imperativas como C, C++, Java. Principalmente estudantes de Ciência da Computação ou Sistemas de Informação.

Além disso, os ambientes de projeto com FPGA são ferramentas complexas com muitas janelas e menus, como o Quartus da Altera ou o ISE/Vivado da Xilinx. O nosso objetivo é simplificar o projeto para ampliar a divulgação inicial. A proposta do MIPSFPGA é exportar de forma transparente a implementação do projeto com a geração automática do código em Verilog. A princípio, o estudante pode sintetizar um MIPS sem ter ainda conhecimento de Verilog ou VHDL. O projeto é então carregado na ferramenta de síntese FPGA que será usada apenas para carregar o *bitstream* no FPGA como ilustrado na Figura 9.

Como o MIPSFPGA segue uma metodologia incremental, permite aos estudantes e professores acrescentar novas extensões. Para desenvolver um novo projeto que necessite de novos componentes que não estejam disponíveis nas bibliotecas do Hades e MIPSFPGA, primeiro é necessário criar o componente. O mais simples é fazer o reuso ou seguir o padrão de desenvolvimento descrito no manual do Hades [9]. O segundo passo é testar o projeto no ambiente de simulação. Como já mencionado, o componente é descrito em Java, com a qual implementa-se a descrição comportamental deste através da orientação a eventos. Ou seja, toda vez que os sinais de entrada são alterados, o componente é chamado para tratar a entrada e escalonar as saídas correspondentes de forma reativa. O terceiro passo é a criação de um arquivo em Verilog que implemente a funcionalidade e interface do componente. Finalmente, o gerador do MIPSFPGA pode ser chamado para gerar os arquivos verilog do novo projeto. O projeto é então carregado em uma ferramenta de FPGA que fará a síntese e a geração do *bitstream* para prototipação em FPGA. Trabalhos futuros irão incluir a possibilidade de desenvolver o componente em Verilog e importar na simulação do Hades.

5.1. Entrada e Saída para Depuração

Para simplificar a verificação da implementação no FPGA, já que o subconjunto de instruções apresentado no livro é bem reduzido, o MIPSFPGA propõe o uso de componentes de visualização durante a simulação e também na prototipação que podem ser encontrados na maioria dos kits.

A Figura 10 mostra o diagrama de uma implementação na qual pode-se ver o processo de simulação e prototipação. Pode-se observar três tipos de componentes. Primeiro, o caminho de dados e controle que é o projeto do processador. No simulador é descrito com os componentes da biblioteca, os fios e barramentos podem ser visualizados passo a passo. Porém, quando é prototipado no FPGA, como ilustra a Figura 10, o estudante passa a não ter acesso as partes internas para verificar o funcionamento, pois tudo está programado dentro do FPGA.

O segundo tipo de componentes são os recursos de entrada e saída presentes na placa. Como a maioria das placas disponibiliza chaves e *displays*, o MIPSFPGA propõe uma abordagem simples para permitir a depuração. A ideia é priorizar a exibição do banco de registradores, do ciclo

atual do *clock*, do contador de programa PC e do conteúdo da memória. Basicamente, usa-se de forma compartilhada os *displays* numéricos para exibir os valores. Como muitas placas tem pelo menos quatro botões, a proposta é cada um selecionar uma opção: registradores, pc, memória ou *clock*. Se a memória ou registradores forem selecionados com o botão, precisamos de um paramêtro adicional para saber qual posição será visualizada. A solução é usar as chaves que irão selecionar qual o endereço de memória ou registro. Por exemplo, como ilustra na Figura 10, se as chaves tem o valor 2 e o botão de registradores estiver pressionado, o *display* irá exibir o conteúdo do registrador 2. Assim, o estudante pode ter acesso a visualização do conteúdo de qualquer registrador ou posição de memória de uma forma simples e direta.

Como a simulação e depuração são realizadas passo a passo, o *clock* também é controlado pela interface proposta. O estudante pode observar os valores dos registradores e da memória, a cada ciclo de *clock*. Para a escolha de quantos pulsos de *clock* serão executados, o aluno seleciona um valor que serão injetados através das chaves. O quarto botão, ao ser pressionado indica, através do *display*, o valor do contador de programa (PC).

Para que o ambiente seja amigável, antes de prototipar, o estudante tem no simulador o mesmo ambiente de verificação da placa de FPGA. O terceiro tipo de componentes ilustrado na Figura 10 são os componentes do kit FPGA executando no simulador MIPSFPGA. Estes componentes não são sintetizados, apenas parte deles, pois são componentes que já existem na placa e são externos ao FPGA. Pode-se visualizar os quatro seletores (botões) para selecionar: registradores, *clock*, PC e memória. As chaves possuem uma interface mais simples, basta digitar o valor, o *display* e o contador de ciclos de *clock* já executados.

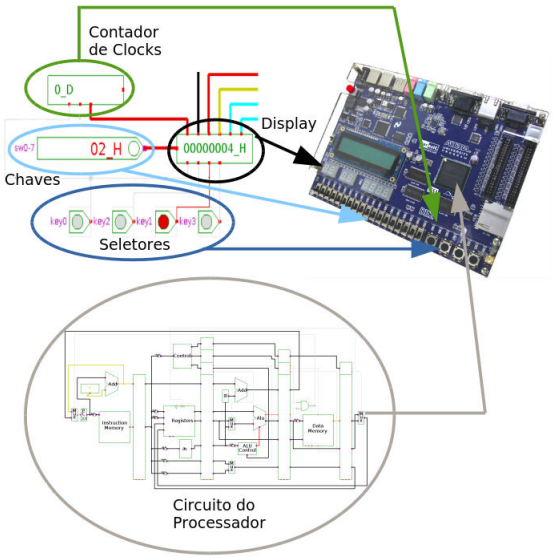

Figura 10. Componentes de depuração no simulador e no kit de FPGA DE-2 Altera

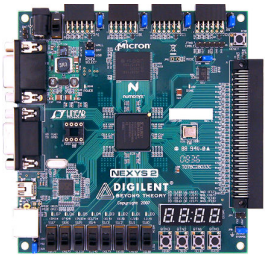

Figura 11. Kit Digilent Nexys 2. [4]

5.2. Kits para Validação

Como existem dois grandes fabricantes de FPGA, para a validação da proposta foram selecionados dois Kits, um da Altera e outro da Xilinx. Os kits são Altera DE2 *Development and Education Board*; e Digilent Nexys 2 com FPGA Xilinx. São kits populares disponíveis em várias universidades. Colaboradores podem fazer o reuso dos componentes da biblioteca do MIPSFGPA para incluir outros kits de prototipação de FPGA.

O Kit Nexys 2, que é ilustrado na Figura 11, é um kit desenvolvido pela Digilent com um FPGA Spartan 3E-500 FG320 da Xilinx. O kit possui quatro *displays* de sete segmentos, quatro botões e sete chaves, além de conectores, memória RAM dentre outros. Como o MIPS é de 32 bits, os valores a serem exibidos precisariam de oito *displays*. Como a placa posui apenas quatro *displays*, a cada segundo metade dos bits são exibidos, ora os dezesseis menos significativos e ora os mais significativos), convertidos em hexadecimal. Este recurso é implementado no processo de geração do Verilog para o kit Nexsys2 de forma transparente.

O Kit Altera DE2, que pode ser visto na Figura 10, é um kit desenvolvido pela Altera que traz o FPGA Ciclone II com 35K LEs. Este kit possui mais recursos que o primeiro kit com oito *displays* de sete segmentos (o que permite exibir números de 32 bits diretamente), quatro botões e dezoito chaves, além de *display* LCD, Memória RAM, leitor de cartões SD dentre outros. Extensões podem ser propostas para visualização mais amigável com o uso da saída VGA e/ou *display* LCD.

Como já mencionado, os quatro botões foram utilizados para selecionar a opção do depurador a ser executada: o botão 3 é responsável por disparar os pulsos de *clock* em uma quantidade escolhida através das chaves. O botão 2 exibe nos *displays* o valor atual contido na saída de PC. O botão 1 exibe nos *displays* o conteúdo do registrador

endereçado pelo valor das chaves e, por fim, o botão 0 exibe nos *displays* o conteúdo da posição de memória endereçada pelo valor das chaves. Quando nenhum botão for pressionado, os *displays* exibirão o valor especificado pelas posições das chaves.

5.3. Gerador de Código em Verilog

O Hades foi concebido com compatibilidade de simulação com VHDL. Um gerador simples de VHDL é incluso no Hades, porém o VHDL gerado não é compatível com as ferramentas da Altera e Xilinx. O MIPSFPGA adotou a linguagem Verilog para exportar o projeto para a prototipação. Contudo, é possível usar a mesma metodologia para exportar em VHDL. Cada componente da biblioteca possui dois arquivos. Um arquivo em Java que descreve o comportamento na simulação e um arquivo em Verilog, que é usado na síntese do projeto.

O projeto no Hades é descrito por um arquivo texto estrutural composto por duas partes: componentes e sinais. A parte de componentes tem a lista de componentes do projeto. Cada componente tem seu tipo, nome da instância e paramêtros para simulação (ex: número de bits, valor inicial, etc). A parte de sinais descreve a conexão entre os componentes. Um *parser* foi implementado para converter o formato do Hades para um arquivo estrutural Verilog. Como já mencionado, o *parser* realiza algumas funcionalidades específica para tornar transparente a geração de componentes que são externos ao FPGA (chaves, *display*, etc). O *parser* foi implementado em Java e tem código aberto para extensões, como por exemplo, inclusão de novos kits.

5.4. Conclusão

Este artigo apresenta uma nova abordagem baseada em simulação para auxiliar no ensino do processador do MIPS. Diferente dos outros simuladores, o MIPSFPGA inova ao usar uma metodologia incremental, ser simples e genérico para ser facilmente estendido. Ademais, a ferramenta inclui síntese e validação em FPGA, apoio para ensino com exercícios integrado no ambiente, exemplos na sequência do livro texto [13] como correspondência um para um nas ilustrações e depuração com kits FPGAs. A ferramenta tem sido utilizado em sala de aula em nível de simulação. A síntese e validação em FPGA está em fase de teste e será aplicada em sala de aula. Para trabalhos futuros, pretende-se acrescentar novas formas para a visualização dos dados, modelagem de novos componentes em Verilog tanto para prototipação como para simulação no Hades com acesso a um FPGA remoto.

6. Agradecimentos

Este projeto foi financiado pelas agências FAPEMIG (Universal 2014), CNPq e CAPES e a empresas Intel/Brazil e Gapso.

Referências

[1] F. A. Alves, D. Almeida, L. Bragança, A. B. Gomes, R. S. Ferreira, and J. Nacif. Ensinando arquiteturas vetoriais utilizando um simulador de instruçoes MIPS. *International Journal of Computer Architecture Education*, 2016.

[2] E. Z. Bem and L. Petelczyc. Minimips: a simulation project for the computer architecture laboratory. *ACM SIGCSE Bulletin*, 35(1):64–68, 2003.

[3] I. Branovic, R. Giorgi, and E. Martinelli. Webmips: a new web-based mips simulation environment for computer architecture education. In *Workshop on Computer architecture education*. ACM, 2004.

[4] DIGILENT. Nexys 2 spartan-3e fpga trainer board.

[5] A. F. Felix, C. V. Pousa, and M. Carvalho. Dimipss: Um simulador didático e interativo do MIPS. In *Workshop sobre Educação em Arquitetura de Computadores*, pages 49–52, 2006.

[6] R. Ferreira, J. M. Cardoso, and H. C. Neto. An environment for exploring data-driven architectures. In *Int. Conference on Field-Programmable Logic and Applications (FPL)*, 2004.

[7] R. Ferreira, J. M. Cardoso, A. Toledo, and H. C. Neto. Data-driven regular reconfigurable arrays: design space exploration and mapping. In *Int. Conf. on Embedded Computer Systems Architectures, Modeling and Simulation SAMOS*, 2005.

[8] R. Ferreira, J. Nacif, S. Magalhaes, T. de Almeida, and R. Pacifico. Be a simulator developer and go beyond in computing engineering. In *Frontiers in Education Conference (FIE)*. IEEE, 2015.

[9] N. Hendrich. A java-based framework for simulation and teaching: Hades—the hamburg design system. In *Microelectronics Education*, pages 285–288. Springer, 2000.

[10] M. Kabir, M. Bari, and A. Haque. Visimips: Visual simulator of MIPS32 pipelined processor. In *Computer Science & Education (ICCSE)*. IEEE, 2011.

[11] J. R. Larus. *Spim s20: A mips r2000 simulator*. Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin, 1990.

[12] P. Marwedel, K. Cong, and S. Schwenk. Ravi: Interactive visualization of information system dynamics using a java-based schematic editor and simulator. 2002.

[13] D. A. Patterson and J. L. Hennessy. *Computer organization and design: the hardware/software interface*. Morgan Kaufmann, 2013.

[14] A. Torres and A. Brito. Ferramenta de auxílio no ensino de organização e arquitetura de computadores: extensão ptolemy para fins educacionais. *Int. Journal of Computer Architecture Education*, 2012.

[15] K. Vollmar and P. Sanderson. Mars: an education-oriented MIPS assembly language simulator. In *ACM SIGCSE Bulletin*, volume 38, pages 239–243. ACM, 2006.