UM PROCESSADOR DE CÓDIGO ABERTO PERSONALIZÁVEL COMPATÍVEL COM O ISA DO NIOS II E O BARRAMENTO AVALON

AN OPEN AND CUSTOMIZABLE PROCESSOR COMPATIBLE WITH NIOS II ISA AND AVALON BUS

Erinaldo Pereira[*]
Paulo Matias[**]
Eduardo Marques[***]

RESUMO

O uso do processador *soft-core* Nios II é muito difundido em projetos de sistemas integrados projetados para FPGAs da Altera, principalmente devido ao seu ferramental de fácil utilização e rica documentação. Apesar de ser um processador configurável, disponível em diferentes versões, com parâmetros ajustáveis, o Nios II não é completamente configurável com exigências específicas, porque ele é fornecido como uma propriedade intelectual de código fechado (IP) e criptografado. Instruções personalizadas só podem ser adicionados de acordo com um modelo predefinido, e mudanças na sua arquitetura não são permitidos. Propomos, portanto, um processador *soft-core* de código aberto chamado Bluespec Soft-processor (BSP), compatível com o conjunto de instruções (ISA) do Nios II e a especificação do barramento Avalon, permitindo que o processador seja integrado ao ecossistema de ferramentas da Altera. Nosso processador foi escrito utilizando o *framework Bluespec*, embora atualmente não competitivo em termos de desempenho com Nios II/f, o BSP é eficaz como base para a exploração do projeto e para o ensino de arquitetura de computadores, até mesmo para os alunos sem experiência prévia em desenvolvimento de projeto de hardware.

Palavras-chave: Soft-core. Nios II. FPGA. Computação Reconfigurável. Bluespec.

ABSTRACT

Usage of the Nios II processor is widespread in integrated system projects designed for Altera FPGAs, mainly due to its easy-to-use tooling and rich documentation. Despite being a configurable processor, available in different versions with adjustable parameters, Nios II is not completely tailorable to specific requirements, because it is provided as an encrypted and closed-source intellectual property (IP) core. Customized instructions can only be added according to a predefined template, and no architectural changes are allowed. We thus propose an alternative open source soft-core processor called Bluespec Softprocessor (BSP), compatible with the Nios II Instruction Set Architecture (ISA) and Avalon Bus specification, allowing it to integrate seamlessly into Altera QSys tool flow. Our processor is the realization of a very simple framework written in na Electronic System Level (ESL) language and therefore, although currently

[*] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (São Carlos-SP). erinaldo@usp.br
[**] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (São Carlos-SP). paulo.matias@usp.br
[***] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (São Carlos-SP). emarques@icmc.usp.br

not competitive in performance with Nios II/f, it is effective as a basis for design exploration and for teaching computer architecture, even to students without prior experience in hardware description languages.

Keywords: Soft-core. Nios II. FPGA. Reconfigurable Computing. Bluespec

Introdução

O software de código aberto impulsionou o desenvolvimento da computação moderna, permitindo maiores níveis de reutilização de código e fomentando a formação de comunidades de desenvolvedores maiores e mais coesas. Projetos de código aberto estão sendo cada vez mais conduzidos de forma profissional, bem diferente do estigma de projeto de tempo livre. Hoje em dia, vários projetos de código aberto que existem seguem estritamente as melhores práticas em engenharia de software, tais como testes automatizados e integração contínua. Estes apresentam-se como um recurso de aprendizagem de valor inestimável para todos os desenvolvedores de software.

Ao contrário de sua contraparte de software, hardware de código aberto não têm grande visibilidade e desenvolvimento ao longo dos anos. Isso decorre do fato de que hardware é composto por objetos físicos, que não podem ser compartilhados, da mesma forma como software. Contudo, o advento de linguagens de descrição de hardware (HDL) e de o conceito de propriedade intelectual reutilizável (IP) permitiu que núcleos de hardware possam ser concebidos de uma forma similar ao software, no sentido de que pode ser escrita sob a forma de código de linguagem de computador, fornecido a um compilador ou sintetizador.

Além disso, os FPGAs (*Field-Programmable Gate Arrays*) oferecem acessibilidade sem precedentes para projetos de hardware em termos de custo e velocidade de desenvolvimento. Com placas experimentais e plataformas de desenvolvimento cada vez mais baratas, FPGAs são adequados para um grande público, desde profissionais e amadores da eletrônica a engenheiros de sistemas de prototipagem em empresas de semicondutores. Empregando uma placa FPGA como a plataforma de destino para a implantação de um projeto de hardware, núcleos IP podem ser escritos e distribuídos de forma semelhante ao software de código aberto. Comunidades e conjuntos de projetos podem, então, formar-se em torno desse ecossistema. Isso já está começando a acontecer, e um exemplo é o NetFPGA, que desenvolve uma plataforma de hardware para projetos orientados a rede, acumulando hoje mais de 2.000 sistemas

implantados em mais de 150 instituições e mais de 40 países ao redor do mundo [1].

Como consequência da crescente complexidade dos projetos de sistemas embarcados, projetando cada componente de um sistema de hardware sem reutilização de componentes existentes tornou-se impraticável e dispendiosa. Assim, a ideia de empregar IPs predefinidos e pré-testados em conjunto com os processadores *soft-core*, para a realização de *co-design* de hardware/software em conjunto com outros componentes, tornou-se uma alternativa atraente e de baixo custo para projetistas e empresas [2].

As HDLs mais difundidas para programar FPGAs são ainda Verilog e VHDL. Quando se pensa em termos de portas lógicas, registros e multiplexadores, elas são completamente adequadas para descrever um projeto, no entanto, esta visão é muitas vezes excessivamente baixo nível para um projeto complexo. Estas linguagens remontam à década de 1980, e foram inicialmente concebidas tendo em conta um ambiente de simulação, em vez de se concentrar em síntese automática, uma área de pesquisa que ainda estava fazendo seus primeiros passos no momento. Portanto, apenas um subconjunto restrito de VHDL e Verilog é sintetizável e recursos suportados por diferentes fornecedores EDA (*Electronic Design Automation*) não são rigorosamente padronizados [3].

Um nível mais alto de abstração tem o potencial de aumentar a produtividade de desenvolvedores e diminuir a incidência de erros de projeto, resultando em uma redução do tempo de lançamento do mesmo no mercado. Por esta razão, várias HLS (*High-Level Synthesis*), ferramentas que possibilitam a síntese em alto nível, têm sido propostas ao longo da última década com o objetivo de facilitar a elaboração de projetos de hardware. Elas têm um objetivo comum: elevar o nível de abstração em que o usuário pode escrever código e gerar seu correspondente VHDL ou Verilog sintetizável [3].

Até recentemente, uma grande barreira para a adoção do Bluespec era a licença restrita do compilador. Apesar de vários núcleos IP escritos em Bluespec foram desenvolvidos de forma *open source*, mesmo alguns projetados por instituições relacionadas com o fornecedor, o compilador é proprietário e licenças estavam disponíveis livre de encargos apenas para as instituições de ensino e pesquisa – indivíduos sem um vínculo com instituições de ensino/pesquisa não são capazes de obter uma licença sem custos. No entanto, com o objetivo de promover a adoção do Bluespec[5], um servidor de compilação foi disponibilizado para qualquer código-fonte publicamente hospedado, em parceria com o projeto Connectal [6].

O ato de dar acesso ao código-fonte e torná-lo disponível abertamente, juntamente com o uso de um nível mais alto e mais fácil de entender a HDL, estabelece uma grande oportunidade tanto para a ensino de arquitetura de computadores e para pesquisa. Permite-se uma melhor compreensão das informações intrínsecas da arquitetura e operações do processador. Além disso, faz com que muitos projetos financeiramente viáveis, oferecendo aos profissionais envolvidos no processo de desenvolvimento a oportunidade de melhorar diretamente qualquer componente do processador, assim como portar o projeto para diferentes plataformas de FPGA. Este artigo contribui para essa ideia, apresentando um processador *soft-core* de código aberto compatível com o conjunto de instruções do Nios II, desenvolvido em Bluspec e integrado as ferramentas de desenvolvimento da Altera.

1 Uma rápida revisão sobre o Nios II

O *soft-core* Nios II é um processador de propósito geral baseado em uma arquitetura RISC (*Reduced Instruction Set Computing*) de 32 bits. Ele está disponível em três versões: o Nios II *fast* (Nios II/f), otimizado para oferecer um melhor desempenho; o Nios II *economy* (Nios II/e), otimizado para ocupar a quantidade mínima de recursos no FPGA; e o Nios II *standard* (Nios II/s), projetado para prover um equilíbrio entre desempenho e tamanho [7]. Embora cada um seja otimizado para uma classe específica de área e desempenho, eles apresentam o mesmo conjunto de instruções, e, portanto, o mesmo código binário pode ser executado em qualquer versão.

O ISA (*Instruction Set Architecture*) do Nios II é composto por 87 instruções, divididas em três tipos de formatos distintos: *I-Type*, *R-Type* e *J-Type* [7]. A Figura 1 apresenta o formato de palavra binária para cada tipo. Além destas instruções, a linguagem *Assembly* do Nios II também inclui 19 pseudoinstruções que são implementados usando instruções equivalentes a partir do conjunto regular.

I-Type				
Bits	31:27	26:22	21:6	5:0
Field description	A	B	IMM16	OP

R-Type					
Bits	31:27	26:22	21:17	16:6	5:0
Field description	A	B	C	OPX	OP

J-Type		
Bits	31:6	5:0
Field description	IMM26	OP

Figura 1. Formato palavra binária por tipos de instruções I, R e J.
Fonte: Adaptado de: [7].

A principal característica da instrução *I-Type* é a presença de um valor imediato incorporado dentro da palavra de instrução. O campo *IMM16* consiste na maior parte dos casos de um inteiro assinado, exceto no caso de operações e comparações lógicas, em que é interpretado como um inteiro sem sinal. Os campos A e B são utilizados para representar uma fonte e um destino (resultado) do operando. Finalmente, o campo OP especifica o código de operação da instrução. Exemplos de instruções *I-Type* são: operações lógicas e aritméticas envolvendo uma constante, de *load* e *store*, e as instruções de gerenciamento da memória cache.

Nas instruções *R-Type*, ambos os operandos origem e de destino são especificados como registros. Na maioria dos casos, A e B são os operandos de origem e C mantém o resultado. Apenas algumas das instruções *R-Type* apresentam um OPX (campo de código da operação estendido) imediato de 5 bits. Se OPX não é usado, ele é preenchido com zeros. Exemplos de instruções *R-Type* são: operações lógicas e aritméticas envolvendo instruções com dois registradores de origem, de comparação e instruções personalizadas.

Instruções *J-Type* são utilizadas para o controle de fluxo. Essas instruções compreendem um código de operação de 6 bits e um valor imediato de 26-bits. Instruções tais como *call* e *jmpi* são exemplos de instruções *J-Type*.

2 O Processador *Bluespec Soft-processor*

O *Bluespec Soft-processor* (BSP) é a nossa implementação do ISA do Nios II em Bluespec. A implementação foi baseada em uma implementação de um MIPS simples originalmente concebido para auxiliar a introdução da tecnologia FPGA em turma de graduação na disciplina de arquitetura de computadores para físicos computacionais [8], que por sua vez foi fortemente inspirada no SMIPS [9] processador do MIT, também desenvolvido para fins pedagógicos.

A arquitetura MIPS [10] foi escolhida como ponto de partida, devido à estreita semelhança entre o Nios II e o conjunto de instruções do MIPS. Adaptação do código-fonte do processador original MIPS a um ISA do Nios II ISA foi inteiramente conduzida pelo primeiro autor deste trabalho, que não tinha experiência prévia em HDLs no momento. Este fato ajuda a ilustrar o potencial da nossa ferramenta (BSP) como auxílio educacional.

O pipeline foi concebido como uma simplificação da arquitetura tradicional

MIPS-I, e é composto por 3 estágios são eles: *Fetch*, *Execute* e *WriteBack* (ver Figura 2). O estágio de *Fetch* consulta palavras da memória cache de instruções e os envia para o estágio *Execute*. O estágio *Execute* decodifica as instruções, lê a partir do conjunto de registros (*RegisterFile*) o valor de quaisquer registros usados como operandos de origem, executa qualquer aritmética que se destina ou operações lógicas, e envia os resultados para o estágio WriteBack. Por sua vez, o estágio WriteBack escreve qualquer resultado no conjunto de registradores ou na memória.

Embora as versões do Nios II/s e Nios II/f incluírem um preditor de desvios, a versão atual do processador proposto não tem esse recurso. Um processador MIPS-compatível não se beneficiaria de um preditor de desvios usando um pipeline proposto em três estágio por causa do tempo de espera especificado em cada estágio pela arquitetura. A arquitetura Nios II, no entanto, além de ser muito semelhante a arquitetura MIPS, não especifica nenhum tempo de espera. A implementação de um preditor de desvios ligado ao estágio de *Fetch* seria, portanto, um ótimo exercício para os alunos de uma classe de arquitetura de computadores baseada em BSP.

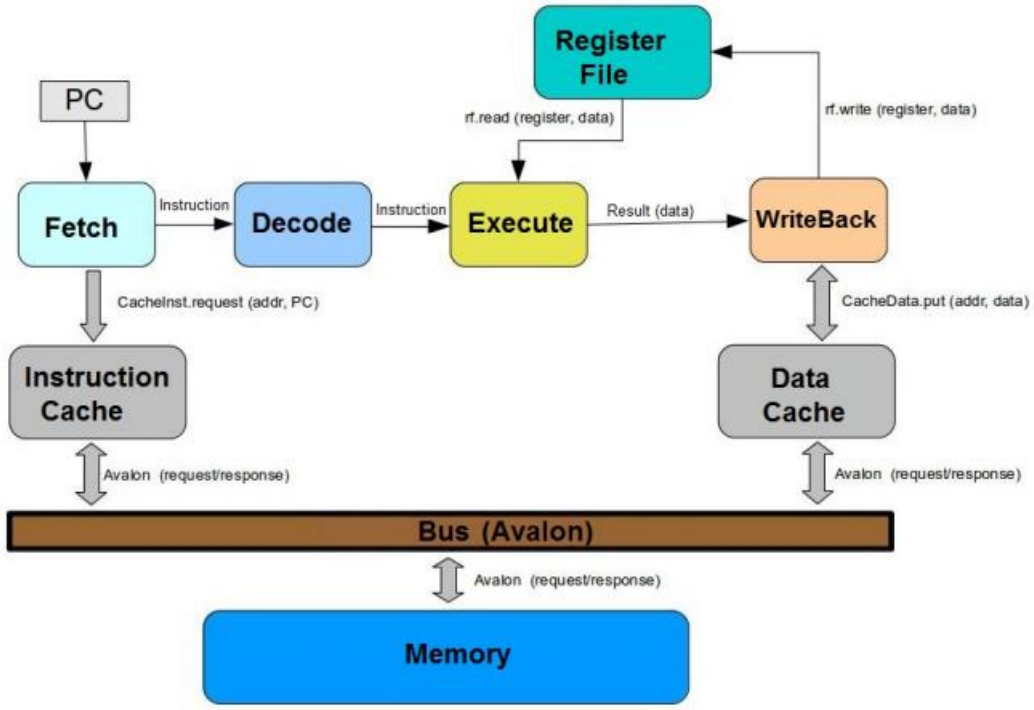

Figura 2. Diagrama de blocos do processador proposto.

O conjunto de registros é implementado atualmente como um conjunto de registros de flip-flop de 32 bits, exceto pelo registro zero (R0), que é programado para

um valor nulo, tal como exigido pelo ISA. Implementar o conjunto de registros usando blocos de memória RAM iria melhorar muito o desempenho do processador, mas também exigiria um *pipeline* mais complexo.

Nossa arquitetura é projetada para que a maioria das instruções possa alcançar uma taxa de transferência de uma instrução por ciclo de clock, o que é uma propriedade muito desejável para uma arquitetura do tipo RISC. No entanto, as instruções de divisão foram implementadas usando um divisor sequencial, uma vez que um divisor de combinações iria apresentar um caminho crítico muito grande, diminuindo a frequência de clock do processador atingível. O divisor sequencial utiliza um esquema de desvio e é capaz de transportar divisões de 32 bits em até 8 ciclos de clock. Por outro lado, as instruções de multiplicação foram implementadas de forma combinacional usando blocos DSP do FPGA.

Como um exercício para ensinar a importância de habilitar e desabilitar linhas em um barramento, implementamos o barramento Avalon-Master em sua forma mais simples. Para contornar a simplicidade da implementação, as instruções que armazenam bytes ou meias-palavras (16 bits) de dados (ou seja, *STB* e *STH*) precisa ocupar o estágio *WriteBack* por mais de um ciclo de clock. O estágio precisa esperar a resposta de pedido de leitura de uma palavra a partir da cache de dados antes de emitir o pedido de escrita de memória. Para simplificar o projeto, este perigo não é tratado, e o *pipeline* é parado até à sua conclusão.

A memória cache é implementada de uma forma muito simples. Existem caches de instruções e dados separadas. Ambas as caches são implementadas usando blocos de memória RAM, que pode responder às solicitações dentro de um ciclo de clock. A cache de instrução é conectado a um circuito de pré-busca que lê palavras sequencialmente a partir do barramento de memória sempre que ele estiver ocioso.

Acessos as caches de instruções e dados caches para o barramento de memória são arbitrados em regime turnos - se ambos os caches tentam acessar o barramento no mesmo ciclo, a que ganha é o que tinha perdido no conflito anterior. A política de substituição de dados na memória cache é o menos usado recentemente (LRU - *Least Recently Used*), ou seja, os itens menos utilizados recentemente são descartados quando necessário.

Uma versão do processador, em que cada memória cache (dados e instruções) é ligada a um barramento Avalon-Master distinto também foi implementada para fornecer uma arquitetura do tipo Harvard, tal como o processador oficial Nios II da Altera.

Cada estágio de *pipeline* do processador é implementado em Bluespec como uma regra, ou como um conjunto de regras que se excluem mutuamente. Estágios comunicam-se entre si através de filas *FIFO*. O estágio de *Fetch* é implementado pela regra *fetchJump*, que é acionado quando o estágio *Execute* solicitou um salto durante o ciclo anterior, e por *fetchAhead*, que aciona o contrário. O *Execute* é implementado pelas regras *execute* e *dividerGetResult*. Este último é acionado quando as execuções diferidas para o divisor sequencial estão concluídas. *WriteBack* é implementado pelas regras *wbFromExec* e *wbLoadResult*.

3 Trabalhos Relacionados

Alguns processadores *soft-core* de código aberto já podem ser encontrados na literatura. Leon-3 [11] é um *soft-core open source* baseado na arquitetura SPARC V8. Além de reutilizar ferramentas como compiladores e depuradores que já estavam disponíveis para a arquitetura, que permite a configuração dos parâmetros do processador de forma semelhante ao Nios II.

Plavec [12] realizou o trabalho que mais se assemelha a nossa proposta - uma implementação em Verilog da primeira geração do processador Nios chamado UT Nios. Esse projeto foi otimizado para FPGAs Altera da família Stratix. Não é tão personalizado como Nios e é composto por dois módulos principais: o caminho de dados e a unidade de controle. O caminho de dados consiste de um *pipline* de quatro estágios: *Fetch* (F), *Decode* (D), *Operand* (O) e *Execute* (X).

O projeto SecretBlaze [13] desenvolveu um processador *soft-core* de código aberto compatível com o ISA do MicroBlaze da empresa Xilinx. Esse processador foi usado como um estudo de caso para mostrar a eficiência de diferentes estratégias de implementação a nível RTL, e realizar uma análise sobre a eficiência de ocupação dos recursos do FPGA.

O processador BlueJEP [14] é um *soft-core* com um ISA personalizado inspirado em Java bytecode. Os autores foram capazes de inserir o processador em ferramentas de integração de sistemas Xilinx, mas só depois de alguma configuração manual das interfaces. Em contraste, o nosso projeto segue a convenção de nomenclatura do barramento Avalon, o que torna mais perfeita a integração do processador as ferramentas da Altera.

4 Discussão da Nossa Metodologia

4.1 A Linguagem Bluespec SystemVerilog

Olhando para a história da engenharia de software, pode-se observar o desenvolvimento contínuo de mecanismos para lidar com a crescente complexidade dos projetos, sobre questões como a modularidade e acoplamento de componentes. No entanto, este desenvolvimento não foi tão expressivo para projetos de hardware. As duas HDLs mais comuns - VHDL e Verilog - estão muito atrás de linguagens de programação modernas em termos de abstrações estruturais, tipos de dados, encapsulamento, parametrização e reutilização de código. Até mesmo seus descendentes, como SystemVerilog e SystemC, sofrem com essas limitações e não são totalmente sintetizáveis [4].

O processador apresentado neste trabalho foi desenvolvido em Bluespec SystemVerilog (BSV), que é uma linguagem de descrição de hardware baseado no paradigma ESL (*Electronic System Level*). Além de ser 100% sintetizável [15], Bluespec também fornece um ambiente rico para a simulação do projeto.

BSV é uma HDL, que reutiliza as capacidades do SystemVerilog. As suas extensões estruturais, envolvendo tipos mais expressivos, sobrecarga, encapsulamento e parametrização são inspirados no Haskell [16], uma linguagem de programação funcional. Sua semântica comportamental é descrita em termos de ações atômicas vigiadas, um formalismo que é apropriado para descrever hardware complexo.

Bluespec proporciona um ambiente para conceber, verificar e implementar uma arquitetura com um alto nível de abstração e suporta síntese RTL, como ilustrado pela Figura 3. Além disso, Bluespec permite construir um SoC (*System-on-a-chip*) complexo num curto período de tempo, quando comparado com um projeto baseado em Verilog ou VHDL. Ele também permite simulações com precisão de ciclo para ser executada em alta velocidade. Quando compilado, código Bluespec gera um código correspondente em Verilog para síntese em hardware, ou em C, para a simulação.

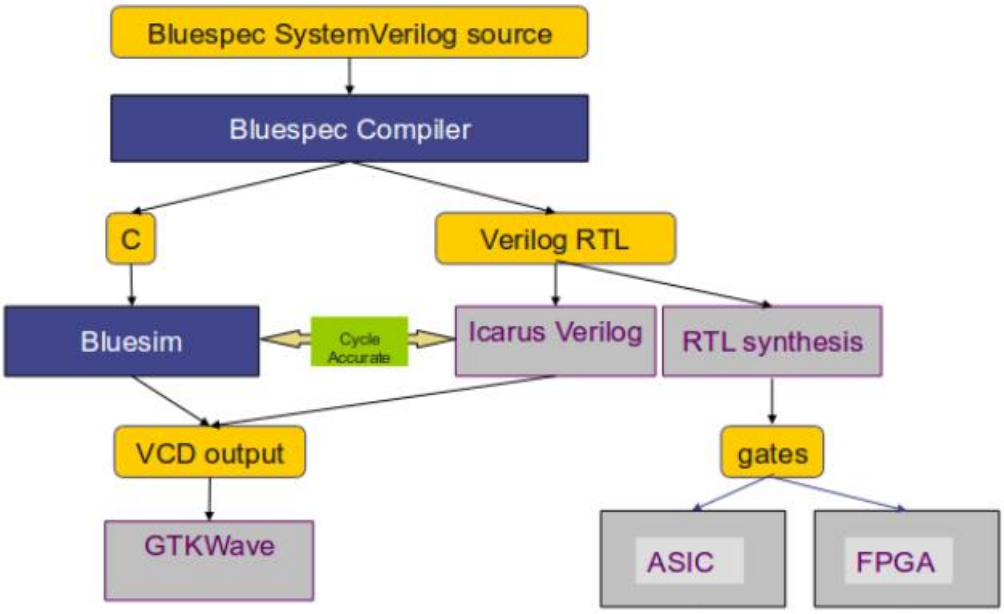

Figura 3. Fluxo de compilação Bluespec.
Fonte: Adaptado de: [15].

Apesar de ser uma HDL relativamente nova, Bluespec tem captado a atenção tanto da academia e da indústria. Isto tem sido demonstrado pelo aparecimento de muitos projetos escritos em BSV, tais como [17], [18], [19], [20], [21].

4.2 Desenvolvimento do BSP

O desenvolvimento deste processador tem sido feito de forma modular. Cada componente da arquitetura pode ser concebido e desenvolvido separadamente como um sistema independente. Isto é possível porque BSV representa a comunicação entre os módulos como transações, de acordo com a especificação do projetista. O desenvolvimento modular facilita a descrição do projeto, a validação das peças que compõem o projeto, testes e documentação. O estado atual do processador pode ser visto na Figura 4.

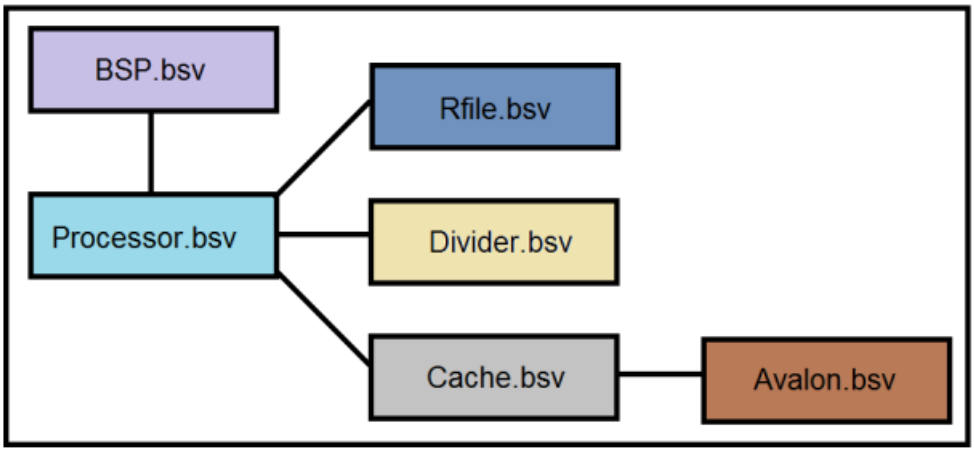

Figura 4. Hierarquia de arquivos para os módulos do processador proposto.

As instruções do processador foram implementadas mantendo o mesmo formato e tipos como os das instruções do Nios II (ver Figura 1), especificada em seu manual. A implementação do conjunto de instruções foi baseada na seguinte abordagem: inicialmente, alguns benchmarks foram selecionados (ver Figura 6) e compilados com o compilador Nios II no modo de depuração, a fim de estabelecer qual as instruções do Nios II eram necessárias para executar esses *benchmarks*, priorizando assim a sua implementação. Esta abordagem inicial destina-se a validar o nível de compatibilidade de instruções para a execução desses benchmarks pelo BSP.

Depois de validar a implementação das instruções presentes nos benchmarks, uma nova abordagem foi definida para implementar as instruções restantes. Esta decisão foi tomada porque os benchmarks selecionados compreendem apenas 10% do Nios II ISA. Nesta nova abordagem, as instruções restantes foram implementadas na mesma ordem em que eles apareceram no manual de instruções do Nios II.

Outra ferramenta de fundamental importância para o desenvolvimento do processador foi BlueSim (simulador Bluespec), que simula os projetos em alta velocidade com precisão ciclo de clock, e no nosso caso foi até duas vezes mais rápido do que a simulação de nível RTL (o que era esperado, devido à BSV ser um ESL [15]).

A simulação dos *benchmarks* no processador foi realizada como se segue: primeiro, o compilador Nios II foi utilizado para gerar o programa binário, o qual foi, em seguida, convertido para um arquivo hexadecimal com extensão *.mem* (o mesmo formato que é utilizado para colocar os dados na memória de inicialização descrita em Verilog). Então, o BlueSim carrega esse arquivo para uma memória simulada e executa

o código do processador. A simulação permitiu-nos verificar se o processador se comportou como esperado por meio de um *trace* de execução gerado pelas rotinas *$display* e *$format* que nós embutimos no código do processador. A Figura 5 mostra o fluxo da geração de arquivos e utilização da ferramenta Bluespec para simular um programa C no BSP.

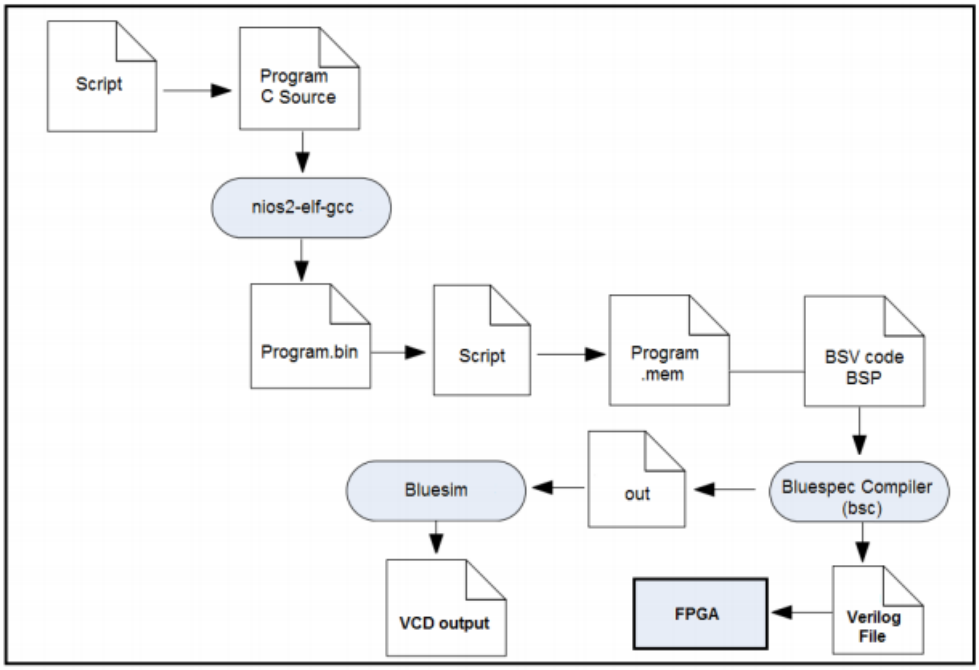

Figura 5. Fluxo para simular um programa C no BSP.

5 Resultados

O processador BSP é um *soft-core* de código aberto funcional, que é totalmente personalizável e não restringe o usuário a uma parametrização específica, como faz o Nios II oficial. Ele pode atualmente ser considerado como um *framework* muito simples para ensino/aprendizagem de arquitetura de computadores/processadores utilizando a linguagem *Bluespec SystemVerilog*, o que é especialmente útil para a educação e para iniciar os estudos na exploração de projeto. Apesar de ser constituída por componentes simples, o BSP é completo, no sentido de que outros módulos podem ser acoplados e as estruturas necessárias para a construção de um processador completo.

A Tabela 1 mostra um exemplo de personalização feita para o nosso processador - um monitor de cache. Ele permite medir as taxas de *miss* e *hit* da cache de dados e de instruções. O Nios II oficial da Altera não dispõe de mecanismos que possibilitem medir esses valores, e este recurso não pode ser implementado pelo usuário usando seus meios

de personalização disponíveis.

Tabela1. Medidas da cache obtidas no BSP

Program	I-hit	I-miss	D-hit	D-miss
	(BSP/N2)	(BSP/N2)	(BSP/N2)	(BSP/N2)
adpcm_coder	41746/-	67/-	2912/-	678/-
adpcm_decoder	33309/-	57/-	3018/-	572/-
rgb_to_hsv	28360/-	69/-	5257/-	751/-
sobel	496/-	52/-	8132/-	496/-

O processador foi simulado utilizando o simulador BlueSim para executar programas escritos na linguagem C e compilados com o compilador *nios2-elf-gcc*. Esta abordagem de simulação foi de fundamental importância para o desenvolvimento deste trabalho, uma vez que permitiu a verificação da compatibilidade com BSP com o código objeto originalmente gerado para o Nios II.

Nós também executamos código em uma placa Altera Stratix V FPGA, a fim de testar e verificar a compatibilidade do BSP com o ecossistema da Altera. Apesar de ser totalmente compatível com QSys, o processador ainda não está integrado ao recurso de depuração da Altera IDE (Nios II EDS) para desenvolvimento C/C ++, pois isso exigiria uma engenharia reversa do protocolo de comunicação JTAG Debug Module. Por esta razão, nós carregamos todos os programas a serem executados no BSP através de uma interface de linha de comando. Criamos dois scripts distintos para realizar essa tarefa: um invoca e executa o compilador Nios II e outro carrega (faz o *load*), do programa executável na memória do BSP.

A Figura 6 apresenta o *speedup* das diferentes versões Nios II quando comparado com o nosso processador. O BSP apresenta melhor desempenho do que Nios II/s e Nios II/f em alguns casos. Isto pode ser explicado pelo fato de que, nestes casos, os algoritmos realizam uma grande quantidade de operações aritméticas. A instrução de multiplicação do BSP, por exemplo, foi implementada utilizando blocos DSP FPGA. Os resultados de síntese de BSP apontam o uso de dezessete desses blocos, enquanto que nas versões Nios II/s e Nios II/f utilizam apenas dois. No entanto, em geral, o BSP tem atualmente um melhor desempenho global do que a versão Nios II/e, e pior desempenho em relação as versões Nios II/s e Nios II/f.

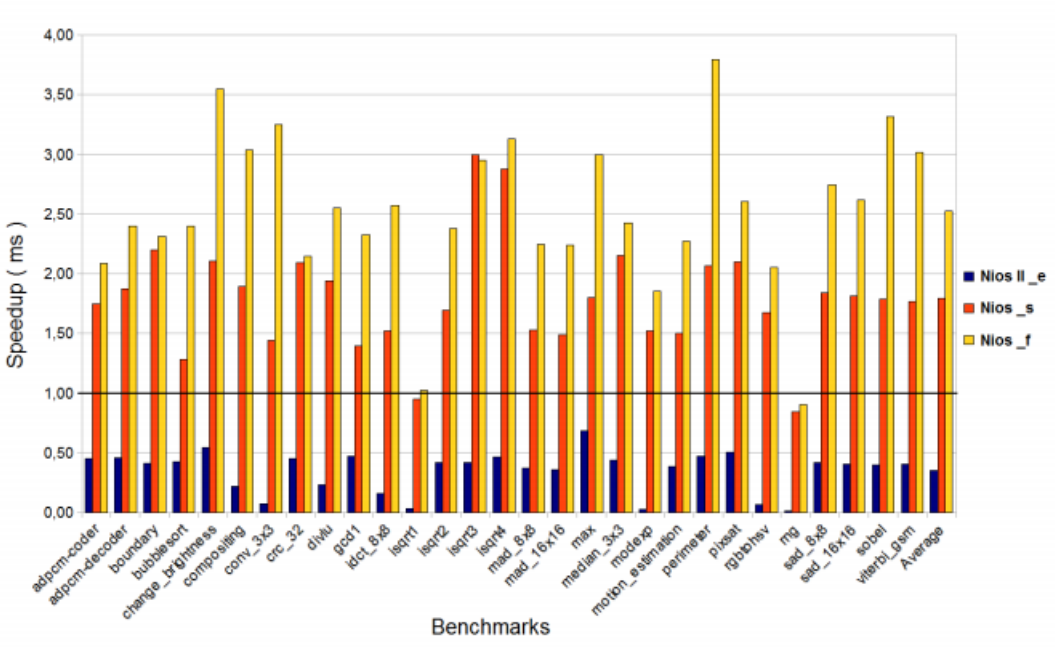

Figura 6. Nios II vs. BSP medições de *speedup* em vários benchmarks.

A Figura 7 ilustra o *trace* de um programa em execução, que pode ser usado para detectar *stall* no *pipeline*, uma característica interessante para auxiliar no ensino e aprendizagem em um curso de arquitetura de computadores. O simulador também mostra as taxas de erro (*miss*) e acerto (*hit*) no acesso a dados nas memorias *caches* de dados e instruções, o número de ciclos de execução e outras medidas importantes.

```
[Trace] [Fetch 00010e]
[Trace] [Exec  00010e] [1009883a] add r4, r2, r0
[Trace] [wbFromExec] WbREG{r:4,data:00000090}
[Trace] [Fetch 00010f]
[Trace] [Exec  00010f] [1805883a] add r2, r3, r0
[Trace] [wbFromExec] WbREG{r:2,data:000000e9}
[Trace] [Fetch 000110]
[Trace] [Exec  000110] [28fffa0e] bge r5, r3, 0xffe8
[Trace] [Fetch 000111]
[Trace] [Exec  000111] [0005883a] add r2, r0, r0
[Trace] [wbFromExec] WbREG{r:2,data:00000000}
[Trace] [Fetch 000112]
[Trace] [Exec  000112] [f800283a] ret
[Trace] [Fetch 000113] [stall] [jump 000102]
[Trace] [Fetch 000102]
[Trace] [Exec  000102] [0001c032] custom 0,zero,zero,zero
[Display] - dmisses          0
[Display] - dhits            0
[Display] - imisses          4
[Display] - ihits           81
[Display] - cycles        2331
Simulation finished
```

Figura 7. Exemplo de um *trace* de execução do BSP quando executado no modo simulado.

Conclusões e Trabalhos Futuros

Este trabalho apresentou o BSP, um *soft-core* projetado para ser totalmente compatível com o ISA do Nios II e disponível em forma de código aberto. É modular e descrito numa linguagem de alto nível. Essas características, combinadas com a simplicidade de projeto, destinam-se a permitir que os estudantes de arquitetura de computadores e grupos de pesquisa possam modificar o código fonte, a realização de aulas e experimentos possam contribuir para o desenvolvimento do projeto.

Uma vez que é concebido como um componente para a construção de sistemas embarcados dentro de um FPGA, este *soft-core* pode ser totalmente personalizado para atender às exigências de projeto ou para satisfazer fins educacionais. É possível usar o BSP de um modo semelhante ao processador Nios II, isto é, a execução de programas C/C++ integrados com todos os núcleos IP e ferramentas disponíveis da Altera (atualmente excluindo o recurso de depuração da IDE Nios II EDS).

O trabalho futuro vai concentrar-se na implementação de uma unidade de gestão de memória (MMU) e de suas instruções relacionadas, o que permitiria rodar um sistema operacional completo (como o Linux) no processador. Além disso, pretendemos desenvolver dentro do processado um módulo de depuração do hardware.

O código fonte do BSP está disponível em [22].

Referências

[1] NetFPGA. **NetFPGA**, 2014. Available: <http://www.netfpga.org/>

[2] XIE, F.; YANG, G.; SONG, X. Component-based hardware/software co-verification for building trustworthy embedded systems. **Journal of Systems and Software**, v. 80, n. 5, p. 643–654, sept. 2007.

[3] BACON, D.; RABBAH, R.; SHUKLA, S. FPGA programming for the masses. **Queue**, v. 11, n. 2, p. 40-52, feb. 2013. Available: <http://doi.acm.org/10.1145/2436696.2443836>

[4] NIKHIL, R. S. Abstraction in hardware system design. **Queue**, v. 9, n. 8, p. 40-54, aug. 2011. Available: <http://doi.acm.org/10.1145/2016036.2020861>

[5] MOILANEN, J. Emerging hackerspaces – peer-production generation. In: HAMMOUDA, I. et al (Eds). **Open Source Systems:** Long-Term Sustainability, ser. IFIP Advances in Information and Communication Technology. Berlin: Springer Berlin Heidelberg, 2012. V. 378. p. 94–111.

[6] WANG, H.; HICKS, J.; BRASK, A. T. **Connectal buildbot,** 2015. Available: <http://connectalbuild.qrclab.com>

[7] Altera, Nios II Processor Reference Handbook, 2011. Available: <http://www.altera.com/literature/hb/nios2/n2cpu nii5v1.pdf>

[8] MATIAS, P. **Simple MIPS microcontroller for educational purposes,** 2014. Available: <http://dx.doi.org/10.5281/zenodo.10320>

[9] ARVIND; ASANOVIC, K. **Complex digital systems**, 2005. Available: <http://csg.csail.mit.edu/6.884/index.html>

[10] HARRIS, D. M.; HARRIS, S. L. **Digital Design and Computer Architecture**. São Francisco: Elsevier, Incorporated, 2007.

[11] ZAIDI, I. et al. Power/area analysis of a FPGA-based open-source processor using partial dynamic reconfiguration: digital system design architectures, methods and tools. In: EUROMICRO, 11, 3-5 sept.. 2008. p. 592–598.

[12] PLAVEC, F. Experiences with Soft-core processor design. In: IEEE INTERNACIONAL, **Proceedings**, 19., 4-8 apr. 2005. p. 167b.

[13] BARTHE, L et al. "Optimizing an opensource processor for FPGAs: a case study". In: FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL). 5-7 sep. 2011, p. 551-556.

[14] GRUIAN, F.; WESTMIJZE, M. VHDL vs. Bluespec SystemVerilog: a case study on a Java embedded architecture. In: **SAC '08**: Proceedings of the 2008 ACM symposium on Applied computing. New York: ACM, 2008. p. 1492–1497.

[15] BLUESPEC. **Bluespec SystemVerilog user guide**, 2011. Available: <http://www.bluespec.com/forum/download.php?id=107>

[16] JONES, S. P. **Haskell 98 Language and Libraries the Revised Report**, 2014. Available: <http://www.haskell.org/definition/ haskell98-report.pdf>

[17] DAVE, N. H. **Designing a processor in Bluespec**. 2005. 67 p. Dissertação (Mestrado em Engenharia Elétrica) – Instituto de Tecnologia de Massachusetts, Massachusetts, 2005. Disponível em: <http://hdl.handle.net/1721.1/30174>

[18] GRUIAN, F.; WESTMIJZE, M. BlueJEP: A flexible and highperformance java embedded processor. In: **Proceedings of the JTRES 2007.** Vienna: ACM Press, 2007. p. 222–230.

[19] EKANADHAM, K. TSENG, J.; PATTNAIK, P. IBM PowerPC design in Bluespec. IBM Research Division, Tech. Rep., 2008.

[20] CHU, S.-L.; LI, G.-S.; LIU, R.-Q. Caliburn: a MIPS32 VLIW processor with hardware instruction morphing mechanism. **Przeglad Elektrotechniczny**, v. 89, n. 3 B,

p. 10–12, 2013.

[21] WATSON, R. N. M. **Bluespec extensible RISC implementation (BERI)**, 2014. Available: <http://www.cl.cam.ac.uk/research/security/ctsrd/beri.html>

[22] PEREIRA, P. **BSP – Bluespec Soft-Processor**, 2015 (to be replaced with permanent archive in camera-ready version). Available: <https://www.dropbox.com/s/l8d0wpzjmlps7qg/BSP.tar.gz?dl=0>