ReonV: uma versão RISC-V do processador SPARC LEON3

Lucas Castro, Rodolfo Azevedo

lcbc.lucascastro@gmail.com, rodolfo@ic.unicamp.br

Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251 – 13083-852 – Campinas – SP – Brasil

Abstract. *This paper endorses the importance of reuse and contribution for open source hardware, offering as example the development of a RISC-V soft-core processor, named ReonV, which was developed by reusing all IP cores from a well designed SPARC 32-bit processor changing only its ISA in order to obtain a fully operational RISC-V processor that inherits all other modules and Board Support Package (BSP) from the original SPARC core.*

Resumo. *Este artigo reforça a importância de reutilização e contribuição para o desenvolvimento de hardware de código aberto, mostrando o exemplo de desenvolvimento de um processador soft-core RISC-V, chamado ReonV, que foi desenvolvido reutilizando todos os IP cores de um processador SPARC V8 de 32 bits já consolidado, modificando apenas seu pipeline para a nova ISA e herdando todos os outros módulos e o Board Support Package (BSP) do processador original.*

1. Introdução

A comunidade da computação está bastante familiarizada com reutilização de *software* em larga escala, o que é uma das grandes vantagens de se desenvolver software de código aberto. Por outro lado, quando trata-se de *hardware*, especialmente em grandes projetos como os de processadores, reutilização normalmente é limitada a pequenos módulos ou é feita para adicionar extensões na ISA que o processador suporta - geralmente o desenvolvimento de novos *cores* é feito do zero, construindo todo o processador e o seu *Board Support Package* (BSP). Esta abordagem trás obstáculos desnecessários, como recriar módulos já existentes em outros processadores (como periféricos e controladores) além de enfrentar problemas de compatibilidade quando for necessário estender o BSP para novas FPGAs ou outros ambientes de simulação e síntese.

Essa foi a motivação desta pesquisa em mostrar um exemplo de reutilização de *hardware* em larga escala construindo um novo processador *soft-core* RISC-V [1], o qual chamamos de ReonV, reutilizando os *IP cores* da GRLIB, uma biblioteca de módulos em VHDL que também contém o processador SPARC V8 LEON3 [2, 3, 4]. A abordagem tomada foi de alterar apenas o *pipeline* do processador original para implementar a ISA RISC-V em vez de SPARC, herdando todos os outros componentes já existentes, como memória, controladores de memória, suporte a periféricos, *debug support unit* (DSU), *scripts* de síntese para diversas FPGAs de diferentes fabricantes e outros.

Com isto, além de um exemplo bem sucedido de reutilização de *hardware*, também é possível contribuir com a comunidade RISC-V lançando uma versão de um processador amplamente documentado, testado e utilizado em aplicações reais implementando esta nova ISA.

2. GRLIB e o LEON3

A *GRLIB IP Library* é um conjunto integrado de IP *cores* reutilizáveis, feitos visando o desenvolvimento de *system-on-chip* (SOC), lançada sob licença GNU GPL pela Aeroflex Incorporated. Os módulos IP são posicionados ao redor de um *bus* compartilhado e utilizam um método coerente, configurável e automatizado de simulação e síntese. A GRLIB também provê *templates* de *designs* para diversas placas de FPGA e *scripts* para facilitar a síntese e implementação nas placas suportadas [3, 4, 5].

A GRLIB contém o LEON3, que é um modelo sintetizável, escrito em VHDL, de um processador de 32 bits que utiliza a arquitetura SPARC V8. Este modelo é altamente configurável e desenvolvido especialmente para aplicações do processador como um SOC, utilizando os demais módulos da GRLIB [3, 4, 6].

A figura 1 mostra uma representação do processador LEON3. Pode-se notar o quão configurável ele é, já que pode ser sintetizado com um mínimo de 5 módulos cruciais com opções minimalistas ou com até 17 módulos altamente configuráveis [3, 6].

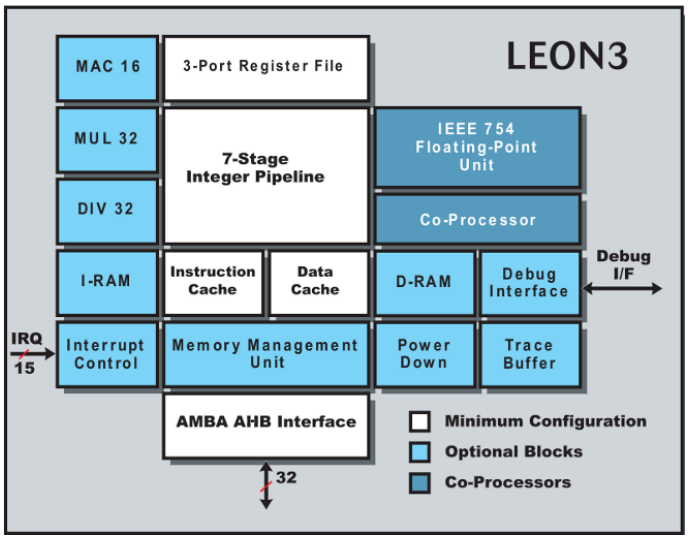

Figura 1. Representação do processador LEON3 [7].

A figura 2, por sua vez, mostra uma representação da estrutura do SOC do processador LEON3 nos *templates* da GRLIB, além de evidenciar o grande leque de periféricos e controladores existentes no pacote provido nesta biblioteca. Esta figura também evidencia que existem ganhos consideráveis ao reutilizar esta estrutura no desenvolvimento de um novo processador RISC-V, uma vez que este a herdará sem necessidade de modificações.

3. ReonV - Uma versão RISC-V do processador LEON3

ReonV é o nome dado ao processador RISC-V de 32 bits obtido após alterar a ISA do LEON3 de SPARC para RISC-V. Ele possui um repositório aberto[1] e foi lançado sob licença GNU GPL [8].

O ReonV foi desenvolvido reutilizando o processador LEON3, bem como a estrutura do seu SOC provido pela GRLIB, alterando apenas o seu *pipeline* de instruções

[1]Repository available at: https://github.com/lcbcFoo/ReonV

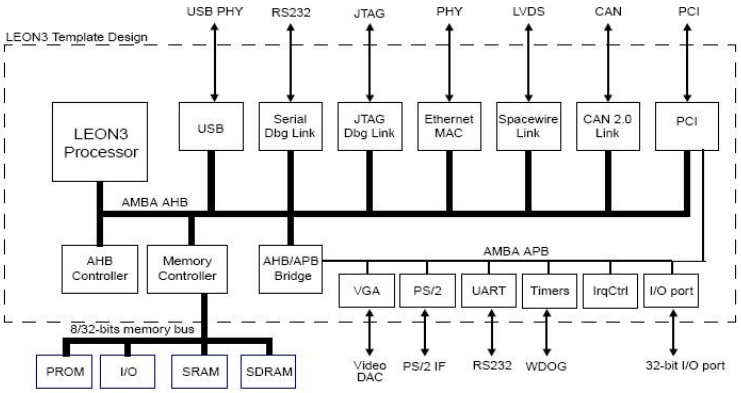

Figura 2. Representação dos periféricos e da estrutura do SOC do LEON3 [6].

de inteiros de 7 estágios (observe a figura 1) a fim de implementar a ISA RV32I [1] sem instruções privilegiadas em vez da ISA original SPARC V8, mantendo todos os outros módulos e recursos providos pela GRLIB intocados. Com isto, buscamos obter um processador RISC-V que contivesse todo o suporte de síntese para diferentes FPGAs, periféricos e demais recursos que o processador LEON3 possui sem a necessidade de desenvolvê-los novamente para um novo processador.

4. Metodologia e Resultados

A abordagem de desenvolvimento se baseou no princípio de reutilizar o máximo possível do processador LEON3 original. Foi possível manter todas as alterações internas ao *pipeline* do processador. Como não houve modificações nos demais módulos, garantimos que estes continuarão corretos e viabilizamos que o novo processador RISC-V herde o suporte construído para o LEON3. Uma consequência desta abordagem é que foi preciso garantir, neste primeiro momento, que eventuais alterações no *pipeline* não alterassem o comportamento esperado deste módulo em relação aos demais, de forma que foi preferível realizar alterações pouco eficientes mantendo a estrutura do *pipeline* original, priorizando corretude e deixando análises e otimizações de eficiência para trabalhos futuros.

Esta abordagem permitiu que utilizássemos *software* desenvolvido para o LEON3, como ferramentas de síntese, *scripts* e até mesmo o seu monitor de depuração, o GRMON, para facilitar a comunicação com o processador e o seu desenvolvimento [9]. O GRMON possui uma interface que interage com a *Debug Support Unit* (DSU) do processador, permitindo facilmente carregar e rodar programas, ler regiões de memória e registradores, além de outras operações de depuração úteis [6, 9]. Mesmo sendo um monitor feito para um processador SPARC e, portanto, tendo incompatibilidades com a arquitetura RISC-V, ter uma ferramenta como o GRMON já no estágio de desenvolvimento do processador foi uma vantagem significativa desta abordagem.

4.1. Alterando a ISA

A alteração da ISA se aproveitou do fato de o *pipeline* estar corretamente implementado, de forma que quaisquer modificações puderam ser facilmente testadas, especialmente tendo a DSU e o GRMON em mãos. Modificamos apenas o necessário para obter um processador funcional compatível com a RV32I. Por isto, mantivemos a estrutura do

pipeline com 7 estágios (*Fetch*, *Decode*, *Registers Access*, *Execute*, *Memory Access*, *Exception* e *Write Back*), conseguindo restringir a maior parte das modificações, porém não todas, ao estágio de decodificação, já que houve necessidade de remover e incluir instruções para atender a especificação da nova ISA.

Ocorreram problemas de instruções e convenções incompatíveis, sendo as mais graves a respeito da *endianness* do processador (SPARC utiliza *big endian* e RISC-V *little endian*) e das instruções de desvio de fluxo condicionais (*branches*). Foi possível tratar todas elas de forma satisfatória, sem perda significativa de desempenho, com exceção dos *branches*.

Os *branches* na ISA SPARC usam bits de aritmética (C, N, V, Z) que já foram previamente setados por uma instrução anterior para apenas validar se a condição é verdadeira e realizar ou não o desvio [2], por outro lado, na arquitetura RISC-V, as instruções de *branch* devem ler 2 registradores, calcular se a relação entre eles é verdadeira e só depois realizar ou não o desvio [1]. Desta forma, as opções possíveis para implementar estas instruções eram: modificar drasticamente os estágios do *pipeline* para atender de forma eficiente à especificação RISC-V ou desabilitar o mecanismo de *branch prediction* do processador e inserir *stalls* (ou bolhas) no *pipeline* para garantir que o cálculo da condição fosse correto. Como inicialmente optou-se por garantir corretude com o menor número possível de modificações, foi realizada a segunda opção - o *branch predictor* foi desabilitado e inserimos 3 bolhas no *pipeline*.

Após finalizadas as alterações, todas as instruções, com exceção dos *branches*, utilizam o mesmo número de ciclos que as suas equivalentes na arquitetura SPARC utilizavam no processador original, como mostrado na tabela 1. Nela, BP se refere à *Branch Prediction* e as abreviações seguem as especificações RISC-V [1]. As instruções de *store* utilizam 2 ciclos em ambos os modelos, pois o *design* do *pipeline* precisa de um ciclo para carregar a posição de memória e outro para enviar os dados. É importante notar que no processador LEON3, caso o BP erre, as instruções de *branch* consomem 3 ciclos, ao passo que no ReonV todos os *branches* consomem 4.

Tipo	**Instruções**	**Ciclos (ReonV / Leon3)**
Relativa ao PC	AUIPC	1 / não implementado
Controle de fluxo	JALR, JAL	1 / 1
Controle de fluxo condicional	BEQ, BNE, BLT, BGE, BLTU, BGEU	4 (sem BP) / 1 (BP correto) 3 (BP errado)
Manipulação de memória	LB, LBU, LH, LHU, LW	1 / 1
Manipulação de memória	SB, SH, SW	2 / 2
Lógicas e aritméticas	ADD(I), SUB, XOR(I), OR(I), AND(I), SLL(I), SRL(I), SRA(I), SLT(I), SLTU(I), LUI	1 / 1

Tabela 1. Instruções implementadas pelo ReonV e comparação dos ciclos necessários entre elas e suas equivalentes SPARC no LEON3.

Embora não tenham sido feitos testes específicos de desempenho, é visível que o processador ReonV apresenta uma perda significativa de eficiência devido aos 4 ciclos necessários para instruções de *branch*. Por outro lado, ainda é possível reverter tal perda com mudanças mais incisivas no *pipeline*, como será discutido na seção 5.

O projeto utilizou a ferramenta de desenvolvimento *Xilinx Vivado Design Suite* [10] e a placa FPGA Nexys4DDR, da AVNET [11], para sintetizar e rodar o *design* do processador. A tabela 2 compara a utilização de recursos da placa entre os *designs* do LEON3 e do ReonV com as mesmas configurações de síntese. Pode-se notar que, ao passo que o ReonV utiliza menos LUT, ele demanda mais *flip-flops*, entretanto, como a diferença é pequena, consideramos que não houve mudança significativa de consumo de recursos devido à alteração da ISA.

Recurso	**ReonV**	**LEON3**	**Disponível na Nexys4DDR**	**Utilização % (Reonv / LEON3)**
LUT	8665	8820	63400	13.67 / 13.91
LUTRAM	97	98	19000	0.51 / 0.52
FF	5346	4977	126800	4.22 / 3.93
BRAM	23	23	135	17.04
I/O	81	81	210	38.57
BUFG	11	11	32	34.38
PLL	5	5	6	83.33

Tabela 2. Comparação dos recursos utilizados pelo LEON3 e pelo ReonV. A coluna de porcentagem de utilização é em relação ao total disponível na placa Nexys4DDR.

4.2. Desenvolvendo software para suporte de execução

Com a ISA RISC-V implementada, obteve-se um processador completo, configurável e com suporte para execução em diferentes FPGAs. Entretanto, o processo de compilação e linkagem de todo código era, inicialmente, manual, devido às manobras necessárias para utilizar o monitor do processador LEON3, o GRMON. Dessa forma, foram desenvolvidas as seguintes estruturas de software para automatizar o processo:

1. Código de inicialização (*crt0*) simples, para setar as posições de memória, os valores iniciais dos registradores e a pilha do programa.
2. Camada de "chamadas de sistema", com implementações simplificadas de algumas funções do padrão POSIX. Não foram implementadas *syscalls* até o momento (já que para tal é preciso que o processador suporte instruções privilegiadas e nível de supervisor), estas funções são chamadas como funções comuns, e fornecem apenas uma interface já testada para leitura e escrita de regiões de memória.
3. Interface de software para comunicação serial, aproveitando os módulos UART já implementados pela GRLIB, permitindo que os códigos sendo executados no processador enviem mensagens serialmente que são recebidas e exibidas pelo GRMON.
4. Scripts de automação do processo de compilação, linkagem e execução de binários utilizando o GRMON para comunicação com o processador.

4.3. Benchmarks de validação

Visando avaliar a corretude da mudança de ISA, foram feitos *benchmarks* de validação utilizando o ambiente automatizado de síntese e compilação descrito na seção 4.2. Os testes adotados foram adaptações do conjunto disponível nos *benchmarks* da WCET [12], escolhidos por apresentarem em detalhes o conjunto de estruturas de código que cada programa avalia em sua execução.

O ReonV foi testado e corretamente executou cada um dos programas descritos na tabela 3. Todos os testes foram executados com o processador sintetizado na forma descrita na seção 4.1. O código fonte de cada teste, bem como todo o ambiente utilizado para executá-los estão disponíveis no repositório do projeto [8]. Foi utilizada a versão oficial da *RISC-V Toolchain* [13] compilada para a ISA RV32I para compilar os programas de teste. A tabela 3 também mostra quais estruturas de código são avaliadas por cada um deles, de acordo com a seguinte legenda:

- L - Possui loops
- N - Possui loops aninhados
- B - Manipulação de bits
- A - Utilização de vetores ou matrizes
- R - Possui recursão

Arquivo	Descrição	L	N	A	R	B
bs.c	Busca binária	✓		✓		
bsort.c	Bubble sort	✓	✓	✓		
cover.c	Fluxos condicionais	✓				
expint.c	Exponenciação de inteiros	✓	✓			
fac.c	Fatorial	✓			✓	
fibcall.c	Fibonacci	✓				
insertsort.c	Insertion Sort	✓	✓	✓		
complex.c	Laços aninhados	✓	✓			
matmult.c	Multiplicação de matrizes	✓	✓	✓		
ndes.c	Manipulação de bits	✓		✓		✓
prime.c	Testa primos	✓				
qsort-exam.c	Quick Sort	✓	✓	✓		
recursion.c	Recursão				✓	

Tabela 3. Programas utilizados para validar o ReonV. Todos são adaptações dos *benchmarks* da WCET [12].

5. Discussão

Dado que todos os testes foram executados corretamente, dá-se que as alterações no *pipeline* e a alteração de ISA para RV32I foram bem sucedidas. Embora nem todos os periféricos tenham sido testados, pôde-se atestar durante a execução dos *benchmarks* que módulos importantes do processador funcionaram naturalmente, sem modificações necessárias, entre eles: o banco de registradores, as *caches* de dados e instruções, as instâncias

de memória (SRAM e DRAM) , a DSU, a interface AMBA AHB, a interface serial UART, entre outros. Isto reforça que a proposta inicial de herdar a infraestrutura do processador original também foi alcançada.

Somado a isto, a reutilização dos módulos da GRLIB permitiu utilizar a DSU do processador original e um monitor de depuração desde o começo do desenvolvimento, uma vantagem significativa para o projeto. Além disso, como o LEON3 possui extensa documentação e está no mercado há alguns anos, pôde-se assumir que os demais módulos, não alterados, mantiveram-se corretos, restringindo análises de depuração ao próprio *pipeline*.

É importante ressaltar que a versão inicial do ReonV, apresentada neste artigo, ainda não implementa nenhuma extensão de ISA para instruções como multiplicação, divisão e ponto flutuante. Por outro lado, os módulos para implementar tais operações já existem no processador, já que foram herdados do LEON3, de forma que seria preciso apenas reincorporá-los ao *pipeline* para estender o suporte à essas instruções. Além disso, instruções privilegiadas e controle de modos (supervisor, usuário e de máquina) também não foram implementados até o momento.

Embora tenham existido problemas já esperados de incompatibilidade durante a mudança de ISA que fizeram esta versão do processador pouco eficiente, especialmente se tratando das instruções de *branch*, não existem restrições de projeto que impeçam modificações mais profundas na estrutura do *pipeline* a fim de melhorar o desempenho. Além disso, tais incompatibilidades são específicas das ISAs em questão (SPARC e RISC-V) e poderiam não ocorrer em abordagens semelhantes de alteração do *pipeline* em outras arquiteturas; por exemplo, MIPS e RISC-V não encontrariam a mesma incompatibilidade nas instruções de *branch* dado que a especificação destas instruções no MIPS é semelhante a do RISC-V [1, 14]. Por isso, pode-se dizer que a abordagem proposta de reutilização de *hardware* não é responsável pela perda de eficiência causada por incompatibilidades - esta é consequência das ISAs em questão e da escolha de realizar o menor número de alterações na versão original do processador.

6. Possibilidades de trabalhos futuros

A seguir é mostrado um panorama de algumas áreas que podem ser exploradas em trabalhos futuros com base no processador apresentado neste artigo, os problemas enfrentados e visando a capacidade de executar sistemas de alta complexidade no futuro:

- Aprimorar a eficiência das instruções de *branch*, ativar o mecanismo de *Branch Prediction*.
- Aproveitar as unidades de multiplicação, divisão e ponto flutuante herdadas do LEON3 para desenvolver suporte à estas intruções na arquitetura RISC-V.
- Deixar de usar o monitor GRMON devido aos problemas de incompatibilidade com RISC-V, seja adaptando a DSU do processador para se comunicar com outras ferramentas de depuração com suporte a RISC-V ou criando uma ferramenta similar ao próprio GRMON que contenha tal suporte.
- Desenvolver as camadas de intruções privilegiadas, exceções e interrupções do ReonV para o padrão descrito pela ISA RISC-V.
- Desenvolver o tratamento de *system calls*, implementar as funções do padrão POSIX e portar um sistema operacional compatível com POSIX para o ReonV.

7. Conclusão

Mesmo existindo incompatibilidades previstas devido às diferenças nas especificações das ISAs SPARC e RISC-V, esta pesquisa foi bem sucedida ao desenvolver um processador RISC-V a partir de um *soft-core* SPARC já consolidado no mercado, mostrando que é possível aplicar os conceitos de reutilização e compartilhamento de código entre diferentes comunidades de desenvolvedores de *hardware* de código aberto a exemplo do que é feito com *software*.

Além disso, todo o trabalho desenvolvido para tal alteração não adicionou restrições que impeçam projetos futuros de utilizar, modificar e estender o que foi feito até o momento, mantendo a possibilidade de aprimorar a eficiência do processador apresentado, além de aumentar o seu suporte de instruções e de desenvolver camadas de *software* de baixo nível para oferecer compatibilidade a padrões de interface com sistemas operacionais, como o POSIX.

Este trabalho conclui-se contribuindo com a comunidade de desenvolvimento da arquitetura RISC-V ao lançar um modelo em VHDL de código aberto de um *soft-core* RISC-V que trás consigo todo o suporte de *hardware* e síntese de uma biblioteca já documentada e validada como consequência direta da sua abordagem de desenvolvimento: reutilização de *hardware*.

8. Agradecimentos

Agradecemos todo o suporte dado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (processo 2017/04018-2), CNPq e CAPES (processo 2966/2014) e do Instituto de Computação da Unicamp para a realização deste trabalho.

Referências

[1] *The RISC-V Instruction Set Manual - Volume I: User-Level ISA - Document Version 2.2*. RISC-V Foundation, 1300 Henley Court, Pullman, WA 99163, 509.334.6306. URL `https://riscv.org/specifications/`. Editors Andrew Waterman and Krste Asanovic. Accessed on 07/2018.

[2] *The SPARC Architecture Manual, Version 8*. Sparc International Inc., 535 Middlefield Road - Suite 210 - Menlo Park - CA 94025.

[3] *GRLIB IP Library User's Manual - Version 2018.1*. Cobham Gaisler AB, Kungsgatan 12, 411 19 Gothenburg, Sweden, . URL `www.cobham.com/gaisler/products/grlib/grlib.pdf`. Accessed on 07/2018.

[4] Cobham Gaisler AB. Grlib ip library. URL `www.gaisler.com/index.php/products/ipcores/soclibrary`. Accessed on 07/2018.

[5] *GRLIB IP Core User's Manual - Version 2018.1*. Cobham Gaisler AB, Kungsgatan 12, 411 19 Gothenburg, Sweden, . URL `www.cobham.com/gaisler/products/grlib/grip.pdf`. Accessed on 07/2018.

[6] *Configuration and Development Guide - Version 2018.1*. Cobham Gaisler AB, Kungsgatan 12, 411 19 Gothenburg, Sweden, . URL `www.cobham.com/gaisler/products/grlib/guide.pdf`. Accessed on 07/2018.

[7] Sven Åke Andersson. Leon3 32-bit processor core. URL `www.rte.se/blog/blogg-modesty-corex/leon3-32-bit-processor-core/1.5`. Accessed on 07/2018.

[8] Lucas Castro. ReonV - A RISC-V version of LEON3. URL www.github.com/lcbcFoo/ReonV. Accessed on 07/2018.

[9] *GRMON2 User's Manual*. Cobham Gaisler AB, Kungsgatan 12, 411 19 Gothenburg, Sweden, . URL www.gaisler.com/doc/grmon2.pdf. Accessed on 07/2018.

[10] *Vivado Design Suite User Guide - Release Notes, Installation, and Licensing*. Xilinx, Inc. URL www.xilinx.com.

[11] *Nexys 4™ FPGA Board Reference Manual*. Digilent Inc, 1300 Henley Court, Pullman, WA 99163, 509.334.6306.

[12] Mälardalen WCET research group. Wcet benchmark. URL http://www.mrtc.mdh.se/projects/wcet/benchmarks.html. Acessado em 30/04/2018.

[13] RISC-V Foundation. Software Tools. URL www.riscv.org/software-tools. Accessed on 07/2018.

[14] *MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual*. Imagination Technologies LTD.