ProcessorCI: Integração Contínua para processadores RISC-V em FPGAs*

Julio N. Avelar[1], Victor P. Lago[1], Ângelo R. P. Malaguti[1], Rodolfo Azevedo[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal – 13.083-852 – Campinas – SP – Brazil

{j241163, v194784, a165429}@dac.unicamp.br, rodolfo@ic.unicamp.br

Abstract. *This paper presents an automated infrastructure for the verification of RISC-V processors, using continuous integration techniques combined with FPGAs from multiple vendors, such as Altera/Intel, Xilinx/AMD, Gowin, and Lattice. We designed a robust and scalable environment that facilitates early detection of faults in various RISC-V processor implementations, ensuring compliance with ISA specifications. Preliminary results demonstrate the effectiveness of the approach, highlighting the methodology's ability to quickly identify errors and gather essential data for processor selection and verification.*

Resumo. *Este artigo apresenta uma infraestrutura automatizada para a verificação de processadores RISC-V, utilizando técnicas de integração contínua combinadas com FPGAs de múltiplos fornecedores, como Altera/Intel, Xilinx/AMD, Gowin e Lattice. Foi projetado um ambiente robusto e escalável que facilite a detecção precoce de falhas em diversas implementações de processadores RISC-V, garantindo a conformidade com as especificações da ISA. Resultados preliminares demonstram a eficácia da abordagem, destacando a capacidade da metodologia em identificar rapidamente erros e coletar dados essenciais para a seleção e verificação de processadores.*

1. Introdução

A arquitetura de computadores está passando por uma grande revolução, marcada principalmente pelo surgimento de conjuntos de instruções abertos (Instruction Set Architectures - ISAs), hardwares de domínio específico e metodologias ágeis de desenvolvimento [Hennessy and Patterson 2019]. Essas mudanças estão transformando a forma como hardware é projetado e verificado, criando a necessidade de abordagens mais flexíveis e escaláveis para lidar com a crescente complexidade e diversidade dos sistemas modernos.

O advento de ISAs abertas, especialmente com a popularização do RISC-V, estabeleceu um novo paradigma no desenvolvimento de hardware. Ao oferecer um conjunto de instruções livre de *royalties*, o RISC-V abriu as portas para uma ampla variedade de implementações de processadores, cada uma adaptada a diferentes requisitos de desempenho, consumo de energia e custo. Essa flexibilidade, contudo, também introduz novos desafios. Com uma lista oficial que já contabiliza mais de 100 variantes do RISC-V,

*Este trabalho foi parcialmente financiado com recursos dos projetos CNPq 316067/2023-7 e FAPESP 2013/08293-7.

o ecossistema se torna cada vez mais diversificado, com implementações utilizando diferentes linguagens de descrição de hardware, ferramentas e abordagens de projeto, além de serem destinadas a uma gama variada de plataformas como FPGAs (Field Programmable Gate Array), ASICs (Application Specific Integrated Circuits) e ambientes de simulação.

Esse cenário diversificado e em rápida evolução exige novas metodologias de verificação que sejam capazes de validar eficientemente um grande número de implementações. A complexidade aumenta ainda mais quando se considera a necessidade de garantir a compatibilidade e a corretude dessas variantes, muitas das quais são desenvolvidas em ambientes de código aberto, onde as garantias de qualidade podem variar significativamente. Enquanto o processo de verificação anterior envolvia ambientes fechados específicos e distintos para cada ISA, agora, com as múltiplas implementações do mesmo ISA, há a clara necessidade de ambientes compartilhados para promover melhor desempenho e cobertura das implementações.

Diante desse panorama, este artigo propõe uma nova abordagem para a verificação de processadores RISC-V, utilizando uma combinação de técnicas de integração contínua (CI) e uma infraestrutura baseada em FPGAs de múltiplos fornecedores, incluindo Altera/Intel, Xilinx/AMD, Gowin e Lattice. A infraestrutura proposta visa fornecer um ambiente robusto e escalável para a validação automática de múltiplas implementações de processadores, facilitando a detecção precoce de falhas e garantindo a conformidade com as especificações do ISA. Além disso, ao integrar diversas ferramentas de síntese, tanto proprietárias quanto *open-source*, o ambiente permite uma análise comparativa detalhada das implementações, fornecendo informações importantes para o refinamento e otimização dos *designs*.

2. Trabalhos Relacionados

Um dos passos mais críticos no processo de desenvolvimento de circuitos integrados é a fase de verificação, que consome muitos recursos, tanto em tempo quanto em processamento [Rülling 2003]. Quando se trata de processadores, a perspectiva se torna mais complicada pela diferenciação dos mesmos, tanto nas conexões externas quanto internamente (ISA, microarquitetura). Muitas metodologias de teste acabaram se tornando proprietárias e muito específicas da implementação [Jones et al. 1995]. Apesar de esforços na direção de generalização [Sawada 2000, Bertran et al. 2012, Schubert et al. 2018, Bruns et al. 2023], uma nova geração de trabalhos só se tornou visível com o advento de arquiteturas e implementações abertas em larga escala, especialmente proporcionadas pela arquitetura RISC-V [Armstrong et al. 2019, Herdt et al. 2020], bem como geradores auxiliados por IA [Orenes-Vera et al. 2023].

O conjunto de instruções RISC-V [Waterman et al. 2014] foi desenvolvido para suportar a pesquisa e educação em arquitetura de computadores. Algumas características importantes deste ISA são: uma arquitetura desacoplada de implementações específicas, suporte a extensões de ISA de propósitos especializados, espaços de endereço de 32, 64 e 128 bits disponíveis tanto para software quanto para hardware, suporte a virtualização, entre outras características [Cui et al. 2023]. Não há códigos de condição nem *delay slots*. O ISA também suporta implementações de ponto flutuante. O conjunto básico de 32 bits é o ponto de partida para outras expansões, permitindo implementações como ponto flutuante, espaços de endereço de 64 ou 128 bits, codificação de comprimento variável e

versões de código compactado.

Junto com a especificação da arquitetura RISC-V, surgiram formatos e metodologias de verificação relevantes, ganhando suporte da comunidade. A primeira delas é o RISC-V Formal Interface (RVFI) [Joannou et al. 2024]. O RVFI funciona com base no monitoramento de alguns sinais do processador a cada ciclo de *clock*, com esses sinais sendo expostos para fora do *core* através de alguns sinais de saída que podem ser comparados com um modelo de referência. Apesar de ser uma técnica extremamente promissora, ela se mostra invasiva ao modelo de hardware, enquanto nosso objetivo é o desenvolvimento de uma técnica mais simples, com uma quantidade menor de modificações nos sinais do modelo.

Com base nos sinais de saída da interface, resta apenas verificar contra os valores corretos através de outro processador ou de um simulador. A comunidade RISC-V utiliza tanto o simulador `spike` quanto o modelo em SAIL [Armstrong et al. 2019], que são capazes de alimentar ferramentas de verificação formal.

Como resultado, o uso do RVFI implicaria em *patches* ao modelo original do processador, que nem sempre são aceitos pelos desenvolvedores no tempo desejado, para realizar a tarefa de verificação. Nosso projeto procura ser menos invasivo, executando um conjunto de programas, como nos casos anteriores, mas realizando toda a verificação de forma externa ao *core*, como será detalhado posteriormente.

3. Metodologia

A infraestrutura proposta é composta por cinco partes principais: 1) O sistema de integração contínua responsável por gerenciar todo o fluxo de testes e a integração contínua do projeto; 2) As ferramentas de síntese e gravação para FPGAs responsáveis pela síntese e gravação dos designs nas FPGAs; 3) Os módulos de hardware que são encarregados de monitorar e controlar o processador dentro das FPGAs; 4) Os programas de teste; e 5) Os softwares responsáveis pela comunicação e análise dos dados fornecidos pelas FPGAs além de processar as informações coletadas.

3.1. Sistema de Integração Contínua

Para a implementação do processo de integração contínua, foi utilizado o software *Jenkins*, que automatiza o download das implementações diretamente do GitHub do desenvolvedor do processador, além de executar os scripts de síntese, gravação e coleta de dados. Com o *Jenkins*, é possível visualizar o fluxo completo de execução dos testes em um *pipeline*, como ilustrado na Figura 1. Além disso, o *Jenkins* oferece a capacidade de manter um histórico das execuções, permitindo a análise detalhada de falhas e sucessos ao longo do tempo.

3.2. Ferramentas de Síntese e Gravação para FPGAs

A infraestrutura oferece suporte a FPGAs de diferentes fornecedores, permitindo testar a mesma implementação em várias tecnologias e coletar dados estatísticos, como área e frequência máxima para cada uma delas.

Para suportar essas FPGAs, foram utilizadas várias *toolchains*, tanto abertas quanto proprietárias. Em casos onde foi possível, optou-se por usar a *toolchain open*

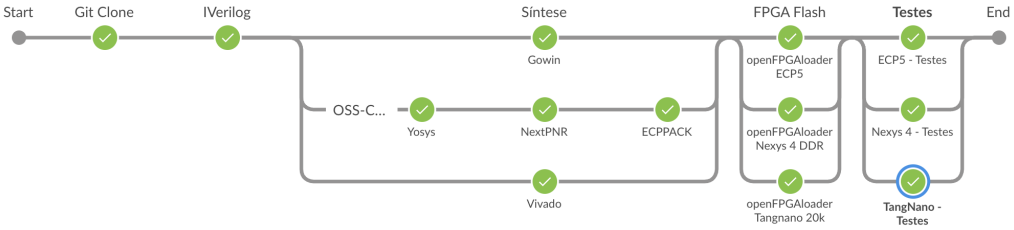


Figura 1. *Pipeline* gerada pelo *Jenkins* com o fluxo de *build* e teste, considerando apenas um único processador, realizando a síntese em três tecnologias distintas com Gowin, Yosys e Vivado como ferramentas.

source YosysHQ, sozinha ou em combinação com toolchains proprietárias. Para as outras FPGAs, foram empregadas as ferramentas fornecidas pelos próprios fabricantes, destacando-se o Quartus para FPGAs Altera/IntelFPGA, o Vivado para FPGAs Xilinx/AMD e o GowinEDA para FPGAs Gowin. A utilização combinada de *toolchains* abertas e fechadas permite também avaliar o estado de qualidade das ferramentas abertas para as múltiplas tecnologias.

3.3. Módulos de Hardware para Monitoramento e Controle do Processador

No momento atual, a funcionalidade de execução de código remotamente está específica para cada processador utilizado e com pouco controle externo, o qual será necessário para os testes de versões avançadas dos processadores. Dessa forma, foram desenvolvidos módulos de hardware auxiliares para controlar o processador e coletar os dados produzidos por ele. Esses módulos facilitam a comunicação e também o controle do sistema, permitindo que o computador realize ações sobre a implementação em FPGA, como alterar valores na memória e controlar sinais de entrada do processador como *clock*, *reset* e *halt*.

Essa infraestrutura de hardware possibilita três abordagens de testes: a execução consecutiva de vários testes carregados na memória, a execução até um ponto de parada (*breakpoint*), e testes manuais, nos quais o computador envia instruções passo a passo, como resetar o processador e controlar ciclos de *clock*. O diagrama de blocos com os principais módulos pode ser visto na Figura 2.

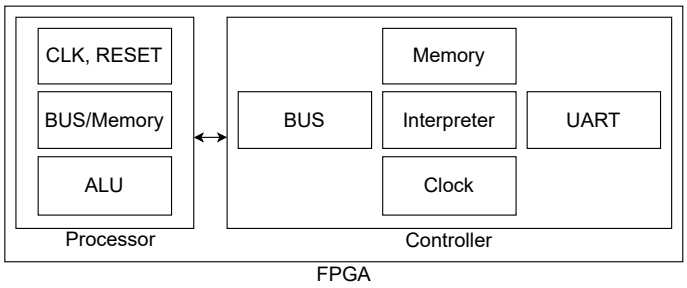


Figura 2. Diagrama de blocos de ligação entre o controlador e o processador.

3.4. Programas de testes

Para validar a implementação do processador, é necessário um conjunto de testes escritos em linguagem de máquina. Na fase atual do projeto, foi priorizada a utilização de

conjuntos de testes já disponíveis em outros projetos, e geradores de testes aleatórios como o *RISC-V DV* e o *RISC-V Torture*. Em fases futuras, planeja-se desenvolver testes específicos para ampliar a cobertura do conjunto de instruções e suas extensões.

3.5. Softwares para Comunicação e Análise de Dados

Um protocolo de comunicação foi desenvolvido para permitir a interação entre o hardware auxiliar dentro da FPGA e a máquina *host*. Esse protocolo consiste em um conjunto de comandos (instruções) que possibilitam à máquina *host* enviar ordens, além de transmitir e solicitar informações ao controlador na FPGA.

Para utilizar esse protocolo, é necessário um software na máquina *host* que o interprete e envie os testes ao hardware na FPGA. Esse software também é responsável por analisar e comparar os dados coletados, além de informar ao *Jenkins* se o processador produziu o resultado esperado. No momento, existe uma versão preliminar com as primeiras funcionalidades já implementadas.

3.6. Visão geral da Infraestrutura

Fisicamente, a infraestrutura atual consiste de um computador com 3 modelos de FPGAs conectadas via interface USB, que fornece acesso a uma interface JTAG e a uma porta serial UART, conforme ilustrado no diagrama da Figura 3. Em versões futuras, planeja-se adicionar a possibilidade de comunicação via PCIe e SPI, aumentando a velocidade de comunicação entre a máquina *host* e a FPGA.

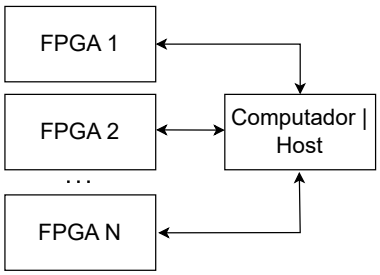


Figura 3. Diagrama de blocos da conexão das FPGAs à máquina *host*.

O fluxo de verificação começa com um modelo de processador armazenado em um repositório *git* que, ao receber um commit dos desenvolvedores, dispara um evento no *Jenkins*. Esse evento aciona múltiplos *pipelines* de síntese para diferentes FPGAs. À medida que as sínteses são concluídas, os *bitstreams* resultantes são carregados nas respectivas FPGAs e os testes são iniciados. Com a conclusão dos testes, obtém-se um atestado de verificação para cada uma das tecnologias suportadas, garantindo que os designs são compatíveis com as plataformas testadas.

Na Figura 4, uma visão geral da infraestrutura é apresentada, mostrando os módulos de hardware auxiliares, o processador dentro da FPGA, e todos os softwares em execução na máquina *host*.

4. Resultados

Como resultado atual, foram integrados 15 processadores à infraestrutura que realiza o processo de *build* de forma totalmente automatizada. Esse processo nos fornece um

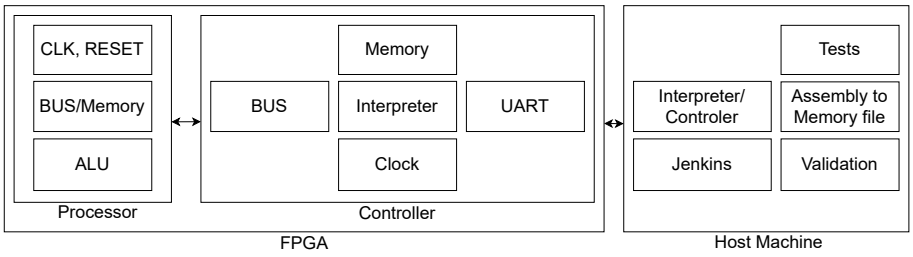


Figura 4. Diagrama de blocos dos módulos na FPGA e na máquina *Host*.

histórico detalhado de *builds* desses processadores, permitindo a detecção rápida de erros introduzidos por modificações no design da implementação. A Figura 5 ilustra esse histórico de *builds* de um processador, destacando os pontos em que falhas foram identificadas.

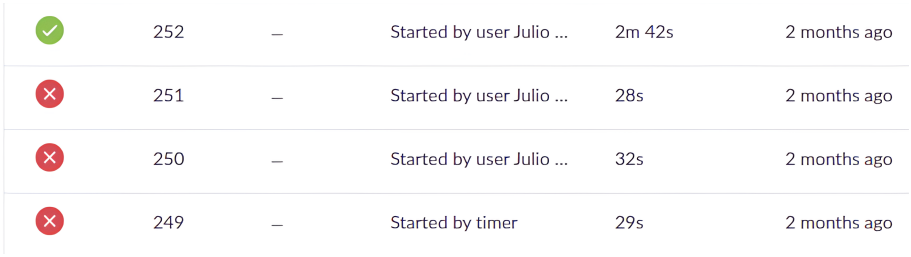
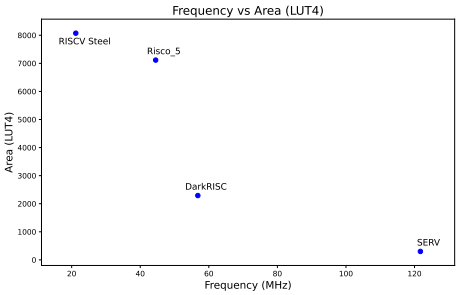


Figura 5. Parte do histórico de *build* de um processador no *Jenkins*.

Além da detecção de falhas, o processo de síntese em diferentes ferramentas aumenta a precisão da verificação, utilizando distintas características de frequência e área, permitindo explorar pontos diferentes do espaço de projeto. A Figura 6a mostra um comparativo entre frequência (MHz) e área (LUT4) para várias implementações em uma FPGA Lattice ECP5, permitindo também a escolha de uma implementação conforme os requisitos de projeto desejados.



(a) Frequência vs LUT4 na placa FPGA Colorlight i9.

Name	Links	Status	Full Log
Risco 5	Github,Website	build passing	Log
DarkRISCV	Github	build failing	Log
MRISCV	Github	build failing	Log
NeoRV32 Verilog	Github	build failing	Log
Pequeno Risco 5	Github	build passing	Log
RISC-V Steel	Github,Website	build passing	Log
SERV	Github	build passing	Log

(b) Trecho da página do site contendo as informações de *build*.

Figura 6. Resultados do ambiente de integração contínua.

Toda a infraestrutura está disponível publicamente, com o código-fonte sendo disponibilizado no GitHub sob uma licença aberta. Os desenvolvedores têm acesso completo aos *logs* e resultados de *builds* e testes, além de poderem incorporar um selo com os resul-

tados do processo em seus repositórios. A documentação do projeto e os dados gerados pela infraestrutura também estão disponíveis em um site público[1], conforme Figura 6b.

5. Discussão

Os resultados obtidos até o momento demonstram a eficácia da infraestrutura e metodologia proposta. O uso de técnicas de integração contínua permite uma rápida detecção de falhas e uma escalabilidade de forma mais simples. Os dados coletados sobre as implementações podem ser úteis para a construção de um seletor de processadores, permitindo comparações do grau de verificação, área, frequência, tecnologias compatíveis, desempenho, etc.

A recente descoberta da vulnerabilidade *GhostWrite*, que afeta alguns processadores RISC-V, ilustra como a utilização de técnicas de integração contínua podem ser úteis para detecção de falhas. Esta vulnerabilidade permite que atacantes não privilegiados leiam e escrevam em qualquer parte da memória e controlem dispositivos periféricos, comprometendo a segurança do sistema [Thomas et al. 2024]. Com uma infraestrutura de verificação em escala, uma vez que a falha é identificada, ela pode ser verificada em todas as implementações suportadas pela infraestrutura. Além disso, ao executar os mesmos testes em diversas implementações, é possível observar diferenças nos resultados. Como todas as implementações seguem o mesmo ISA, qualquer variação nos resultados pode indicar uma falha de implementação.

Apesar desses resultados preliminares positivos, atualmente destacamos as duas principais limitações: 1) O conjunto de *software* em execução nos processadores precisa ser ampliado significativamente e categorizado para facilitar a identificação de futuros *bugs*; 2) Ampliação do fluxo de verificação de todos os processadores para todas as FPGAs disponíveis, incluindo outras que ainda não estão sendo utilizadas no ambiente.

6. Conclusão

A infraestrutura e a metodologia desenvolvidas já permite realizar testes básicos de 15 modelos de processadores RISC-V em 3 tipos de FPGAs e o processo de verificação é ativado a cada vez que o repositório de um dos projetos é atualizado através dos mecanismos de integração contínua do *Jenkins*. Esse suporte inicial a diversos processadores e FPGAs ilustra a escalabilidade da infraestrutura, permitindo sua expansão para novos modelos e configurações.

Pretende-se ampliar a infraestrutura em todos os eixos já desenvolvidos com a inclusão de 1) novos *cores*; 2) novos programas de testes; 3) novas FPGAs; 4) novos módulos de hardware para auxiliar a verificação; 5) novas interfaces com ambientes de desenvolvimento para publicar os testes. Além dessas possibilidades de expansão, é igualmente importante ressaltar que a infraestrutura possui potencial para incorporar novas técnicas de verificação, como a co-simulação [Bruns et al. 2023], permitindo, no futuro, a aplicação de abordagens distintas para a verificação de partes específicas do processador.

Futuramente, pretende-se também desenvolver o seletor de *cores*, utilizando os parâmetros capturados pelo processo de verificação de forma similar ao mostrado na Figura 6a.

[1]Disponível em: https://processorci.ic.unicamp.br

Referências

Armstrong, A., Bauereiss, T., Campbell, B., Reid, A., Gray, K. E., Norton, R. M., Mundkur, P., Wassell, M., French, J., Pulte, C., Flur, S., Stark, I., Krishnaswami, N., and Sewell, P. (2019). ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS. *Proc. ACM Program. Lang.*, 3(POPL).

Bertran, R., Buyuktosunoglu, A., Gupta, M. S., Gonzalez, M., and Bose, P. (2012). Systematic Energy Characterization of CMP/SMT Processor Systems via Automated Micro-Benchmarks. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 199–211.

Bruns, N., Herdt, V., and Drechsler, R. (2023). Processor Verification using Symbolic Execution: A RISC-V Case-Study. In *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6.

Cui, E., Li, T., and Wei, Q. (2023). RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Access*, 11:24696–24711.

Hennessy, J. L. and Patterson, D. A. (2019). A new golden age for computer architecture. *Commun. ACM*, 62(2):48–60.

Herdt, V., Große, D., Jentzsch, E., and Drechsler, R. (2020). Efficient Cross-Level Testing for Processor Verification: A RISC- V Case-Study. In *2020 Forum for Specification and Design Languages (FDL)*, pages 1–7.

Joannou, A., Rugg, P., Woodruff, J., Fuchs, F. A., van der Maas, M., Naylor, M., Roe, M., Watson, R. N. M., Neumann, P. G., and Moore, S. W. (2024). Randomized Testing of RISC-V CPUs Using Direct Instruction Injection. *IEEE Design Test*, 41(1):40–49.

Jones, R. B., Dill, D. L., and Burch, J. R. (1995). Efficient validity checking for processor verification. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '95, page 2–6, USA. IEEE Computer Society.

Orenes-Vera, M., Martonosi, M., and Wentzlaff, D. (2023). From RTL to SVA: LLM-assisted generation of Formal Verification Testbenches.

Rülling, W. (2003). *Circuit Verification*, pages 210–218. Springer US, Boston, MA.

Sawada, J. (2000). Processor Verification with Precise Exceptions and Speculative Execution.

Schubert, K.-D., Abrar, S. S., Averill, D., Bauman, E., Brown, A. C., Cash, R., Chatterjee, D., Gullickson, J., Nelson, M., Pasnik, K. A., and Sugavanam, K. (2018). Addressing verification challenges of heterogeneous systems based on IBM POWER9. *IBM Journal of Research and Development*, 62(4/5):11:1–11:12.

Thomas, F., Hetterich, L., Zhang, R., Weber, D., Gerlach, L., and Schwarz, M. (2024). RISCVuzz: Discovering Architectural CPU Vulnerabilities via Differential Hardware Fuzzing. `https://ghostwriteattack.com/`.

Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2014). The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley.