Processor CI Inspector: Detectando parâmetros automaticamente em processadores RISC-V *

Gabriel Oliveira [1], Julio N. Avelar[1], Enzo P. Bertoloti[1], Gabriel Gomes[1], Rodolfo Azevedo[1]

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

{g247700, j241163, e248361, g212736}@dac.unicamp.br,
rodolfo@ic.unicamp.br

Abstract. *This paper presents a tool, to be integrated into a continuous integration environment, designed to automatically detect parameters of RISC-V processors. By leveraging open-source software such as hardware description language simulators (e.g., Icarus Verilog and Verilator) and cosimulation frameworks (e.g., Cocotb), a scalable application was developed to support the selection among different CPU implementations. Preliminary results confirmed the effectiveness of the tool, highlighting the methodology's capability to identify essential features for processor selection in diverse contexts. This tool was able to correctly detect 6 parameters of 9 distinct processor implementations.*

Resumo. *Este artigo apresenta uma ferramenta, a ser incorporada em um ambiente de integração contínua, que visa, automaticamente, detectar parâmetros de processadores RISC-V. Utilizando-se de softwares open-source como simuladores de linguagem de descrição de hardware (e.g. Icarus Verilog e Verilator) e frameworks de cossimulação (e.g. Cocotb) foi projetada uma aplicação escalável que auxilia na escolha entre diferentes implementações de CPUs. Resultados preliminares constataram a eficácia da ferramenta, realçando a capacidade da metodologia em identificar características essenciais à seleção de processadores em diferentes contextos. Foi possível detectar 6 parâmetros de configuração de 9 implementações abertas distintas com sucesso.*

1. Introdução

O setor de arquitetura de computadores tem testemunhado um crescimento significativo no desenvolvimento de circuitos integrados com maior complexidade e especificações rigorosas. Nesse cenário, em que hardwares de domínio específico (Application-Specific Integrated Circuits - ASICs) e sistemas em chip (System on Chips - SoCs) tornaram-se predominantes, é crucial haver mudanças na forma como processadores são projetados, avaliados e selecionados para diferentes aplicações.

A popularização de conjuntos de instruções abertos (Instruction Set Architectures - ISAs), como o RISC-V [Waterman et al. 2014], veio acompanhada de novas normas e metodologias de desenvolvimento de hardware. Em um ecossistema dominado por

*Este trabalho foi parcialmente financiado com recursos dos projetos CNPq 3160607/2023-7, CNPq 148337/2025-2 e FAPESP 2013/08293-7.

empresas como AMD, Intel e ARM, com seus respectivos ISAs proprietários, o RISC-V destaca-se por sua natureza aberta e livre de royalties, permitindo que desenvolvedores e pesquisadores tenham acesso irrestrito às especificações do conjunto de instruções. Diante desse cenário, diversas implementações de processadores RISC-V começaram a surgir (com centenas de modelos disponíveis na página oficial da RISC-V Foundation [International 2025b]), cada uma com características distintas, como diferentes níveis de desempenho, consumo de energia, área ocupada no silício, entre outros.

Perante esse panorama, a seleção do processador mais adequado para uma aplicação específica tornou-se um desafio complexo que exige uma análise detalhada de múltiplos parâmetros. Tradicionalmente, essa seleção é realizada por meio de avaliações manuais, que podem ser demoradas e propensas a erros, especialmente quando se lida com um grande número de opções. Além disso, a falta de padronização na documentação e na apresentação das características dos processadores RISC-V dificulta ainda mais esse processo, o que demonstra a necessidade de ferramentas automatizadas que possam auxiliar na análise e comparação dessas implementações.

Diante desse contexto, este artigo apresenta o desenvolvimento de uma ferramenta que automatiza a detecção de parâmetros essenciais de processadores RISC-V (e.g. licença, tamanho de palavra, presença de pipeline etc), integrando-se ao ambiente de integração contínua (Continuous Integration - CI), Processor CI[Avelar et al. 2024]. A ferramenta utiliza simuladores de hardware *open-source*, como Icarus Verilog e Verilator, juntamente com *frameworks* de cossimulação, como Cocotb [coc 2025], para analisar o comportamento dos processadores em diferentes cenários. O objetivo é fornecer uma solução escalável e eficiente que facilite a seleção do processador mais adequado para diversas aplicações, reduzindo o tempo e o esforço necessários para essa tarefa.

2. Trabalhos Relacionados

Existe uma imensidão de trabalhos na literatura que abordam a problemática da verificação e validação de processadores [Sawada 2000, Schubert et al. 2018], contudo poucos focam no contexto da arquitetura RISC-V [Herdt et al. 2020, Joannou et al. 2024]. Nesse contexto, surgiram iniciativas que buscam padronizar e facilitar o desenvolvimento e a verificação de processadores *open-source* como *frameworks* e infraestruturas de integração contínua (CI) [Deutschbein and Stassinopoulos 2025]. Dentre essas iniciativas, destaca-se o Processor CI, uma infraestrutura que automatiza a verificação funcional de processadores RISC-V *open-source*. O Processor CI utiliza uma abordagem baseada em diversas metodologias de verificação diferentes, incluindo simulação e testes em FPGA, para garantir a conformidade dos processadores com as especificações do conjunto de instruções RISC-V. A infraestrutura é composta por uma série de módulos que permitem a execução automática de testes funcionais, análise de cobertura e geração de relatórios detalhados sobre o desempenho e a conformidade dos processadores testados.

Nesse contexto de verificação de processadores, vê-se a necessidade de detectar parâmetros de forma automática desses hardwares a fim de facilitar a escolha de metodologias de testes para cada implementação. Embora existam iniciativas que auxiliem a análise desses *cores* RISC-V, como o RISC-V Compliance Suite [Lee Moore 2019], que verifica a conformidade dos processadores com as especificações do conjunto de instruções, não há ferramentas que automatizem a detecção de parâmetros essenciais para a

seleção de processadores.

Com base nesse cenário, técnicas de análise estática e dinâmica de código-fonte têm sido exploradas para extrair informações relevantes sobre o comportamento e as características desses circuitos escritos em HDL [Lee et al. 2020]. A análise estática envolve a inspeção do código-fonte sem a necessidade de execução, permitindo a identificação de padrões e propriedades específicas. Já a análise dinâmica, por meio de simulação e cossimulação, possibilita a observação do comportamento do processador em tempo real, fornecendo informações valiosas sobre seu desempenho e funcionalidade.

Na análise dinâmica, o uso de *frameworks* de cossimulação, como o Cocotb, fundamenta a capacidade de monitorar e controlar o processador em teste de forma padronizada, facilitando a execução de testes automatizados e contornando um recorrente problema na literatura: a falta de padronização na interface dos processadores. Essa abordagem permite a criação de testes em Python baseados em técnicas de *pipelining* automático [Marinescu and Rinard 2001, Cortadella et al. 2015], que podem ser adaptados para diferentes implementações de processadores RISC-V.

Diante disso, este trabalho propõe uma ferramenta que integra essas técnicas de análise estática e dinâmica, utilizando simuladores de hardware *open-source* e *frameworks* de cossimulação, para automatizar a detecção de parâmetros essenciais de processadores RISC-V. A automatização desse processo é essencial para permitir a escalabilidade na análise e seleção de processadores, uma vez que, com o crescimento do número de implementações disponíveis no Processor CI, surge uma expectativa da ferramenta possibilitar uma rápida comparação entre diferentes versões de um mesmo processador, que seria inviável se fosse necessário o preenchimento manual desses parâmetros. A ferramenta visa uma metodologia escalável e não intrusiva, que possa ser facilmente integrada ao ambiente de integração contínua do Processor CI, facilitando a seleção do processador mais adequado para diferentes aplicações.

3. Metodologia

A Figura 1 ilustra o fluxo de trabalho da ferramenta, que contém duas etapas principais:

1. Análise estática do código-fonte do processador;
2. Cossimulação dinâmica do processador com um conjunto de testes padronizados.

3.1. Análise Estática

Para a análise estática, foram desenvolvidos *scripts* em Python que examinam o código-fonte do processador antes de sua simulação. Esses programas realizam uma análise léxica e sintática dos arquivos do repositório, buscando, pelo uso de expressões regulares, trechos de texto que possam identificar o tipo de licença utilizada (e.g. Apache 2.0, GPL3, etc.). Para a determinação da linguagem de descrição de hardware (HDL) utilizada, o *script* verifica os arquivos necessários para a simulação e identifica a quantidade de *tokens* associados a cada linguagem (e.g. Verilog, VHDL, SystemVerilog, etc.). A linguagem com o maior número de *tokens* é então selecionada como a linguagem predominante do projeto. Essa análise é possibilitada por meio de um arquivo de configuração fornecido pela infraestrutura do Processor CI, que lista os arquivos relevantes para a simulação de cada processador, o que permite um fluxo semi-automático de todo fluxo de trabalho (com o único passo manual sendo a ligação do módulo de monitoramento ao processador em teste).

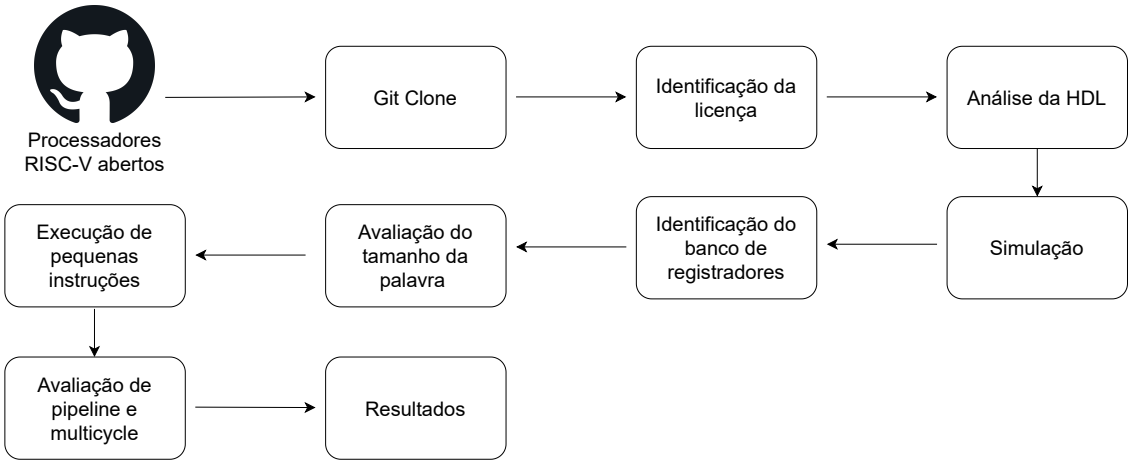

Figura 1. Fluxograma exemplificando os passos realizados pelo Processor CI Inspector

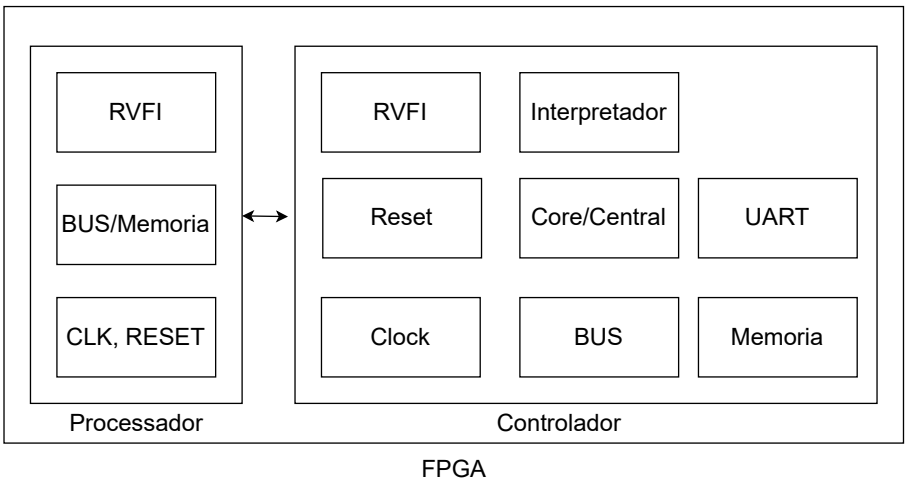

Figura 2. Diagrama da conexão entre o controlador e o processador

3.2. Cossimulação Dinâmica

A segunda etapa, que consiste na cossimulação dinâmica do processador, é realizada utilizando o *framework* Cocotb, que permite a criação de testes em Python para módulos descritos em HDLs como Verilog e VHDL. A ferramenta desenvolvida faz uso desse sistema através de um módulo de hardware fornecido pela infraestrutura do Processor CI que é ligado ao processador em teste, permitindo seu monitoramento e controle de forma padronizada. O esquema de conexão entre o processador e o módulo de monitoramento é ilustrado na Figura 2.

A cossimulação é conduzida nos seguintes passos:

1. Localização do banco de registradores do processador, que é essencial para a execução e validação dos testes, através da análise dos sinais do processador. Essa identificação é realizada pela análise dos sinais do processador, buscando por arrays de registradores que correspondam ao tamanho esperado (e.g. 32 registradores para RISC-V);

2. Avaliação do tamanho do banco de registradores, que pode indicar o tamanho da palavra do processador (e.g. 32 bits, 64 bits, etc.) com base na quantidade de bits de cada registrador;
3. Execução de um pequeno conjunto de testes padronizados, que incluem instruções aritméticas, lógicas e de controle de fluxo, para observar o comportamento do processador;
4. Observação de sinais específicos, como sinais de *clock* e PC (Program Counter), para determinar a presença e a profundidade do pipeline, bem como a existência de *multicycle*. A presença de pipeline é inferida pela análise do comportamento do sinal de PC em relação ao *clock*, enquanto a profundidade do pipeline é estimada observando o número de ciclos necessários para completar uma instrução, observando o atraso entre a emissão de uma instrução e a escrita do resultado no banco de registradores. A detecção de *multicycle* é realizada verificando se o processador leva múltiplos ciclos de *clock* para completar certas instruções.

4. Resultados

Resultados iniciais foram obtidos a partir da análise de 9 processadores RISC-V *open-source*, que foram selecionados por sua diversidade em termos de arquitetura (e.g. pipeline, *multicycle*, etc.), complexidade e popularidade na comunidade (e.g. Tinyriscv com mais de 1.3k estrelas no Github).A ferramenta desenvolvida foi capaz de detectar automaticamente os seguintes parâmetros:

Tipo de licença: Identificação dos tipos de licença utilizados no projeto do processador (e.g. Apache 2.0, GPL3, etc.);

Tamanho da palavra: Determinação do tamanho da palavra do processador (e.g. 32 bits, 64 bits, etc.) com base no tamanho do banco de registradores;

Implementação pipeline: Verificação se o processador possui um pipeline implementado;

Profundidade do pipeline: Estimativa da profundidade do pipeline, caso presente;

Implementação *multicycle*: Verificação se o processador utiliza um design de *multicycle* ao realizar operações;

Linguagem de descrição de hardware: Identificação da linguagem HDL predominante utilizada no projeto (e.g. Verilog, VHDL, etc.).

Dentre os processadores analisados, destacam-se modelos como AUK-V-Aethia, Picorv32, Risco-5, RVX, SparrowRV e Tinyriscv. A Tabela 1 apresenta um resumo dos resultados obtidos para cada processador analisado.

A ferramenta foi integrada ao ambiente de integração contínua (CI) do Processor CI, permitindo a execução automática dos testes sempre que um novo processador é adicionado à infraestrutura. Isso garantiu que os resultados fossem atualizados constantemente, facilitando a comparação entre diferentes implementações. O formato de saída dos resultados foi padronizado em JSON, conforme ilustrado na Figura 3, facilitando a leitura e interpretação dos dados por outras ferramentas ou *scripts*. É importante ressaltar que o JSON ilustrado pode ser ampliado para incluir parâmetros que ainda não são detectados pela ferramenta, como tipo de barramento, cache, ISA, superescalar, entre outros.

O código-fonte e a documentação da ferramenta desenvolvida estão disponíveis publicamente no Github sob uma licença aberta, permitindo que outros pesquisadores

```json
{
    "tinyriscv":  {
        "license_types":  [
            "Custom License",
            "Apache 2.0"
        ],
        "bits":  32,
        "language":  "Verilog",
        "multicycle":  False,
        "pipeline":  True,
        "pipeline_depth": 5,
    }
}
```

Figura 3. Exemplo de *json* com os parâmetros a serem detectados

e desenvolvedores possam contribuir para seu aprimoramento e adaptação a diferentes necessidades.

Processador	Licença	Bits	Pipeline	Profundidade	Multicycle	Linguagem
AUK-V-Aethia	GPLv3	32	Sim	5	Não	Verilog
Grande Risco-5	CERN-OHL-P-2.0	32	Sim	5	Não	SystemVerilog
Hornet	MIT	32	Sim	5	Não	Verilog
Kronos	Apache 2.0	32	Sim	3	Não	SystemVerilog
Picorv32	ISC	32	Não	-	Sim	Verilog
Risco-5	CERN-OHL-P-2.0	32	Não	-	Sim	Verilog
RV3N	Apache 2.0	32	Sim	8 [1]	Não	Verilog
RVX	MIT	32	Não	-	Sim	Verilog
Tinyriscv	Apache 2.0	32	Sim	5	Não	Verilog

Tabela 1. Resultados obtidos para os processadores analisados.

5. Discussão

Os resultados adquiridos com a ferramenta desenvolvida foram promissores, demonstrando a eficácia da abordagem proposta. O uso de *scripts* em Python, aliados a simuladores de hardware *open-source* e *frameworks* de cossimulação, permitiu a análise automática de diversos processadores RISC-V, identificando parâmetros essenciais de forma rápida e precisa ao serem integrados ao ambiente de integração contínua (CI) do Processor CI.

Com a expansão de processadores RISC-V no mercado, a exemplo da NVIDIA produzindo um bilhão de núcleos RISC-V no ano de 2024 [International 2025a], o Processor CI Inspector se mostra uma ferramenta relevante para desenvolvedores e pesquisadores que buscam otimizar a seleção de processadores para suas aplicações específicas. A capacidade de detectar automaticamente características como tipo de licença, tamanho

[1] O RV3N possui pipeline de tamanho configurável, esses resultados foram obtidos com a configuração padrão

da palavra, presença e profundidade do pipeline, entre outros, facilita a comparação entre diferentes implementações, economizando tempo e esforço.

Embora os resultados preliminares tenham sido positivos, é importante reconhecer as limitações atuais da ferramenta: 1) A análise foi realizada em um número limitado de processadores (9 modelos); 2) A ferramenta atualmente detecta apenas um conjunto restrito de parâmetros; 3) A precisão da detecção automática pode variar dependendo da complexidade do processador analisado e dos componentes externos ao core. Pretende-se aprimorar a ferramenta para endereçar esses desafios em trabalhos futuros.

6. Conclusão

A ferramenta desenvolvida demonstrou ser eficaz na detecção automática de parâmetros essenciais de processadores RISC-V, com 9 modelos analisados com sucesso. A integração com o ambiente de integração contínua (CI) também mostrou-se viável, permitindo uma análise rápida e eficiente das CPUs testadas. Essa implementação inicial destaca a escalabilidade da ferramenta, que pode ser expandida para suportar um número maior de processadores e parâmetros adicionais.

Planeja-se, como trabalhos futuros, a ampliação da ferramenta desenvolvida para incluir a análise de mais *cores* RISC-V, bem como a incorporação de novos parâmetros que possam ser relevantes para diferentes aplicações (e.g. cache, ISA, superescalar, etc.).Além disso, busca-se a implementação de técnicas de aprendizado de máquina para aprimorar a precisão e a eficiência da detecção automática de características dos processadores que não são facilmente mensuráveis por meio de simulação (e.g. configurabilidade, tipo de barramento, etc.). Acredita-se que essas melhorias contribuirão significativamente para a seleção otimizada de processadores RISC-V em diversos contextos, atendendo às necessidades específicas de diferentes aplicações.

Referências

(2025). Cocotb - an open source coroutine-based cosimulation testbench environment for verifying vhdl and systemverilog rtl using python.

Avelar, J., Lago, V., Ângelo Malaguti, and Azevedo, R. (2024). Processorci: Integração contínua para processadores risc-v em fpgas. In *Anais Estendidos do XXV Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 1–8, Porto Alegre, RS, Brasil. SBC.

Cortadella, J., Galceran-Oms, M., Kishinevsky, M., and Sapatnekar, S. S. (2015). Rtl synthesis: From logic synthesis to automatic pipelining. *Proceedings of the IEEE*, 103(11):2061–2075.

Deutschbein, C. and Stassinopoulos, A. (2025). "test, build, deploy-- a ci/cd framework for open-source hardware designs.

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.

International, R.-V. (2025a). How nvidia shipped one billion risc-v cores in 2024. https://riscv.org/blog/2025/02/how-nvidia-shipped-one-billion-risc-v-cores-in-2024/.

International, R.-V. (2025b). Risc-v international. https://riscv.org/.

Joannou, A., Rugg, P., Woodruff, J., Fuchs, F., Van Der Maas, M., Naylor, M., Roe, M., Watson, R., Neumann, P., and Moore, S. (2024). Randomized testing of risc-v cpus using direct instruction injection.

Lee, J., Nie, P., Li, J. J., and Gligoric, M. (2020). On the naturalness of hardware descriptions. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 530–542, New York, NY, USA. Association for Computing Machinery.

Lee Moore, I. S. (2019). Compliance methodology and initial results for risc-v isa implementations.

Marinescu, M.-C. and Rinard, M. (2001). High-level automatic pipelining for sequential circuits. In *International Symposium on System Synthesis (IEEE Cat. No.01EX526)*, pages 215–220.

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.

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.