Shell interativo com carregador para RISC-V em FPGA

Luiz Carlos da S. N. Vartuli[1], **Marcus Vinicius Lamar**[1], **Alba Cristina M. A. de Melo**[1]

[1]Departamento de Ciência da Computação
Universidade de Brasília (UnB) – Brasília – DF – Brazil

luizcsnvartuli@gmail.com, {lamar,alves}@unb.br

Resumo. *Este trabalho apresenta o desenvolvimento de um shell para um processador RISC-V implementado em FPGA com memória extremamente limitada. O sistema permite o carregamento e execução de programas via interface serial RS232, substituindo o uso do In-System Memory Content Editor do Intel Quartus® Prime. A interface textual segue o modelo REPL e inclui comandos como* `echo`*,* `clear`*,* `help`*,* `exit` *e* `exec`*, que carrega arquivos binários diretamente na memória. Os testes confirmaram a funcionalidade do shell, incluindo tratamento de erros, e compatibilidade com o RISC-V Assembler and Runtime Simulator (RARS).*

Abstract. *This work presents the development of a shell for a RISC-V processor implemented on an FPGA with extremely limited memory. The system enables loading and executing programs via an RS232 serial interface, replacing the use of the Intel Quartus® Prime In-System Memory Content Editor. The text interface follows the REPL model and includes commands such as* `echo`*,* `clear`*,* `help`*,* `exit`*, and* `exec`*, which loads binary files directly into memory. Tests confirmed the shell's functionality, including error handling and compatibility with the RISC-V Assembler and Runtime Simulator (RARS).*

1. Introdução

FPGAs (*Field-Programmable Gate Arrays*) são circuitos integrados reconfiguráveis amplamente utilizados no ensino e pesquisa em arquitetura de computadores, permitindo projetar e testar processadores desde conceitos básicos de microarquitetura até sistemas completos. Nos projetos do laboratório de arquitetura de computadores da Universidade de Brasília (UnB) utilizam-se FPGAs do modelo DE1-SoC [Terasic Technologies 2025], que possibilitam a implementação prática de processadores idealizados por professores e estudantes. Entre esses projetos destaca-se o processador baseado na arquitetura RISC-V, uma ISA (*Instruction Set Architecture*) aberta e extensível que tem servido como base para experimentos de execução de programas diretamente no FPGA e se destaca no contexto da soberania computacional em nível nacional [Ministério da Ciência, Tecnologia e Inovação 2024] e internacional [Pelé 2025]. A versão utilizada neste trabalho é a ISA RV32IMF multiciclo.

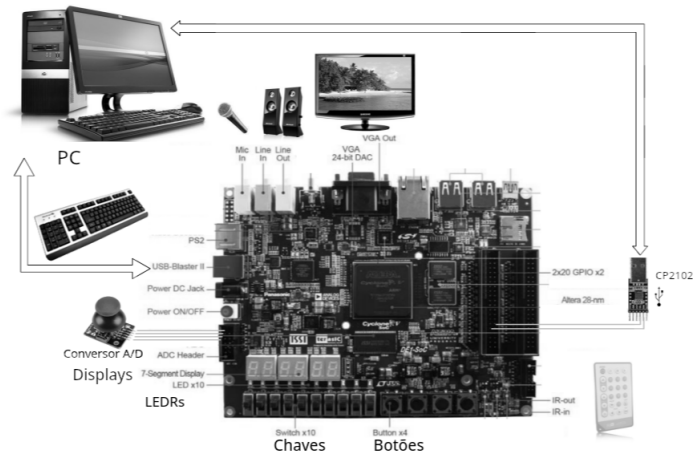

Figura 1. Ambiente de laboratório da disciplina de OAC

O kit DE1-SoC oferece diversos recursos de hardware integrados, como saída VGA, entrada para teclado PS/2, saída de áudio analógico P2, conversor analógico-digital e interface serial RS232 para comunicação entre o processador RISC-V e um computador externo por meio de um conversor USB–RS232 (CP2102). Também está disponível um sistema operacional mínimo com chamadas de sistema básicas, como entrada pelo teclado e saída de texto no monitor. Entretanto, o ambiente apresenta restrições de memória (64 KiB para código e 128 KiB para dados) que dificultam a implementação de funcionalidades mais complexas, exigindo soluções otimizadas.

O carregamento atual de programas no processador é realizado manualmente por meio da ferramenta *In-System Memory Content Editor* do Intel Quartus® Prime, em um processo repetitivo, pouco intuitivo e sujeito a erros. Este trabalho propõe um *shell* simples, executado diretamente no processador RISC-V implementado no FPGA, que se comunica com um computador com Windows por meio da conexão USB-RS232 citada acima, possibilitando o carregamento interativo de arquivos binários para as memórias de instruções e dados. O sistema foi concebido para substituir o fluxo tradicional por uma alternativa mais flexível e automatizada, contemplando o desenvolvimento de um *shell* funcional capaz de ler e executar comandos localmente, a implementação de chamadas de sistema para interação com arquivos armazenados no computador e a criação de um carregador que interprete os arquivos recebidos via comunicação serial e carregue o programa corretamente na memória do FPGA.

2. Trabalhos Relacionados

Diversas iniciativas exploram processadores RISC-V implementados em FPGA para fins educacionais e experimentais, especialmente em ambientes com recursos limitados. Alguns projetos compartilham elementos com esta proposta, como o uso de comunicação serial e simplificação dos fluxos de carregamento de programas.

O IOb-SoC [Fernandes 2021] implementa um SoC RISC-V open-source baseado no núcleo PicoRV32, com memória SRAM interna e comunicação UART serial, semelhante à interação serial entre PC e FPGA neste trabalho. De modo parecido, o SoC HULK-V [Valente et al. 2023] utiliza RISC-V para soluções eficientes em IoT, substituindo controladores DDR por HyperRAMs para reduzir consumo, enquanto nosso shell

automatiza o carregamento manual via In-System Memory Editor por uma interface serial REPL. Ambos focam em otimização de recursos escassos e interfaces simplificadas, demonstrando a versatilidade do RISC-V em contextos restritos.

Esta proposta integra esse cenário, destacando-se pela simplicidade e economia de memória, com um interpretador leve executado diretamente no processador embarcado. Assim, permite interações interativas via REPL e carregamento direto de arquivos binários, mantendo-se adequada a plataformas limitadas como a DE1-SoC.

3. Metodologia

Esta seção descreve os métodos e estratégias empregados no desenvolvimento do sistema proposto. O ambiente de interação com o usuário é realizado por meio de um monitor externo conectado ao FPGA, no qual é exibida a interface do *shell*. O PC participa apenas como um auxiliar para o acesso a arquivos, sendo utilizado por meio de um programa em linguagem C que se comunica via RS232 com o FPGA e atende às chamadas de sistema solicitadas pelo processador.

3.1. Shell

O *shell* implementado neste trabalho fornece uma interface interativa básica, permitindo a leitura de comandos digitados pelo usuário e sua execução imediata, no estilo REPL (*Read-Eval-Print Loop*). A leitura da entrada é feita caractere por caractere, com suporte a edição simples (como uso de *backspace*) e término com a tecla Enter. Essa funcionalidade foi encapsulada em uma chamada de sistema, de modo a facilitar sua reutilização por outros alunos.

A identificação dos comandos ocorre por meio de uma verificação de prefixo, dispensando a necessidade de um *parser* completo, dado o conjunto reduzido e simples de instruções suportadas. Foram implementados os comandos `echo`, `clear`, `exit`, `help` e `exec`, permitindo desde a exibição de mensagens até a execução de programas externos. Além disso, o *shell* conta com uma rolagem automática da tela, que desloca o conteúdo duas linhas para cima ao atingir o final da área visível.

3.2. Sistema de Arquivos

3.2.1. Comunicação RS232

A comunicação serial no PC foi implementada em C com a biblioteca de Teuniz van Beelen[1], que fornece funções básicas como `PollComport`, `SendByte` e `SendBuf`. Para atender às necessidades do projeto, foram criadas rotinas adicionais de leitura blocante (`ReadByte`, `ReadInt`, `ReadString`, `ReadBuf`) e envio de inteiros (`SendInt`). A função `ReadByte` é a única que acessa diretamente a biblioteca, usando `PollComport`, enquanto `SendInt` se baseia em `SendBuf`. Todos os inteiros são transmitidos no formato *big-endian*.

No FPGA, a comunicação RS232 foi escrita em *Assembly* RISC-V, utilizando uma interface mapeada em memória. Foram implementadas funções básicas de leitura e escrita de bytes (`ReadByte`, `SendByte`), que servem de base para rotinas mais elaboradas,

[1] `https://www.teuniz.net/RS-232/`

como `ReadInt`, `ReadBuf`, `SendInt`, `SendString` e `SendBuf`. Essas funções de nível mais alto operam em laços sobre as rotinas básicas, permitindo a transmissão e recepção de dados estruturados byte a byte.

3.2.2. Programa em C para chamadas de sistema de arquivos

Foi desenvolvido um programa em C que é executado no PC e aguarda comandos enviados pelo RISC-V via RS232. O programa interpreta o *byte* recebido para identificar a chamada desejada, recebe os argumentos por meio das funções de leitura implementadas e executa a função correspondente da biblioteca `stdio.h`. Após a execução, os valores de retorno são enviados de volta ao RISC-V.

Enquanto a biblioteca `stdio.h` utiliza ponteiros de arquivo (`FILE*`) para suas operações, as chamadas de sistema do RISC-V utilizam apenas descritores de arquivos (`int`). Sendo assim, foi necessário um mapeamento entre descritores e ponteiros de arquivos. Para isso, foi utilizado o vetor `FILE *files[1024]`. A função `syscall_open`, invocada para executar a chamada `Open`, armazena o ponteiro do arquivo na posição indexada pelo seu descritor, como visto na linha 9 da Listagem 1.

```
1  void syscall_open() {
2      ...
3      int fd;
4      FILE *f = fopen(filepath, mode);
5      if (!f) {
6          /* tratamento de erros */
7      } else {
8          fd = fileno(f);
9          files[fd] = f;  /* salva FILE* no indice do descritor */
10     }
11     ...
12 }
```

Listagem 1. Trecho da função `syscall_open`.

3.2.3. Chamadas de sistema de arquivos no FPGA

As chamadas de sistema `Open`, `Close`, `LSeek`, `Read` e `Write` foram implementadas com a mesma semântica das chamadas de sistema presentes no simulador RARS. Sua implementação no processador RISC-V (conforme mostrado na Listagem 2) foi feita de forma simples: o RISC-V envia o byte identificador da chamada (Linha 14) seguido dos argumentos (Linhas 16 e 18) via RS232, e aguarda a resposta do PC com os valores de retorno (Linhas 19 e 23).

```
# Read
## a0 = descritor do arquivo
## a1 = endereco do buffer onde os bytes serao escritos
## a2 = quantidade de bytes a serem lidos
# retorno
## a0 = quantidade de bytes lidos
Read: DE1(s8, Read.DE1)
    ecall
    ret
Read.DE1: addi sp, sp, -4
    sw ra, 0(sp)
    mv s0, a0
    li a0, READ
    jal RS232_SendByte  # manda o codigo da chamada
    mv a0, s0
    jal RS232_SendInt   # manda descritor do arquivo
    mv a0, a2
    jal RS232_SendInt   # manda qntd de bytes a serem lidos
    jal RS232_ReadInt   # le quantos bytes foram lidos
    mv s0, a0
    mv a0, a1
    mv a1, s0
    jal RS232_ReadBuf
    mv a0, s0
    lw ra, 0(sp)
    addi sp, sp, 4
    ret
```

Listagem 2. Implementação da chamada de sistema Read no FPGA.

3.3. Carregador

3.3.1. Separação da memória

Para garantir a execução correta do código do usuário, é necessário que ele seja carregado no mesmo endereço em que foi montado. O montador do RARS, no entanto, sempre monta o código a partir do endereço `0x00400000`. Sendo assim, para impedir que o código do usuário sobrescreva o *kernel*, surge a necessidade de montá-lo a partir de um outro endereço. Foi então escolhido o endereço `0x0041000` como endereço inicial do código do *kernel* (seção `ktext`). Similarmente, foi escolhido o endereço `0x10030000` como o endereço inicial da seção de dados do *kernel* (seção `kdata`). Estes endereços foram definidos a partir das restrições de quantidade de memória interna existente no FPGA, onde 128KiB são reservados ao segmento `data` e 64KiB ao segmento `text`

Para possibilitar a montagem do *kernel* nesses endereços, foram desenvolvidas duas novas diretivas no montador do RARS: `ktext` e `kdata`. Essas diretivas funcionam de forma análoga às diretivas `text` e `data`, sinalizando para o montador que o trecho de código ou de dados à seguir deve ser montado a partir do endereço inicial da seção correspondente à diretiva.

3.3.2. Comando `exec`

Com a seção de programa do usuário reservada, foi adicionado o comando `exec` ao *shell*, responsável por carregar o código. Como o RARS não gera arquivos em formato executável (ELF, PE, etc.), é necessária a geração de dois arquivos binários separados: `programa.bin` (código) e `programa.dat` (dados), com `programa` sendo o nome do programa do usuário.

Inicialmente, o carregador utiliza a chamada `Open` para abrir o arquivo `programa.bin`. Caso não o encontre, o *shell* imprime uma mensagem de erro comunicando isso ao usuário, e volta ao REPL. Depois, o carregador utiliza a chamada `LSeek` para determinar o tamanho do arquivos. Caso exceda o espaço disponível, outra mensagem de erro é exibida. Caso contrário, o arquivo será lido.

Se o *shell* estiver sendo executado no processador RISCV, o arquivo é lido diretamente na seção `text`. Se o *shell* estiver rodando no RARS, isso não é possível, então o conteúdo do arquivo é temporariamente carregado na seção `data` e copiado instrução por instrução para a seção correta. Após a leitura do arquivo de código, o mesmo procedimento é feito para o arquivo de dados. Caso obtenha sucesso no carregamento, os arquivos são fechados e a execução é transferida para o início da seção `text`.

3.3.3. Chamada de sistema `exit`

A chamada de sistema `exit` foi modificada para permitir que o *shell* continue funcional após a execução de um programa, unificando o comportamento no RARS e no processador FPGA. Antes, no simulador, ela encerrava a execução, enquanto no FPGA resultava em um laço infinito, sem retorno ao *shell*. Na nova implementação, ao finalizar o programa, é exibida a mensagem "Pressione qualquer tecla para continuar" e o sistema aguarda a entrada de um caractere via `readChar`, permitindo ao usuário visualizar a saída. Em seguida, o ponteiro de pilha (`sp`) é restaurado, o endereço do procedimento `clear` do *shell* é carregado no registrador `uepc` e a instrução `uret` é executada, garantindo o retorno correto segundo a convenção de exceções e interrupções da arquitetura RISC-V.

4. Resultados

Os componentes do sistema foram projetados com atenção ao uso eficiente da memória, de modo a garantir o funcionamento do *shell*, das chamadas de sistema e ainda reservar espaço adequado para os programas de usuário. O *shell* completo, incluindo os comandos internos e a interface de entrada interativa, ocupa 1140 *bytes*, enquanto a implementação das chamadas de sistema utiliza 7548 *bytes*. Em contraste, o espaço reservado para os programas de usuário corresponde a 64 KiB (65536 *bytes*), valor significativamente maior do que o do *kernel* (8688 *bytes*). Essa distribuição evidencia que a maior parte da memória está disponível para execução de aplicações externas, ao passo que o *kernel* minimalista permanece compacto, respeitando os recursos limitados do FPGA e facilitando sua reutilização em projetos futuros.

O código do *shell* e das chamadas de sistema está disponível em um repositório

do *Github*[2]. Vídeos demonstrando o funcionamento do trabalho estão disponíveis no *YouTube*[3,4].

4.1. Execução do Shell no Processador RISC-V

A execução do sistema inicia com o carregamento do programa do *shell* na memória do FPGA pelo *In-System Memory Content Editor*, seguido da configuração do registrador utvec e da transferência de controle para o endereço inicial do *shell*. A interface textual é exibida diretamente na tela conectada ao FPGA e permite a interação com o usuário por meio de um teclado conectado ao sistema.

O *shell* implementa os quatro comandos internos: `echo`, `clear`, `help` e `exit`. Cada comando foi testado individualmente no processador, utilizando entrada por teclado e exibição direta em vídeo. Comandos inválidos são reconhecidos, e o *shell* informa o usuário que o comando é inválido, retornando imediatamente ao estado de espera por nova entrada.

4.2. Chamadas de Sistema de Arquivos

Cada chamada de sistema implementada foi submetida a testes utilizando diferentes parâmetros de entrada, e em todas as situações observou-se o comportamento correto esperado. Após essa validação funcional, também foi realizada uma avaliação de desempenho das chamadas `Read` e `Write`, medindo o tempo necessário para a transferência de blocos de dados de 1 KiB.

Para garantir resultados consistentes, o procedimento foi repetido 100 vezes e, a partir desses dados, calculou-se a média dos tempos medidos. Destaca-se que, em todo o desenvolvimento deste trabalho, a comunicação serial RS232 foi configurada para operar a uma taxa de transmissão (*baud rate*) de 115200 b/s. A Tabela 1 apresenta os valores obtidos nesse teste de desempenho.

Tabela 1. Tempo de execução e taxa de transferência das chamadas `Read` e `Write` para blocos de 1 KiB.

Operação	Tempo médio de execução	Taxa de transferência
Read 1 KiB	114,90 ms	8,91 KB/s
Write 1 KiB	122,09 ms	8,38 KB/s

Essas chamadas são essenciais para o funcionamento do carregador de programas, permitindo que os arquivos binários e de dados sejam lidos do computador e transferidos para a memória interna do FPGA.

5. Conclusão

Este trabalho desenvolveu um *shell* para um processador RISC-V em FPGA, projetado para facilitar o carregamento e execução de programas externos via conexão RS232, superando as limitações do método tradicional com o *In-System Memory Content Editor* do

[2]https://github.com/ziul123/risc-v-loader-shell
[3]https://youtu.be/2BpYoFsBRDc
[4]https://youtu.be/fo6uGAqDrww

Quartus® ao oferecer uma solução mais automatizada. Como o ambiente possui recursos bastante limitados, especialmente de memória, foi necessária uma abordagem enxuta e otimizada em todas as fases do desenvolvimento.

O sistema resultante provê uma interface textual no formato REPL, capaz de interpretar comandos básicos como `echo`, `clear`, `help` e `exit`, além de permitir o carregamento de arquivos binários por meio do comando `exec`. A implementação das chamadas de sistema para acesso a arquivos por meio da comunicação RS232 entre FPGA e PC possibilitou a transferência de dados e programas, validando a proposta mesmo em um ambiente com restrições severas.

6. Trabalhos Futuros

Como trabalhos futuros, podem ser implementadas funcionalidades que ampliem o *shell*, incluindo histórico de comandos, variáveis de ambiente, redirecionamento de entrada e saída, além de novas chamadas de sistema para melhorar a integração entre os programas e o *shell*, com chamadas de sistema de leitura e escrita na interface do shell. Também é possível introduzir escalonamento para permitir a execução concorrente de múltiplos programas, com troca de contexto e gerenciamento de processos.

Outra extensão seria a implementação de um sistema de paginação para memória real, possibilitando a execução de programas maiores que a memória física disponível no FPGA por meio do carregamento sob demanda de páginas. Além disso, o carregador poderia ser adaptado para suportar arquivos ELF, facilitando o uso de montadores tradicionais, e o programa residente no computador poderia ser aprimorado para oferecer funcionalidades de sistema de arquivos remoto, como listagem e alteração de diretórios.

Referências

Fernandes, J. H. F. (2021). Bootloader for a RISC-V processor that uses Flash Memory. Master's thesis, Instituto Superior Técnico.

Ministério da Ciência, Tecnologia e Inovação (2024). Brasil se torna membro da Aliança RISC-V International. `https://www.gov.br/mcti/pt-br/acompanhe-o-mcti/noticias/2024/03/brasil-se-torna-membro-da-alianca-risc-v-international`.

Pelé, A.-F. (2025). CEA Backs RISC-V for Sovereign, Scalable Computing. `https://www.eetimes.eu/cea-backs-risc-v-for-sovereign-scalable-computing/`.

Terasic Technologies (2025). DE1-SoC Development Kit. `https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836\#contents`. Accessed: 2025-06-29.

Valente, L., Tortorella, Y., Sinigaglia, M., Tagliavini, G., Capotondi, A., Benini, L., and Rossi, D. (2023). HULK-V: a Heterogeneous Ultra-low-power Linux capable RISC-V SoC. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6.