VIRTUALIZAÇÃO DO INSTRUMENTO VENSPEC-M DA MISSÃO VERITAS POR MEIO DE EMULAÇÃO DE HARDWARE COM QEMU E INTEGRAÇÃO *SPACEWIRE* VIA TCP/IP

Felipe Fazio da Costa [1]; Rodrigo Marco França [2]

[1] Aluno de Iniciação Científica do Instituto Mauá de Tecnologia (IMT);
[2] Professor do Instituto Mauá de Tecnologia (IMT).

Resumo. *Este artigo apresenta o desenvolvimento de um sistema virtualizado do instrumento VenSpec-M, parte da missão VERITAS da NASA, utilizando o emulador QEMU e fermentas open-source e integrando interfaces SpaceWire via TCP/IP. O objetivo é permitir o desenvolvimento e teste de software de voo espacial em um ambiente Software-in-the-Loop (SIL), minimizando a necessidade de hardware físico. O sistema proposto, denominado QEMULA, fornece interfaces de controle agnósticas e busca se integrar com ferramentas de Integração Contínua/Entrega Contínua (CI/CD), facilitando o desenvolvimento e validação de software para missões espaciais. O trabalho inclui o desenvolvimento do ambiente de virtualização, definição de formatos de dados, implementação de máquinas de estados, mecanismos de verificação de erros e integração com hardware real utilizando o SpaceWire Brick da STAR-Dundee. A virtualização do VenSpec-M contribui para otimizar o desenvolvimento de software no ambiente espacial, reduzindo custos e riscos associados ao uso de hardware físico, além de criar novas possibilidades de testes automatizados.*

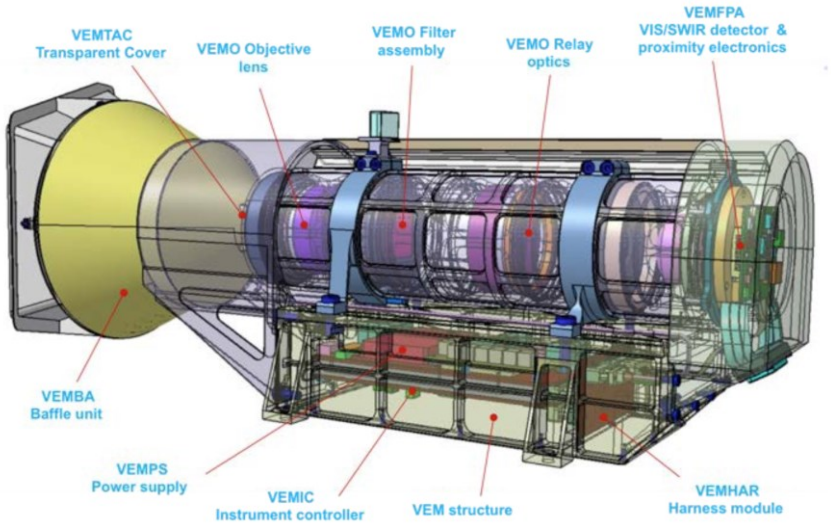
Introdução

A exploração espacial tem avançado significativamente nas últimas décadas, impulsionada por missões que visam compreender melhor os planetas do nosso sistema solar. A missão VERITAS, mostrada na Figura 1, acrônimo de *Venus Emissivity, Radio Science, InSAR, Topography, and Spectroscopy*, é uma iniciativa liderada pelo Laboratório de Propulsão a Jato (JPL) da NASA, em colaboração com agências espaciais internacionais, incluindo o Centro Aeroespacial Alemão (DLR). O principal objetivo da missão é mapear a superfície de Vênus com alta resolução, utilizando radar de abertura sintética e espectrometria, para compreender sua evolução geológica, processos tectônicos, vulcanismo e estrutura interna (Smrekar *et al.*, 2016).

Figura 1 – Imagem conceitual da missão VERITAS

O *Venus Emissivity Mapper* (VEM) é um instrumento crítico para a missão VERITAS, desenvolvido pela DLR. De acordo com Helbert *et al.* (2019a), Helbert *et al.* (2019b) e Hagelschuer *et al.* (2024), ele é responsável por analisar a composição mineralógica da superfície venusiana por meio de medições de emissividade no infravermelho próximo. Essas medições fornecem dados essenciais para a compreensão da história geológica de Vênus e sua comparação com a Terra. O VEM é composto de uma unidade ótica e uma unidade eletrônica denominada VenSpec-M, que pode ser vista na Figura 2.

Figura 2 – Imagem conceitual do instrumento VenSpec-M



O desenvolvimento de *software* para instrumentos espaciais como o VenSpec-M enfrenta desafios significativos. Testes e validação de *software* de voo espacial normalmente requerem *hardware* específico, o que pode ser dispendioso, limitado em disponibilidade e complexo de operar. De acordo com Geletko *et al.* (2019), o uso de simuladores operacionais, como o NOS3, permite superar essas limitações, fornecendo um ambiente flexível para o desenvolvimento e teste de *software* sem a dependência direta de equipamentos físicos.

Este trabalho apresenta o desenvolvimento do QEMULA, um sistema de virtualização do VenSpec-M baseado no emulador QEMU. Conforme demonstrado por Carvalho *et al.* (2020) e Howard & Irvin (2023), o QEMU é uma ferramenta poderosa para emulação de arquiteturas de *hardware* específicas, permitindo a virtualização de sistemas para testes de *software*. O QEMULA incorpora interfaces de controle via TCP/IP, tornando-se agnóstico em relação às ferramentas de desenvolvimento utilizadas e facilitando a integração em pipelines de CI/CD. Além disso, o sistema busca permitir a integração com *hardware* real, utilizando-se de ferramentas como o *SpaceWire Brick* da STAR-Dundee para estabelecer comunicações via *SpaceWire*, padrão amplamente utilizado em sistemas espaciais (Hagelschuer *et al.*, 2024).

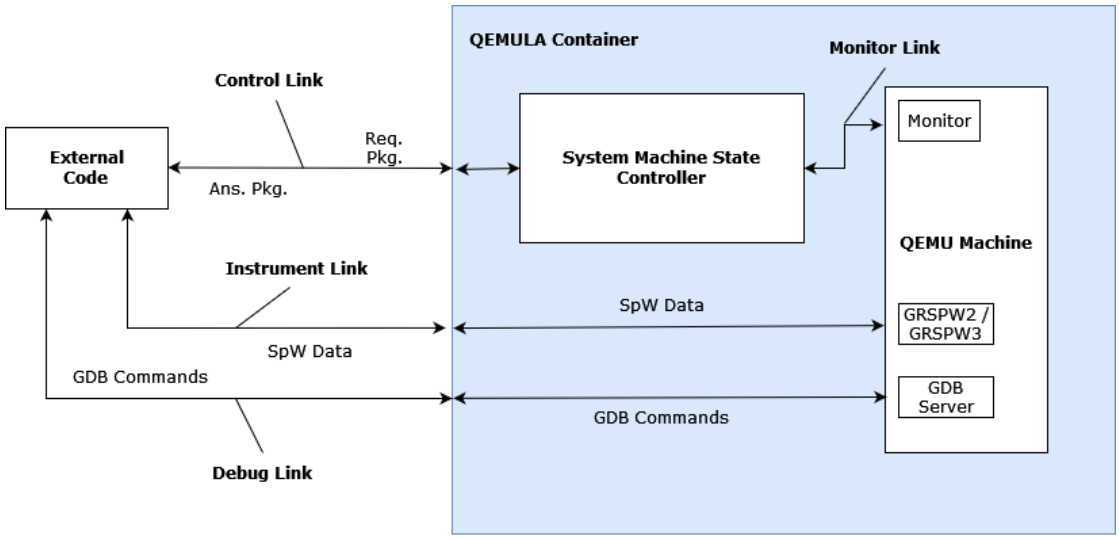
Material e Métodos

O QEMULA foi desenvolvido utilizando o QEMU, um emulador de código aberto que permite a virtualização de arquiteturas de *hardware* específicas. Conforme mostrado por Pfandzelter & Bermbach (2022) e Howard & Irvin (2023), a utilização de ambientes virtualizados é essencial para o desenvolvimento e teste de sistemas embarcados. O VenSpec-M utiliza o processador LEON3, baseado na arquitetura SPARC V8, já disponível no QEMU. No entanto, para emular completamente o instrumento, foi necessário virtualizar diversas interfaces de *hardware* adicionais, especialmente aquelas relacionadas à comunicação via *SpaceWire*.

A arquitetura geral do QEMULA é apresentada na Figura 3. O sistema possui dois estados internos principais:

- **CONFIG:** Neste estado, a simulação do sistema virtualizado pode ser configurada. O usuário pode definir parâmetros como interfaces de comunicação, configurações de rede e outros aspectos relevantes para o ambiente de teste.
- **RUN:** No estado RUN, a simulação é efetivamente executada. O sistema virtual é iniciado com as configurações estabelecidas no estado CONFIG, permitindo a execução do *software* de voo espacial e a interação com o ambiente externo.

Figura 3 – Diagrama de blocos mostrando os fluxos principais de comunicação do QEMULA com entidades externas

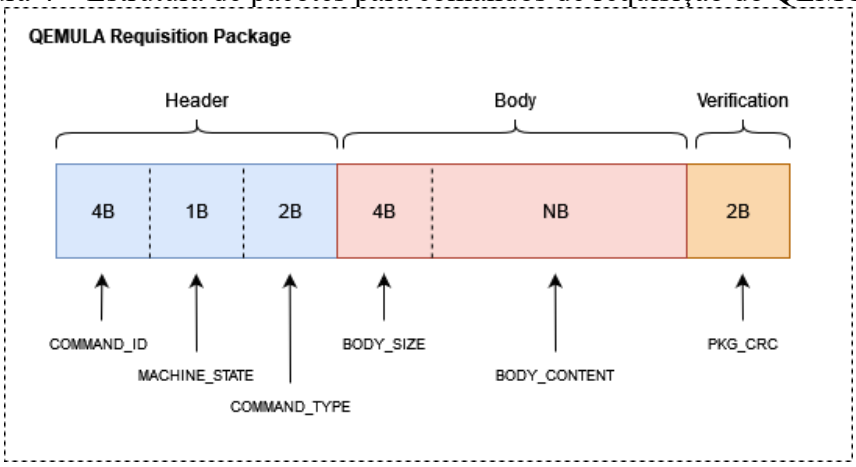


Essa estrutura de estados permite flexibilidade e controle sobre o ambiente de simulação, facilitando a adaptação a diferentes cenários de teste.

Para garantir uma interface de controle agnóstica e genérica, todas as comunicações com o QEMULA são realizadas através de portas TCP/IP. Seguindo abordagens semelhantes às descritas por Geletko *et al.* (2019), essa abordagem permite a integração com diversas ferramentas e aplicações, incluindo sistemas de CI/CD.

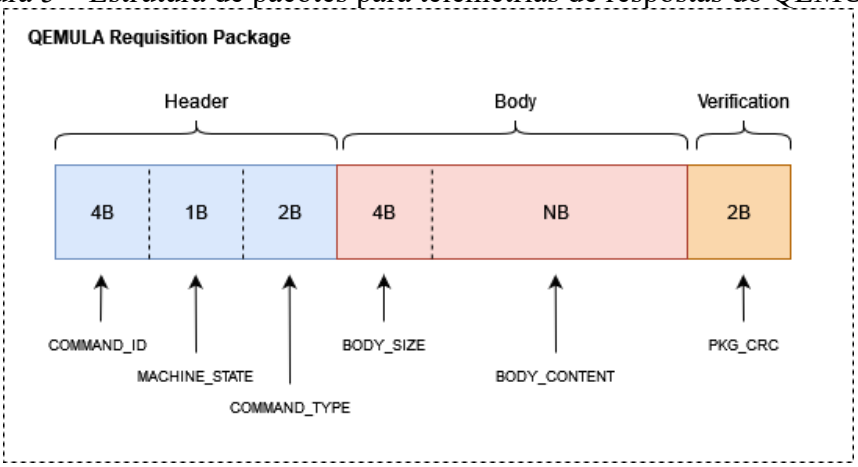
A estrutura dos pacotes de comandos de requisição é ilustrada na Figura 4. Cada pacote de comando inclui campos como identificador de comando, tamanho dos dados, dados específicos do comando e códigos de verificação de erros.

Figura 4 – Estrutura de pacotes para comandos de requisição do QEMULA



A estrutura dos pacotes de telemetria de resposta é apresentada na Figura 5. As respostas incluem campos como status de execução, dados de retorno e códigos de erro, quando aplicável.

Figura 5 – Estrutura de pacotes para telemetrias de respostas do QEMULA



As Tabela 1 e Tabela 2 listam, respectivamente, os comandos disponíveis no sistema e as possíveis respostas correspondentes. Essa padronização facilita a implementação de clientes que interagem com o QEMULA, permitindo automação e integração em processos de desenvolvimento.

Tabela 1 – Lista de comandos para o QEMULA

MACHINE STATE	COMMAND_TYPE	BODY_CONTENT
0xX	0x00 (Verify State)	-
0x0 (CONFIG)	0x01 (Inject .ELF)	.ELF Binaries
0x0 (CONFIG)	0x02 (Config RAM)	RAM in MB
0x0 (CONFIG)	0x03 (Config Debug)	0x0 - Without Debug / 0x1 With Debug
0x0 (CONFIG)	0x04 (Start Emulation)	-
0x1 (RUN)	0x05 (Pause Machine)	-
0x1 (RUN)	0x06 (Unpause Machine)	-
0x1 (RUN)	0x07 (Dump Memory)	0xXXXXXXXX - Mem ADDR + Mem Length (in bytes)
0x1 (RUN)	0x08 (Load Memory)	0xXXXXXXXX - Mem ADDR + Mem DATA
0x1 (RUN)	0x09 (Read I/O)	0xXXXXXXXX - Mem ADDR
0x1 (RUN)	0x0A (Write I/O)	0xXXXXXXXX - Mem ADDR + 0xX IO Value
0x1 (RUN)	0x0B (Quit Emulation)	-

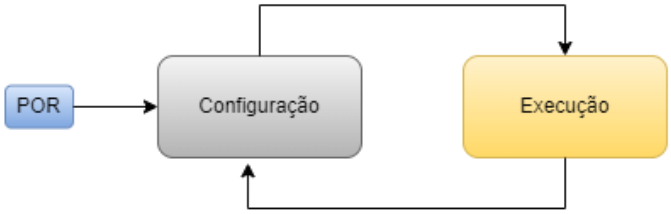
O processamento dos comandos no QEMULA segue a lógica de uma máquina de estados interna, conforme ilustrado na Figura 6. A máquina de estados gerencia as transições entre os estados CONFIG e RUN, bem como as ações associadas a cada comando recebido.

Tabela 2 – Lista de respostas para o QEMULA

Prev. MACHINE STATE	Prev. COMMAND_TYPE	BODY_CONTENT
0xX	0x00 (Verify State)	0x0 - CONFIG / 0x1 - RUN
0x0 (CONFIG)	0x01 (Inject .ELF)	-
0x0 (CONFIG)	0x02 (Config RAM)	-

0x0 (CONFIG)	0x03 (Config Debug)	-
0x0 (CONFIG)	0x04 (Start Emulation)	-
0x1 (RUN)	0x05 (Pause Machine)	-
0x1 (RUN)	0x06 (Unpause Machine)	-
0x1 (RUN)	0x07 (Dump Memory)	Mem DATA
0x1 (RUN)	0x08 (Load Memory)	-
0x1 (RUN)	0x09 (Read I/O)	I/O Value
0x1 (RUN)	0x0A (Write I/O)	-
0x1 (RUN)	0x0B (Quit Emulation)	-

Figura 6 – Diagrama da máquina de estados interna de simulação do QEMULA



Os principais estados e transições do sistema são estruturados para permitir um controle preciso sobre o ciclo de vida da simulação. O sistema inicia no estado CONFIG, aguardando comandos de configuração. Neste estágio de inicialização, ele está pronto para receber instruções que ajustam os parâmetros do sistema. Após a configuração necessária, ao receber o comando para iniciar a simulação, o sistema transita para o estado RUN. No estado RUN, o sistema executa o *software* de voo espacial e processa comandos relacionados à simulação em andamento. Quando o comando para parar a simulação é recebido, o sistema retorna ao estado CONFIG, permitindo novas configurações ou o encerramento do ciclo. Essa estrutura garante que as operações sejam realizadas em ordem lógica e consistente, facilitando o gerenciamento e a estabilidade da simulação.

Para garantir a integridade das comunicações, foram implementados mecanismos robustos de verificação de erros, incluindo códigos de redundância cíclica (CRC) e tratamento de exceções, conforme práticas recomendadas em sistemas críticos (Naia, 2015).

A Tabela 3 detalha os tipos de erros que podem ser reportados pelo sistema em resposta aos comandos recebidos. Os erros incluem:

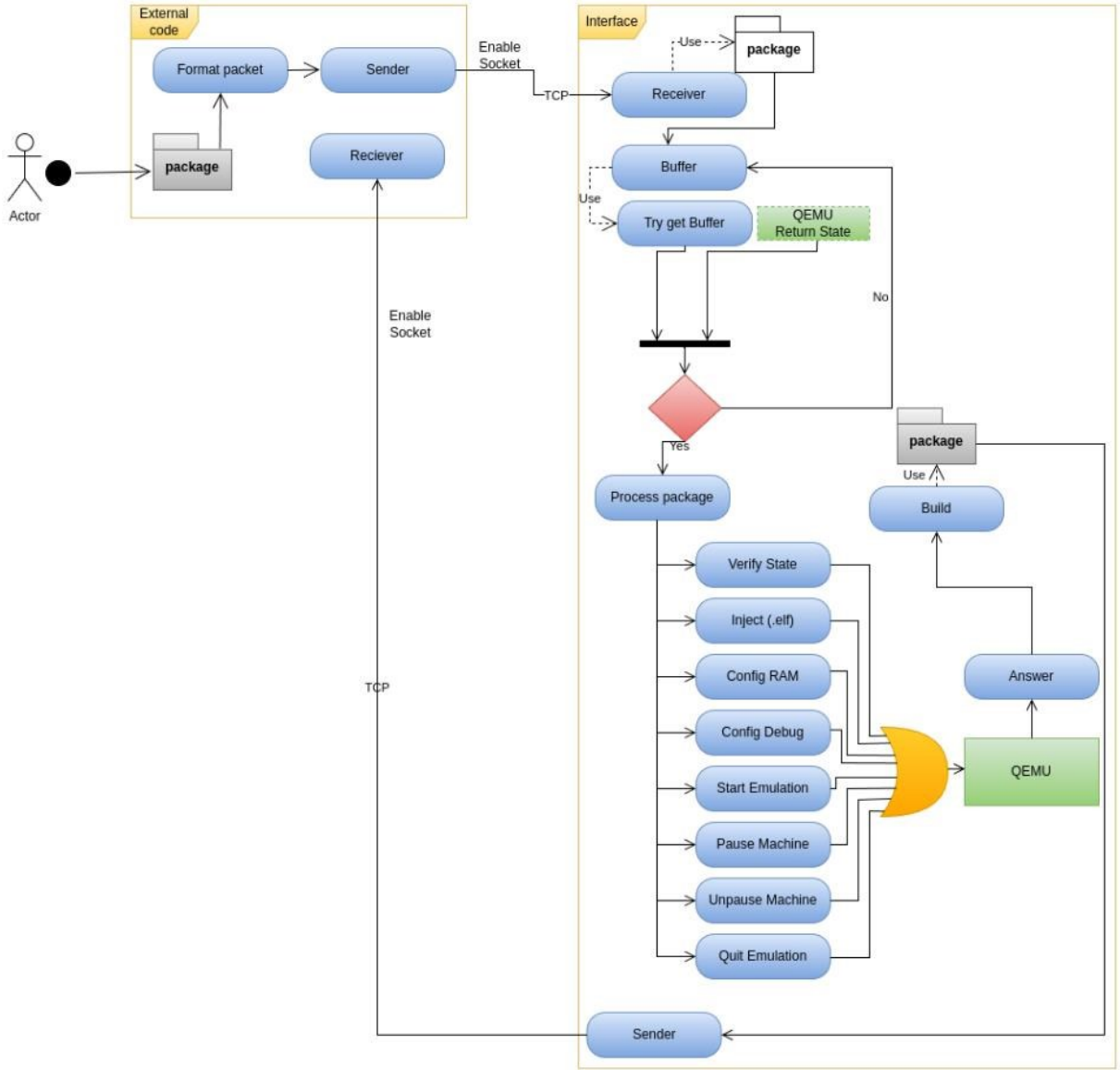
- **Erros de Formato de Pacote**: Detecção de pacotes malformados ou com campos inválidos.
- **Erros de Comando Inválido**: Comandos não reconhecidos ou não suportados no estado atual.
- **Erros de Execução**: Falhas durante a execução de um comando válido.

Tabela 3 – Tabela de erros que podem ser reportados nas respostas de comandos do QEMULA

ANSWER_STATUS	Execution Status
0x01	QEMU Error
0x02	Executed Succesfully
0x03	CRC Error
0x04	Invalid COMMAND_TYPE
0x05	Invalid BODY_CONTENT
0x06	Timeout
0x07	Invalid MACHINE_STATE
0x08	Invalid COMMAND_ID

O fluxo de dados do processamento de comandos no QEMULA é ilustrado na Figura 7 e se inicia com o recebimento do pacote, onde o sistema recebe o pacote de comando via TCP/IP. Em seguida, ocorre a validação do pacote, na qual se verifica a integridade e o formato do pacote utilizando o CRC e outros mecanismos de verificação. Após a validação bem-sucedida, o sistema procede ao processamento do comando, executando a ação correspondente ao comando recebido, sempre considerando o estado atual da máquina de estados. Posteriormente, há a geração da resposta, momento em que o sistema cria o pacote de resposta contendo os dados solicitados ou códigos de erro apropriados. Finalmente, ocorre o envio da resposta ao cliente via TCP/IP. Esses mecanismos asseguram que os comandos sejam executados corretamente e que quaisquer erros sejam detectados e comunicados ao usuário de forma clara.

Figura 7 – Diagrama do fluxo de dados do processamento de comandos do QEMULA



Uma possibilidade interessante que também foi explorada é a integração de sistemas virtualizados com sistemas reais através de links *SpaceWire* de alta velocidade. Essa abordagem é fundamental para garantir a compatibilidade entre o *software* desenvolvido e o *hardware* real (Hagelschuer *et al.*, 2024). Para permitir a comunicação entre o sistema virtualizado e *hardware* real, foram utilizados os equipamentos *SpaceWire Brick* STAR-Dundee, especificamente os modelos Mk2 e Mk3 (Figura 8). Esses dispositivos permitem que

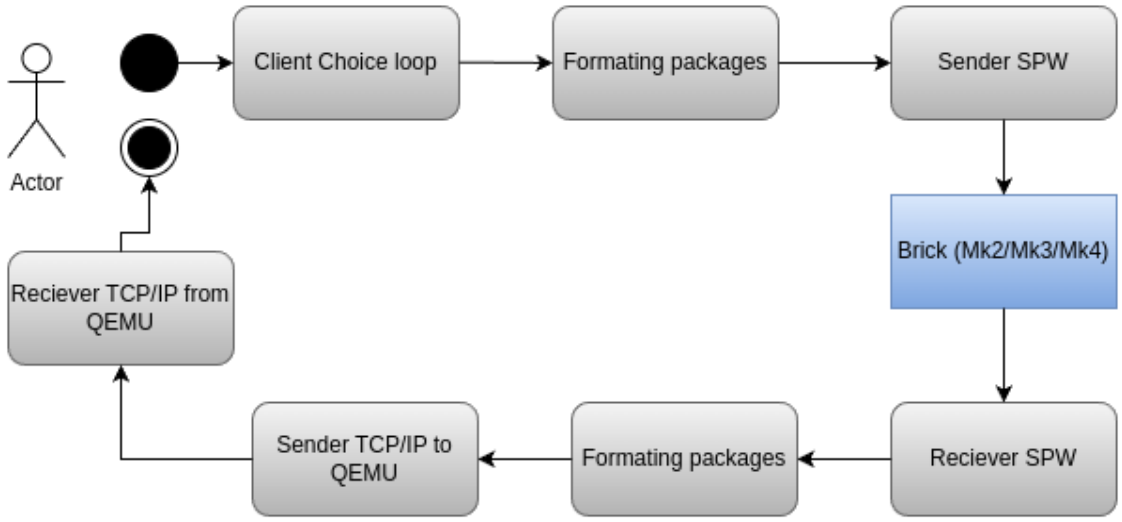
um computador tenha acesso a interfaces *SpaceWire* através de uma porta USB, facilitando a integração com sistemas reais.

Figura 8 – Fotografia do SpaceWire Brick Mk3 da STAR-Dundee

Com as APIs fornecidas pela STAR-Dundee, foram desenvolvidos códigos que conectam uma interface TCP/IP aos links *SpaceWire* do *Brick*, desempenhando funções essenciais. Esses códigos configuram o *Brick*, incluindo inicialização, estabelecimento de conexões e ajuste de parâmetros operacionais. Também realizam a formatação de pacotes, convertendo dados de TCP/IP para o formato *SpaceWire*, com transformações numéricas e separação por bytes para compatibilidade com o protocolo.

Esses códigos desempenham várias funções essenciais. Primeiramente, realizam a configuração do *Brick*, que envolve a inicialização do dispositivo, o estabelecimento de conexões e o ajuste de parâmetros de operação necessários para o funcionamento adequado. Em seguida, executam a formatação de pacotes, convertendo os dados recebidos via TCP/IP para o formato adequado do *SpaceWire*, o que inclui a transformação de bases numéricas e a separação dos dados por bytes para compatibilidade com o protocolo. Além disso, implementam a transmissão e recepção de dados, onde os pacotes formatados são enviados através do link *SpaceWire*, e o sistema mantém uma escuta contínua para a recepção de dados provenientes do *hardware*. Por fim, incluem o tratamento de erros e exceções, implementando mecanismos para lidar com travamentos de *threads* e perdas de pacotes, garantindo a robustez e estabilidade da comunicação. O diagrama de fluxo de informações para o sistema desenvolvido é apresentado na Figura 9.

Figura 9 – Diagrama do fluxo de informações para o sistema desenvolvido para interface TCP/IP e SpaceWire



Essa integração permite que o QEMULA interaja com instrumentos físicos, possibilitando testes híbridos que combinam elementos virtuais e reais, embora essa funcionalidade completa ainda não tenha sido testada.

Foi implementado um menu de opções que permite ao usuário configurar o sistema para operar de acordo com diferentes perfis de uso:

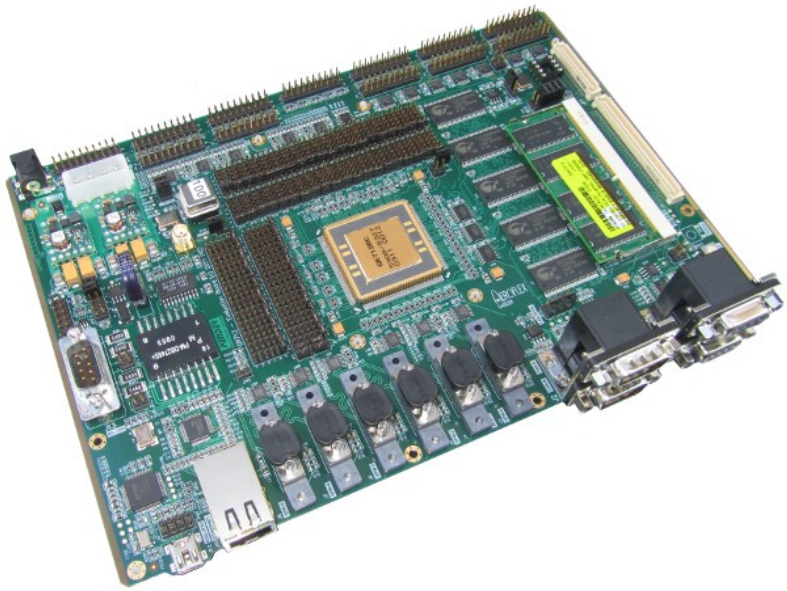
- **Configurações do *hardware***:
    - **Resetar**: Reinicializa o dispositivo para o estado padrão.
        - **Configurar Looping**: Define o modo de operação:
        - **Envio Completo (*SpaceWire* e TCP/IP)**: O sistema envia e recebe dados através de ambos os protocolos.
        - **Semi-Loop (*SpaceWire* ou TCP/IP)**: Opera apenas com um dos protocolos.
        - **Loop de Estresse**: Realiza testes intensivos para verificar a robustez do sistema.

Essas opções proporcionam flexibilidade para atender a diferentes necessidades de teste e desenvolvimento.

Para garantir que as interfaces virtualizadas funcionem como o *hardware* real, foi utilizada a placa de desenvolvimento GR712RC da Frontgrade Gaisler, equipada com um processador LEON3 dual-core, similar ao do VenSpec-M. A placa, mostrada na Figura 10, inclui interfaces como *SpaceWire*, UART e controladores de memória, servindo como referência para validar os módulos virtualizados. Esse método de validação é consistente com práticas estabelecidas em pesquisas anteriores (Carvalho *et al.*, 2020; Pfandzelter & Bermbach, 2022).

Testes realizados no QEMULA verificaram integridade, compatibilidade e desempenho das comunicações, garantindo que o *software* desenvolvido no ambiente virtualizado seja compatível com o *hardware* real. Embora o desenvolvimento e validação completos dos módulos de *hardware* estejam fora do escopo deste trabalho, a abordagem adotada assegura a fidelidade da emulação e a confiabilidade do sistema para testes e desenvolvimento de *software*.

Figura 10 – Placa de desenvolvimento GR712RC utilizado para testar os sistemas virtualizados contra um sistema real



Para facilitar a distribuição e utilização do QEMULA, foi criada uma imagem Docker otimizada. O uso de contêineres Docker para a distribuição de ambientes de desenvolvimento

é uma prática recomendada que tem sido adotada em outros trabalhos (Howard & Irvin, 2023). O Docker é uma plataforma de virtualização leve que permite criar, implantar e executar aplicativos em contêineres, garantindo que o ambiente seja consistente em diferentes sistemas.

Inicialmente, o ambiente Docker criado, baseado em Ubuntu, resultou em uma imagem de 2,2 GB, considerada grande para o projeto. Diversas otimizações foram realizadas, como a substituição do sistema operacional para Debian Bookworm-Slim, a remoção de componentes desnecessários do QEMU e o uso do ambiente virtual Python (venv) para incluir apenas pacotes essenciais. Essas mudanças reduziram o tamanho da imagem para 214 MB, mais de 10 vezes menor, tornando o sistema mais ágil, prático para download e instalação.

A utilização do Docker visa permitir que o QEMULA seja facilmente integrado em pipelines de CI/CD, como aqueles disponíveis no GitHub Actions ou GitLab CI. Isso busca possibilitar que novos códigos sejam automaticamente testados e validados, acelerando o ciclo de desenvolvimento e aumentando a confiabilidade do *software*.

Uma das especificações fornecidas pelo DLR foi a necessidade de depurar o código emulado por meio do servidor GDB integrado ao QEMU. Como o QEMU já possui uma porta de depuração, foi necessário apenas expô-la externamente no contêiner Docker. Isso permite que os desenvolvedores se conectem remotamente ao GDB para analisar e corrigir falhas no *software* emulado, facilitando o processo de desenvolvimento (Howard & Irvin, 2023).

Resultados e Discussão

O QEMULA apresentou resultados significativos como um ambiente virtualizado funcional e eficiente, capaz de emular *hardware* baseado em LEON3 com alta fidelidade. Sua arquitetura, estruturada em estados (CONFIG e RUN), oferece flexibilidade e controle para atender a diferentes necessidades de teste. Apesar de ainda estar em desenvolvimento, com conclusão prevista para 2025, o sistema promete incorporar novas funcionalidades, como injeção de falhas e maior fidelidade ao *hardware* VenSpec-M.

A implementação de interfaces de controle agnósticas baseadas em TCP/IP trouxe flexibilidade e acessibilidade, permitindo integração com ferramentas como CI/CD e acesso remoto por equipes distribuídas. Além disso, mecanismos como CRC e tratamento de exceções garantem a confiabilidade e integridade das comunicações. O ambiente Docker também foi significativamente otimizado, com a redução do tamanho da imagem de 2,2 GB para 214 MB, tornando o QEMULA mais prático para distribuição e instalação, o que incentiva sua adoção por equipes de desenvolvimento.

A validação do sistema foi realizada utilizando a placa GR712RC, confirmando a precisão da emulação. O *software* desenvolvido no QEMULA demonstrou comportar-se de forma consistente tanto no ambiente virtualizado quanto no *hardware* real, assegurando a representatividade dos testes. Esses resultados são consistentes com aqueles observados em trabalhos semelhantes, como o de Geletko *et al.* (2019) e Pfandzelter & Bermbach (2022), que destacam a importância de ambientes virtualizados na modernização do desenvolvimento de *software* espacial.

Por fim, o QEMULA se destaca como uma contribuição relevante para a modernização do desenvolvimento de *software* espacial. Ao oferecer um ambiente *Software-in-the-Loop* (SIL) totalmente baseado em ferramentas gratuitas e de código aberto, o sistema reduz custos e riscos ao minimizar a necessidade de *hardware* físico, facilita o desenvolvimento colaborativo entre equipes em diferentes localidades, automatiza testes e validações por meio de integração com sistemas de CI/CD, e permite a simulação de condições adversas. Essas características aprimoram a robustez do *software* e aceleram o ciclo de desenvolvimento, tornando o QEMULA uma ferramenta promissora para o setor espacial.

Conclusões

O QEMULA busca oferecer um ambiente flexível e robusto para o desenvolvimento e teste de *software* de voo espacial, reduzindo a dependência de hardware físico e integrando-se a ferramentas de CI/CD. A criação de interfaces de controle agnósticas, a implementação de mecanismos robustos de verificação de erros e a possibilidade de integração com sistemas reais ampliaram as opções de validação. Além disso, a otimização do ambiente Docker tornou o sistema mais acessível para diversas equipes de desenvolvimento.

O projeto contribui para o avanço das metodologias de desenvolvimento na área aeroespacial, apoiando futuras missões espaciais e fortalecendo a colaboração entre instituições de pesquisa e agências espaciais. O desenvolvimento continuará até, pelo menos, o final de 2025, com planos de incorporar funcionalidades como injeção de erros, simulação de condições adversas mais complexas, maior integração com *hardware* real, e melhorias nas interfaces de controle para torná-las ainda mais intuitivas.

Embora o QEMULA esteja sendo desenvolvido para a missão VERITAS, espera-se que futuramente seja disponibilizado como uma ferramenta de código aberto, incentivando a colaboração e o progresso na pesquisa aeroespacial.

Referências Bibliográficas

Carvalho, H.; Nelissen, G.; Zaykov, P. (2020) mcQEMU: Time-Accurate Simulation of Multi-core platforms using QEMU. *23rd Euromicro Conference on Digital System Design (DSD)*, 81-88.

da Silva, P.C.; Gonçalves, S. (2021) A QEMU-based Approach to Hardware-Assisted Virtualization. *Dissertação de Mestrado, Faculdade de Engenharia - Universidade do Porto (Portugal)*.

Geletko, D.M.; Grubb, M.D.; Lucas, J.P.; Morris, J.R.; Spolaor, M.; Suder, M.D.; Zemerick, S.A. (2019) NASA Operational Simulator for Small Satellites (NOS3): The STF-1 CubeSat Case Study. *arXiv preprint arXiv:1901.07583*.

Hagelschuer, T.; Pertenais, M.; Walter, I.; Dern, P.; del Togno, S.; Säuberlich, T.; Peter, G. (2024) The Venus Emissivity Mapper (VEM): Instrument Design and Development for VERITAS and EnVision. *Infrared Remote Sensing and Instrumentation XXXII*, **13144**, 116-123.

Helbert, J.; Dyar, M.D.; Walter, I.; Rosas-Ortiz, Y.M.; Widemann, T.; Marcq, E.; Ghail, R. (2019a) The Venus Emissivity Mapper-Obtaining Global Mineralogy of Venus from Orbit on the ESA EnVision and NASA VERITAS Missions to Venus. *50th Lunar and Planetary Science Conference*.

Helbert, J.; Vandaele, A.C.; Marcq, E.; Robert, S.; Ryan, C.; Guignan, G.; Lara, L. (2019b) The VenSpec Suite on the ESA EnVision Mission to Venus. *Infrared Remote Sensing and Instrumentation XXVII*, **11128**, 18-32.

Howard, M.; Bruce Irvin, R. (2023) ESP32: QEMU Emulation Within a Docker Container. *Proceedings of the Future Technologies Conference*, 63-80.

Naia, N.P.D. (2015) Real-Time Linux and Hardware Accelerated Systems on QEMU. *Dissertação de Mestrado, Universidade do Minho (Portugal)*.

Pfandzelter, T.; Bermbach, D. (2022) Celestial: Virtual Software System Testbeds for the LEO Edge. *23rd ACM/IFIP International Middleware Conference*, 69-81.

Smrekar, S.; Dyar, M.; Hensley, S.; Helbert, J.; VERITAS Science Team (2016) VERITAS (Venus Emissivity, Radio Science, InSAR, Topography and Spectroscopy): A Proposed Discovery Mission. *AAS/Division for Planetary Sciences Meeting Abstracts# 48*, **48**, 216-07.