Interface de rede AMBA-AXI para uma rede-em-chip confiável

Gustavo Henrique Stahl Müller
Universidade do Vale do Itajaí, Brasil
gustavomuller@edu.univali.br

Thiago Haas Rausch
Universidade do Vale do Itajaí, Brasil
thiagorausch@edu.univali.br

Douglas Rossi Melo
Universidade do Vale do Itajaí, Brasil
drm@univali.br

ABSTRACT

The integrated systems used in space environments and other critical environments feature increasing cores. Because of the environment in which these systems operate, they must use fault tolerance techniques to improve reliability. XINA (eXtensible Interconnect Network Architecture) is a reliable network-on-chip for space systems, which has already been verified and tested. However, for its effective use in real integrated systems, it is necessary to have a network interface that implements communication services. Therefore, this work presents the development of a fault-tolerant network interface compatible with the XINA that also implements communication services commonly desired by cores in integrated systems. To facilitate and accelerate the core integration process with XINA, the interface implements the AMBA-AXI 5 communication protocol. We observed that the reliability techniques allow the interface to function correctly even in the presence of faults in the controllers and buffers. However, they cause an increase of 63% in LUTs, an increase of 16% in FFs, and a decrease of 56% in the maximum operation frequency. The main contribution of this work is to enable the effective application of the XINA network-on-chip in multi-core fault-tolerant integrated systems.

PALAVRAS-CHAVE

Redes-em-Chip, Interfaces de Rede, Tolerância a Falhas.

1 INTRODUÇÃO

A melhoria contínua da tecnologia vem permitindo um aumento constante na quantidade de componentes presentes em um único sistema integrado, também conhecido como System-on-Chip (SoC). Esses sistemas são implementados por meio de modelos de hardware pré-projetados, também chamados de núcleos ou *cores* [1].

A conexão entre os núcleos de um SoC é um problema chave no projeto de sistemas multi-núcleos, e o modelo de conexão escolhido causa um impacto cada vez maior no desempenho geral do sistema, visto que o impacto causado é proporcional à quantidade e complexidade dos núcleos presentes. Por bastante tempo, núcleos foram conectados através de hierarquias de barramentos, mas tais soluções se tornaram cada vez mais difíceis de serem projetadas e verificadas a medida que as necessidades e restrições de projetos foram ampliadas [2]. Consequentemente, é necessária a criação de métodos de comunicação cada vez mais robustos. Um dos métodos de comunicação desenvolvidos é o uso de redes chaveadas, conhecidas como redes-em-chip ou Networks-on-Chip (NoCs) [3].

Uma das aplicações de NoCs é em sistemas integrados orientados à ambientes espaciais, que assim como os sistemas integrados usados na Terra, também apresentam um aumento na complexidade e quantidade de núcleos. Com a atual miniaturização das tecnologias de construção de circuitos, aumenta-se a suscetibilidade a erros causados por interferências eletromagnéticas [4]. Devido a esses fatores, o uso de técnicas de tolerância a falhas que garantem confiabilidade são necessárias para que esses sistemas sejam usados efetivamente [5].

Um exemplo de arquitetura de rede-em-chip que provê confiabilidade é a rede XINA (eXtensible Interconnect Network Architecture) desenvolvida no Laboratório de Sistemas Embarcados e Distribuídos da Universidade do Vale do Itajaí [6]. A arquitetura foi projetada com um foco em regularidade e flexibilidade, e implementas as camadas de rede, enlace e física do modelo OSI (Open Systems Interconnection). Os controladores presentes na arquitetura fazem uso de técnicas de tolerância a falhas como TMR e codificação Hamming.

Apesar da rede XINA já ter sido verificada e testada, para que essa possa ser usada efetivamente em sistemas integrados multi-núcleo, é necessária a criação de uma interface de rede que implemente serviços de comunicação que atendem às necessidades desses núcleos.

Dessa forma, este trabalho descreve o desenvolvimento de uma interface de rede para a rede XINA. A interface desenvolvida implementa serviços de comunicação exigidos por núcleos de sistemas integrados tolerantes à falhas e habilita a integração de múltiplos núcleos à rede.

Este artigo está organizado da seguinte forma. A Seção 1 apresentou a introdução deste trabalho. Na Seção 2 é apresentada a fundamentação teórica, seguida dos trabalhos relacionados na Seção 3. A Seção 4 detalha o desenvolvimento da interface de rede confiável. Na Seção 5 são apresentados e discutidos os resultados da implementação. Por fim, a Seção 6 contém as conclusões e considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

A base teórica deste trabalho consiste nos conceitos de redes-em-chip, interfaces de rede e padrões de comunicação.

2.1 Redes-em-chip

As redes-em-chip são um modelo específico de arquitetura de interconexão que soluciona os gargalos resultantes da integração de uma grande quantidade de núcleos em um SoC [7]. Em uma rede-em-chip, os núcleos de um SoC se comunicam por meio de uma rede interligada por roteadores, através do envio e recebimento de mensagens que são divididas em pacotes. Um pacote é constituído por *flits* (flow control units), a unidade sobre a qual é realizado o controle de fluxo. Portanto, uma arquitetura típica de uma rede-em-chip consiste de múltiplos roteadores e enlaces ponto-a-ponto que conectam esses roteadores de uma maneira estruturada [8].

Uma rede-em-chip é caracterizada pelos seguintes elementos: a topologia, expressa como um grafo que descreve a disposição dos roteadores e enlaces; o roteamento, responsável por determinar o caminho de um pacote da origem ao destino; o controle de fluxo, que regula o tráfego de entrada e saída do roteador; o chaveamento, que especifica a conexão entre portas de entrada e saída; a arbitragem,

encarregada de coordenar o uso de canais e buffers para os pacotes a serem chaveados; e a memorização, que se refere à técnica utilizada para armazenar pacotes durante a espera pelo escalonamento no circuito de arbitragem [8].

A NoC usada nesse trabalho é a XINA, proposta por [6]. A XINA é uma rede projetada para aplicação em SoCs confiáveis usados em aplicações espaciais. A rede faz uso de roteamento XY, controle de fluxo por meio de um *handshake* de quatro estágios, chaveamento *wormhole* e arbitragem *round-robin*. A arquitetura de roteador da XINA faz uso de técnicas de tolerância a falhas, como TMR (Triple Modular Redundancy) nos controladores e código Hamming nos *buffers* de entrada.

2.2 Interfaces de rede

Em uma rede-em-chip, cada núcleo é conectado a um roteador por meio de uma interface de rede. O objetivo principal da interface é de desacoplar a computação do núcleo com a comunicação da rede. Isso é realizado através da conversão do protocolo de comunicação usado pelo núcleo para o protocolo de comunicação específico usado pela rede. Essa conversão também garante maior reusabilidade e facilidade de integração de núcleos [7]. Segundo [9], a arquitetura padrão de uma interface de rede consiste de um *front-end* e um *back-end*.

O *front-end* é o módulo responsável por implementar um protocolo de comunicação ponto-a-ponto padronizado, permitindo o fácil reuso de núcleos entre diversas plataformas. Alguns exemplos de protocolos são o AMBA-AXI (Advanced eXtensible Interface) e o OCP (Open Core Protocol). Tais protocolos são responsáveis por especificar quais transações os núcleos podem executar e quando as sessões entre dois núcleos são abertas e fechadas. O *back-end* é responsável por implementar todas as camadas abaixo da camada de Sessão, sendo essas a camada de Transporte, Rede, Enlace e Física [9].

O modelo de comunicação padrão entre os núcleos é do tipo mestre-escravo, onde mestres iniciam transações por meio de requisições, que representam comandos a serem executados (como escrita ou leitura). Um ou mais escravos executam essas transações quando uma requisição é recebida. Um escravo também pode opcionalmente enviar uma resposta ao seu mestre para retornar dados ou para confirmar que a transação foi completada [9].

2.3 Padrão de comunicação AMBA

O AMBA (Advanced Microcontroller Bus Architecture) é um padrão de comunicação desenvolvido pela ARM Limited cuja especificação é considerada um padrão para a interconexão de núcleos em sistemas integrados. Um dos protocolos definidos pela especificação AMBA é o AXI (Advanced eXtensible Interface), projetado para uso em sistemas mestre-escravo que possuem alto desempenho e alta frequência de operação.

O protocolo AMBA-AXI foi escolhido como alvo de implementação na interface deste trabalho devido à sua ampla adoção por núcleos pré-existentes no mercado e na literatura. A especificação 5 foi escolhida por ser a mais recente e possuir um conjunto maior de funcionalidades, enquanto mantém compatibilidade com versões anteriores..

3 TRABALHOS RELACIONADOS

Para embasar teoricamente o tema de pesquisa deste trabalho, foram examinados estudos correlatos com o objetivo de identificar suas contribuições, os serviços proporcionados pelas interfaces desenvolvidas e os resultados alcançados.

No trabalho de \cite{das2012}, foram descritas duas arquiteturas de interfaces de rede tolerantes a falhas para uma NoC baseada em SDM (Spatial Division Multiplexing). Essas arquiteturas incluem um modelo de tolerância a falhas centralizado e outro baseado em um modelo distribuído. Os autores também implementaram uma interface de rede para cada arquitetura, com o propósito de comparação.

O trabalho [10] descreveu a arquitetura e implementação de uma interface de rede para a rede-em-chip SoCIN, compatível com o barramento Avalon. A arquitetura da interface foi baseada na divisão definida por \cite{bertozzi2006}, que separa a arquitetura em front-end, composto da camada Específica, e back-end, contendo as camadas Genérica e de Rede.

Em [11] foi descrita a arquitetura e implementação de uma versão modificada da interface XIRU, compatível com o barramento AMBA-AHB. O front-end foi o principal alvo de modificações, pois é responsável pela adaptação de protocolos na interface. O back-end também passou por alterações para permitir a transferência de pacotes em rajadas. Por fim, o formato de pacote da rede (SoCIN) precisou ser modificado.

O trabalho [12] estendeu a interface modificada de [11], adicionando técnicas de tolerância a falhas de informação e espaciais. Primeiramente, foi adicionada a codificação de Hamming na camada de rede. Isso envolve a codificação dos pacotes que saem da interface para o roteador e a decodificação dos pacotes que chegam do roteador. Isso exigiu uma ampliação do canal de comunicação com o roteador. Em seguida, foi adicionado TMR nos componentes de controle da camada de rede e da camada genérica.

Os autores de [13] descreveram a arquitetura e implementação de uma interface de rede para a rede-em-chip LISNoC, implementada em Verilog. Essa interface suporta canais virtuais e oferece alta flexibilidade nas configurações de roteadores, fazendo uso de roteamento *wormhole* e arbitragem *round-robin*. A interface criada implementa o protocolo AXI para comunicação com os núcleos.

4 DESENVOLVIMENTO

A interface de rede proposta tem como principal objetivo fornecer um meio de comunicação com a rede-em-chip XINA por meio do protocolo de barramento AMBA-AXI 5. A interface é compatível com esse protocolo, com o propósito de facilitar e acelerar o processo de integração de núcleos pré-existentes à rede-em-chip. Como a interface é projetada para uso em ambientes espaciais e críticos, ela incorpora técnicas de tolerância a falhas. A Figura 1 demonstra o contexto em que este trabalho está inserido, mostrando o lugar da interface em um SoC baseado em NoC e uma visão simplificada da sua arquitetura.

A arquitetura da interface é baseada na divisão de *front-end* e *back-end*, conforme descrito por [9]. O *front-end* é responsável pela comunicação com o núcleo, enquanto o *back-end* lida com a comunicação com o roteador da NoC.

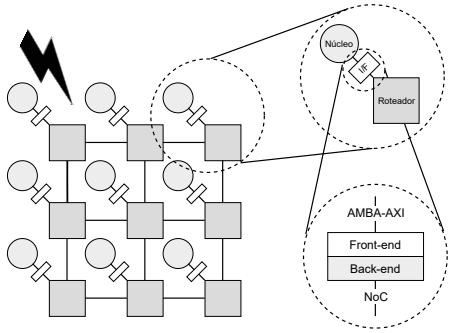

Figura 1: Visão geral da interface de rede.

4.1 *Front-end*

O objetivo do *front-end* é adaptar os sinais da especificação AMBA-AXI vindos do núcleo para os sinais genéricos usados para a comunicação com o *back-end*. Essa lógica de adaptação isolada permite que a interface seja facilmente modificada para suportar outros padrões de comunicação. As versões mestre e escravo possuem sinais de comunicação idênticos, mas com direções opostas. A única exceção é o sinal `CORRUPT_PACKET`, que indica se o pacote recebido pela interface está corrompido. Esse sinal é sempre destinado ao núcleo, seja ele mestre ou escravo. As Figuras 4 e 5 ilustram os sinais presentes na interface para núcleos mestres e escravos, respectivamente.

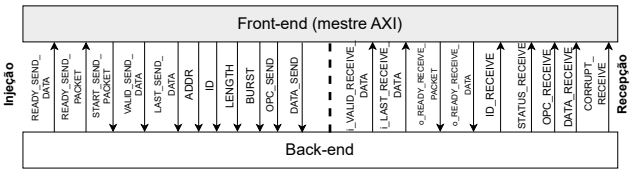

Figura 2: Sinais do *front-end* da interface mestre.

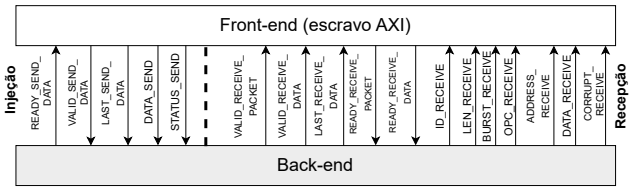

Figura 3: Sinais do *front-end* da interface escravo.

4.2 *Back-end*

O *back-end* possui os seguintes objetivos: empacotar os dados recebidos do *front-end*, desempacotar os pacotes recebidos do roteador da NoC, armazenar temporariamente os pacotes recebidos (tanto do *front-end* quanto da NoC), e realizar o controle de fluxo para envio e recebimento de *flits* do roteador. As técnicas de tolerância a falhas são implementadas nessa camada.

Assim como o *front-end*, o *back-end* possui duas variações diferentes (mestre e escravo). As Figuras 4 e 5 ilustram os sinais entre o *front-end* e *back-end* das interfaces mestre e escravo, respectivamente. Nas duas versões, há a presença de sinais de controle e sinais de dados. Os sinais de controle iniciam e encerram as transações e realizam as transferências, enquanto os sinais de dados contém os dados da carga útil (*payload*) e os valores dos campos do cabeçalho de interface. Os sinais de controle relacionados à injeção de *flits* possuem o sufixo *SEND*, enquanto os sinais relacionados à recepção possuem o sufixo *RECEIVE*.

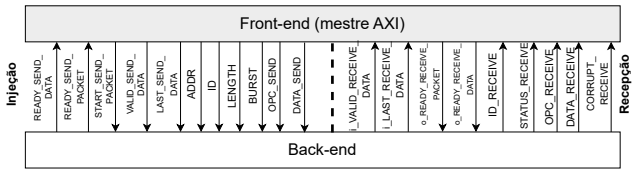

Figura 4: Sinais do *front-end* da interface mestre e o *back-end*.

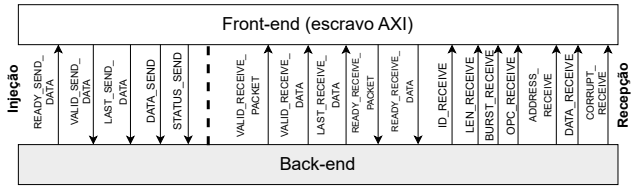

Figura 5: Sinais do *front-end* da interface escravo e o *back-end*.

4.3 Formato do pacote

O formato do pacote usado pela interface segue o padrão definido por [6], sendo subdividido em camada de enlace, rede, interface e de sistema. O cabeçalho do pacote inicia com a camada de rede, seguido pela camada de interface. A camada de sistema compõe a carga útil do pacote (*payload*) e possui uma quantidade variável de *flits*. Foi adicionado um *flit* abaixo da camada de sistema que representa o *trailer*.

O formato do pacote proposto é ilustrado na Figura 6. Os primeiros dois *flits* do pacote representam a camada de rede e possuem os endereços X/Y de destino e origem do pacote, respectivamente. O terceiro *flit* representa a camada de interface, que contém as informações da transação vindas do núcleo. Os campos da camada de interface foram elaborados com base nos seguintes parâmetros que caracterizam as transações:

- Operação: Escrita (WR) ou Leitura (RD);
- Tipo: Requisição (REQ) ou resposta (RESP);
- Status: Status de resposta da transação de escrita ou leitura;
- Agrupamento: Rajada fixa (FIXED), rajada incremental (INCR), rajada modular (WRAP) ou sem rajada (transação simples);
- Tamanho do *payload*: Quantidade de *flits* no *payload*; e
- Identificador: Número que identifica a transação unicamente.

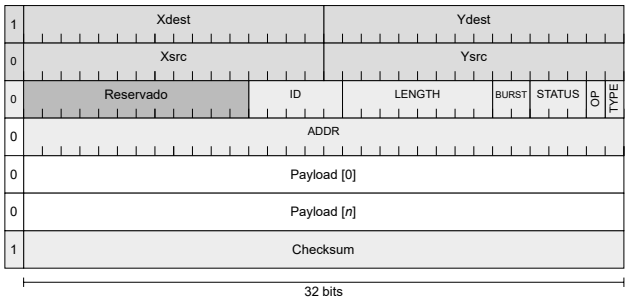

Figura 6: Formato de pacote proposto.

4.4 Organização interna

A Figura 7 ilustra o diagrama de blocos do *back-end*, junto de uma visão geral da organização da interface. Internamente, o *back-end* é composto de um empacotador, desempacotador, controladores de fluxo, codificador e decodificador Hamming e *buffers* FIFO para armazenamento temporário dos pacotes. O módulo TR (tabela de roteamento) é responsável por transformar o endereço do núcleo destinatário em um par de valores X/Y, que são utilizados posteriormente pelos roteadores da rede. O módulo CF (controlador de fluxo) é responsável por controlar o envio e recebimento de *flits* para o roteador, implementando as mesmas técnicas de fluxo adotadas pela rede. Por último, o módulo CI (controlador de integridade) é responsável por computar o *checksum* do pacote.

As técnicas TMR e codificação Hamming foram escolhidas para prover tolerância a falhas para a interface. O TMR é aplicado em todos os controladores (empacotador, desempacotador, de fluxo e de integridade). A codificação Hamming é aplicada pelo controlador Hamming (CH) em cada *flit* escrito nos *buffers* FIFO, e a decodificação (DH) é aplicada para cada *flit* lido dos *buffers* FIFO.

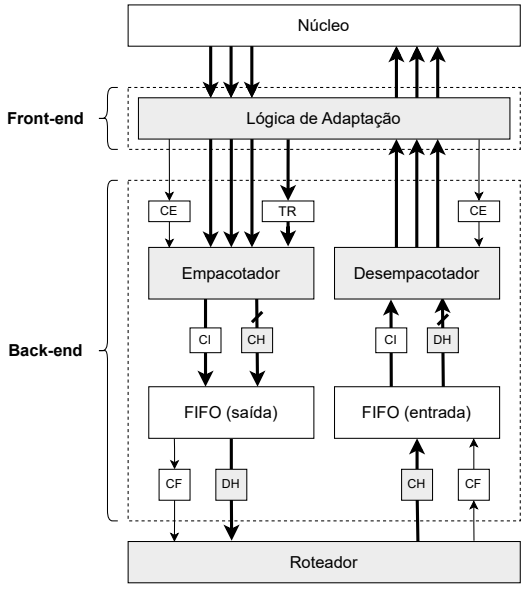

Figura 7: Organização interna da interface de rede.

5 RESULTADOS

Nesta Seção, são apresentados os resultados da síntese e simulação da interface, com posterior discussão dos resultados da integração da interface com os núcleos AMBA-AXI existentes.

5.1 Síntese

Para obter os custos das interface em elementos lógicos, as versões mestre e escravo foram sintetizadas na ferramenta Quartus Prime Lite Edition 22.1std. A síntese teve como dispositivo alvo o FPGA 5CGXFC9E7F35C8 da família Cyclone V. A Tabela 1 apresenta um comparativo da distribuição de recursos das duas versões da interface (I/F) com e sem as técnicas de confiabilidade aplicadas, considerando um roteador XINA em configuração padrão.

Componente	LUTs	FFs	Fmax (MHz)
I/F mestre	359	788	208,25
I/F mestre (confiável)	(+63%) 586	(+16%) 918	(-56%) 112,57
I/F escravo	449	788	179,95
I/F escravo (confiável)	(+44%) 651	(+15%) 913	(-44%) 102,17
Roteador XINA	812	484	145,18

Tabela 1: Resultados de síntese.

Em comparação à um roteador da rede XINA, a versão mestre corresponde a 44% da quantidade de LUTs, enquanto a interface escravo corresponde a 52%. A interface mestre suporta uma frequência máxima 43% maior, enquanto a interface escravo suporta um frequência máxima 23% maior. Porém, ambas as versões da interface necessitam de uma quantidade de FFs 62% maior. Essa quantidade maior de FFs é causada pela profundidade padrão dos *buffers* da interface, que é igual à 10. Enquanto isso, os *buffers* de um roteador XINA padrão possuem profundidade igual à 4. Esse valor maior de profundidade foi escolhido porque ele evita gargalos na interface ao enviar pacotes com múltiplos *flits* de carga útil.

5.2 Verificação

A fim de verificar o comportamento da interface, foram criadas simulações para as diferentes versões da interface. As simulações foram executadas após a síntese por meio da ferramenta Questa - Intel FPGA Starter Edition 2021.2.

Hamming

Para verificar a eficácia dos *buffers* Hamming, foi injetada uma falha no *testbench* `tb_master_injection_write`. A Figura 8 ilustra o diagrama de ondas do *testbench* após injetar a falha. A simulação inclui os seguintes eventos:

(1) Ciclo 2 (150ns): O *flit* de endereço de destino do pacote é salvo no *buffer* de injeção.
(2) Injeção de falha (200ns): Nesse instante, uma falha é injetada no *flit* de endereço de destino. O último bit é invertido (forçando o sinal em simulação), trocando o *flit* de `53BBBBBBBB` para `53BBBBBBBA`.
(3) Ciclo 3 (250ns): Mesmo com o dado corrompido dentro do *buffer*, o flit que está entrando na rede (presente no sinal `l_in_data_i`) ainda é `53BBBBBBBB`. Isso significa que o *buffer* corrigiu o erro de bit único quando o *flit* foi lido pelo roteador da rede.

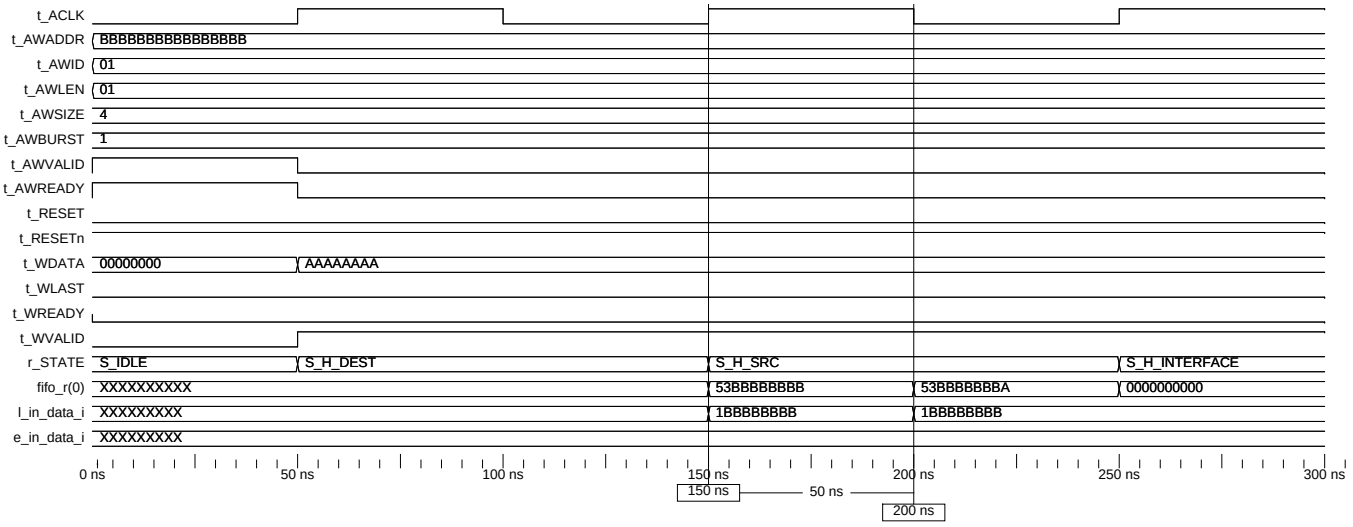

Figura 8: Verificação do funcionamento da codificação Hamming.

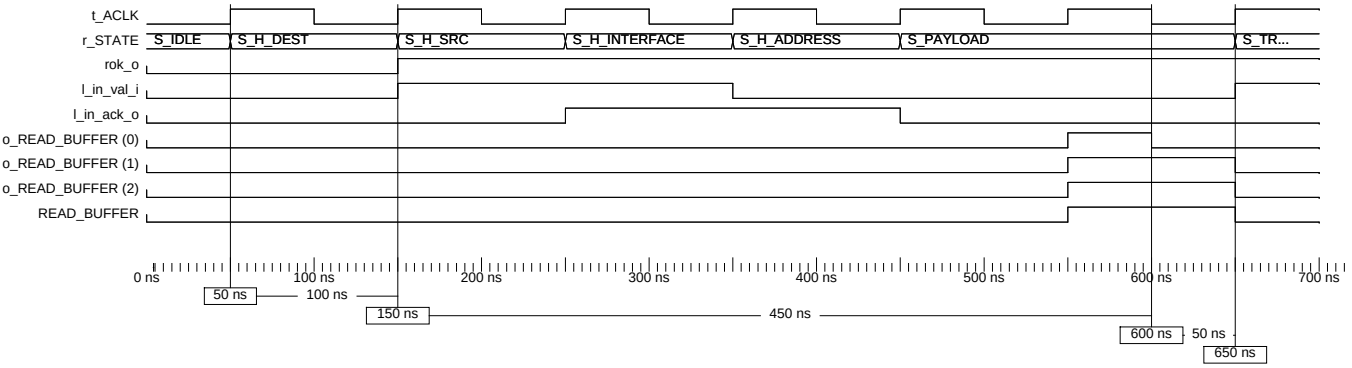

Figura 9: Verificação do funcionamento do TMR.

TMR

Para verificar a eficácia do TMR nos controladores, foi injetada uma falha no controlador de fluxo de envio no *testbench* `tb_master_injection_write`. A Figura 9 ilustra o diagrama de ondas do *testbench* após injetar a falha. A simulação inclui os seguintes eventos:

(1) Ciclo 1 (50ns): O empacotador muda para o estado `S_H_DEST`. Esse estado é responsável por escrever o *flit* de endereço de destino no *buffer* de injeção.

(2) Ciclo 2 (150ns): O *flit* de endereço de destino é escrito no *buffer*. Nesse instante até o instante 450ns, o controle de fluxo realiza o *handshake* com a rede para enviar o *flit* (por meio dos sinais `l_in_val_i` e `l_in_ack_o`).

(3) Injeção de falha (600ns): Nesse instante, o *flit* de endereço de destino já foi enviado para a rede e o controle de fluxo irá consumir o *flit* de destino do *buffer*. O sinal responsável por consumir *flits* do *buffer* é o `READ_BUFFER`. O sinal vindo do primeiro controlador interno (`o_READ_BUFFER (0)`) é invertido (forçando o sinal em simulação), enquanto os sinais dos outros dois controladores permanecem iguais.

(4) Ciclo 7 (650ns): Mesmo após a inversão do sinal do primeiro controlador, o sinal `READ_BUFFER` ainda é igual à 1, o que corresponde à maioria. Isso significa que a falha no primeiro controlador foi mascarada corretamente.

Checksum

Para verificar a eficácia do *checksum*, foi injetada uma falha no pacote de requisição recebido pela interface escravo no *testbench* `tb_slave_write`. O ID presente no *flit* de interface do pacote foi modificado de 1 para 3 por meio de uma inversão de bit. Isso causa uma mudança de *checksum* de `0x599B1A37` para `0x599C1A37`. A Figura 10 ilustra o diagrama de ondas do *testbench* após injetar a falha.

Além dos eventos originais do *testbench*, no instante 4250ns a interface ativa o sinal `CORRUPT_PACKET`, que indica que o pacote recebido possui um *checksum* diferente ao *checksum* computado pela interface. O núcleo escravo poderia usar essa informação para desfazer os efeitos causados pela transação e realizar uma resposta de escrita com código de resposta apropriado (como `SLVERR`, `DECERR` ou `DEFER`).

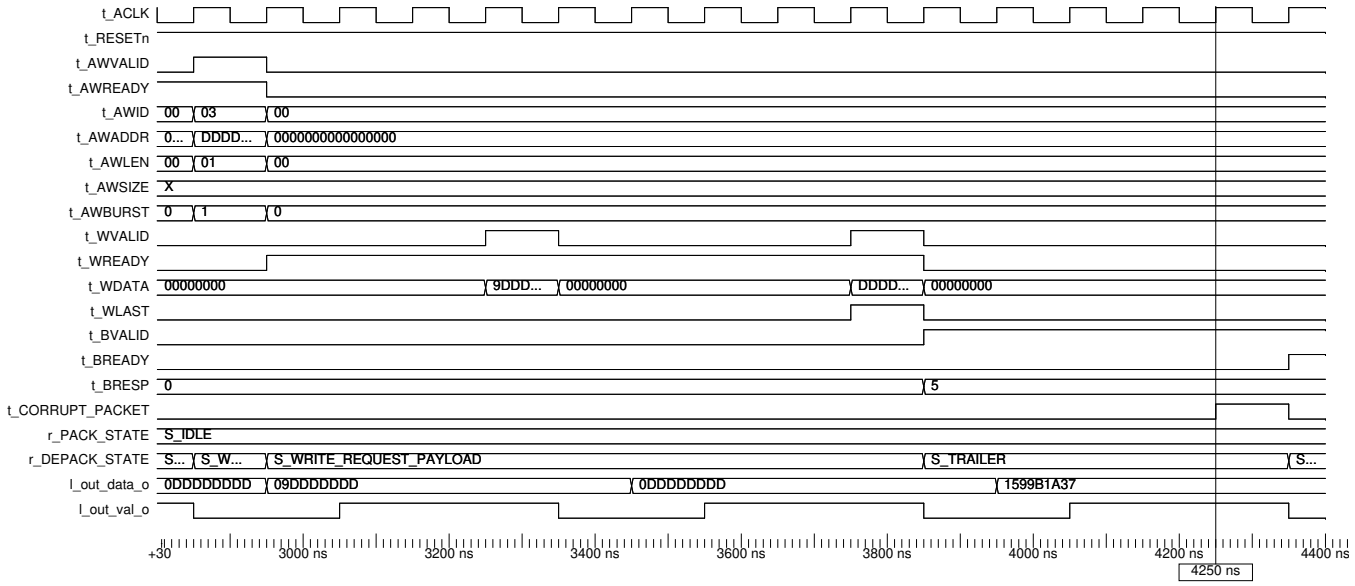

Figura 10: Verificação do funcionamento do *checksum*.

5.3 Integração com compressor CCSDS 123

Foi realizada uma integração com o compressor CCSDS 123 implementado por [14]. O CCSDS 123 é um algoritmo de compressão para imagens hiperspectrais [15], as quais são usadas para coletar imagens da Terra e fornecer dados climáticos [16]. O compressor implementa o padrão 123.0-B-2 sem perda (*lossless*) do algoritmo, além do protocolo AMBA-AXI 4 para fácil integração em SoCs.

A lógica interna de compressão do núcleo possui quatro entradas: `t`, `z`, `sample` e `neighbour`. Para fins de teste, foram definidos quatro valores fixos para essas entradas, como mostra a Tabela 2. Por último, o compressor possui uma única saída de 16 bits que é lida por meio de transações de leitura.

Entrada	Tamanho	Valor de teste
`t`	8 bits	70
`z`	3 bits	2
`sample`	16 bits	809
`neighbour`	16 bits	12330

Tabela 2: Configurações do compressor CCSDS 123.

O sinal de *start* que inicia a compressão internamente é mapeado para o primeiro bit da primeira posição de uma memória interna. Visto que as transações de escrita escrevem para essa memória interna, para que a compressão inicie, basta que o mestre escreva o valor 1 por meio de uma transação de escrita.

Para realizar a integração, o compressor foi conectado à rede por meio de uma interface escravo. Na rede também foi conectado um mestre responsável por enviar uma requisição de escrita e uma de leitura para o compressor. O *testbench* criado (`tb_integration_compressor`) é separado em duas etapas: uma requisição de escrita que inicia a compressão e uma requisição de leitura que lê o resultado da compressão.

A Figura 11 mostra o diagrama de ondas do compressor recebendo a requisição de escrita. Essa etapa compreende o período entre 3250ns e 4550ns, e inclui os seguintes eventos:

(1) Ciclo 33 (3250ns): A interface inicia a transação de escrita por meio do sinal `AWVALID`. A transação possui ID igual à 1, escreverá no endereço 0 e transmitirá somente uma palavra.
(2) Ciclo 37 (3650ns): A interface realiza a única transferência da transação por meio do sinal `WVALID`. O dado à ser escrito é o número 1, que iniciará a compressão.
(3) Ciclo 38 (3750ns): O valor 1 é escrito na memória local do compressor (sinal `mem_data_out`). No próximo ciclo, a compressão é iniciada.
(4) Ciclo 42 (4150ns): O compressor realiza a resposta da transação de escrita com status igual à 0 (`OKAY`).
(5) Ciclo 46 (4550ns): O resultado da compressão (`0x00005A02`) é armazenado em um registrador local.

Após a compressão, basta que o mestre realize uma transação de leitura para ler o resultado. A Figura 12 mostra o diagrama de ondas do mestre recebendo a resposta do compressor. O diagrama possui somente um evento, que é no instante 13550ns, onde o mestre recebe uma resposta de leitura com dado igual à `0x00005A02`, que é idêntico ao resultado computado internamente no compressor.

6 CONCLUSÃO

Neste trabalho, foi explorada a necessidade de arquiteturas de comunicação mais robustas devido ao aumento constante de componentes em sistemas integrados. O foco foi na rede-em-chip XINA (eXtensible Interconnect Network Architecture), uma arquitetura intra-chip confiável para sistemas integrados tolerantes a falhas.

Para efetivamente implementar a XINA em sistemas existentes, desenvolveu-se uma interface de rede compatível. Esta interface, baseada no padrão de comunicação AMBA-AXI 5, fornece serviços essenciais, como interfaceamento, empacotamento, desempacotamento e transferência em rajadas, além de detecção de erros por

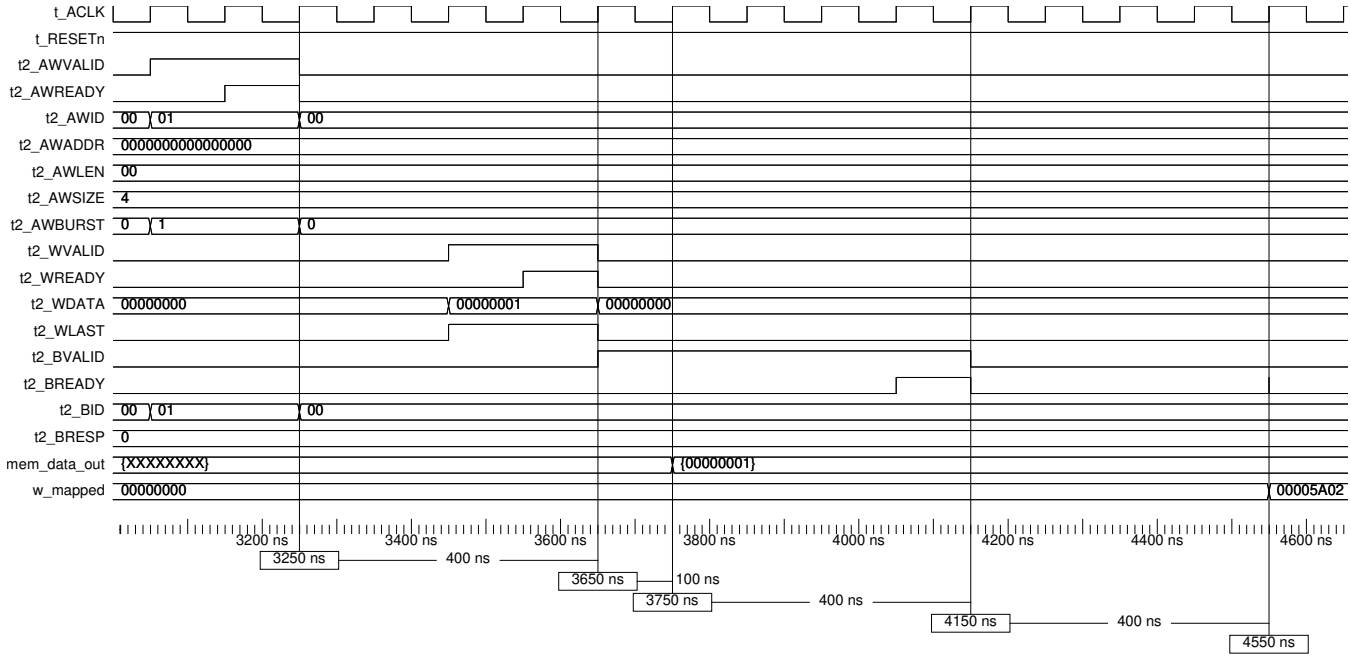

Figura 11: Verificação do compressor CCSDS 123 recebendo a requisição de escrita.

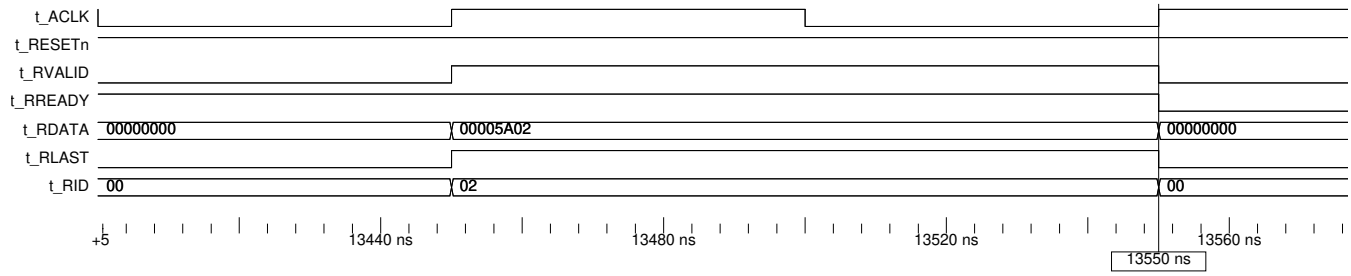

Figura 12: Verificação do mestre recebendo a resposta do compressor CCSDS 123.

meio de *checksum*. Técnicas de tolerância a falhas, como redundância espacial (TMR) e redundância de informação (código Hamming) foram incorporadas.

Como perspectivas futuras, sugerem-se modificações na interface para suportar outros padrões de comunicação, a substituição do *checksum* por algoritmos como o CRC (Cyclic Redundancy Check) e a realização de uma campanha de injeção de falhas para uma avaliação mais aprofundada da confiabilidade da interface desenvolvida.

REFERÊNCIAS

[1] Rajesh Gupta and Y. Zorian. Introducing core-based system design. *Design Test of Computers, IEEE*, 14:15 – 25, 11 1997. doi: 10.1109/54.632877.

[2] G. De Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini. Networks on chips: From research to products. In *Design Automation Conference*, pages 300–305, 2010. doi: 10.1145/1837274.1837352.

[3] Axel Jantsch and Hannu Tenhunen. Networks on chip. 2003.

[4] Chris Baraniuk. The computer errors from outer space, 2022. URL https://www.bbc.com/future/article/20221011-how-space-weather-causes-computer-errors.

[5] Fernanda Kastensmidt and P. Rech. *FPGAs and Parallel Architectures for Aerospace Applications*. 01 2016. ISBN 978-3-319-14352-1. doi: 10.1007/978-3-319-14352-1.

[6] Douglas Melo, Cesar Zeferino, Luigi Dilillo, and Eduardo Bezerra. Maximizing the inner resilience of a network-on-chip through router controllers design. *Sensors*, 19(8):5416, 2019.

[7] S. Kundu and S. Chattopadhyay. *Network-on-chip: The next generation of system-on-chip integration*. 01 2014. doi: 10.1201/b17748.

[8] Cesar Albenes Zeferino. Redes-em-chip: arquiteturas e modelos para avaliação de área e desempenho. 2003.

[9] Luca Benini and Giovanni De Micheli. Networks on chips - technology and tools. In *The Morgan Kaufmann series in systems on silicon*, 2006.

[10] Douglas Melo, Michelle Wangham, and Cesar Zeferino. Xiru: Interface de rede extensivel para integração de núcleos a uma rede-em-chip. *Revista de Informática Teórica e Aplicada*, 21(2), 2014.

[11] Fernando Luiz Gaya. Interface de comunicação amba-ahb para integração de núcleos em uma rede-em-chip. 2016.

[12] Thalyson Zatt Silva. Implementação e análise de técnicas de tolerância a falhas na interface de rede xiru. 2017.

[13] Rakotojaona Nambinina, Daniel Onwuchekwa, Sabikun Nahar, Darshak Sheladiya, and Roman Obermaisser. Extension of the lisnoc (network -on-chip) with an axi-based network interface. In *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 682–686, 2022. doi: 10.1109/ICCMC53470.2022.9753813.

[14] Wesley Grignani, Douglas A. Santos, Luigi Dilillo, Felipe Viel, and Douglas R Melo. A low-cost hardware accelerator for ccsds 123 lossless hyperspectral image compression. In *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2023.

[15] CCSDS. The consultative committee for space data systems (ccsds), 2023. URL https://public.ccsds.org/.

[16] Lingli Zhu, Juha Suomalainen, Jingbin Liu, Juha Hyyppä, Harri Kaartinen, and Henrik Haggrén. *A Review: Remote Sensing Sensors*. 05 2018. ISBN 978-1-78923-108-3. doi: 10.5772/intechopen.71049.