UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ARTUR BERNARDO MALLMANN

EXTRAÇÃO DE DADOS E MONITORAMENTO DE EVENTOS NA REDE INTRA-CHIP BRNOC

CHAPECÓ
2021

ARTUR BERNARDO MALLMANN

EXTRAÇÃO DE DADOS E MONITORAMENTO DE EVENTOS NA REDE INTRA-CHIP BRNOC

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Dr. Luciano Lores Caimi

CHAPECÓ
2021

Mallmann, Artur Bernardo

Extração de dados e Monitoramento de eventos na rede intra-chip BrNoC / Artur Bernardo Mallmann. – 2021.
63 f.: il.

Orientador: Dr. Luciano Lores Caimi.
Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2021.

1. Depuração. 2. MPSoC. 3. NoC. 4. SoC. 5. BrNoC. I. Caimi, Dr. Luciano Lores, orientador. II. Universidade Federal da Fronteira Sul.

E-mail: arturbmallmann@estudante.uffs.edu.br

ARTUR BERNARDO MALLMANN

EXTRAÇÃO DE DADOS E MONITORAMENTO DE EVENTOS NA REDE INTRA-CHIP BRNOC

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Dr. Luciano Lores Caimi

Aprovado em: 11/11/2021 .

BANCA AVALIADORA

Dr. Luciano Lores Caimi – UFFS

MSc. Adriano Sanick Padilha - UFFS

Dr. Emílio Wuerges - UFFS

AGRADECIMENTOS

Primeiramente, queria agradecer a Deus, por me dar forças e permitir que meus objetivos fossem alcançados, pela saúde e determinação para não desanimar durante a realização deste trabalho.

Agradeço também, ao professor Dr. Luciano Lores Caimi, pelo excelente trabalho realizado como meu orientador e dizer que graças ao seu incentivo e paciência permitiu-me apresentar um melhor desempenho no meu processo de formação profissional. Também aos membros das bancas que muito contribuíram para a execução e aprimoramento deste estudo: MSc. Adriano Sanick Padilha e Dr. Emílio Wuerges.

E finalmente, agradeço a minha família, por terem me apoiado, de forma incondicional, e que compreenderam minhas ausências nos dias dedicados à realização deste trabalho. E principalmente, minha companheira, que sempre esteve ao meu lado durante todo meu percurso acadêmico.

“Don’t Stop Me Now”

(Queen)

RESUMO

Com o aumento contínuo da densidade de transistores nos circuitos integrados, circuitos com maior complexidade se tornam possíveis, tais como os System on Chip (Sistema em Chip) (SoC). Com o crescimento da exigência por dispositivos mais complexos e o avanço da fabricação de chips com múltiplos microprocessadores, surgem os sistemas em chip com múltiplos processadores Multiprocessor System-on-Chip (Sistema Multiprocessado em Chip) (MPSoC). Nos MPSoCs, os barramentos se tornam uma limitação, o que nos trás as redes intra-chip Network on Chip (Rede Intra-chip) (NoC). A Broadcast Network on Chip (Rede de Broadcast Intra-chip) (BrNoC) (WACHTER et al., 2017) é uma rede NoC que, por ser um recente, ainda não conta com uma ferramenta de depuração especialmente desenvolvida. Neste trabalho o objetivo foi a implementação de uma interface gráfica para o acompanhamento de eventos em simulações da rede BrNoC a fim de contribuir para o estudo de redes intra-chip com suporte a *broadcast*. Entre as atividades realizadas no desenvolvimento do trabalho estão: a instrumentalização do código na camada de descrição Hardware com mecanismos de extração dos dados e de captura de eventos; a definição de um padrão de comunicação entre as camadas da ferramenta; e a disponibilidade de uma interface gráfica para análises das informações coletadas. Como resultados da implementação proposta, pode-se observar que, apesar da simplicidade, ela acaba sendo robusta, por não ignorar nenhum sinal e operar, independentemente, do estado das máquinas de estado do roteador, coletando os sinais que tiverem valores alterados, permitindo assim, acompanhar todo o estado da rede e seus roteadores. Em relação ao desempenho da simulação, observa-se que houve um aumento médio no tempo de simulação entre 22.6% e 34.1%.

Palavras-chave: Depuração. MPSoC. NoC. SoC. BrNoC.

ABSTRACT

With the continuous increase in transistor density in integrated circuits, circuits with greater complexity become possible, such as System on Chip (SoC). multiple microprocessors, on-chip systems with multiple processors arise Multiprocessor System-on-Chip (MPSoC). In MPSoCs, buses become a limitation, which brings us Networkon Chip intra-chip networks (NoC). The Broadcast Network on Chip (BrNoC) (WACHTER et al., 2017) is a NoC network that, being a recent one, does not yet have a specially developed debugging tool. In this work, the objective was to implement a graphical interface for monitoring events in simulations of the BrNoC network, in order to contribute to the study of intra-chip networks with Broadcast support. Among the activities carried out in the development of the work are: the instrumentation of the code in the Hardware description layer with mechanisms for extracting data and capturing events; the definition of a communication standard between the tool layers; and the availability of a graphical interface for analyzing the information collected. As a result of the proposed implementation, it can be observed that, despite its simplicity, it ends up being robust, as it does not ignore any signal and operates, regardless of the state of the router's state machines, collecting the signals that have changed values, thus allowing , monitor the entire state of the network and its routers. Regarding simulation performance, it is observed that there is an average increase in simulation time between 22.6% and 34.1%.

Keywords: Debug. MPSoC. NoC. SoC. BrNoC.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

brAll Broadcast to All (Broadcast para todos).
BrNoC Broadcast Network on Chip (Rede de Broadcast Intra-chip).
brTgt Broadcast with a Target (Broadcast com um alvo).
brWt Broadcast without a target (Broadcast sem um alvo).

CAM Content Addressable Memory (Memória de conteúdo endereçável).

flit Flow Control Unit (Unidades de Controle de Fluxo).

HeMPS Many-core Modeling Platform for Heterogenous SoCs(Plataforma de modelagem de SoCs com diversos núcleos).

I-FSM Input finite State Machine (Máquina de Estados de Entrada).
IP Intellectual Property (Propriedade Intelectual).

MPSoC Multiprocessor System-on-Chip (Sistema Multiprocessado em Chip).

NI Network Interface (Interface de Rede).
NoC Network on Chip (Rede Intra-chip).

O-FSM Output Finite State Machine (Máquina de Estados de Saída).

PE Processing Element (Elemento processador).

SoC System on Chip (Sistema em Chip).

SUMÁRIO

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

Com o aumento da densidade de transistores nos circuitos integrados, cada vez mais, sistemas com maior complexidade se tornam possíveis em um único chip. Conforme Pasricha e Dutt (2008), sistemas em chip SoCs são circuitos integrados que consistem de diversos componentes, tais como: processador, memória, dispositivos de entrada e saída, aceleradores, decodificadores, entre outros dispositivos personalizados para atender demandas específicas.

Os SoCs, devido ao fato de incorporarem em um só chip recursos que antes abrangiam uma placa eletrônica completa, tiveram de adotar barramentos para a comunicação interna. Entre os barramentos mais conhecidos estão ARM AMBA (processadores ARM) e IBM CoreConnect (processadores PowerPC e POWER processors) (WOLF; JERRAYA; MARTIN, 2008). Com o aumento da exigência por dispositivos mais complexos e o avanço da fabricação de chips com múltiplos microprocessadores, surgem os sistemas em chip multiprocessados (MPSoC).

De acordo com Benini e De Micheli (2002) antes do uso de múltiplos processadores, os barramentos simples atendiam bem a comunicação entre o processador e os componentes do SoC. Porém, com o advento dos MPSoCs esse tipo de comunicação passa a se mostrar incapaz de atender a evolução destes sistemas em termos de largura de banda, escalabilidade e paralelismo na comunicação. Como alternativa aos barramentos, as redes NoCs (*Network on Chip*) tomam forma. As NoCs substituem as conexões diretas e arbitradas por um elemento mestre, pelo uso de roteadores que se comunicam sem ocupar todo o recurso de comunicação que faz uso. Nas NoCs a comunicação se dá por pacotes e quem fornece conexão entre os demais elementos são os roteadores. O primeiro projeto comercial a usar NoC foi o Sonics SiliconBackplane (WOLF; JERRAYA; MARTIN, 2008).

No Brasil o Grupo de Apoio ao Projeto de Hardware (GAPH) da PUC-RS desenvolve estudos de redes NoCs, entre elas a rede HeMPS (CARARA et al., 2009), Memphis (RUARO, Marcelo et al., 2019) e BrNoC (WACHTER et al., 2017). Sendo a rede BrNoC, objeto de estudo deste trabalho. Conforme Wachter et al. (2017) a BrNoC é uma rede que utiliza broadcast como mecanismo de transmissão, paralela e totalmente desacoplada da NoC de dados, desta forma, possui baixa intrusividade. De acordo com o autor, ela tem como objetivo atender as demandas de segurança, tolerância a falhas e gerenciamento de sistemas.

1.2 PROBLEMATIZAÇÃO

A BrNoC, por ser um trabalho em desenvolvimento, ainda não conta com uma ferramenta de depuração especialmente desenvolvida. Porém, tanto nos MPSoCs quanto nas NoCs ocorrem múltiplos eventos simultaneamente. Isso é devido às características de paralelismo inerente de processamento e comunicação. Estas características são acentuadas em uma rede em broadcast,

devido a propagação de eventos criados por uma mensagem na rede. Neste contexto, acompanhar e depurar as sequências de eventos que se propagam na rede (enlaces, buffer, roteadores, etc) é uma tarefa bastante árdua.

Atualmente, nas simulações da BrNoC, a depuração se dá por análise das formas de onda geradas pela simulação. Para se analisar o funcionamento da rede precisamos estar a par de que instante as requisições ocorrem, qual o comportamento da interface de rede, qual o tempo que os dados levam para trafegar na rede, o que está limitando o desempenho, endereço das instruções carregadas pelos processadores e assim por diante.

Conforme M. Ruaro et al. (2016) para testar, desenvolver, depurar e visualizar o comportamento de uma rede NoC é de extrema importância que se tenha uma ferramenta de depuração de alto nível, que abstraia as informações de baixo nível e assim permita que esforços sejam aplicados em detalhes mais importantes da simulação. O estudo de M. Ruaro et al. (2016), bem como Friederich, Heisswolf e Becker (2014) e Ciordas et al. (2004), servem para elucidar detalhes importantes para o desenvolvimento de uma ferramenta para observação dos componentes.

Dito isto, depois de executado este projeto, tem-se a possibilidade de acompanhar os eventos de forma organizada e transparente na BrNoC, o que trará facilidade no seu uso, contribuindo assim, com o avanço dos estudos que dependem dela, como o trabalho de Caimi e Moraes (2019), que faz uso da mesma para gerenciar as zonas opacas de uma rede segura e fazer o tratamento de falhas.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Implementação de uma interface gráfica para o acompanhamento de eventos em simulações da rede BrNoC a fim de contribuir para o estudo de redes intra-chip com suporte a broadcast.

1.3.2 Objetivos Específicos

- Apresentar a rede BrNoC, demonstrando suas principais características;
- Instrumentalizar o código na camada de descrição Hardware com mecanismos de extração dos dados e de captura de eventos;
- Determinar o padrão dos dados reportados;
- Definir quais eventos serão reportados;
- Criar estruturas de dados para armazenamento e pesquisa dos eventos;

- Desenvolver uma interface gráfica para acompanhamento de eventos.

1.4 JUSTIFICATIVA

No início do século XXI, Benini e De Micheli (2002) já apontavam um caminho sem volta: a necessidade do desenvolvimento das redes intra-chip. Tendo em vista que barramentos começavam a se mostrar limitados em sistemas multiprocessados, conseguindo ainda atender bem a demanda dos SoCs com até, no máximo, cinco processadores. Os sistemas multiprocessados trouxeram vantagens de desempenho e menor consumo de energia, entretanto, além de estudar o paralelismo inerente destes sistemas. Conforme Yoo, Lee e Kim (2008) as NoCs ajudaram a fugir das limitações técnicas das interconexões convencionais. Com o advento do processo de fabricação nanométricas, os barramentos se mostram bastante ineficientes na tarefa de transferir os dados com desempenho satisfatório. Em 2017 vemos um passo importante da intel®, que adotou como topologia de comunicação dos processadores Xeon® de arquitetura Skylake a NoC de topologia em malha (em inglês, mesh).

A rede BrNoC, é uma rede NoC de topologia *mesh*, com o propósito de ser uma rede auxiliar, encarregada de assuntos administrativos (CAIMI; MORAES, 2019). Uma grande carência atualmente no projeto da BrNoC é uma interface gráfica que auxilie na compreensão do seu comportamento. Atualmente, para a análise do seu funcionamento, é necessário que se analise as formas de ondas relativas aos sinais de cada roteador, o que torna a compreensão dos eventos da simulação difícil. Simular variados cenários se torna uma tarefa árdua. A existência de uma ferramenta que supra esta necessidade, se torna essencial.

1.5 METODOLOGIA

Para a instrumentalização do código existente, se fez necessário revisar o conhecimento nas linguagens de descrição de hardware VHDL (IEEE std 1076, 2019) e *SystemC* (IEEE std 1666, 2019) . Para então, conhecer a estrutura de diretórios do projeto, e arquivos que descrevem os principais elementos. Compreender os elementos internos, estrutura de armazenamento, máquinas de estado, sinais e protocolos de comunicação.

Em um segundo momento, definiu-se quais sinais e em qual momento seriam salvos. Também, onde os mesmos seriam capturados, se os eventos da rede seriam registrados em um único arquivo, ou em diversos e a forma que ocorreria a geração dos arquivos de *log*.

O padrão de dados do arquivo então foi definido, de forma que fosse possível implementar sua escrita sem o auxílio de bibliotecas específicas na instrumentalização do hardware. E também com um bom suporte a leitura no lado do cliente gráfico.

Um aspecto muito importante, considerado neste processo, foi que o código de instrumentalização não afetasse diretamente o comportamento do funcionamento da rede BrNoC, e

que mesmo assim, houvesse a mesma robustez de uma ferramenta *white-box*, porém, em uma abordagem *black-box*.

Para o desenvolvimento da aplicação de interface gráfica, a utilização de uma linguagem robusta e com boas bibliotecas se faz necessário. Tanto para o suporte a bons formatos de arquivos para *log*, e principalmente para a criação de janelas e desenho dos componentes da rede e dos roteadores.

Um elemento decisivo para o projeto foi a reconstituição da simulação a partir dos arquivos de *log*. Ainda, quais estruturas de dados serviriam para representar os sinais de tal forma que o desempenho e as funcionalidades fossem atendidas de forma satisfatória.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados alguns termos, conceitos e taxonomia da área de estudo. Inicialmente será abordada a evolução dos mecanismos de interconexão e comunicação entre os componentes dos SoCs, até o surgimento das redes intra-chip.

2.1 SISTEMAS MULTIPROCESSADOS EM CHIP - MPSOC

Conforme Wolf, Jerraya e Martin (2008), dispositivos MPSoC emergiram nas últimas décadas. São SoC com múltiplos processadores que equipam dispositivos móveis, multimídia e comunicação como: celulares, set-top box de tv a cabo, televisores, entre outros. Nestes dispositivos, a comunicação interna do chip pode se dar de diversas formas e topologias. Pasricha e Dutt (2008) mostram cada elemento computacional de um MPSoC sendo definido como bloco Intellectual Property (Propriedade Intelectual) (IP) (Intellectual Property). Onde cada bloco IP tem suas funcionalidades individuais e pode ser reutilizado em diferentes sistemas.

Culler, Singh e Gupta (1998 apud WOLF; JERRAYA; MARTIN, 2008) no final da década de 90, baseados no estado da arte, veem a relação da lei de Moore com a evolução da arquitetura de sistemas multiprocessados, em especial, do modelo de comunicação interno utilizado nos mesmos. De forma geral, se converge ao aumento da quantidade de processadores e memórias interconectados em uma rede, onde a memória compartilhada é unificada e os processadores conversam entre si.

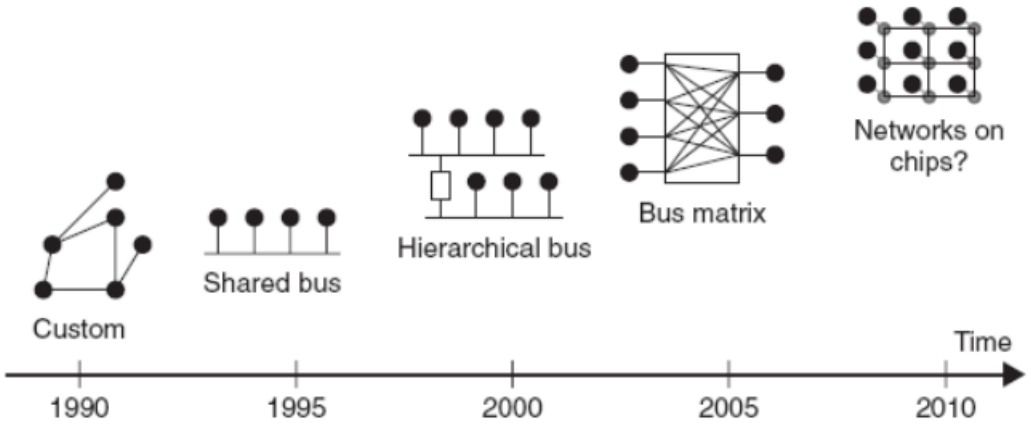


Figura 1 – Evolução da comunicação dos SoCs, Pasricha e Dutt (2008).

Conforme ilustrado na Figura 1, Pasricha e Dutt (2008) já previam que a comunicação intra-chip se daria cada vez mais através de NoCs, onde a comunicação não é mais direta, ficando a cargo dos roteadores levar os pacotes das interfaces de rede de origem até as de destino.

2.1.1 Redes Intra-chip - NoC

Antes do uso de múltiplos processadores, os barramentos atendiam bem a comunicação entre o processador e os demais componentes do sistema aceleradores, memórias, conversores, etc. Porém, com os MPSoCs esse tipo de comunicação acaba restringindo o desempenho e consumindo uma quantidade razoável de energia. As redes NoC (Network on Chip), como alternativa aos barramentos, fazem uso de roteadores para gerenciar a comunicação e o transporte dos dados, o que se dá por meio de pacotes. O primeiro projeto comercial a usar NoC foi o Sonics SiliconBackplane (WOLF; JERRAYA; MARTIN, 2008).

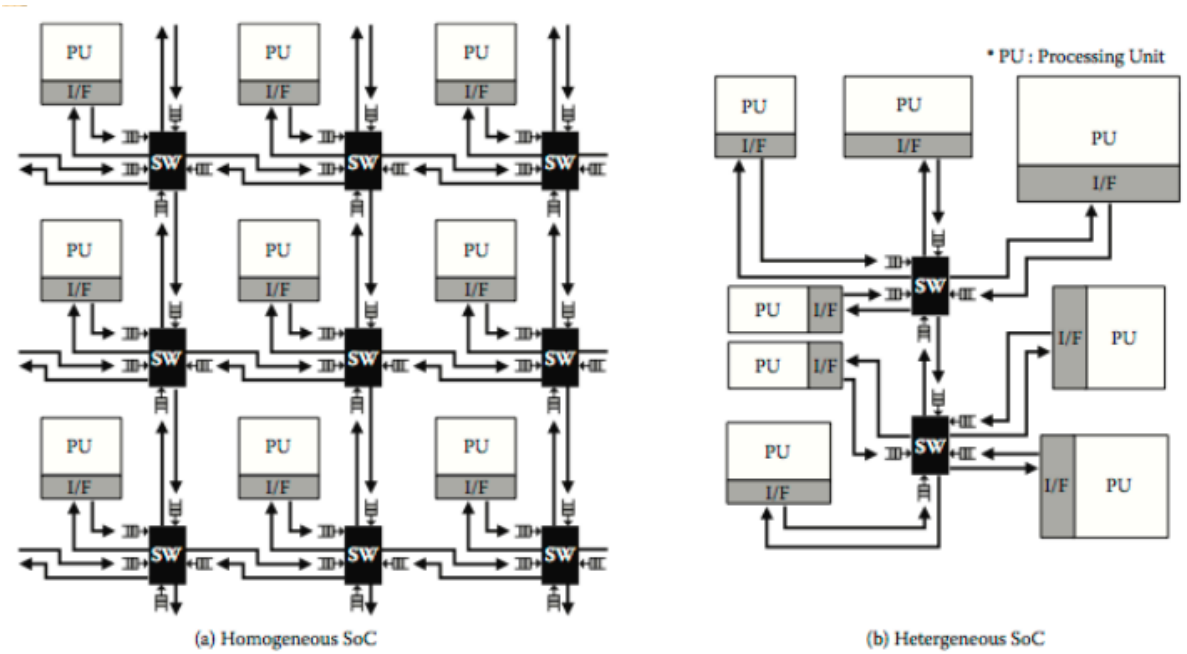


Figura 2 – Topologias das redes NoC reproduzido de Yoo, Lee e Kim (2008).

Conforme Bezerra (2009), as NoCs adaptam aos SoCs os complexos princípios oriundos das áreas de sistemas distribuídos, redes de computadores e processamento paralelo. O uso destes conceitos entre núcleos IP mostram vantagens em relação aos barramentos tradicionais, como: eficiência energética, confiabilidade, reusabilidade, comunicação não-bloqueante e escalabilidade de largura de banda.

Assim como os barramentos, as redes NoC podem ter as mais diversas topologias. Mas de forma mais abstrata pode-se categorizá-las como homogêneas ou heterogêneas. Na Figura 2, vemos duas topologias que se distinguem desta forma (YOO; LEE; KIM, 2008).

O controle de fluxo da NoC, define como os dados transitam na rede. Para entender isso, é importante entender como a informação é estruturada. A Figura 3 mostra a estrutura de uma mensagem. O papel de cada estrutura é explicado por Pasricha e Dutt (2008): as mensagens em NoCs são geradas pelos Processing Element (Elemento processador) (PE)s em forma de pacotes (em inglês, *packets*); estes pacotes são divididos em unidades de controle de fluxo (em inglês, Flow Control Unit - Flow Control Unit (Unidades de Controle de Fluxo) (flit)); os flits

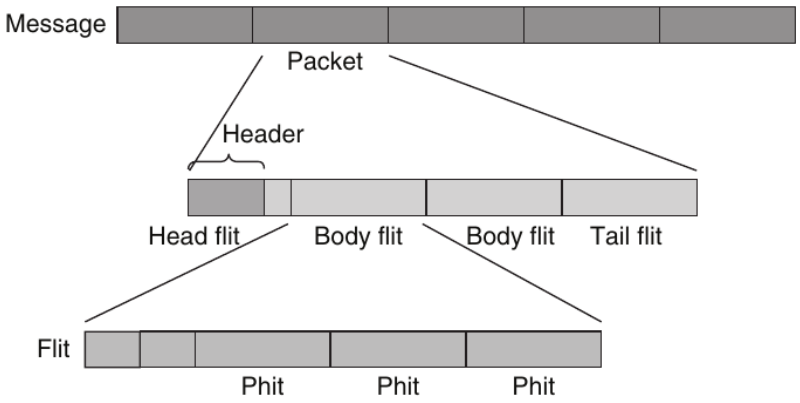


Figura 3 – Representação dos Pacotes de Dados (PASRICHA; DUTT, 2008).

desempenham o papel de controlar a sincronização entre cada roteador; a menor unidade de dados das NoCs é o *phit*, sendo a unidade de dados transferida dentro do tempo de um ciclo.

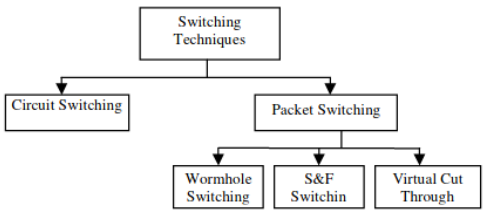


Figura 4 – Controles de fluxo (AGARWAL; SHANKAR, 2009).

Agarwal e Shankar (2009) nos mostram técnicas conhecidas de controle de fluxo (Figura 4). A primeira é a de comutação de circuito (em inglês, Circuit Switching), onde todo o caminho é reservado até que a mensagem chegue ao destino. Do outro lado, temos as comutações por pacotes (em inglês, Packet Switching), que somente reservam os enlaces usados naquele instante. São eles:

- Burraco de minhoca (em inglês, Wormhole): O primeiro flit é o único a sofrer com a latência, os outros flits de um mesmo pacote seguem o caminho do flit de cabeçalho;
- Gravar e Avançar (em inglês, Store & Foward - S&F): Os pacotes apenas são mandados se houver espaço no buffer do PE de destino. Este modelo não necessita de mecanismos para dividir os pacotes em flits;
- Fatia Virtual (em inglês, Virtual Cut Through - VCT): Na transmissão VCT os pacotes são divididos em flits, onde o primeiro flit continua avançando e não espera todos os flits do pacote chegarem. Porém, se não houver espaço para o pacote completo no *buffer* do próximo roteador, todos os flits são mantidos em *buffer* (PASRICHA; DUTT, 2008).

2.2 A REDE INTRA-CHIP BRNOC

Nesta seção será abordada a rede BrNoC (WACHTER et al., 2017). A BrNoC é uma rede intra-chip que utiliza transmissão em broadcast, usada como rede auxiliar para envio de mensagens de controle. Seu objetivo é atender as demandas de segurança, tolerância a falhas e gerenciamento de sistemas. Ela foi concebida para ser totalmente desacoplada da rede NoC de dados, desta forma, tem baixa intrusividade na rede de comunicação principal.

O mecanismo básico de funcionamento da BrNoC se dá da seguinte forma: ao receber uma mensagem por uma das portas, o controle da mesma verifica se a mensagem já está armazenada na memória interna, caso sim, a mensagem recebida é ignorada; caso a mensagem não esteja na memória ela é armazenada e um *flag* interno (flag pending) é ativado indicando que a mensagem foi recebida e precisa ser repassada. Em um segundo momento, a mensagem com o *flag* ativado é colocada em todas as portas de saída do roteador, com exceção da porta que a mesma foi recebida. Após a confirmação de recebimento pelos roteadores vizinhos, a mensagem permanece na memória até que uma mensagem de limpeza da mesma seja recebida pelo roteador. Assim, de forma geral, a mensagem é recebida por uma porta e reenviada por todas as demais portas do roteador, configurando assim a transmissão em broadcast.

Caimi e Moraes (2019) mostram a adoção de uma configuração de dois roteadores (Figura 5). A Figura representa um PE equipado com dois roteadores: o primeiro, um roteador de dados (em inglês, Data NoC Router) com sinais físicos duplicados; o segundo, um roteador de controle (em inglês, Control NoC Router). A rede de dados é uma rede intra-chip tipo *mesh*, com roteamento de pacotes usando o algoritmo XY, controle de fluxo tipo *wormhole* e arbitragem *round-robin*.

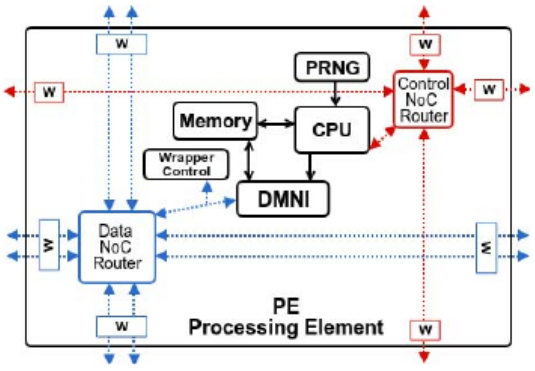


Figura 5 – Representação de um PE equipado com dois roteadores, *Data NoC Router* e *Control NoC Router* por Caimi e Moraes (2019).

A rede de controle (BrNoC) também utiliza topologia *mesh*. Sua finalidade é implementar funções de gerenciamento através de broadcasts. Ela é composta por roteadores com cinco portas full-duplex interconectadas com quatro outros roteadores vizinhos e uma porta local, ligada ao processador local. Os pacotes de mensagem são compostos por flits únicos e a troca delas se dá por meio de um protocolo de *handshake* (req, ack e nack).

A BrNoC possui uma memória tipo CAM (Content Addressable Memory), representada na Figura 6. Ela substitui o *buffer* local de cada porta e seu tamanho é definido no projeto. As entradas e saídas da CAM são acessadas de forma cíclica, o que garante que nenhum canal fique bloqueado. Também vemos na figura as Máquinas de Estado, que fazem o controle dos fluxos de entrada (em inglês, input Finite state Machine - I-FSM) e de saída (em inglês, output Finite state Machine - O-FSM). E por fim, a arbitragem de entrada (em inglês, Input Arbiter) e de saída (em inglês, Output Arbitrer).

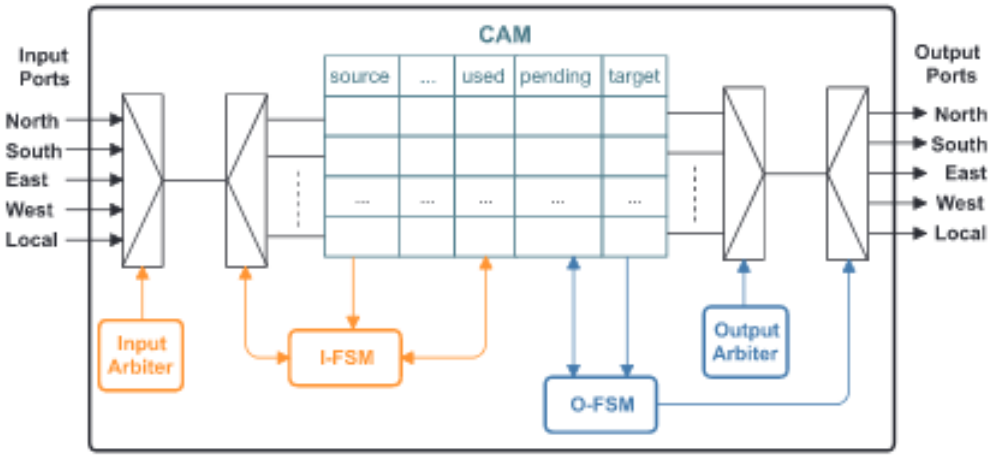


Figura 6 – Memória CAM reproduzido de Wachter et al. (2017).

O procedimento para carregamento na CAM é o seguinte: primeiramente o arbitro escolhe uma porta para receber a mensagem; quando uma requisição chega, a I-FSM verifica se a *TAG* da mensagem já está na CAM e se há espaço livre para armazená-la (e assim o faz), se a mensagem já está na CAM ou se porta está isolada ou com falha; se qualquer uma destas condições for satisfeita, é dado o sinal *ack* para a origem; se a *TAG* não estiver na CAM e não houver espaço a resposta será *nack*.

A O-FSM usa o mesmo protocolo para controlar saída. O árbitro de saída escolhe uma linha, a O-FSM verifica se ela ainda está pendente e encaminha uma requisição (*req*) para seus devidos alvos definidos por *target*. Ele persiste com a mensagem na CAM enquanto houver destinos pendentes, ou seja, respondendo *nack*. Quando todos os destinos responderem *ack* não haverão mais mensagens pendentes e o espaço de memória será liberado.

A CAM armazena em sua estrutura os cabeçalhos de mensagem e o campo de controle. O conteúdo de cada entrada é estruturado conforme a Figura 7.

Wachter et al. (2017) desenvolveram a rede BrNoC para fazer a comunicação de dados de controle, como alternativa a soluções baseadas em multicast. Pelo fato de as transmissões multicast dependerem de algoritmos determinísticos, que fazem roteamento através de caminhos ou árvores, em caso de falhas, poderiam não entregar as mensagens para todos os destinatários. No broadcast, os destinatários são atingidos através do *flooding*, isto é, os pacotes são retransmitidos para todas as saídas do roteador, exceto para a porta de origem.

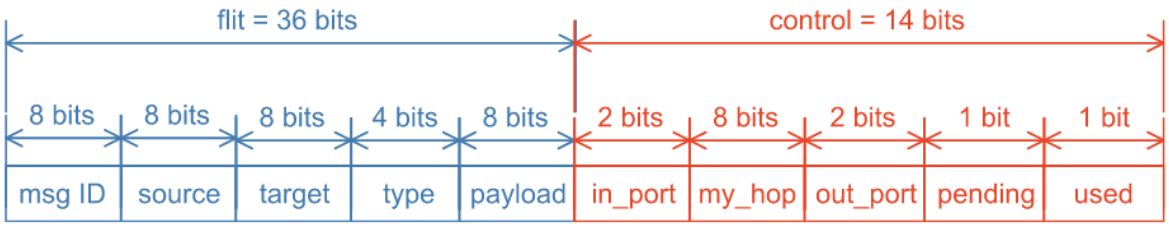


Figura 7 – *Flit* e Memória CAM reproduzido de Wachter et al. (2017).

A BrNoC possui quatro modos de transmissão mensagem, três deles transmitindo para todos (em inglês, broadcast):

- **brTgt**, Broadcast com um alvo (em inglês, Broadcast with a Target - **brTgt**). A mensagem é transmitida em broadcast a todos os PEs da rede, porém apenas o PE alvo a consome. Mensagens que operam neste modo: *BROKEN_PATH*, *SEARCH_PATH* e *END_TASK*.
- **brWt**, Broadcast sem um alvo (em inglês, Broadcast without a target - **brWt**). A mensagem é transmitida em broadcast para todos os roteadores da rede, mas não é consumida por nenhuma das Network Interface (Interface de Rede) (NI)s. Mensagem relacionada ao controle da própria BrNoC, como limpar as estruturas de dados.
- **brAll**, Broadcast para todos (em inglês, Broadcast to All - **brAll**). A mensagem é transmitida para todos os PEs e consumidas por todos eles. Mensagens: *SET_SECURE_ZONE*, *END_SECURE_ZONE* e *START_APP*
- **unicast**, Transmissão direta ou ponto-a-ponto (em inglês, unicast). A mensagem é transmitida diretamente de um PE para outro. É executada como resposta a um **brTgt**, o caminho construído através de *backtracking* do caminho encontrado. Mensagem: *BACKTRACK*.

A seguir serão apresentados exemplos de uso da BrNoC para tolerância a falhas, segurança e gestão de sistema.

2.2.1 Tolerância a falhas

Na Figura 8, vemos a ilustração prática de como a BrNoC executa as operações de tolerância a falhas sobre um rede com zonas interrompidas(indicadas com o símbolo raio). Os PEs 1 e 15 têm sua comunicação interrompida quando o PE 11 sofre uma falha de comunicação, o PE 7 comunica através da mensagem de *BROKEN_PATH*, fazendo broadcast, que teve sua comunicação interrompida com o PE 11.

Para contornar o problema, é disseminada a mensagem *SEARCH_PATH*, que busca caminhos ininterruptos, através de um broadcast disparado a partir do PE 1. Após 6 hops o PE 15 é encontrado. Quando o destino é encontrado, é feita a operação de *BACKTRACK*, dado

por uma comunicação em *unicast*, sobre o caminho válido encontrado, assim a comunicação é restabelecida por este novo caminho (WACHTER et al., 2017).

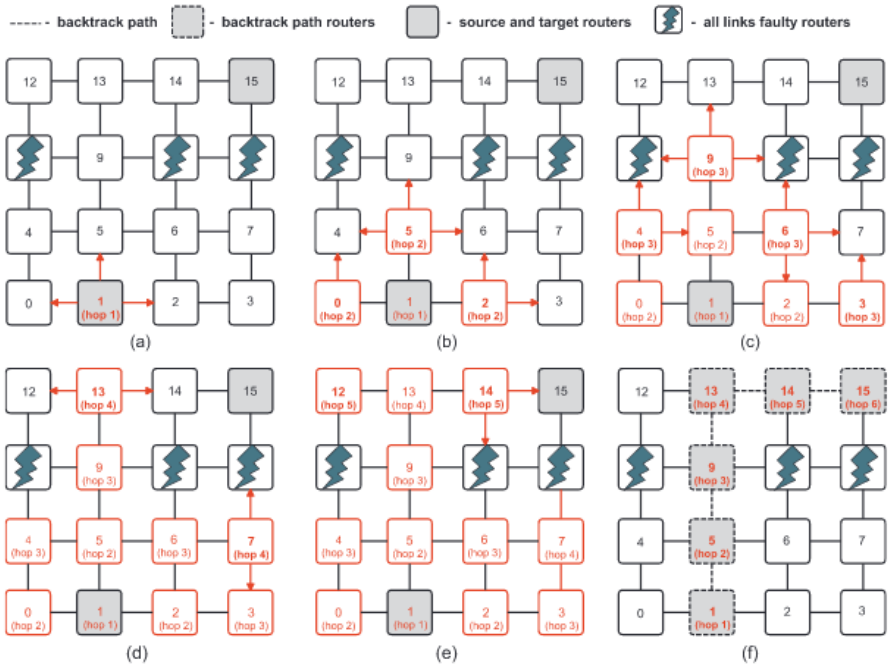


Figura 8 – Tolerância a falhas na BrNoC reproduzido de Wachter et al. (2017).

2.2.2 Segurança

Para aplicações seguras é de extrema importância que outras aplicações não utilizem as regiões reservadas por elas, seja para evitar ataques que inutilizem as suas portas, até ataques que possam comprometer a segurança dos dados. Esta preocupação é abordada por Caimi e Moraes (2019).

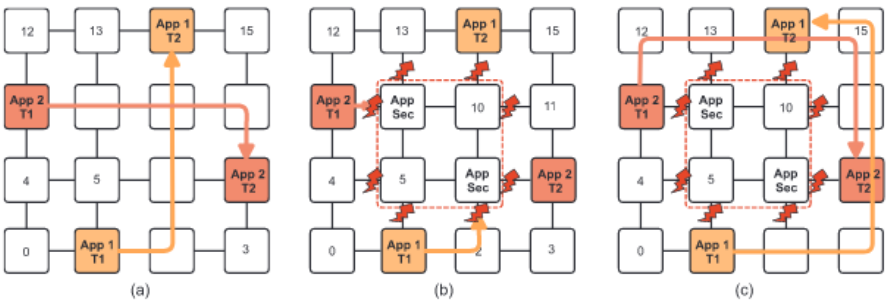


Figura 9 – Zonas seguras na BrNoC reproduzido de Wachter et al. (2017).

A BrNoC utiliza duas mensagens de controle para determinar zonas seguras, uma mensagem para iniciar e outra para finalizar. Ao definir uma zona segura, nenhuma comunicação é permitida de cruzar a zona opaca. Conforme demonstra a Figura 6, onde as aplicações

1 e 2 tiveram de mudar os caminhos por onde se comunicam. Para ativar zonas ocultas *SET_SECURE_ZONE*, para desativar *END_SECURE_ZONE*.

2.2.3 Gestão de sistema

Este é o último tipo de mensagem suportada pela BrNoC. A gestão de sistemas faz uso das mensagens *START_APP* e *END_TASK*. A mensagem *START_APP* envia mensagem de um PE gerenciador para todos os PEs a identificação da nova aplicação a ser iniciada e o alvo, que então fará o seu carregamento. Já a mensagem *END_TASK* é enviada de um PE para um gerenciador a identificação da tarefa a ser finalizada.

2.3 TRABALHOS RELACIONADOS

Nesta seção são apresentados alguns trabalhos sobre monitoramento de eventos e extração de dados em redes NoC. Os quais ajudaram no desenvolvimento da depuração para a BrNoC.

2.3.1 A Data Extraction and Debugging Framework for Large-Scale MPSoCs

Esta seção é destinada a mostrar a implementação da depuração usada na NoC HeMPS (CARARA et al., 2009). Na implementação proposta por M. Ruaro et al. (2016) a ferramenta de depuração é composta por dois elementos principais: o Modelo de Extração de dados e a Interface Gráfica(*GUI*).

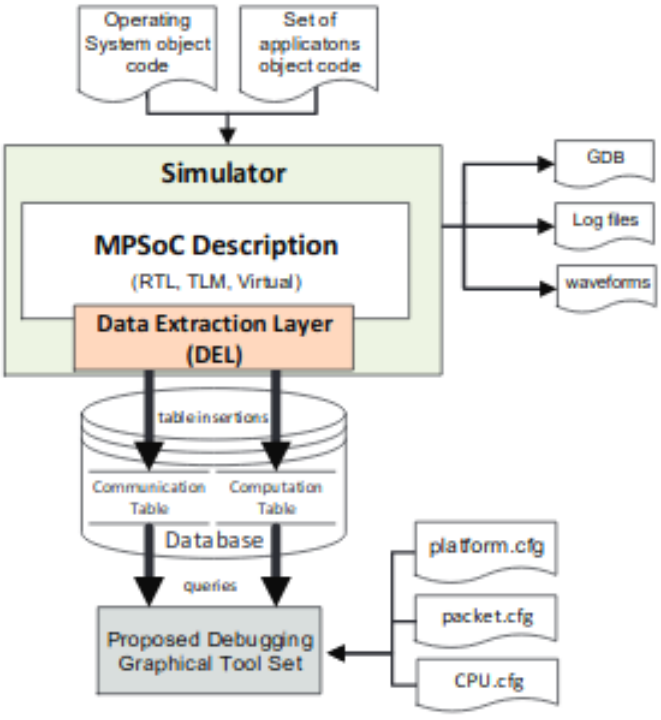


Figura 10 – Diagrama do projeto de M. Ruaro et al. (2016).

A extração dos dados se dá por um modelo de composto por dois elementos: a Camada de Extração de Dados (em inglês, Data Extraction Layer - *DEL*) e o gerenciamento de base de dados e a ferramenta gráfica de depuração (em inglês, Grafical Debugging Toolset - *debugger*). A cada simulação, a *DEL* captura os dados gerados pela simulação e salva em uma nova base de dados, que depois é acessada pela ferramenta gráfica de depuração de dados.

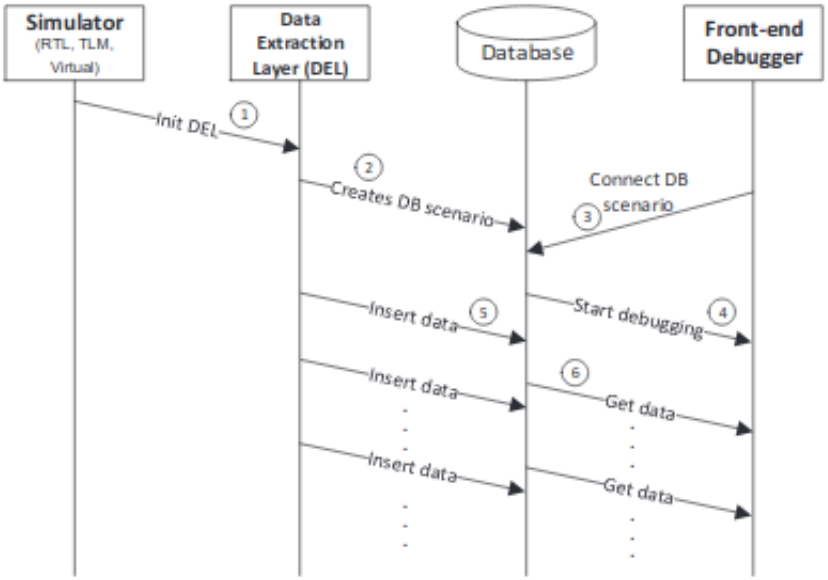


Figura 11 – Diagrama de sequência da gerência do banco de dados (RUARO, M. et al., 2016).

A *DEL* coleta dados de simulação, como latência, taxa de transmissão, e funções desempenhadas, e os grava em um banco de dados. Para isto, ela deve monitorar os dados de comunicação através dos *packets* transferidos e de computação através dos endereços acessados pelo processador. Os *packets* são capturados por *sniffers* nas portas de entrada dos *routers*, já os endereços, com *sniffers* capturando os endereços acessados em cada processador.

A *DEL* é dividida em duas partes: A primeira é implementada na plataforma de hardware de forma não intrusiva, com intuito de monitorar os eventos durante a simulação. A segunda é implementada em *SystemC,* que recebe os sinais e grava os *logs* na base de dados. Se, implementado em uma plataforma descrita em VHDL, necessita-se de um adaptador (em inglês, wrapper).

O *debugger* também tem acesso ao banco de dados, de onde lê os dados de comunicação e computação é expõe a informação para o usuário. Na Figura 11, vemos como ocorre a comunicação entre a *DEL* e o *debugger*. A Figura 12 é um compilado das interfaces gráficas do *debugger*. Nela podemos ver a partir do PE 0x0 (etiqueta 1) a mensagem de alocação de tarefa sendo transferida para o PE 6x4 (2). Depois, na visão de mapeamento de tarefas, ela sendo alocada no PE 6x4 (3). Na visão do processador, ele sai do modo suspenso ao receber uma interrupção (4), para então finalmente o escalonador iniciar a tarefa **p3** (5).

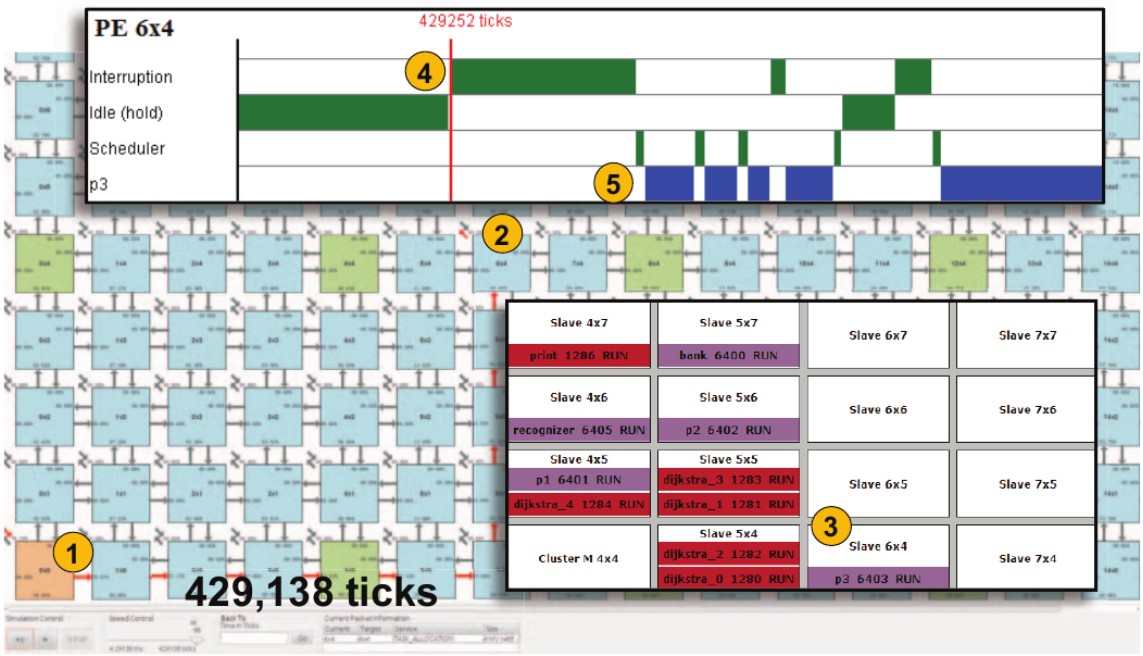


Figura 12 – Elementos gráficos do *debugger* (RUARO, M. et al., 2016).

2.3.2 Hardware/software debugging of large scale many-core architectures

Nesta seção, será discutida uma interface de monitoramento em ambiente prototipado em *FPGA*, proposta por Friederich, Heisswolf e Becker (2014) para a rede *i-NoC* (BENINI; DE MICHELI, 2002). A *i-NoC* (Figura 13), acrônimo para rede intra-chip invasiva (em inglês, Invasive Network-on-Chip), trata-se de uma implementação da infraestrutura de comunicação da arquitetura *InvasIC*, que conta com *routers* organizados na topologia *mesh* e adaptadores de rede (em inglês, Network Adapters - *NA*). O controle de fluxo é *wormhole* e a qualidade de serviço(*QoS*) roda sobre canais virtuais (*VCs*), com conexões ponta-a-ponta.

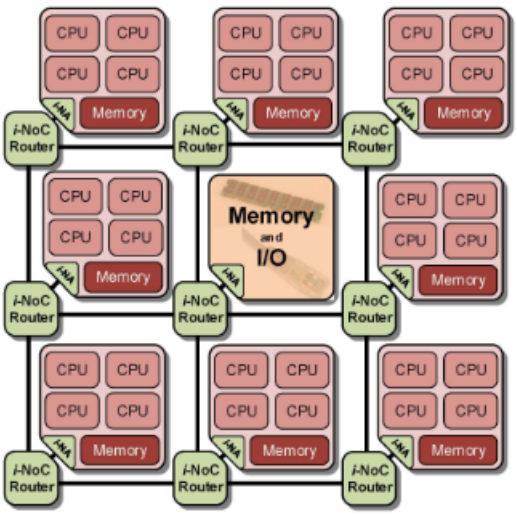


Figura 13 – Rede *i-NoC* por Friederich, Heisswolf e Becker (2014).

Cada interface da rede é conectada diretamente a um *tile*, composto por processadores *LEON*. O barramento utilizado é o *AMBA* Advanced High-performance Bus (*AHB*). Além dos processadores e interfaces de entrada e saída no barramento, também se conecta o módulo Unidade de Suporte a Depuração (em inglês, Debug Support Unit - *DSU*). Abaixo, na Figura 14, vemos a topologia de cada *tile*.

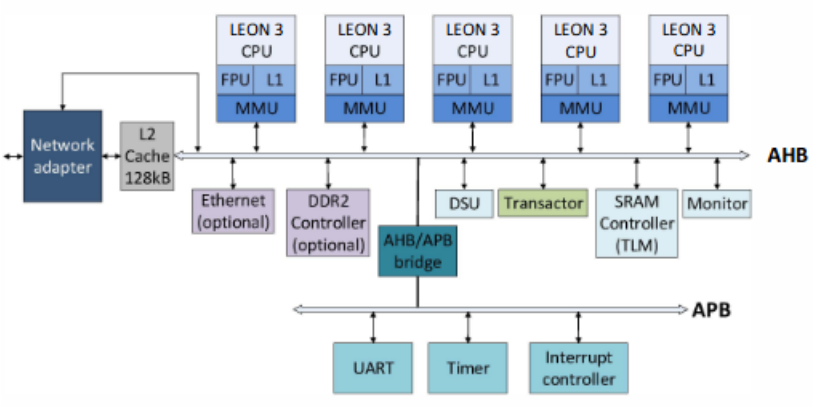


Figura 14 – Arquitetura interna de um *tile* (FRIEDERICH; HEISSWOLF; BECKER, 2014).

A ferramenta de monitoramento dos processadores *LEON* é a *GRMON2*, que é distribuída pela Aeroflex Gaisler (AEOROFLEX. . . , 2014), porém, não foi projetada para NoCs. A *GRMON2* permite: carregar e executar aplicações, gerenciar o buffer, acessar memórias e registradores, etc. O módulo *DSU*s é uma ferramenta de apoio para a depuração e pode ser controlado por qualquer administrador do barramento *AHB*. O *DSU* é o responsável por interconectar a *GRMON2* aos processadores dos *tile*, para tanto, se fez necessário o uso de camadas que adaptam a comunicação pelo protocolo da *UMRbus* ao software e ao master de cada *tile*. Já a depuração da comunicação é feita através de filas de depuração (em inglês, Debug

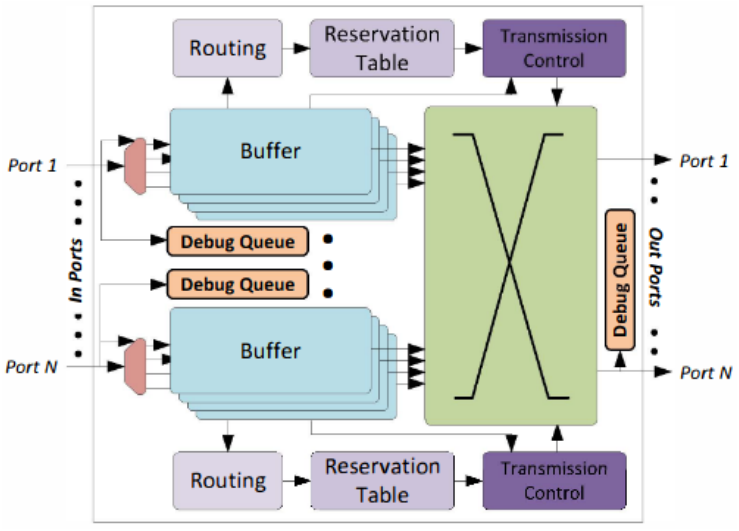


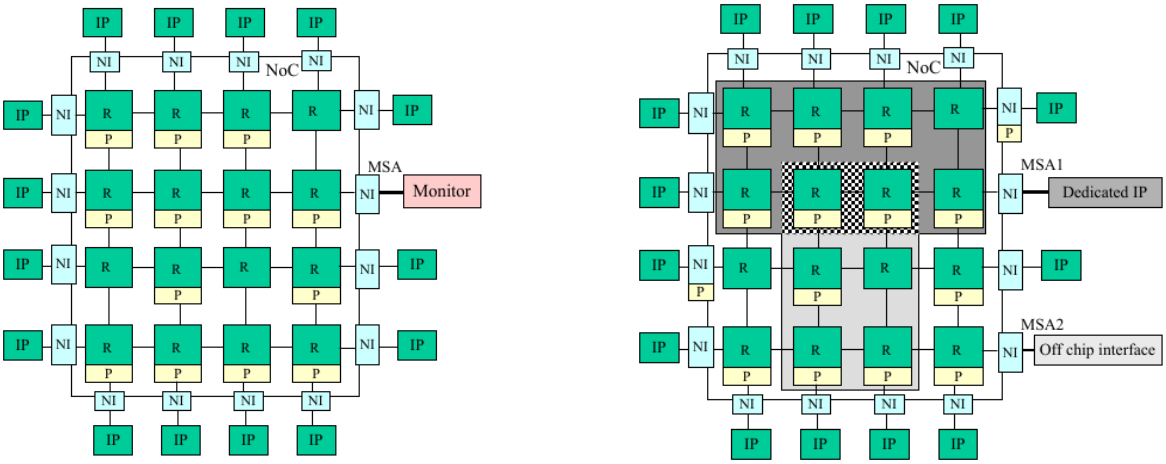
Figura 15 – Diagrama de bloco do *router i-NoC* com depuração por Friederich, Heisswolf e Becker (2014).

Queue), que são registradores *shifters* localizados nas entradas e saídas de cada porta dos *routers*

(Figura 15). Estas filas têm a dimensão definida de acordo com o tamanho do histórico que pretende-se ter. Em caso de erro, a fila é acessível via uma interface mapeada por endereçamento de memória, tal interface é parte da *NA* que é conectada ao barramento *AHB*, portanto acessível pelo *GRMON2*.

2.3.3 An event-based network-on-chip monitoring service

Nesta seção será introduzido o trabalho de Ciordas et al. (2004). Nela é descrita para a NoC *Ætheral* mas que pode ser usada em qualquer NoC. É uma ferramenta de monitoramento baseada em eventos que atende os seguintes requerimentos: escalabilidade, flexibilidade, não intrusão (em inglês, non-intrusiveness), funcionamento em tempo real e baixo custo.



(a) (b)

Figura 16 – (a) Serviço de monitoramento centralizado. (b) Serviço de monitoramento descentralizado (CIORDAS, 2008).

O serviço de monitoramento controla e centraliza os dados coletados pelas sondas de monitoramento, também é utilizado para configurar os tipos de eventos a serem monitorados. Tudo é feito usando da própria infraestrutura da NoC. O serviço pode ser implementado de forma centralizada, em um único IP fixo ou distribuído, e portanto, acompanhar a escalabilidade da rede (Figuras 16a e 16b).

As sondas de monitoramento são utilizadas para capturar os dados que transitam na rede. Elas são anexadas aos *routers* ou NIs. Os serviços disponibilizados são: captura de dados e informações de execução; monitoramento configurável em tempo de execução, que permite abstrair as informações e reduzir o volume de dados; e configuração, controle e monitoramento, usando a estrutura de comunicação da própria NoC.

Como podemos ver na Figura 17, as sondas são compostas por três elementos: *Sniffer*, Gerador de Eventos (em inglês, Event Generator - *EG*) e a Interface de Monitoramento de Rede (em inglês, Monitoring Network Interface - *MNI*).

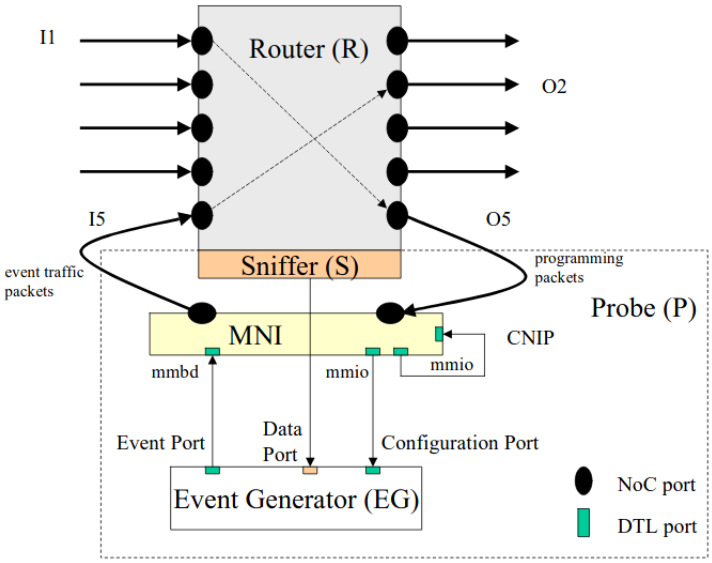


Figura 17 – Arquitetura interna da sonda de monitoramento por Ciordas (2008).

O papel de cada um destes elementos é:

- *Sniffer*, em tradução literal Ferejador: Identificar se a informação que passa pelo *router* é um evento a ser monitorado, se sim, envia para o Gerador de eventos;
- *MNI*: Interface de rede que conecta as portas de entrada e saída do *router* à sonda, por ela, os eventos que programam a sonda, são recebidos. Onde também, os pacotes de eventos são enviados;
- *EG*: Se trata do componente que controla a lógica da sonda, seleciona os eventos importantes recebidos pelo *sniffer*, avalia se esse tipo de evento está configurado para ser coletado e então, repassa a mensagem para a *MNI* enviar os pacotes com os eventos.

3 PLATAFORMA HEMPS, FUNCIONALIDADES, FERRAMENTAS E DEBUG

3.1 DESCRIÇÃO DO AMBIENTE EXISTENTE

O projeto se iniciou com a instalação das dependências e do ambiente de simulação da BrNoC. Depois da instalação do ambiente para simulação, inicia-se a familiarização com a ferramenta e tudo que a compõe. Isso inclui desde a forma que a ferramenta funciona na visão do usuário ao conhecimento da estrutura dos diretórios, códigos fonte, *scripts* etc.

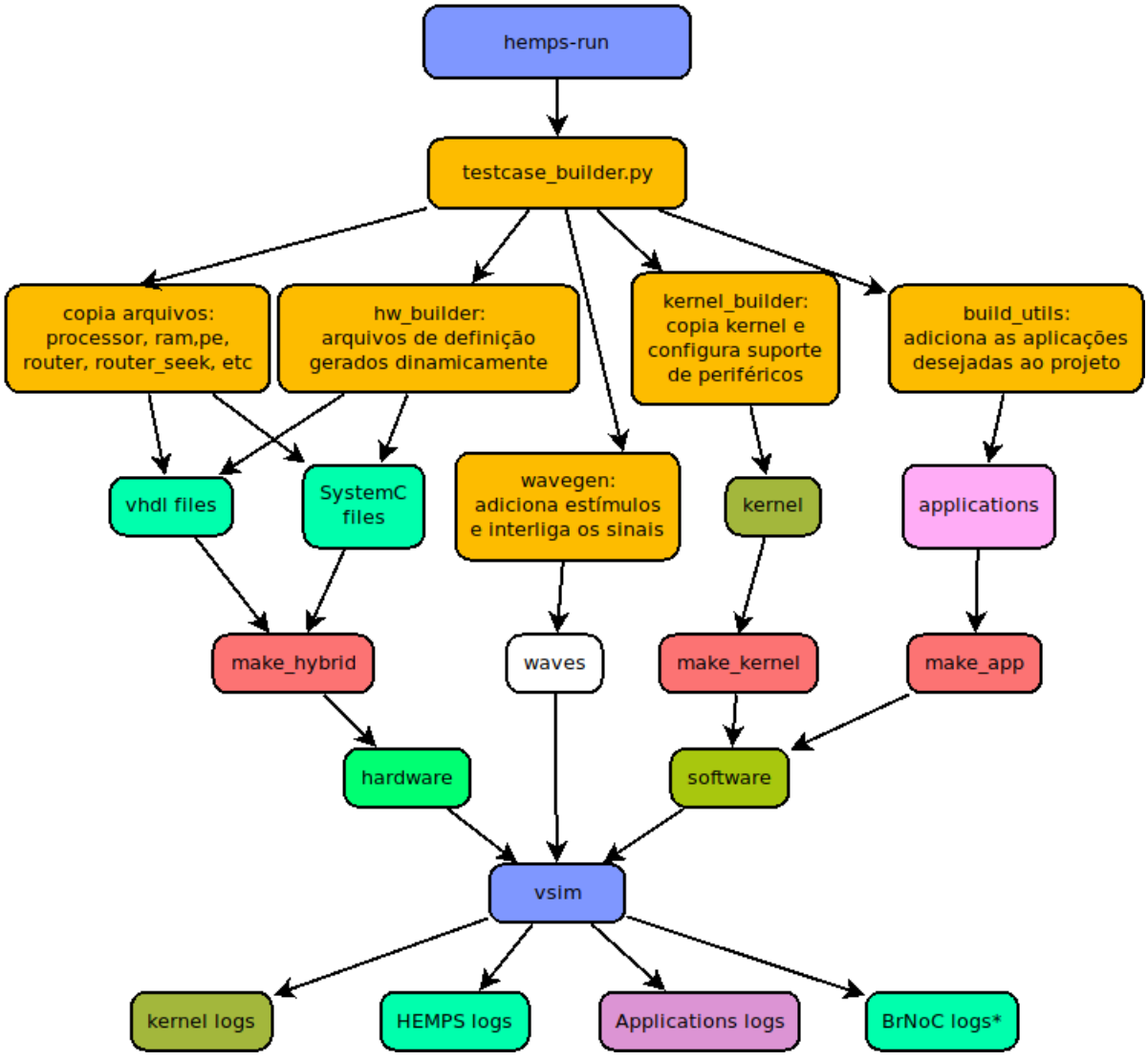


Figura 18 – Visão geral da simulação de *testcases* na HeMPS OSZ.

A versão da BrNoC usada neste projeto vem integrada a HeMPS. O projeto é denominado HeMPS OSZ, devido a sua capacidade de promover zonas seguras. A figura 18 apresenta a cadeia de eventos necessários para a construção de um ambiente de simulação, execução e coleta de dados na mesma. Na figura pode-se observar(na cor bege), os *scripts* para construção do caso de teste (em inglês, testcase). Em tonalidade verde, os componentes relativos ao hardware, os

quais são o foco na etapa de instrumentalização. Mais precisamente o router_seek e módulos que precedem, como PE e router_seek_wrapper.

Na mesma figura(em vermelho), vemos os *makefiles*, responsáveis pela compilação dos componentes. Em tons de rosa, tem-se, a representação das aplicações que rodam na arquitetura e em marrom, o *Kernel* do sistema operacional que hospeda estas aplicações.

A configuração *testcase*, começa pelo carregamento e interpretação do arquivo YAML, que descreve o ambiente e as aplicações no mesmo. O *testcase_builder* recebe esses parâmetros e coordena a chamada dos *scripts* que preparam o ambiente, passando estas informações também para eles. As primeiras tarefas são a preparação dos diretórios e copia para estes os códigos fonte dos principais componentes, tanto de hardware, quanto de software.

A etapa seguinte de preparação é executada pelos *scripts* construtores (em inglês, builders), que são responsáveis pela geração de código dinâmico, baseados nos parâmetros do caso de teste. O tamanho da rede na bancada de testes (em inglês, testbench), a interconexão entre as portas, definição de tamanho de memória, paginação, entre outros atributos de hardware são definidos em arquivos de cabeçalho (SystemC) e packages (VHDL) pelo *script hw_builder* e em arquivos de ondas pelo *wavegen*. Alguns destes atributos dinâmicos do hardware também são necessários serem conhecidos pelo *kernel* e são gerados pelo *kernel_builder*.

3.2 DESCRIÇÃO DO FUNCIONAMENTO DA BRNOC

Entender a BrNoC é parte vital do projeto, no que tange a administração, controle dos eventos administrativos da rede e qualidade de serviço (em inglês, Quality of Service - QoS). Por meio de broadcasts processos são adicionados e removidos, clusters são definidos, zonas seguras são criadas e removidas. Além da criação de rotas de comunicação através de buscas e *backtrackig*. No capítulo dois, especificamente na segunda seção, vemos com mais detalhes o seu funcionamento e modos de operação.

3.2.1 Análise dos sinais e estados das máquinas de estado

Para ajudar no reconhecimento da estrutura de construção dos roteadores, foram feitas duas tabelas de quantitativos de sinais afetados por estado, desta forma os contabilizando. As tabelas seguem anexas ao projeto (apêndices). Elas relacionam estados e sinais, mapeando os instantes em que seus valores são alterados.

Elas abrangem o que tange a sequência de eventos entre a comunicação e os estados das máquinas de estado I-FSM e O-FSM. Estas quais, controlam a entrada e saída dos flits, assim como, o gerenciamento da porta local. Além disso, consegue-se verificar quando são feitas escritas e leituras na tabela *CAM*, que é o elemento de maior importância no controle das mensagens.

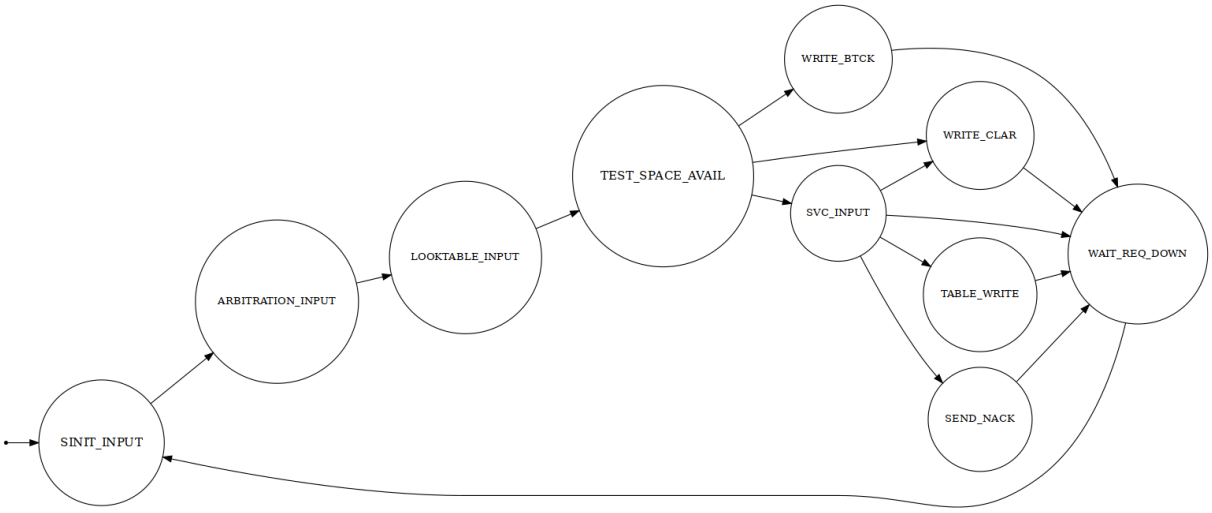


Figura 19 – Diagrama de Estado da I-FSM

As tabelas foram usadas para abstrair os principais sinais para demonstrar o comportamento da rede. Estes sinais são suficientes para mostrar o comportamento da rede, das máquinas de estado e a propagação das mensagens. A abstração resultou em 31 portas monitoradas por cada roteador, e mais dois sinais, que representam os estado da I-FSM e O-FSM.

Para ajudar a entender o funcionamento dos roteadores e suas máquinas de estados e ajudar em estudos futuros futuros, na etapa da construção das tabelas, também foi executado o

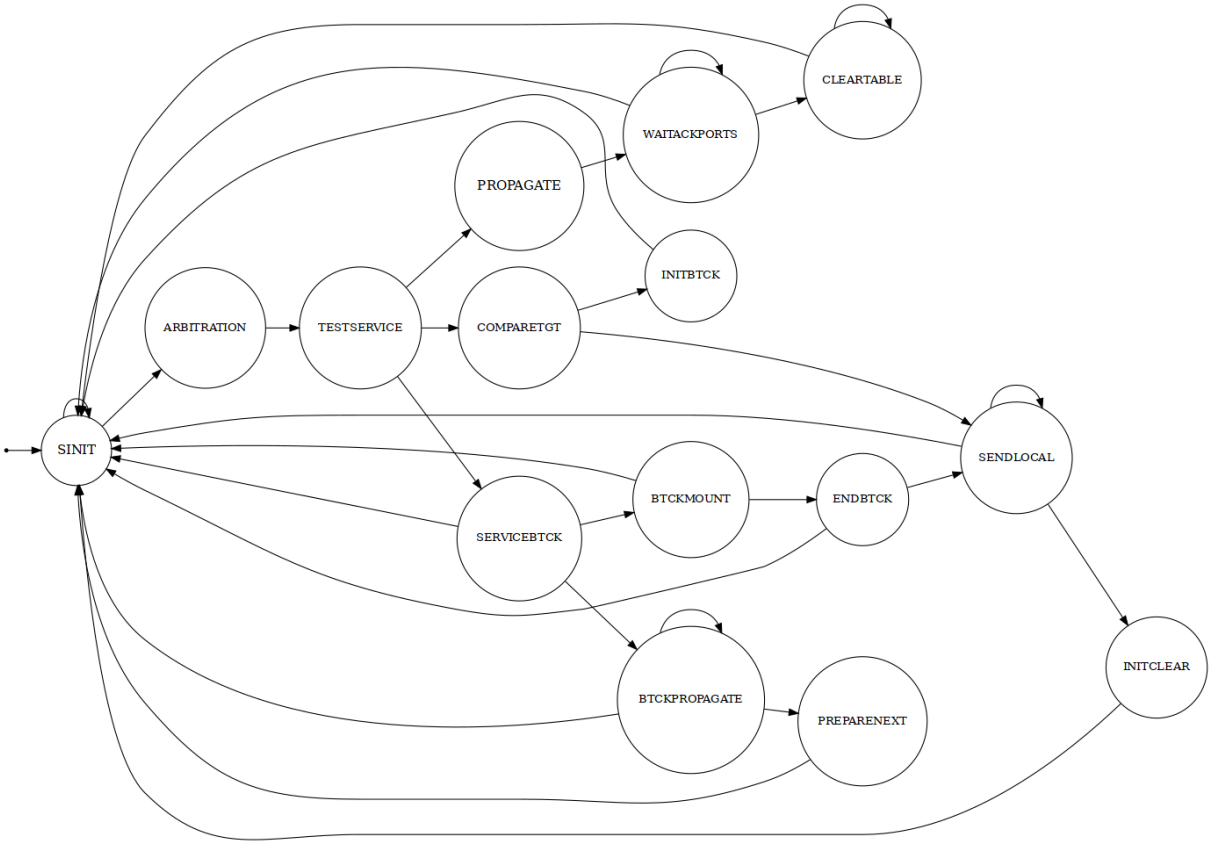


Figura 20 – Diagrama de Estado da O-FSM

reconhecimento das transições. Assim, chegou-se à confecção da representação em diagrama de máquinas de estado. Nestas representações os sinais envolvidos não estão etiquetados, pois são muitos, e a ideia da ferramenta de *debug* é exatamente ajudar na compreensão dos mesmos para fins acadêmicos e de desenvolvimento.

A Figura 19, mostra as transições de estados feitas pela máquina I-FSM, responsável pelo gerenciamento dos pacotes na entrada e escrita na tabela CAM(não são mostrados os eventos responsáveis pela transição). Já na figura 20, temos a representação dos estados da máquina O-FSM, que é responsável pela propagação dos pacotes armazenados na tabela CAM e encaminhamento dos *backtracks*, além da manutenção da tabela CAM.

4 IMPLEMENTAÇÃO DE FERRAMENTA DE DEBUG PARA BRNOC

Neste capítulo é apresentado o desenvolvimento da implementação de uma solução para o auxílio na análise e extração de dados na BrNoC, a fim de contribuir para o estudo de redes intra-chip com suporte a Broadcast.

Atendendo aos objetivos propostos, a implementação iniciou-se com a definição da rede BrNoC e suas principais características. Seguido de uma análise mais aprofundada, sua instrumentalização e os desafios no decorrer destas etapas. Os resultados serão demonstrados no capítulo final, juntamente com a conclusão, objetivando facilitar a visualização dos dados e eventos que ocorrem na rede BrNoC de forma gráfica e intuitiva.

4.1 PROPOSTA DE INSTRUMENTALIZAÇÃO DO CÓDIGO

Após a instalação das dependências, configuração do ambiente de simulação da BrNoC e consequente familiarização com essa ferramenta, iniciou-se a modificação do código existente, começando pelo código referente aos PEs (elemento de processamento). Estes são implementados em *systemC* e representados no módulo PE. Neste, é repassado o sinal de 32 bits referente ao *tick* para dentro dos *router_seek*. Este sinal é necessário no *log* para a localização temporal de cada evento.

Foi adicionado, ao projeto, uma entidade VHDL exclusiva para o monitoramento e construção dos arquivos de *log*. Essa entidade é apenas instanciada em ambiente de simulação (através da anotação *synthesis translate_off*, do vhdl), sendo desativada quando houver a sintetização da BrNoC. Ainda, a entidade é gerada dentro do *router_seek* e todos os sinais, entradas e saídas, são interceptados por ela. A mesma, tem um único *process* que é sincronizado pelo *clock* e estados das duas máquinas, I-FSM e O-FSM. Neste *process*, são comparados os valores de entrada anteriores e atuais. Aqueles que são modificados, naquele instante, são selecionados para gravação posterior no relatório.

A depuração pode ser ativada e desativa para a simulação, tanto nos elementos descritos em SystemC(através do *define* SEEK_LOG), quanto no vhdl (através da definição genérica do valor booleano debug_build).

Algoritmo 1 – Instrumentalização dos módulos existentes - pe.h

```
...
        seek->clock(clock);
        seek->reset( reset );
        // log BrNoC
        #ifdef SEEK_LOG
        seek-> in_tick_counter ( in_tick_counter );
        #endif
...
```

No algoritmo 1, temos as modificações feitas, primeiramente, no arquivo de descrição das unidades de processamento. Onde adiciona-se a ligação do sinal de *tick* com a porta de entrada nova que será adicionada posteriormente ao módulo do roteador.

Algoritmo 2 – Instrumentalização dos módulos existentes - router_seek_wrapper.h

```
...
class router_seek : public sc_foreign_module
{
    public:
      sc_in<bool> clock;
      sc_in<bool> reset;
      #ifdef SEEK_LOG
      sc_in<sc_uint<32>> in_tick_counter;
      #endif
        ...
      router_seek(sc_module_name nm, const char* hdl_name,
          int num_generics, const char** generic_list)
       : sc_foreign_module(nm),
         clock("clock"),
         reset("reset"),
          #ifdef SEEK_LOG
         in_tick_counter("in_tick_counter"),
          #endif
...
```

Os arquivos de código fonte do módulo de tradução de *SystemC* para VHDL (*router_seek_wrapper.h*) também devem ser modificados, para a introdução do sinal de *tick*, que finalmente é conectado ao roteador da BrNoC(*router_seek.vhd*), como podemos no algoritmo 2.

Algoritmo 3 – Instrumentalização dos módulos existentes - router_seek.vhd

```
...
  generic (
      router_address          : regflit
      ; debug_build           : boolean := true -- false para desativar o log
  );
      port(
         clock                          : in  std_logic;
         reset                          : in  std_logic;

         -- log, desativado ao ser sintetizado
         -- synthesis translate_off
         in_tick_counter                : in  std_logic_vector(31 downto 0);
         -- systhensis translate on
...
```

E, finalmente, chegamos no roteador da BrNoC(*router_seek.vhd*), onde as modificações foram mais expressivas, com a introdução do sinal de entrada do *tick* e o atributo genérico *debug_build* para ativação e desativação da depuração (algoritmo 3) e a instanciação do novo módulo *logging.vhd* (algoritmo 4).

Com a introdução da interface de *log*, com os principais sinais do roteador, permite-se o acesso de qualquer um dos sinais internos do roteador da BrNoC.

Algoritmo 4 – Instanciação do módulo logging.vhd - router_seek.vhd

```
...
    -- synthesis    translate_off
    gen_test_count : if (debug_build) generate
      get_signals : entity work.logging
      generic map (
          router_address => router_address
      )
      port map (
          clock => clock,
          reset => reset,
          in_tick_counter => in_tick_counter,
          EA_manager => EA_manager,
...
```

Por fim, temos a implementação da entidade responsável pela construção do relatório. Nela todos os sinais que são recebidos são salvos para posterior comparação. *Procedures*, como a representada no algoritmo 5, fazem o papel de verificar se o sinal analisado foi alterado. Evitando, assim, que o código fique com excessivos *IF's* no *process*. No *process* fica apenas a chamada das *procedures* de cada evento(algoritmo 6).

Algoritmo 5 – Procedure de comparação - logging.vhd

```
procedure log_portas (   linha      : inout line;
                         chave      : in string;
                         pastvalue  : std_logic_vector;
                         realvalue  : std_logic_vector;
                         write_tick : inout bit;
                         size       : integer

) is
begin
    if realvalue /= pastvalue or seek_cycles = 0 then
        format_key_value(chave,'"' & print_port(realvalue, size) & '"', linha);
        write_tick := '1';
    end if;
end procedure;
```

Algoritmo 6 – Chamada das procedures - logging.vhd

```
...
        log_std_logic_vector (linha, "fsm1", past_input_state, input_state, write_tick);
        log_std_logic_vector (linha, "fsm2", past_output_state, output_state, write_tick);
        log_portas (linha, "in_req", past_in_req_router_seek,
            in_req_router_seek, write_tick, NPORT_SEEK);
        log_portas (linha, "in_ack", past_in_ack_router_seek,
            in_ack_router_seek, write_tick, NPORT_SEEK);
        log_integer (linha, "sel_port", past_sel_port, sel_port, write_tick);
...
```

Os estados das máquinas (I-FSM e O-FSM) são enumerações geradas automaticamente com os códigos de identificação de estados que normalmente são imprevisíveis. Portanto, se fez necessário traduzir estes valores de estado para uma forma mais confiável para a leitura. Esta conversão é feita na implementação da arquitetura da entidade do *logging* (algoritmo 7), onde cada nome de estado recebe um valor fixo.

Algoritmo 7 – Identificação dos estados - logging.vhd

```
with EA_manager select
 output_state <=
   "0000" when S_INIT,-- 0
   "0001" when ARBITRATION,-- 1
   "0010" when TEST_SERVICE,-- 2
    ...
```

Para a escrita em arquivo, define-se o local em que este deve se encontrar, para então abrir o mesmo. Cada roteador possui o atributo genérico *router_address*, utilizado para identificar o arquivo correspondente a este. A função, no algoritmo 8, é uma das responsáveis pela construção da *string* de identificação.

Algoritmo 8 – Conversão do valor de endereço do PE para string - logging.vhd

```
function print_router ( vec : std_logic_vector ) -- valor de 16 bits
return string is
    variable output_str : string (1 to 5);
begin
    output_str := CONV_STRING_8BITS(vec(15 downto 8)) & "x" &
        CONV_STRING_8BITS(vec(7 downto 0));
    return output_str ;
end function ;
```

As saídas são traduzidas de sinal lógico para *string*. Como mostra o código, algoritmo 9, para se ter uma *string* resultante de concatenação, acaba-se necessitando o uso de recursões, pois estas não podem ter seu tamanho alterado no escopo.

Algoritmo 9 – Conversão dos valores lógicos para string - logging.vhd

```
function print_port ( vec : std_logic_vector ; size : natural ) -- valor de 16 bits
return string is
    variable output_str : string (1 to size );
begin
    if ( size > 1 ) then
        output_str := print_port (vec( size -1 downto 0), size -1) &
            bit_to_string (vec( size -1)); -- EAST WEST NORTH SOUTH LOCAL
    else
        output_str := bit_to_string (vec( size -1));
    end if ;
    return output_str ;
end function ;
```

E assim a etapa de instrumentalização se dá por completa. Com a abordagem de coletar os eventos independente das máquinas de estado do roteador, do evento e do tipo de dado, bastando apenas adicionar ao corpo do *process* do módulo *logging* uma chamada para a *procedure* do sinal ou representação de estado a ser depurada.

4.1.1 Dados reportados e o padrão de *log*

Os eventos reportados podem ser identificados como a adição e remoção de elementos da CAM, os estados das máquinas de estado I-FSM e O-FSM e todos os sinais de comunicação entre as portas dos roteadores da BrNoC.

```
{"entradas" :[...
{
 "fsm1":9,
 "free_index":1,
 "used_table":"00000001",
 "out_source":"00x00",
 "out_id":"0000000000010010",
 " out_target ":"00x00",
 "out_service":"10010",
 "out_payload":"00000000",
 "pending_table":"00000001",
 "out_ack":"00010",
 "out_opmode":"00000",
 "tick":334
},  ...
{
 "fsm2":9,
 "tick":1337
},
{
 "fsm2":11,
 "out_ack":"00000",
 "tick":1390
 }...]}
```

Figura 21 – Exemplo de saída dos arquivos JSON

Cada roteador tem um arquivo de *log* no formato *JSON* associado, representando seu

histórico. A saída é um vetor composto por objetos que representam as entradas, saídas e sinais. Também são impressos a identificação dos estados das máquinas I-FSM e O-FSM. Além dos valores, cada objeto, também, carrega o valor de *tick* (contagem de ciclos de *clock*) em que ocorreu a mudança do sinal.

Podemos verificar na Figura 21, um exemplo dos dados representados em JSON. Nela vemos o vetor de entradas, onde cada instante em que houve atividade no roteador é representado por um objeto JSON. Vemos, por exemplo, no *tick* 334 o estado da I-FSM representado por *fsm1* com o valor 9 (WAIT_REQ_DOWN), *free_index* posicionado no índice 1, *used_table* indicando ocupação em uma posição da tabela. Já no instante 1337 a O-FSM no estado 9 (COMPARE_TARGET) e no instante 1390 passando para o estado 11(PROPAGATE) e o roteador passando a deixar os sinais de *ack* em valor 0.

4.2 INTERFACE GRÁFICA

Atendendo ao objetivo de desenvolver uma aplicação de interface gráfica para a análise das informações coletadas na simulação, o cliente gráfico foi feito com a linguagem de programação python (PYTHON. . . , 2019). A biblioteca gráfica escolhida foi a Tkinter (TKINTER. . . , 2020), devido sua licença flexível e fácil instalação. Também pela possibilidade de desenhar com mais facilidade os elementos gráficos com o suporte a Canvas, disponibilizado pela mesma. Isso possibilitou desenhar imagens vetoriais e, no caso deste projeto, desenhar os roteadores, com suas principais portas e demais atributos.

O cliente gráfico se comunica com o componente através dos arquivos no formato JSON, permitindo monitorar os *logs*. Estes arquivos de *log* ficam dentro da estrutura de pastas no ambiente de testes, sendo possível ser verificado o resultado da simulação a qualquer momento, ou seja, pode-se acompanhar durante a simulação ou de forma independente em outro momento.

O arquivo de entrada está em formato JSON, gerado na entidade VHDL, criada neste trabalho, e usa da biblioteca nativa do python o módulo JSON. Para fazer o carregamento dos arquivos foi criado um módulo específico, para lidar com os arquivos, chamado fail_handler, onde, primeiramente, ele verifica no diretório dos *testcases* a existência dos arquivos para, então, fazer o carregamento dos *logs*. O roteador BrNoC de cada elemento de processamento tem seus eventos registrados em um arquivo no formato JSON.

4.2.1 Desenvolvimento da interface gráfica

A aplicação visual deve ser capaz de lidar com redes de qualquer tamanho possível, no formato *mesh*, e para isso, foi necessário criar um código que, de maneira interativa, desenha os componentes na tela. O resultante é uma matriz com uma representação de cada um dos roteadores. Os elementos que compõem o roteador são dispostos de maneira pré definida, com

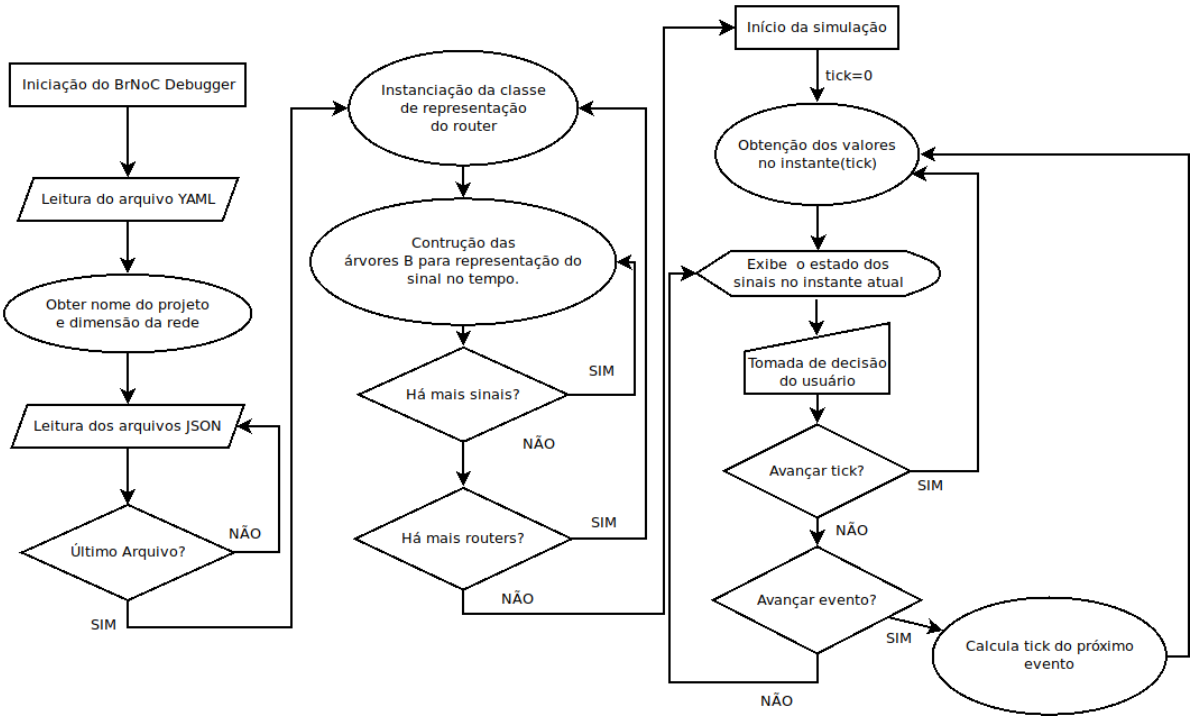


Figura 22 – Fluxograma de iniciação e funcionamento do simulador.

representação das portas de entrada, saída, requisição, mensagem, estado. Como podemos ver na Figura 27.

Já na Figura 22, é apresentado o fluxograma básico do funcionamento da interface gráfica. Primeiramente o arquivo *YAML*(que contém as informações de setup da simulação) é lido e a partir do seu nome a pasta do *testcase* é definida, e do conteúdo as dimensões da rede são conhecidas. Então, é feita a leitura dos arquivos *JSON* e posteriormente a instanciação da classe que representa cada um do roteadores da rede e as árvores B que representarão os sinais no tempo.

A simulação, então, é iniciada e mantida em um *loop* infinito. Na simulação, o *tick* é alterado conforme a velocidade que é escolhida pelo usuário, de forma incremental, ou a partir de comandos manuais do usuário. Os comandos manuais são o incremento ou decremento do *tick* ou o avanço ou retrocesso para a ocorrência de algum evento. Ao procurar um evento o *tick* mais próximo é escolhido, e então, este instante é atribuído ao tempo atual. A cada ciclo do *loop*, se houver uma alteração no *tick*, os componentes gráficos são redesenhados.

A aplicação de interface gráfica do *debug* seguiu o padrão de projeto *MVC* (Modelo, Visão e Controle). Onde o controle é responsabilizado pelo fluxo da execução do programa e lida com o modelo de negócio e os elementos visuais, sem que um tenha dependência direta com o outro. Isto torna o código mais organizado para ser mantido e menos suscetível a problemas futuros caso haja a necessidade de troca de dependências. Na Figura 23 é demonstrado o diagrama de pacotes. As dependências externas são os módulos *Tkinter* e *OOBTree*.

O pacote *Gui* representa a parte gráfica do programa, onde a classe *NetworkView* controla a janela principal e a área de desenho da rede. O *GElements* é uma interface que padroniza os

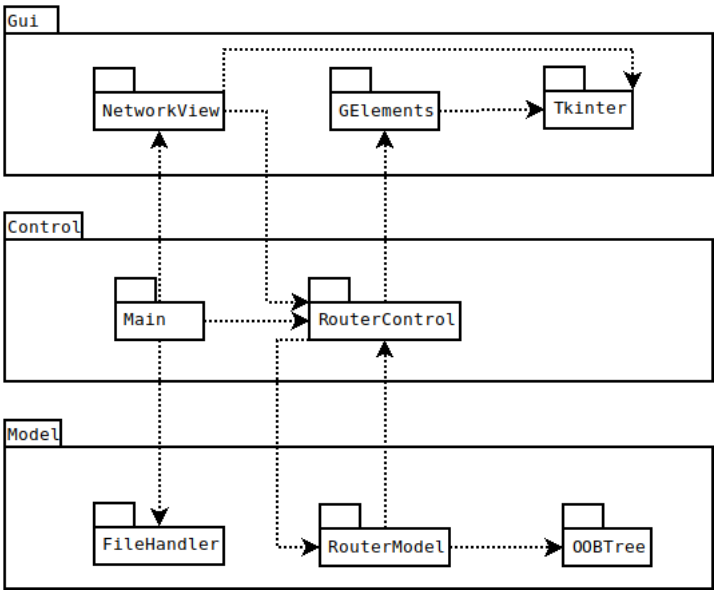


Figura 23 – Diagrama de Pacotes da interface gráfica.

elementos visuais e retiram a dependência direta do controle(Control) com a biblioteca gráfica. Estes elementos podem ser etiquetas, caixas ou flechas e mostram de forma gráfica o estado dos valores dos sinais.

No pacote *Control* temos o módulo *Main*, que inicia a aplicação, requere o carregamento dos dados, instancia a parte gráfica e os objetos da classe *RouterControl*. A classe **RouterControl** é a classe que controla o fluxo da informação a ser exibida por cada roteador, contatando os elementos de representação visual a partir da mudança de estado ocasionada por eventos, e requisitando ao modelo o valor de cada sinal no instante de *tick* da simulação.

Já o modelo é onde as regras de negócio do programa são aplicadas, primeiramente carregando os arquivos com a classe *FileHandler* e posteriormente na sincronização e computação dos eventos reconstituídos a partir dos arquivos de *log*.

4.2.2 Sincronização dos tempos de simulação

Os arquivos de *log* não tem os sinais reportados em todos os instantes, apenas quando houver a modificação dos valores, isto proporciona arquivos de menor tamanho, e também um menor impacto no custo da simulação. A reconstrução da simulação associa o valor de *tick* ao valor do sinal de cada porta como chave e valor. Porém, as lacunas de tempo devem ser preenchidas no momento em que formos reproduzir a simulação na interface gráfica.

Duas alternativas mais óbvias para resolver este problema surgem: reconstruir previamente os valores da simulação com a representação de todos os instantes na memória, ou procurar o valor do sinal no *tick* corrente baseado no último instante que houve uma modificação de valor. A primeira opção conseguiria ter um retorno instantâneo dos valores no *tick* requisitado, porém, afeta diretamente a quantidade de memória consumida pela aplicação. Já a segunda opção apresenta problemas de custos na busca pelos valores no instante a ser represen-

tado, pois exigiria que houvesse uma varredura entre todas as chaves, aquela que tivesse o valor imediatamente inferior ao *tick* requisitado.

Então, neste contexto, a árvore B foi a opção escolhida, como estrutura de dados, pelo fato de lidar bem tanto com a leitura sequencial quanto com a leitura fragmentada (BAYER, 1972) e por possibilitar no futuro que estas estruturas possam ser salvas em disco e manter um bom desempenho. Além de tudo, na estrutura da árvore B, as chaves são mantidas em sequência de grandeza, sendo assim, é fácil referenciar as chaves vizinhas e encontrar o local onde uma chave não existente se encaixaria. Quando a chave não existir, procura-se o valor mais próximo, sem custos adicionais, encontrando a ocorrência mais próxima ao valor da chave alvo, anterior ou posterior.

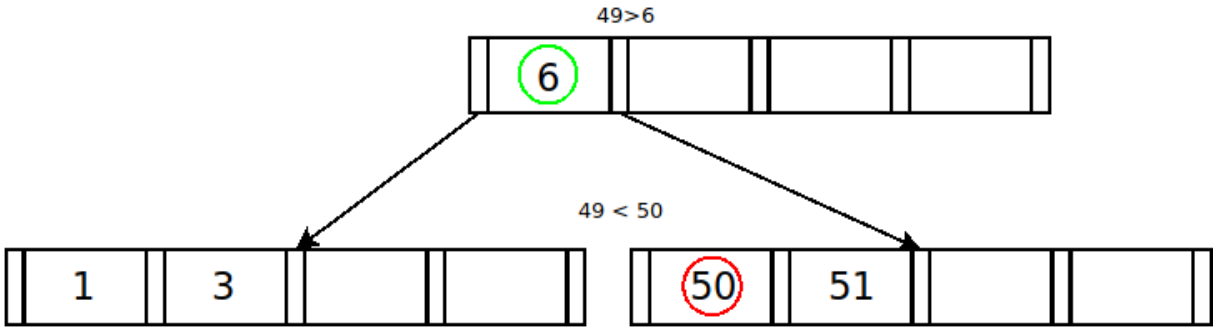


Figura 24 – Representação de busca na árvore B.

Por exemplo, havendo o sinal *in_ack* sido modificado nos instantes 1, 3, 6, 50 e 51, temos a árvore da Figura 24. Se formos, por exemplo, precisamos conhecer o valor do sinal no instante 49, devemos saber que o valor do sinal neste instante ainda é igual ao valor definido no instante 6, pois esta é a última ocorrência de modificação do sinal. Para encontrar este valor, requisita-se à árvore que busque o maior índice anterior ou igual ao 49 registrado. Não encontrando o 49, basta que a chave vizinha à posição que o 49 poderia ser inserido, seja acessada.

A função do modulo *btree* para encontrar a maior chave até um determinado valor é a *maxKey*. Na Figura 25 temos o exemplo da atualização dos valores na tela, onde o valor de cada entrada é atualizado na mudança do *tick* da reprodução da simulação. A interface informa a todos os roteadores que o tempo foi modificado. Na representação do roteador é chamada a função *maxKey* da árvore de cada um dos sinais, as quais retornam a chave relativa à última ocorrência. A chave que é retornada é usada como índice de acesso do valor, que então é adicionado ao dicionário que será repassado ao controle. Após receber o dicionário dos valores atualizados o controle demanda que aos elementos da interface gráfica sejam redesenhados.

Outra funcionalidade possível com o uso da árvore B é a busca por eventos mais próximos, tanto avançando, quanto retrocedendo. Para retroceder, usa-se função *maxKey*, a qual retorna a última ocorrência de cada um dos sinais. Para conhecer-se o evento mais próximo basta comparar as chaves de todos os sinais e escolher a de maior valor, desta forma, sabemos quando ocorreu o último evento no roteador. De forma global, para chegar-se ao evento mais próximo é

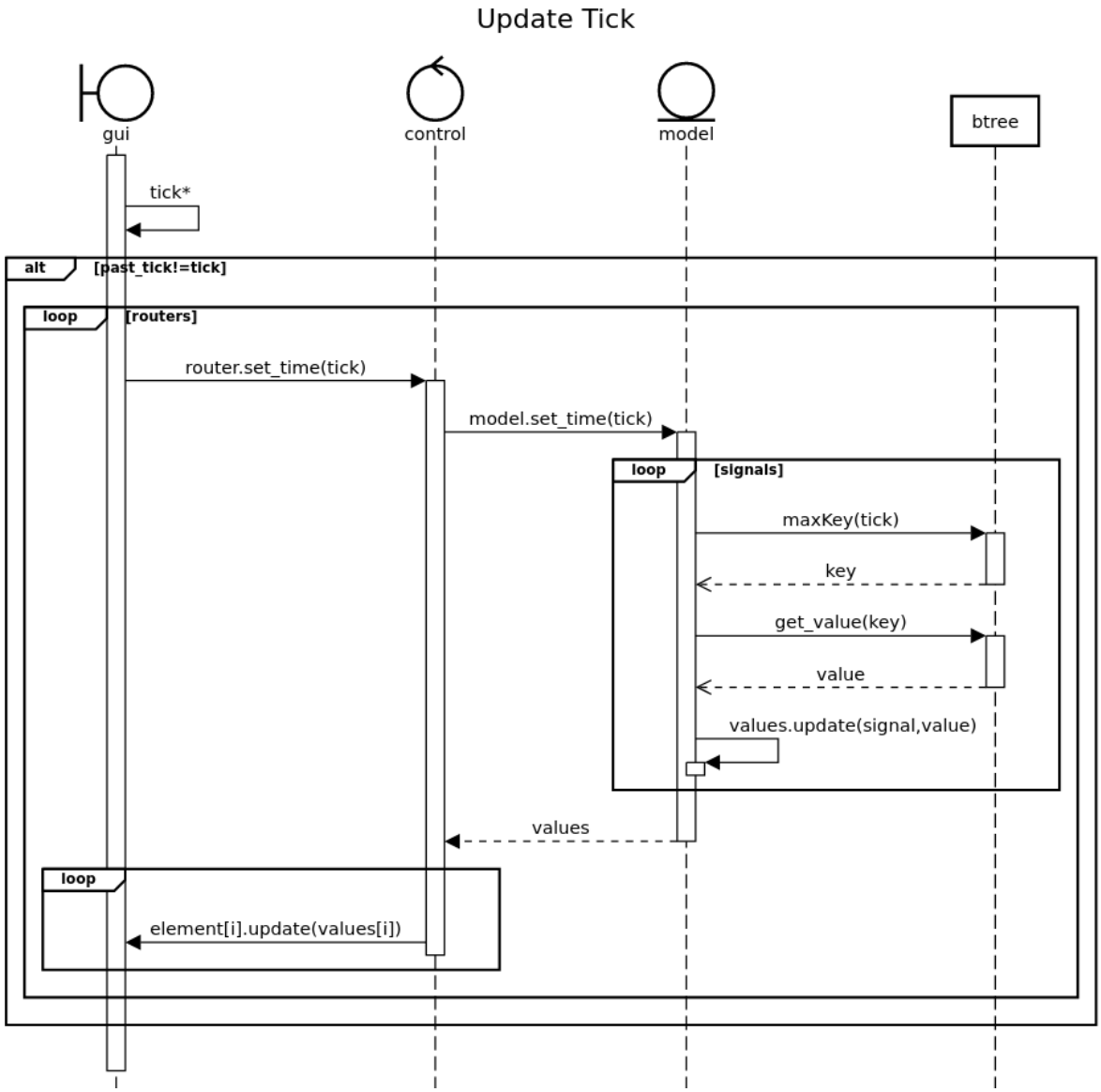


Figura 25 – Diagrama de sincronização dos *ticks*.

feita a comparação entre todos os tempos retornados pelos roteadores e então o *tick* é atribuído pelo maior valor entre todos.

Para avançar no tempo a função da árvore utilizada é a *minKey*. Ela funciona de forma similar à *maxKey*, porém, ao invés de buscar a chave mais próxima igual ou inferior ao valor de chave passado por parâmetro, ela retorna a chave de valor mais próximo de valor igual ou superior à aquele passado por parâmetro. Na Figura 26, temos o exemplo de busca por eventos.

A busca por eventos pode filtrar os sinais que são verificados, para isto, basta limitar os sinais a serem analisados. Isto reduz significativamente os custos da busca e aumenta as chances de encontrar os eventos que mais interessam ao estudo. Limitar os sinais é recomendável para a execução de busca por valores, avançando para o próximo evento de cada um dos sinais até que a ocorrência do valor desejado seja encontrada. Nesta última modalidade de busca, a quantidade de sinais à serem buscados deverá ser mais restrita, visto que além da busca por chaves mais

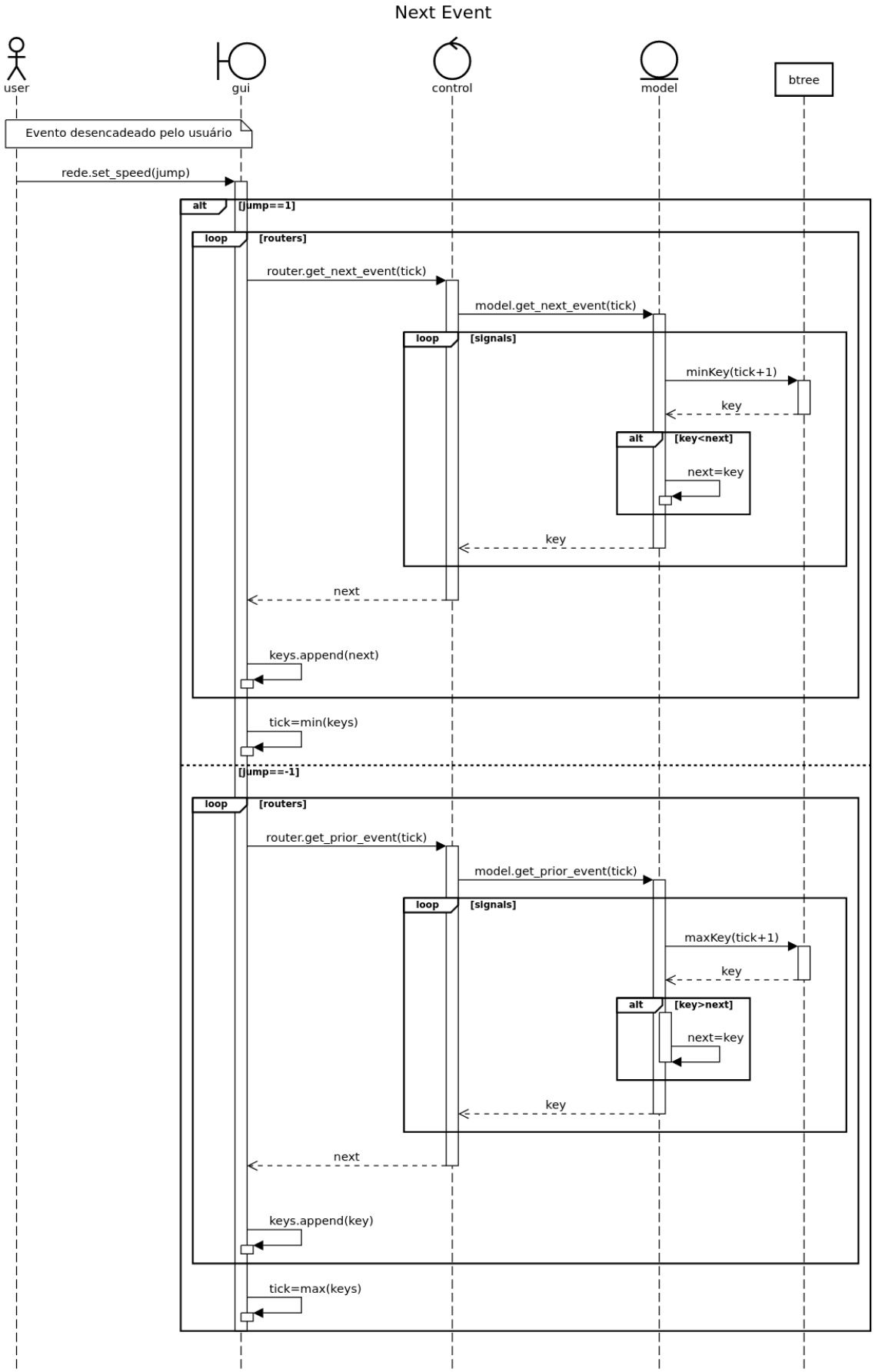


Figura 26 – Diagrama de sincronização por eventos.

aproximadas todos os valores que não forem os esperados pela busca serão descartados, o que pode tornar a busca cara, visto que os valores devem ser checados um à um.

4.2.3 Funcionalidades da Interface gráfica

A Interface gráfica para a BrNoC proporciona as seguintes facilidades de forma organizada e visual:

- Visualização completa da rede;
- Reprodução da simulação;
- Controle de velocidade da reprodução;
- Avanço ou retrocesso da reprodução;
- Busca por eventos;
- Filtragem por sinal;
- Busca por valor de sinal (Nome de serviço, origem, destino, etc);
- Visualização da propagação das mensagens;
- Visualização dos *handshakes*, tanto da porta local, quanto entre roteadores vizinhos;
- Visualização dos atributos das mensagens que entram e saem dos roteadores(origem, destino, *payload* e serviço);
- Contagem da ocupação da tabela CAM;
- Adição e remoção de entradas da tabela CAM;
- Visualização do estado das máquinas de estado I-FSM e O-FSM de cada roteador;
- Visualização do status dos *wrappers* e do sinal *opmode*;
- Visualização do comportamento de mensagens restritas, conforme os sinais dos *wrappers*;
- Acompanhamento da criação e remoção de zonas segura;
- Acompanhamento da Iniciação de processos;
- Acompanhamento da criação de novas rotas de comunicação.

A janela principal (Figura 27) é composta pela área de botões controles (1) e a área de visualização (8). Para controlar o avanço da reprodução da simulação automática são utilizados os botões do bloco 2, sendo o primeiro o responsável pelo avanço contínuo ou a parada do tempo de simulação, já os demais botões do mesmo bloco são responsáveis pela velocidade da reprodução, incrementando-a ou diminuindo-a. A velocidade afeta, diretamente, o salto que se dará para o próximo *tick*, pois ela incrementa o *tick* a cada quadro de tela, podendo também ser negativo, assim decrementando o *tick*, portanto, retrocedendo no tempo. No bloco 7, tanto a velocidade (*ticks* por quadro) e a posição temporal (*tick*) da execução, são exibidos.

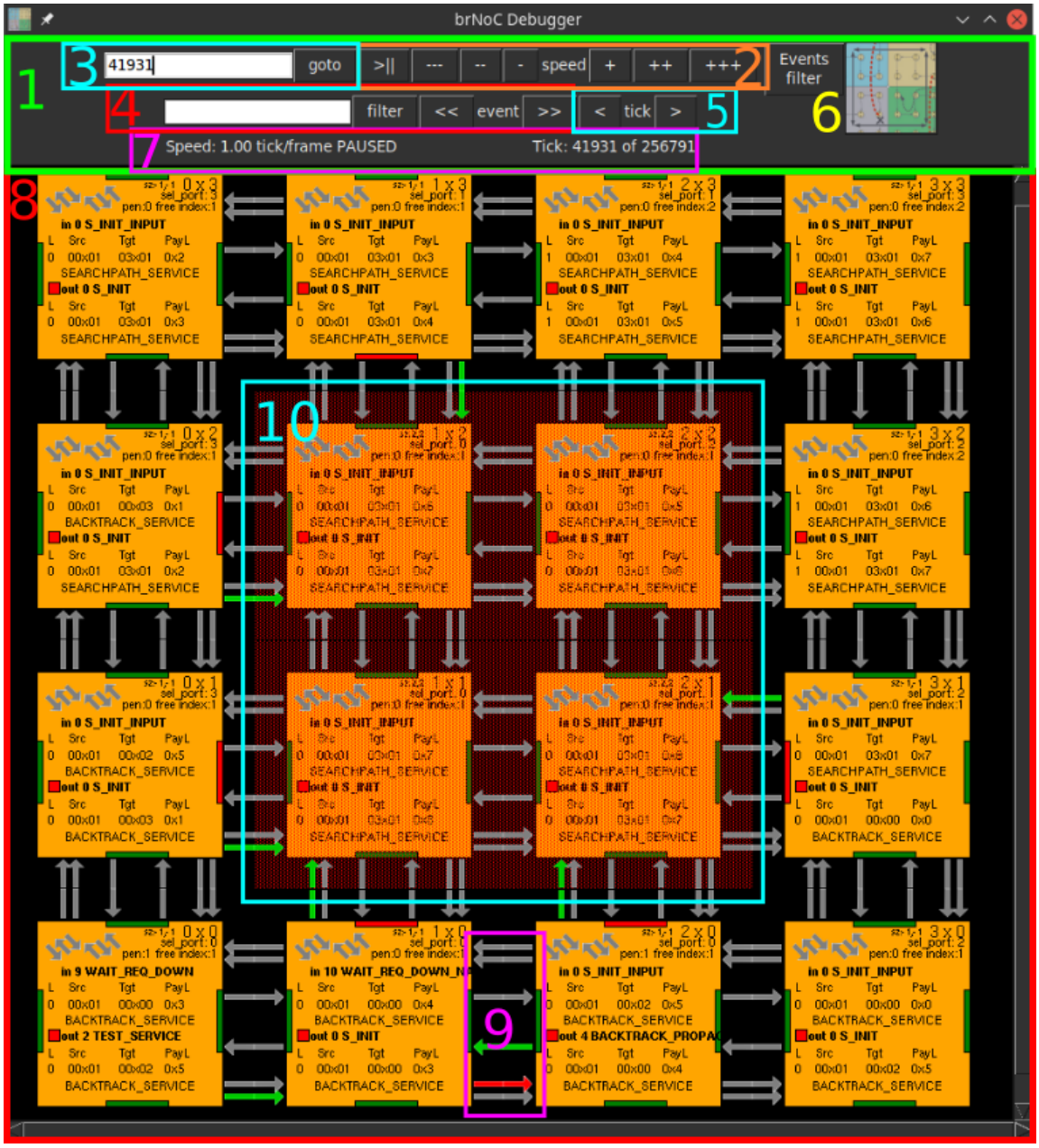


Figura 27 – Captura da janela principal da interface gráfica.

No bloco 4 temos o avanço por eventos, que através das setas duplas avança ou retrocede para o evento mais próximo. Os eventos podem ser filtrados pelo valor na entrada de texto do

mesmo bloco e por sinal (6). Ao filtrar por sinal, somente eventos que ocorrerem nas portas selecionadas irão ser buscados, caso houver uma entrada definida no filtro, por valor, somente os valores informados serão buscados, ignorando eventos onde não houverem esta ocorrência.

No bloco 8 se dá a visualização completa da rede *mesh* BrNoC, onde cada caixa representa o roteador referente a um dos PEs. Eles são numerados de baixo para cima e da esquerda para a direita. Por exemplo, o roteador do PE do canto inferior esquerdo é o 0 x 0, enquanto o do canto superior direito é o 3 x 3, e o roteador do PE da penúltima linha e terceira coluna é o 2 x 1.

As entradas e saídas representadas no nono bloco são as portas responsáveis pelo protocolo de comunicação entre os roteadores, sendo elas: *in_ack_router_seek, in_nack_router_seek, out_req_router_seek, in_req_router_seek, out_nack_router_seek e out_ack_router_seek*. Estas portas implementam o *handshake* entre os roteadores vizinhos. Por fim, vemos a exibição de uma área demarcada como zona segura (10).

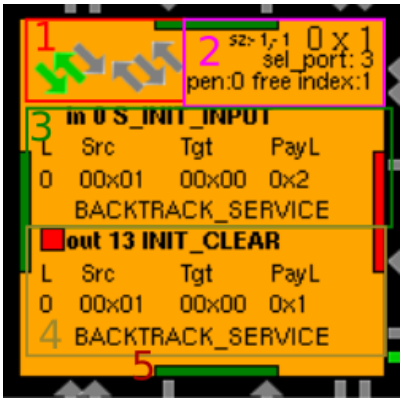


Figura 28 – Representação de um Roteador.

A imagem 28 representa um roteador. O setor demarcado pelo número 1 refere-se a porta local, que permitem ou não, após o *handshake*, a entrada ou saída de pacotes entre o processador e o roteador. No retângulo 2 temos a identificação da zona segura e identificação do PE que este roteador pertence, na segunda linha é através do sinal sel_port que é exposto em qual porta de entrada o valor está sendo lido, e por fim, na terceira linha, é dado o número de linhas pendentes na tabela CAM e a linha em que a próxima entrada na memória CAM será gravada.

O bloco 3 representa a máquina de estados I-FSM. Na primeira linha dá-se o estado atual da máquina de entrada, ao centro, na primeira coluna a última linha da CAM a ser gravada e os seguintes parâmetros do conteúdo da porta de entrada selecionada: linha da escrita, origem, serviço e destino. Na última linha, o *payload* deste pacote.

O bloco 4 representa a máquina de estados O-FSM. Na primeira linha o quadrado representa o *opmode*(vermelho normal, verde global), ao seu lado é exibido o estado atual da máquina de entrada. Nas linhas seguintes é mostrado a linha que está sendo lida na tabela CAM e os seus parâmetros, assim como os exibidos na visualização da máquina I-FSM.

O elemento de número refere-se a porta *in_fail_router_seek*, porta que tem sua cor

alterada de verde para a vermelho caso o *wrapper* do roteador vizinho impeça a passagem de pacotes por ele.

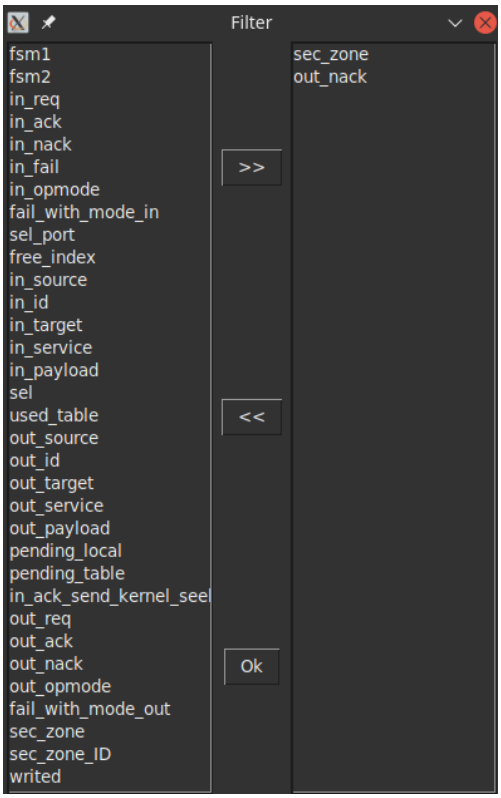


Figura 29 – Janela de personalização dos filtros.

Ao clicar no botão de número 6, da Figura 27 (*Events filter*), a janela de filtros é exibida (Figura 29). À esquerda, são listados os sinais que foram salvos na simulação, ao centro, respectivamente, os botões para adicionar, remover e salvar os sinais escolhidos ao filtro de busca, e à direta os sinais selecionados para a filtragem.

5 RESULTADOS E CONCLUSÃO

Neste capítulo serão demonstradas capturas de tela da interface gráfica, bem como explicar o comportamento da rede BrNoC através destas. Também serão executados alguns testes de desempenho, para verificar o quanto o mesmo é prejudicado pela instrumentalização feita no código fonte do hardware. Por fim, as considerações finais do autor serão apresentadas na conclusão do trabalho.

5.1 VISUALIZAÇÃO DOS MODOS DE OPERAÇÃO

Um dos aspectos mais importantes, tanto para a didática quanto para fins de pesquisa, deste trabalho, foi demonstrar visualmente o comportamento da rede e a forma que os pacotes são transmitidos. Logo abaixo, veremos os quatro modos de transmissão, **brAll**, **brTgt**, **unicast** e **brWt**, demonstrados na interface gráfica.

5.1.1 Broadcast para todos

A mensagem *INITIALIZE_CLUSTER_SERVICE* é operada no modo **brAll**, e é a mensagem escolhida como exemplo para demonstração deste modo. Como previamente explicado, o **brAll** é entregue na porta local de todos os roteadores da rede.

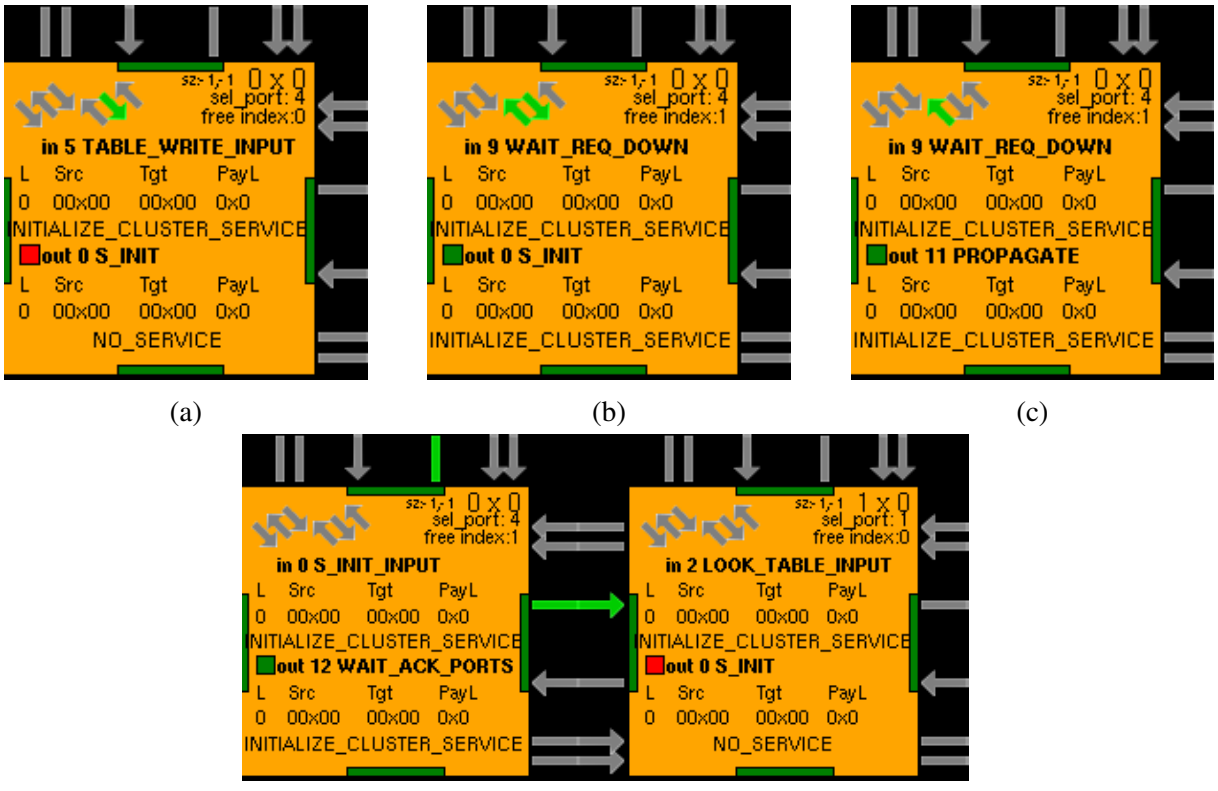


Figura 30 – Recepção da mensagem de origem local.

A mensagem começa sendo ativada localmente com o roteador *master* (0x0), recebendo o pacote (Fig. 30a), ficando o sinal de *req*, na porta local de entrada, ativa. O roteador aprova o recebimento do pacote, ativando o sinal *ack*. O pacote é, então, adicionado a memória CAM (Fig. 30b), o sinal *req* é desativado e a mensagem posteriormente propagada (Fig. 30c). Após o *req* ser desativado, o *ack* também é desativado (Fig. 30d)

Da mesma forma que a mensagem local, a transmissão dos pacotes entre os roteadores também se dá pelos quatro passos do protocolo de comunicação, com a ativação do sinal *req*, pelo roteador que envia (Fig. 31a, neste caso o 0x0). Então, ele obtém o sinal de *ack* dos vizinhos (Fig. 31b), o *req* é desativado (Fig. 31c), o *ack* do vizinho é desativo e o pacote é entregue na porta local. A mensagem é propagada para todos os roteadores, e caso já esteja na tabela CAM, a resposta será um *ack*.

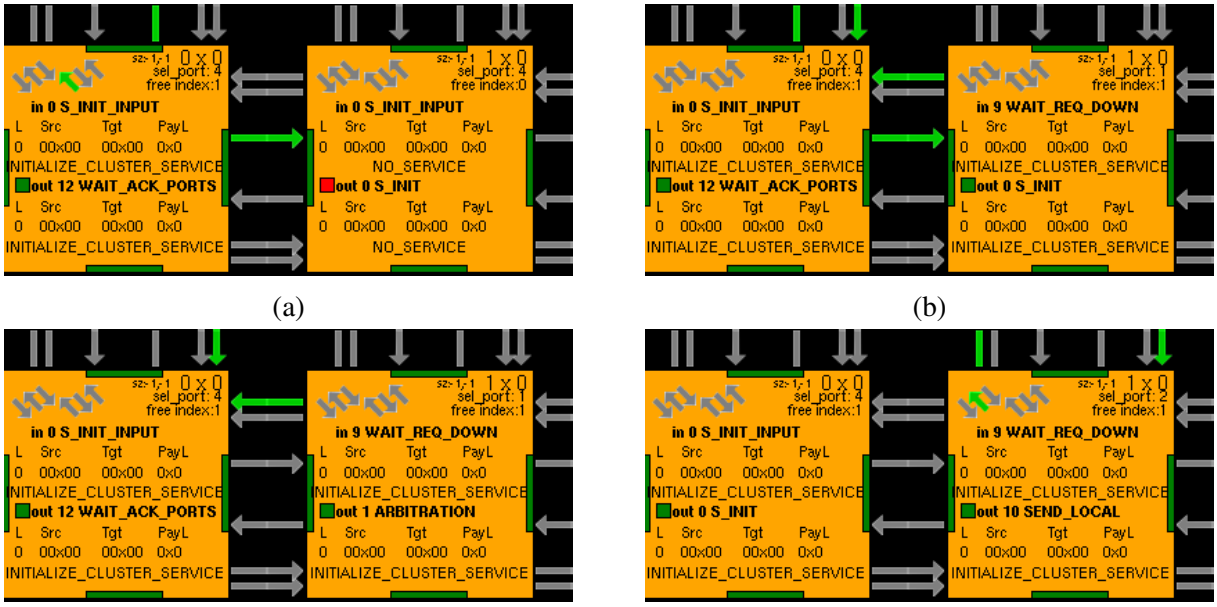


Figura 31 – Propagação do broadcast.

Nas Figuras 32a e 32b a mensagem é aceita pelo roteador e finalmente a requisição é desativada.

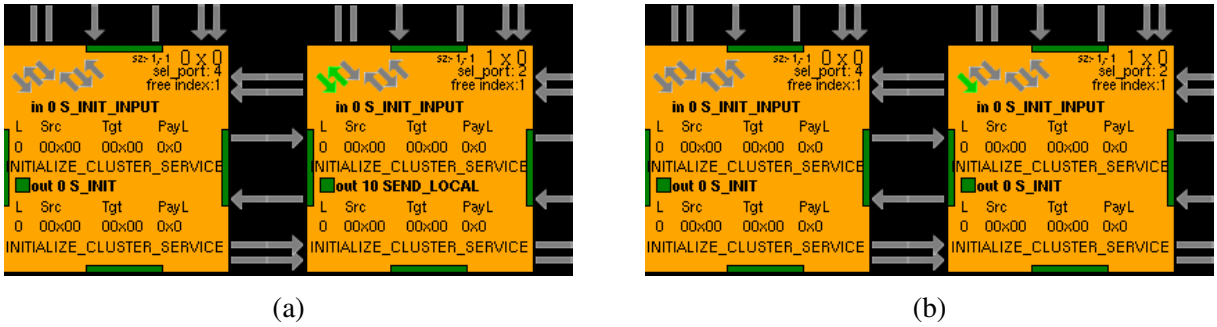


Figura 32 – Admissão nas portas locais.

5.1.2 Broadcast com alvos

A propagação no modo **brTgt** ocorre de forma similar ao do **brAll**, porém apenas no roteador indicado como destino (*target*), o valor é repassado para a porta local, exceto no caso da mensagem de SEARCHPATH_SERVICE. Todavia, o comportamento de forma geral é similar, visto que apenas quando o destino é atingido, ações são tomadas.

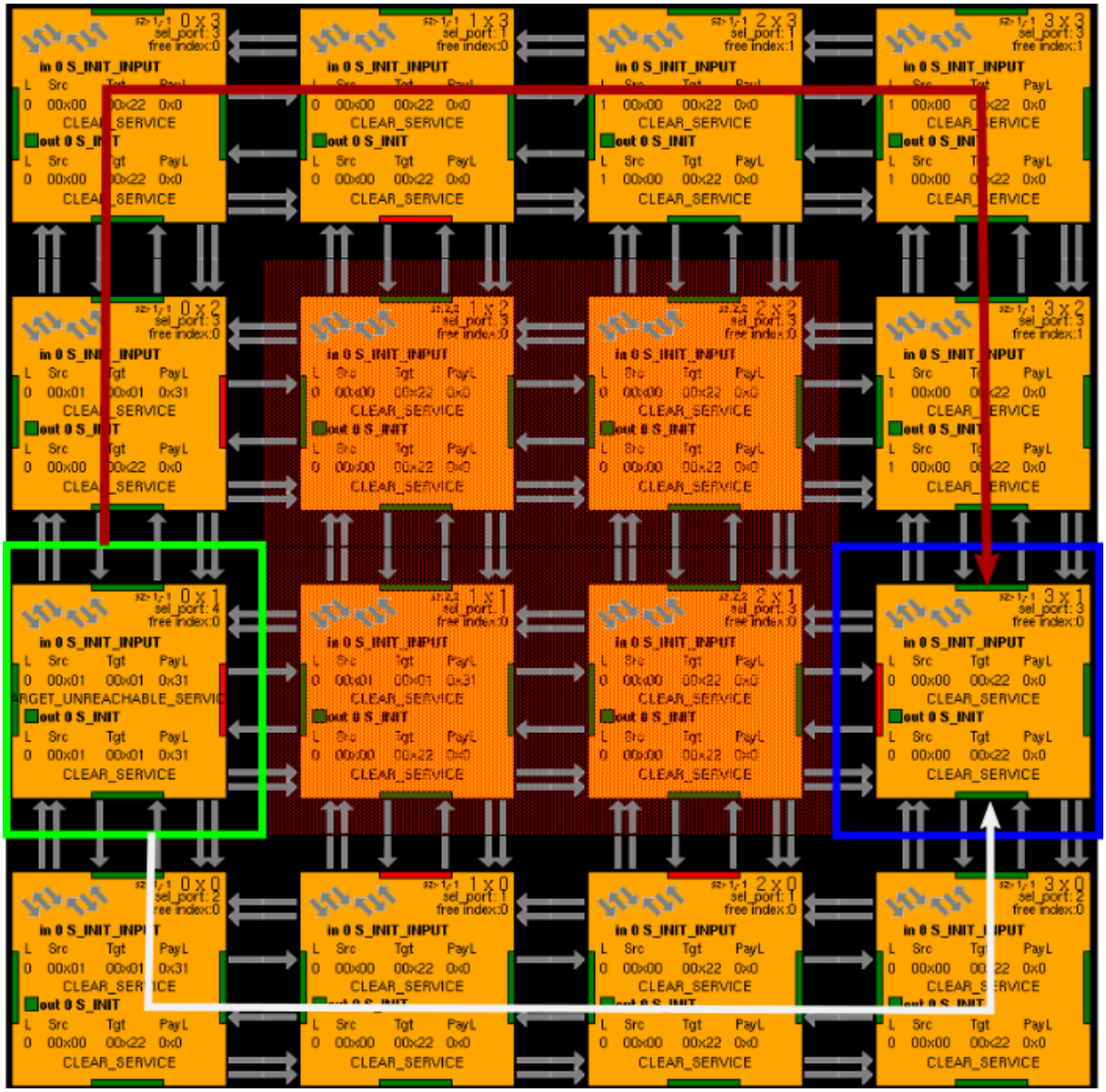


Figura 33 – Perda da comunicação dos roteadores 0x1 e 3x1.

Neste exemplo, será mostrado exatamente o serviço de Search Path. Ele ocorre quando dois processos que se comunicavam, perdem o contato, seja pela ocupação da rede, ou por conta de zonas seguras criadas no antigo caminho. Outro detalhe fundamental desta mensagem é que ela respeita as zonas seguras.

As mensagens SEARCHPATH_SERVICE são as únicas que operam no modo **brTgt** que não são entregues na porta local do roteador de destino, isto porque, o próprio roteador interpreta

a mensagem, não chegando ao conhecimento da aplicação. A operação ocorre após a ocorrência de uma mensagem de destino inalcançável (TARGET_UNREACHABLE_SERVICE).

A figura 33, mostra o estado atual da rede, destacando em verde onde está a origem e em azul, o destino do caminho a ser traçado. Os dois caminhos possíveis são os indicados pelas flechas vermelha e branca.

O roteador de origem (0x1) propaga a mensagem SEARCHPATH_SERVICE (Fig. 34a). A mensagem chega por primeiro até o roteador de destino (3x1) pela porta sul, que é o caminho mais curto (Fig. 34b). A requisição é aceita (Fig. 34c e 34d).

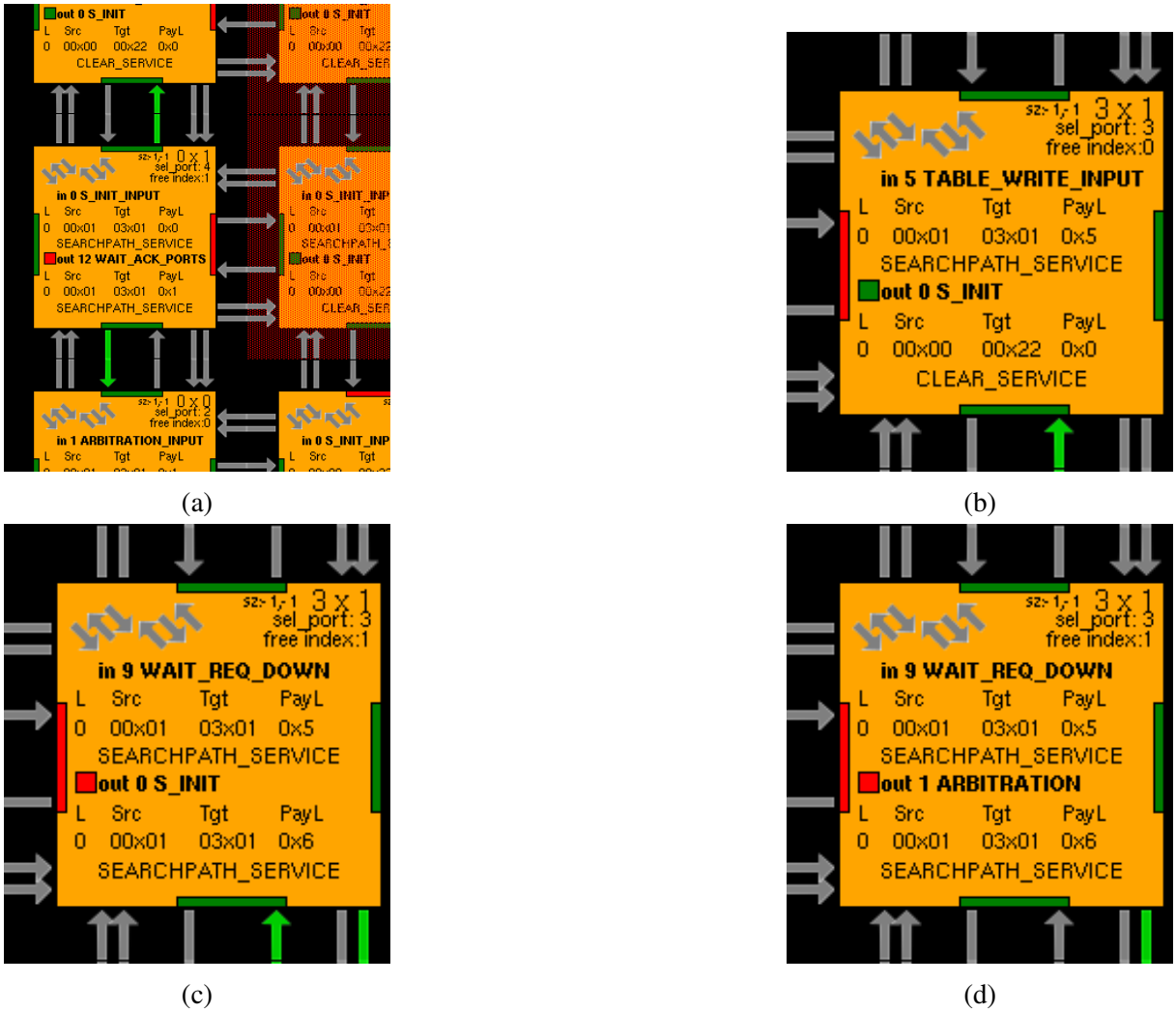
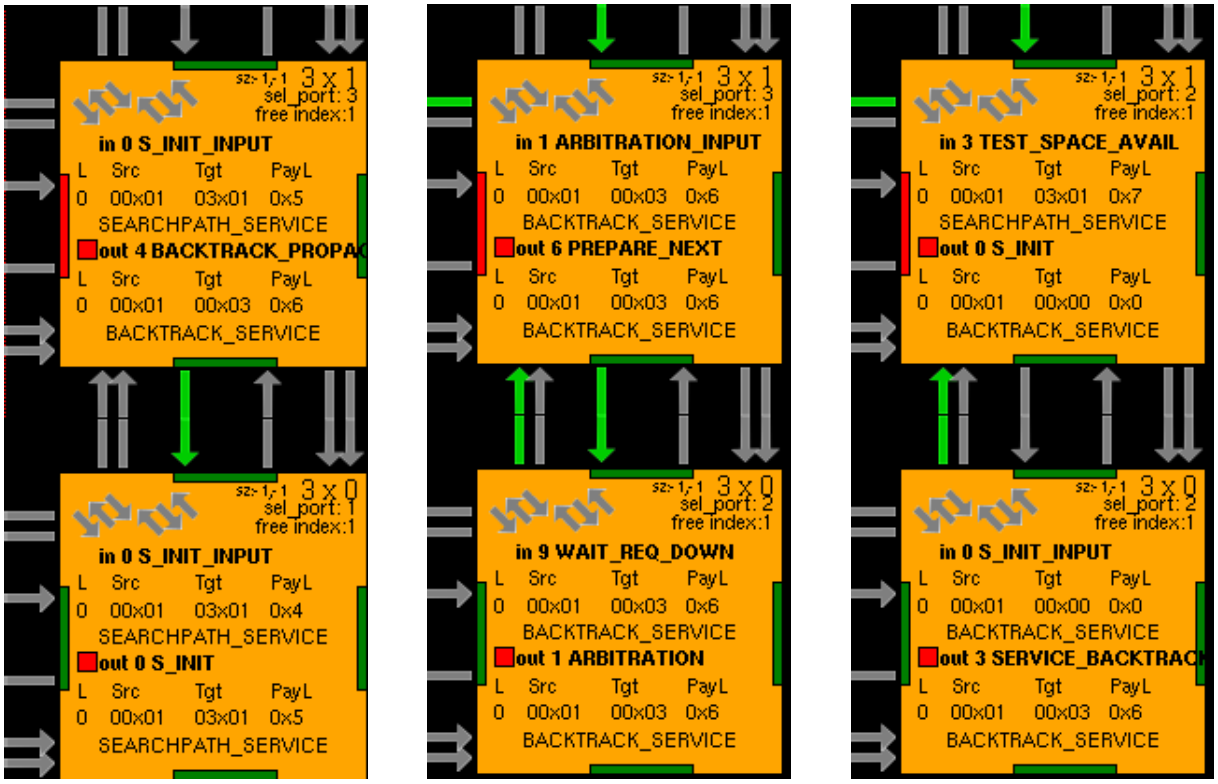


Figura 34 – Serviço de busca de caminhos.

5.1.3 Unicast

O serviço de *backtrack* (BACKTRACK_SERVICE), é o único serviço que opera no modo **unicast**. Esta é uma continuação do serviço de Search Path, mostrado anteriormente. Neste passo, é feita a requisição ao vizinho que deu origem a mensagem de busca (Fig. 35a) e é então, aceita (Fig. 35b), e a requisição é desativada (Fig. 35c). A mensagem é propagada somente para

o roteador de origem, conforme o caminho que a mensagem de SEARCHPATH_SERVICE, de menor tamanho, criou. Após propagar a mensagem vinda do vizinho mais próximo ao destino, cada um dos roteadores dispara uma mensagem de *backtrack* nova, com sua identificação. Deste modo, o roteador de origem, por fim, receberá mensagens originadas de todos os roteadores do caminho, desta forma, conhecendo todo o caminho novo.



(a) (b) (c)

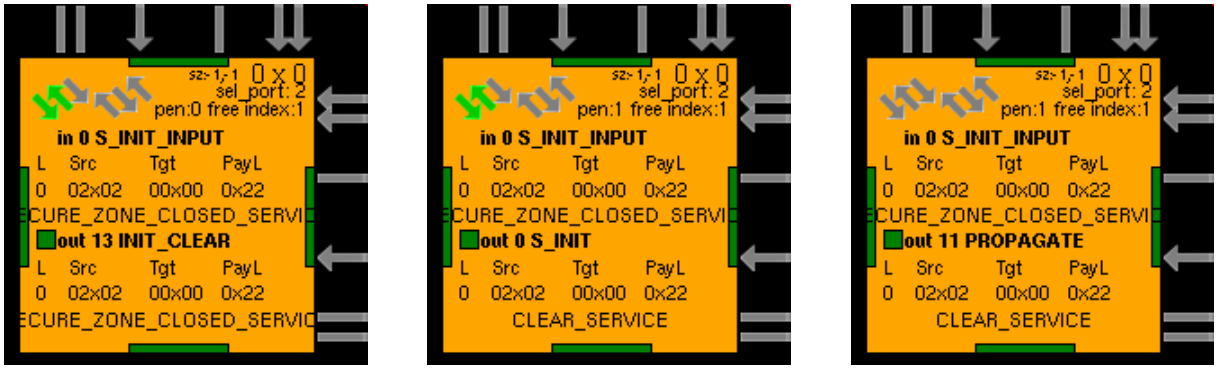
Figura 35 – Execução do serviço de backtrack.

5.1.4 Broadcast sem alvos

Por fim, o modo **brWt**. Este modo de mensagem não tem alvo, tendo como objetivo o gerenciamento das tabelas CAM dos roteadores da BrNoC. As mensagens CLEAR_SERVICE e GMV_READY_SERVICE são as únicas que operam neste modo. Neste exemplo, será mostrado o funcionamento da CLEAR_SERVICE, que faz a limpeza dos serviços.

A mensagem de limpeza é disparada quando alguma mensagem de serviço já atingiu todos os alvos e não é mais necessária, sendo assim, já podendo ser removida das tabelas CAM dos roteadores.

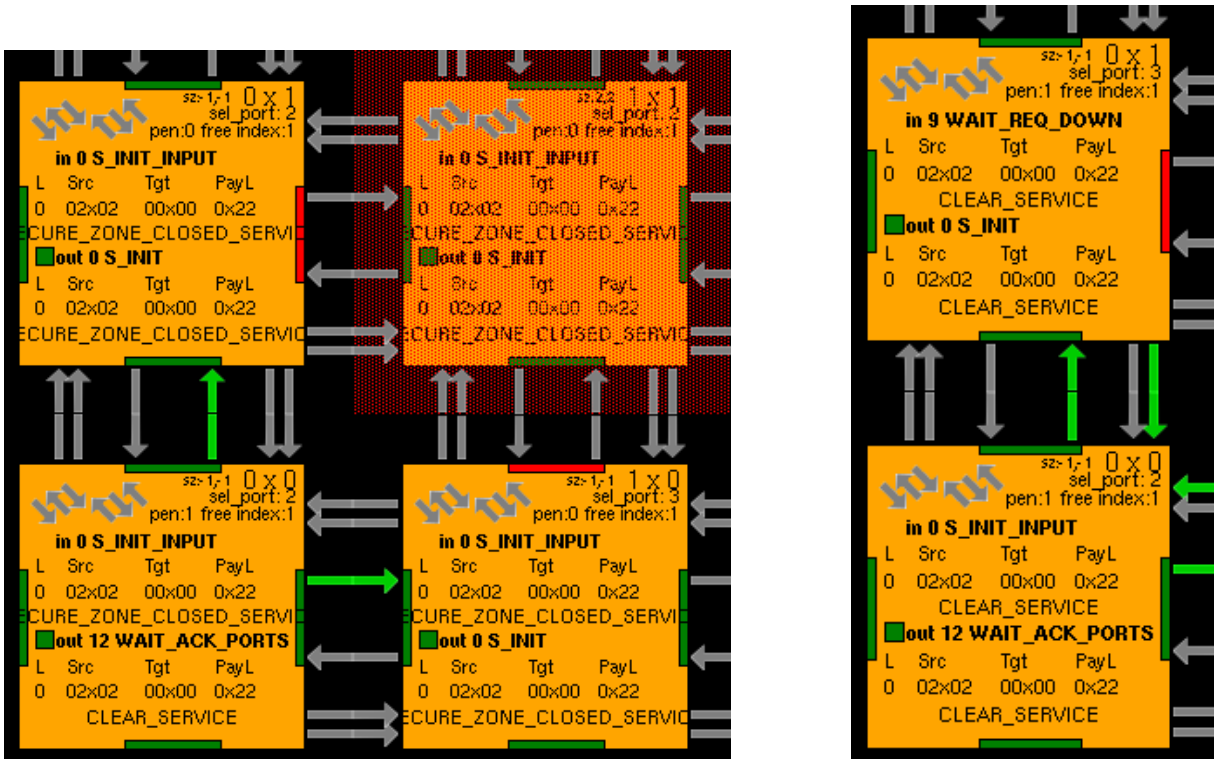
No exemplo, a limpeza de serviços é chamada após o fechamento de uma zona segura. Ao atingir o alvo (*target*), que, no caso da mensagem SECURE_ZONE_CLOSED_SERVICE, é o roteador do PE 0x0. Ao atingir o alvo a máquina de estados O-FSM, entra no estado INIT_CLEAR (Fig. 36a). Neste estado, o serviço de limpeza substitui o antigo serviço a ser apagado (Fig. 36b). A mensagem de limpeza é, então, propagada (Fig. 36c).



(a) (b) (c)

Figura 36 – Início do serviço de limpeza.

O roteador de origem, consequentemente, aciona o *req*, de modo a informar aos roteadores vizinhos que há uma mensagem nova (Fig. 37a). Os vizinhos, então, aceitam a requisição (Fig. 37b).

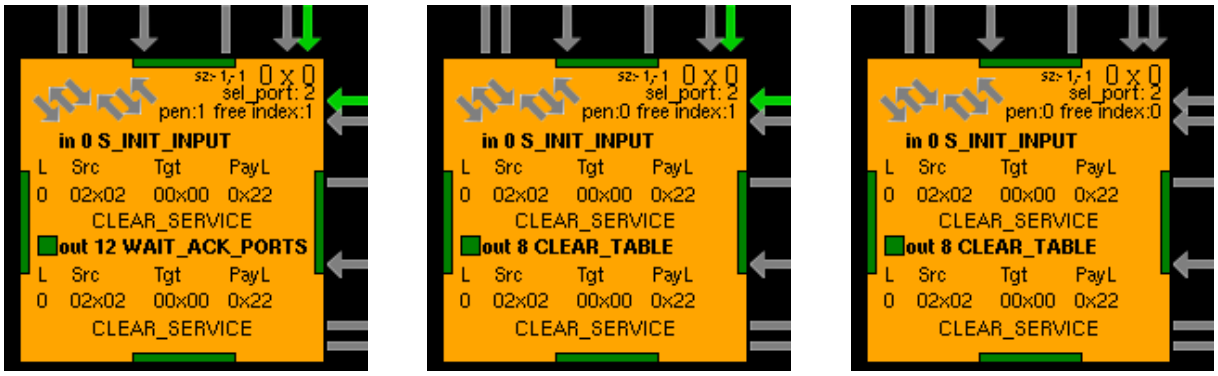


(a) (b)

Figura 37 – Propagação do serviço de limpeza.

Assim que a mensagem é confirmada pelos vizinhos a requisição é retirada (Fig. 38a). Neste momento a limpeza da tabela é inciada, tirando a linha dos elementos pendentes (Fig. 38b) e deste modo, o índice se torna limpo, fazendo com que a linha da tabela que era utilizada pelo serviço se torne elegível a receber novas mensagens (Fig. 38c). Percebe-se neste momento que o *free_index* passa a apontar para a primeira linha da tabela CAM(a prioridade de ocupação da tabela é sempre maior na primeira linha e diminui conforme o índice da linha aumenta). Esta

operação de exclusão se repete em todos os roteadores, até que a mensagem (se existente na tabela CAM) seja excluída de todos os roteadores.



(a) (b) (c)

Figura 38 – Limpeza da tabela CAM.

5.2 AVALIAÇÃO

Nesta seção se objetiva a analisar os impactos de desempenho causados pelo ambiente de depuração na simulação, e também qual a escalabilidade dos arquivos de *log* quando as dimensões da rede variam.

As simulações aqui apresentadas foram todas executadas no mesmo computador. Um AMD FX™ 6100 operando com uma frequência fixa de 3.9 GHz, com capacidade de 24 GB na memória principal, do tipo DDR3 operando em 1600MHz. O armazenamento principal é em um SSD Kingstone v300, com aproximadamente 460 MB/s de leitura e 125MB/s de escrita.

5.2.1 Desempenho e escalabilidade

Para verificar o quanto o desempenho da execução do ambiente durante a simulação é afetado pelo módulo de depuração, e o comportamento da gravação dos arquivos de *log*, foram feitos dois experimentos. O primeiro experimento se deu em duas redes, uma de dimensões 3x3 com 4 processos e uma zona segura, e uma segunda com dimensões 4x4 com o 4 processos e uma zona segura.

Neste experimento foram utilizadas quatro configurações: *log* desativado (usado como referência) e outras três simulações fazendo a gravação em três dispositivos de armazenamento distintos: gravação em disco rígido, gravação em SSD e gravação em um sistema de arquivos montado na memória RAM (RAMFS).

Nos experimentos do cenário 1 (Tabela 1), vemos os tempos do SSD se destacando em relação ao HD e sistema de arquivos RAMFS, com o menor overhead entre os dispositivos de armazenamento, 20.46%.

Tabela 1 – Tabela de desempenhos (cenário 1).

Dimensão	Processos	log	SZ	tempo(min)	overhead
3x3	4	-	1	5:47	-
3x3	4	HD	1	7:01	21.36%
3x3	4	SSD	1	6:58	20.46%
3x3	4	RAMFS	1	7:06	22.77%

Tabela 2 – Tabela de desempenhos (cenário 2).

Dimensão	Processos	log	SZ	tempo(min)	overhead
4x4	4	-	1	6:18	-
4x4	4	HD	1	8:13	30.42%
4x4	4	SSD	1	7:48	23.80%
4x4	4	RAMFS	1	8:04	28.04%

Nos experimentos do cenário 2 (Tabela 2), vemos que a gravação no SSD novamente se destacou, com um *overhead* de 23.8% e o pior tempo ficando com a gravação na RAM. A média geral ficou em 27.42%.

O tipo de dispositivo de armazenamento se demonstrou ser uma variável de pouco peso no desempenho da simulação, possivelmente pela forma que os sistemas operacionais modernos gerenciam o *cache* para a escrita nos sistemas de arquivos.

Para avaliar a escalabilidade do registro dos *logs* com o aumento do tamanho do MPSoC foi proposto um cenário onde a quantidade de processos em execução é constante (4 processos) e aumenta-se o tamanho do MPSoC (desde 3x3 até 6x6). Como as mensagens ocorrem em broadcast e chegam a todos os PEs a quantidade de eventos a serem registrados (logados) aumenta com o aumento no tamanho do MPSoC.

No cenário demonstrado na Tabela 3, além do acréscimo de tempo, outro aspecto analisado é o tamanho dos arquivos gerados nas diferentes dimensões de rede. Houve um acréscimo médio de 12.48% no tamanho dos arquivos de *log* da dimensão 3x3 para a dimensão 6x6, e pelo fato de o número de arquivos escalar junto a quantidade de roteadores este espaço ocupado é multiplicado, resultando em uma variação de 4.5 vezes o espaço total da primeira para o última rede.

Tabela 3 – Tabela de desempenho em diferentes tamanhos de redes.

Dim	Pcs	Log	Menor	Maior	Médio	Total	t c/log	t s/log	Ov
3x3	4	9	52.1KB	76.4KB	63.3KB	570.0KB	4:39	3:28	34.13%
4x4	4	16	50.8KB	77.7KB	65.8KB	1052.5KB	7:46	6:20	22.63%
5x5	4	25	53.3KB	82.7KB	70.2KB	1755.4KB	12:42	9:56	27.85%
6x6	4	36	53.0KB	81.4KB	71.2KB	2564.2KB	20:42	15:54	30.19 %

Observa-se que há um aumento de tempo das simulações, entre 22.63% e 34.13%. O overhead médio, no mesmo cenário variando apenas o tamanho do MPSoC, foi de 28.7%.

5.3 CONCLUSÃO

O presente trabalho objetivou a implementação de uma interface gráfica para o acompanhamento de eventos em simulações da rede BrNoC a fim de contribuir para o estudo de redes intra-chip com suporte a Broadcast. A BrNoC, por se tratar de um trabalho recente, ainda não contava com uma ferramenta de depuração especialmente elaborada. O estudo da rede BrNoC e a correspondente implementação e extração de dados mostrou-se um grande desafio técnico devido a grande complexidade do projeto.

Apesar de todas as etapas terem sido desenvolvidas e analisadas, um fator muito importante a ser ressaltado foram as dificuldades encontradas durante o caminhar deste projeto, em sua maioria ligadas à etapa de instrumentalização. Pois trata-se de um projeto bastante robusto e de difícil compreensão em um primeiro momento.

A interface gráfica atingiu em grande parte os objetivos, tornando a visualização dos eventos muito facilitada, desta forma ajudando na evolução dos estudos das redes intra-chip em malha, mais especificamente redes de broadcast. O uso da árvore B foi um fator importante no sucesso da implementação da aplicação de interface gráfica e de seu bom desempenho, tanto para explorar os instantes da simulação em forma sequencial, quanto aleatória.

Logo, todos os objetivos foram executados, conforme o planejado. Como resultados preliminares da implementação proposta, pode-se observar que, ela acaba por ser bastante robusta, podendo extrair diversos sinais, independentemente, do estado das máquinas de estado do roteador, coletando indiscriminadamente os sinais que tiverem os valores alterados. Em relação ao desempenho da simulação, observa-se que há um aumento médio no tempo de simulação entre 22.6% e 34.1%.

Como estudo futuro, o suporte a *multi-threads* poderia ser adicionado ao aplicativo de interface gráfica, para que o desempenho da reprodução da simulação não seja deteriorado em redes maiores. Outro aspecto, que poderia ser mais explorado, é o comportamento dos *wrappers*, tanto da rede BrNoC (que atualmente são detectados apenas sinais básicos a partir do roteador), quanto da rede principal. Sendo o ideal capturar estes sinais nas próprias interfaces dos *wrappers*, desta forma teríamos um melhor diagnóstico do comportamento.

REFERÊNCIAS

AEOROFLEX Gaisler. GRMON2 User's Manual. 2014. Disponível em: <http://www.gaisler.com/doc/grmon2.pdf>. Acesso em: 8 jan. 2020.

AGARWAL, Ankur; SHANKAR, Ravi. Survey of Network on Chip (NoC) Architectures & Contributions. **Journal of Engineering, Computing and Architecture**, v. 3, jan. 2009.

BAYER, Rudolf. Symmetric binary B-trees: Data structure and maintenance algorithms. **Acta informatica**, Springer, v. 1, n. 4, p. 290–306, 1972.

BENINI, L.; DE MICHELI, G. Networks on chips: a new SoC paradigm. **Computer**, v. 35, n. 1, p. 70–78, 2002. DOI: 10.1109/2.976921.

BEZERRA, Jeronimo Cunha. Verificação e prototipação de redes intrachip : o estudo de caso Hermes-TB, ago. 2009. Disponível em: <http://tede2.pucrs.br/tede2/handle/tede/5081>. Acesso em: 3 jan. 2020.

CAIMI, Luciano; MORAES, Fernando. Security in Many-Core SoCs Leveraged by Opaque Secure Zones, p. 471–476, jul. 2019. DOI: 10.1109/ISVLSI.2019.00091.

CARARA, E. A. et al. HeMPS - a framework for NoC-based MPSoC generation. In: 2009 IEEE International Symposium on Circuits and Systems. [S.l.: s.n.], 2009. p. 1345–1348. DOI: 10.1109/ISCAS.2009.5118013.

CIORDAS, C. **Monitoring-aware network-on-chip design**. 2008. Tese (Doutorado) – Department of Electrical Engineering. ISBN 978-90-386-1475-5. DOI: 10.6100/IR639317.

CIORDAS, C. et al. An event-based network-on-chip monitoring service. In: PROCEEDINGS. Ninth IEEE International High-Level Design Validation and Test Workshop (IEEE Cat. No.04EX940). [S.l.: s.n.], 2004. p. 149–154. DOI: 10.1109/HLDVT.2004.1431260.

CULLER, David; SINGH, Jaswinder Pal; GUPTA, Anoop. **Parallel Computer Architecture: A Hardware/Software Approach**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. ISBN 9780080573076.

FRIEDERICH, Stephanie; HEISSWOLF, Jan; BECKER, Jürgen. Hardware/Software Debugging of Large Scale Many-Core Architectures. In: PROCEEDINGS of the 27th Symposium on Integrated Circuits and Systems Design. Aracaju, Brazil: Association for Computing Machinery, 2014. (SBCCI '14). ISBN 9781450331562. DOI: 10.1145/2660540.2661013.

IEEE 1076 - 2019 - IEEE Standard for VHDL Language Reference Manual. 2019. Disponível em: <https://standards.ieee.org/standard/1076-2019.html>. Acesso em: 8 mai. 2021.

IEEE 1666 - 2011 - IEEE Standard for Standard SystemC Language Reference Manual. 2012. Disponível em: <https://standards.ieee.org/standard/1666-2011.html>. Acesso em: 8 mai. 2021.

PASRICHA, Sudeep; DUTT, Nikil. **On-Chip Communication Architectures: System on Chip Interconnect**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN 012373892X.

PYTHON 3.8.12 documentation. 2019. Disponível em: <https://docs.python.org/3/>. Acesso em: 16 jun. 2021.

RUARO, M. et al. A data extraction and debugging framework for large-scale MPSoCs. In: 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS). [S.l.: s.n.], 2016. p. 616–619. DOI: 10.1109/ICECS.2016.7841277.

RUARO, Marcelo et al. Memphis: a framework for heterogeneous many-core SoCs generation and validation. **Design Automation for Embedded Systems**, v. 23, dez. 2019. DOI: 10.1007/s10617-019-09223-4.

TKINTER — Python interface to Tcl/Tk. 2020. Disponível em: <https://docs.python.org/3/library/tkinter.html>. Acesso em: 18 jun. 2021.

WACHTER, Eduardo et al. BrNoC: A broadcast NoC for control messages in many-core systems. **Microelectronics Journal**, v. 68, p. 69–77, 2017. ISSN 0026-2692. DOI: 10.1016/j.mejo.2017.08.010.

WOLF, W.; JERRAYA, A. A.; MARTIN, G. Multiprocessor System-on-Chip (MPSoC) Technology. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 10, p. 1701–1713, out. 2008. ISSN 1937-4151. DOI: 10.1109/TCAD.2008.923415.

YOO, Hoi-Jun; LEE, Kangmin; KIM, Jun Kyoung. **Low-Power NoC for High-Performance SoC Design**. 1st Edition. [S.l.]: Imprint CRC Press, 2008. ISBN 9781315219530. DOI: 10.1201/9781315219530.

APÊNDICES

APÊNDICE A - TABELA DE QUANTITATIVO DE SINAIS USADOS E AFETADOS POR ESTADO NA I-FSM

Tabela de quantitativo de sinais usados e afetados por estado na FSM1

EA_manager_input	S_INIT_INPUT	ARBITRATION_INPUT	LOOK_TABLE_INPUT	TEST_SPACE_AVAIL	SERVICE_INPUT	TABLE_WRITE_INPUT	WRITE_BACKTRACK_INPUT	TEST_SEND_LOCAL	WRITE_CLEAR_INPUT	WAIT_REQ_DOWN	WAIT_REQ_DOWN_NACK	SEND_NACK
ID Estado	0	1	2	3	4	5	6	7	8	9	10	11
Transições	0,1	2	3	4,6,8,11	5,6,7,9,11	9	9	7,8	9	0,9	0,10	10
EA_manager				EA!sm2 = 12 ?				10 ou 92				
Entradas												
in_source_router_seek[5]						<1>[sel_port]						
in_target_router_seek[5]						<2>[sel_port]	<2>[sel_port]					
in_service_router_seek[5]				Lê	Lê	<3>[sel_port]	<3>[sel_port]					
in_payload_router_seek[5]						<4>[sel_port]	<4>[sel_port]					
in_req_router_seek[5]										Lê	Lê	
in_ack_router_seek						int_in_ack_router_seek						
in_nack_router_seek												
in_fail_router_seek						fail_with_mode_out						
in_opmode_router_seek[5]						<5>[sel_port]						
in_sel_req_backtrack_seek												
in_ack_send_kernel_seek												
Saídas												
out_req_router_seek						(int_out_req_router_seek) and not(fail_with_mode_out)						
out_ack_router_seek						(int_out_ack_router_seek) or (fail_with_mode_in)						
out_nack_router_seek	0[sel_port]											0 ou 1[sel_port]
out_req_send_kernel_seek												
Internos												
fail_with_mode_in						(in_fail_router_seek) and (in_opmode_router_seek)						
int_in_ack_router_seek						(in_ack_router_seek) or (fail_with_mode_out)						
int_in_req_router_seek						(in_req_router_seek) and not(fail_with_mode_in)						
int_out_ack_router_seek												
int_out_ack_router_seek	0[sel_port]			1[sel_port]		1[sel_port]	1[sel_port]		1[sel_port]	1[sel_port]	Lê	
int_out_req_router_seek	Lê	next_port		Lê	Lê	Lê	Lê	Lê	Lê	Lê		
sel_port				Lê	Lê	Lê						
free_index												
in_the_table				Lê	Lê							
req_task									CLR_SVC[source_index]			
req_int	Lê											
space_aval_in_the_table				Lê								
pending_local						0						
sel				Lê								
used_table[8]						1						
sourcet_table[8]						<1>[free_index]						
target_table[8]						<2>[free_index]	<2>[source_index]					
service_table[8]						<3>[free_index]	SCT_SVC[source_index]		CLR_SVC[source_index]			
payload_table[8]						<4>ou(<4>+1)[free_index]	<4>					
opmode_table[8]						<5>[free_index]						
pending_table[8]							1[source_index]		1[source_index]			
backtrack_port_table[8]						sel_port			LOCAL			
my_payload_table							payload_table					
source_router_port_table							sel_port					
source_index				Lê			Lê		Lê			
searchpath_pending_in_table					Lê							
backtrack_pending_in_table				Lê	Lê							
service_pending_in_table					Lê							
PORTAS DE LIDAS:	2	1	0	7	6	7	5	0	2	2	2	1
PORTAS DE ESCRITAS:	2	0	0	1	0	9	6	0	4	1	0	1
TOTAL	4	1	0	8	6	16	11	0	6	3	2	2

Escreve
Lê
É índice
<conectados>
sinal interno
[índice]
escreve[índice]
le[índice]
Alimenta de outros sinais
Saída depente de outros sinais
CAM

APÊNDICE B - TABELA DE QUANTITATIVO DE SINAIS USADOS E AFETADOS POR ESTADO NA O-FSM

Tabela de quantitativo de sinais usados e afetados por estado na FSM2

EA_manager	S_INIT	ARBITRATION	TEST_SERVICE	SERVICE_BACK_TRACK	BACKTRACK_PROPAGATE	INIT_BACKTRACK	PREPARE_NEXT	BACKTRACK_MOUNT	CLEAR_TABLE	COMPARE_TARGET	SEND_LOCAL	PROPAGATE	WAIT_ACK_PORTS	INIT_CLEAR	END_BACKTRACK	COUNT
ID Estado	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Transições	0,1	2	3,9,10,11	0,4,7	0,4,6	0	0	0,14	0,8	5,10,11	0,10,13	12	0,8,10,12	0,15	10	13
EA_manager input									EAfsm1 = 0 ?							
Entradas																
in_ack_router_seek						int_in_ack_router_seek					Lê[LOCAL]					
in_nack_router_seek				Lê[backtrack_port]							Lê[LOCAL]		2[TODAS] = 1?			
in_fail_router_seek								fail_with_mode_out								
in_sel_req_backtrack_seek																
in_ack_send_kernel_seek											Lê					
Saídas																
out_req_router_seek						(int_out_req_router_seek) and not(fail_with_mode_out)										
out_ack_router_seek						(int_out_ack_router_seek) or (fail_with_mode_in)										
out_nack_router_seek																
out_opmode_router_seek						opmode_table[sel]										
out_service_router_seek						service_table[sel]										
out_source_router_seek						source_table[sel]										
out_target_router_seek						target_table[sel]										
out_payload_router_seek						payload_table[sel]										
out_req_backtrack_seek						(reg_backtrack(0)) or (req_backtrack[1]) or (reg_backtrack[2])										
out_req_send_kernel_seek	0										1					
Internos																
fail_with_mode_in							(in_fail_router_seek) and (in_opmode_router_seek)									
int_in_ack_router_seek				Lê[backtrack_port]			(in_ack_router_seek) or (fail_with_mode_out)				Lê[LOCAL]		[1]TODAS] =1?			
int_in_req_router_seek							(in_req_router_seek) and not(fail_with_mode_in)									
int_out_ack_router_seek				1 ou 0 [backtrack_port]												
int_out_req_router_seek	0						0[backtrack_port]		0[sel]		- ou 1[sel]	- ou 1[LOOP]	- ou 0[LOOP]			
sel_port																
in_the_table																
req_task	Lê															
req_int	Lê															
space_aval_in_the_table																
pending_local		Lê							0[sel]		0 ou 1[sel]					
sel		prox Lê	Lê		Lê	Lê	Lê		Lê	Lê	Lê	Lê	Lê	Lê	Lê	
used_table[8]			<2>[sel]	Lê[sel]					Lê	0[sel]						
source_table[8]						<1>[sel]	<2>[sel]	<2>[sel]					Lê[sel]			
target_table[8]			Lê[sel]			BCT_SVC[sel]				Lê[sel]	Lê[sel]					
service_table[8]							<3>[sel]	<1>[sel]		Lê[sel]	Lê[sel]		Lê[sel]	CLR_SVC[sel]		
payload_table[8]																
opmode_table[8]																
backtrack_port_table[8]			<1>[sel]			<1>[sel]							Lê[sel]	4[source_index]		
my_payload_table[8]							<3>[sel]	<1>[sel]								
source_router_port_table[8]							<2>[sel]									
pending_table[8]						1		<1>[sel]	0[sel]		- ou 1[sel]		nack_recv[sel]	1		
source_index														Lê		
searchpath_pending_in_table																
backtrack_port			<1>				Lê									
backtrack_id			<2>	Lê												
backtrack_pending_in_table																
reg_backtrack								<2>							<1>[sel]	
is_my_turn_send_backtrack							<1>								<1>	
service_pending_in_table																
vector_ack_ports		others => 0											Lê - OU 1[1]			
vector_nack_ports		others => 0											Lê - OU 1[2]			
count_clear		others => 0												Lê != 15?]>=15		"0001"
fatal_recv													Lê			
PORTAS DE LIDAS:	2	0	6	3	3	2	4	3	2	3	7	1	9	3	2	0
PORTAS DE ESCRITAS:	5	1	2	0	1	3	4	3	4	0	4	1	3	3	1	1
TOTAL	7	1	8	3	4	5	8	6	6	3	11	2	12	6	3	1