PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

HC-MPSOC: PLATAFORMA DO TIPO CLUSTER PARA SISTEMAS EMBARCADOS

FELIPE GÖHRING DE MAGALHÃES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fabiano Passuelo Hessel

Porto Alegre
2013

M188H Magalhães, Felipe Göhring de
HC-MPSOC : plataforma do tipo cluster para sistemas embarcados / Felipe Göhring de Magalhães. – Porto Alegre, 2013.
83 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Fabiano Passuelo Hessel.

1.Informática. 2. Multiprocessadores. 3. Arquitetura de Computador. I. Hessel, Fabiano Passuelo. II. Título.

CDD 004.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "HC-MPSOC: Plataforma do Tipo Cluster para Sistemas Embarcados" apresentada por Felipe Gohring de Magalhães como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 21/03/2013 pela Comissão Examinadora:

Prof. Dr. Fabiano Passuelo Hessel – Orientador — PPGCC/PUCRS

Prof. Dr. César Augusto Missio Marcon – PPGCC/PUCRS

Prof. Dr. Altamiro Amadeu Susin – UFRGS

Homologada em 25/06/2013, conforme Ata No. 011 pela Comissão Coordenadora.

p/ Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Dedico este trabalho a todos aqueles que, mesmo indiretamente, auxiliaram em seu desenvolvimento.

"One step at a time, one foot in front of the other,
I'm gonna get through this one way or another"
(Four Year Strong - One Step At A Time)

AGRADECIMENTOS

Obrigado ao professor Marcon por me ensinar a diferença entre o justo e o certo.

HC-MPSOC: PLATAFORMA DO TIPO CLUSTER PARA SISTEMAS EMBARCADOS

RESUMO

Sistemas intrachip multiprocessados (MPSoCs) podem ser encontrados em praticamente todos os ramos do mercado e o projeto desses sistemas normalmente apresenta diversas restrições, como por exemplo área do chip utilizada, o que o dificulta. MPSoCs do estado da arte utilizam redes intrachip como meio de comunicação principal, e a tendência é que sistemas baseados em redes intrachip continuem a ser utilizados por um longo tempo, graças a uma maior flexibilidade em seu projeto e também uma alta capacidade de comunicação. Porém, tais sistemas ainda apresentam certas restrições em seu uso, como por exemplo a localização das tarefas que o compõem. Técnicas de mapeamento e particionamento de tarefas de uma aplicação buscam solucionar tais problemas, ou ao menos diminui-los a um ponto não crítico, mas nem sempre são bem sucedidos.

Neste contexto, arquiteturas do tipo *cluster* surgem como uma alternativa viável para MPSoCs, normalmente apresentando uma arquitetura híbrida em sua constituição, utilizando mais de um meio de comunicação, podendo assim agrupar elementos por questões de "afinidade" e ainda assim utilizando meios de comunicação com grande paralelismo, como redes intrachip.

Desta maneira, este trabalho introduz o HC-MPSoC, uma arquitetura *clusterizada* para sistemas intrachip, que utiliza redes intrachip e barramentos de uma maneira conjunta, formando grupos de elementos distribuídos de forma independente por todo sistema. É apresentando ainda, o HellfireOS, sistema operacional de tempo real adaptado para executar sobre a arquitetura, com *drivers* disponibilizados para uso. Todos os módulos do HC-MPSoC, assim como do HellfireOS, e os resultados obtidos utilizando a arquitetura, são apresentados no decorrer do texto.

Palavras Chave: Sistemas embarcados, Barramento, NoC, Cluster, RTOS, Comunicação Embarcada.

ABSTRACT

Multiprocessor System-on-Chip (MPSoC) can be found in virtually all market branches and the design of such systems typically has several restrictions such as chip area used, which hampers. State-of-art MPSoCs uses networks-on-chip as the primary means of communication, and the trend is that systems based on networks intrachip continue to be used for a long time, thanks to greater flexibility in their design and also a high capacity communication. However, such systems also have certain restrictions on its use, such as the location of the tasks that compose it. Mapping and partitioning techiniques seek to solve these problem, or at least decrease it to a non critical point, but are not always successful in this job.

In this context, cluster-based architectures emerges as a viable alternative to MPSoCs. Such systems typically have a hybrid architecture in its constitution, using more than one communication medium, thus being able to group elements by questions of "affinity" and still using high-speed communication medias, such as networks-on-chip.

Thus, this work introduces the HC-MPSoC, an architecture for cluster-based intrachip systems, which uses buses and networks-on-chip in a joint way, forming groups of elements independently distributed throughout the system. The HellfireOS is also presented, a real time operating system adapted to run on the platform, counting with a full set of drivers throughout a high-level API. All HC-MPSoC modules as well as the HellfireOS modules, and the results obtained using the platform are presented along the text.

Keywords: Embedded systems, Bus, Network-on-Chip, Cluster, RTOS, Embedded Communication.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

SE – Sistemas Embarcados

RTS – Real time system

SO – Sistema Operacional

RTOS – Real Time Operating System

SOC – System-on-Chip

MPSOC – Multiprocessor System-on-Chip

NOC – Network-on-Chip

HC-MPSOC – Hellfire Cluster-based MPSoC

API – Application programming interface

SUMÁRIO

1. INTRODUÇÃO

Atualmente, é possível encontrar Sistemas Embarcados (SE) em diversos produtos disponíveis no mercado, como por exemplo, em eletrônicos, na indústria robótica e aviônica e em equipamentos médicos. Uma das principais características destes sistemas é o fato de que, normalmente, são projetados para uma aplicação específica, além de apresentarem alguns requisitos de projeto, como memórias de dados e código pequenas e consumo de energia limitado [14].

Muitos desses sistemas apresentam, além das restrições citadas anteriormente, restrições temporais, onde uma única requisição não atendida no tempo devido coloca em risco o funcionamento do sistema como um todo. Tais sistemas são conhecidos como Sistemas de Tempo Real (RTS) e, a fim de garantir seus funcionamentos, é muito comum a utilização de um Sistema Operacional (OS) que possa atender os requisitos de tempo real, sendo este conhecido como Sistema Operacional de Tempo Real (RTOS). Para tornar este gerenciamento de tempo real possível, um RTOS utiliza uma unidade de escalonamento de tempo real [18] que, além de gerenciar a ordem de execução das tarefas presentes, escalona estas de forma a tentar cumprir todas restrições temporais.

A fim de otimizar, dentre outras coisas, seu custo e desempenho computacional e energético, é desejável que SEs possam ser implementados em uma única pastilha de silício (*chip*), sendo este sistema conhecido como *System-on-Chip* (SoC). Um SoC normalmente é composto de diversos componentes, como: CPUs, memórias, barramentos e DSPs no mesmo *chip*. A evolução tecnológica permitiu, ao longo dos anos, que mais de uma unidade de processamento fosse utilizada em um mesmo chip e o SoC resultante passou a ser denominado Multiprocessor System-on-Chip (MPSoC). Uma das grandes vantagens de se utilizar MPSoCs é a possibilidade da divisão da carga de processamento entre os processadores, o que pode acarretar em um ganho de desempenho energético e também computacional.

A Figura 1.1 mostra um MPSoC formado por quatro unidades de processamento, interligadas por um meio de comunicação qualquer, uma memória compartilhada por todas unidades e acessada via o meio de comunicação e, ainda, um meio de comunicação com o mundo externo (*Input/Output* - I/O).

A comunicação entre os componentes internos de um SoC ou de um MPSoC é dada via um meio de comunicação, dentre os quais podem-se destacar:

- barramento, unidade básica de comunicação em que os pacotes trocados entre os componentes ocupam o meio de comunicação de forma exclusiva, e;
- *redes intra-chip* (NoC) em que os conceitos de redes de computadores, com roteadores que orientam o tráfego de informação, são aplicados dentro de um chip.

Mais detalhes a respeito do funcionamento desses meios de comunicação serão apresentados na Seção 1.1.

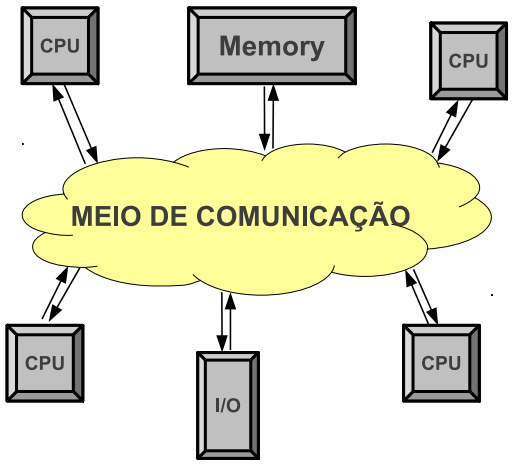

Figura 1.1: Exemplo de um MPSoC

Dentro do contexto apresentado, dois pontos importantes podem ser ressaltados, assim como um questionamento a respeito do futuro de sistemas embarcados:

- a crescente relevância de SEs e sua utilização cada vez maior nos sistemas eletrônicos presentes no cotidiano das pessoas, onde tarefas, que antes eram exclusivas a sistemas de propósito geral, como, por exemplo, processamento de imagens, passaram a ser executadas dentro de um chip;
- o aumento da complexidade das aplicações cresceu conjuntamente com a evolução de tecnologias, porém, ferramentas de suporte ao desenvolvimento de tais aplicações não cresceram no mesmo ritmo [28], e;
- qual será o próximo passo da evolução dos Sistemas Embarcados?

Quanto a este questionamento pode-se dizer que, naturalmente a tendência é incorporar novas funcionalidades ao sistema, tanto em nível de *hardware* quanto em nível de *software*. Por exemplo, se pode imaginar que em um futuro breve dispositivos embarcados sejam capazes de produzir imagens holográficas em tempo real.

Refletindo sobre esta possibilidade, a viabilização deste tipo de aplicação passa pela proposição de novas arquiteturas de *hardware* e *software*. No *hardware* é necessário que a arquitetura permita o aumento de desempenho de execução da aplicação. Isto não somente pelo aumento do número de processadores ou pela utilização de um processador com poder de processamento maior, mas por estruturas que permitam e dêem suporte a isto. Também, os meios de comunicação devem ser repensados e re-projetados para atender a crescente demanda de desempenho das aplicações.

Do ponto de vista do *software* novas propostas, como a virtualização, devem ser estudadas, assim como novas políticas de escalonamento de tarefas e técnicas de mapeamento e particionamento das mesmas. Ainda, se deve considerar a crescente demanda do consumo de energia destes sistemas, fato que nunca pode ser ignorado em sistemas embarcados.

Logo, existem diferentes fronteiras a serem transpostas para a evolução dos sistemas embarcados, isto é, para se chegar a nova geração de sistemas embarcados. Neste sentido, este trabalho

busca transpor a fronteira da arquitetura de *hardware*, propondo uma nova arquitetura para sistemas embarcados que permita aumentar o desempenho de execução de aplicações complexas, ao mesmo tempo em que seja simples de programá-la.

1.1 Meios de Comunicação

Em sistemas embarcados multiprocessados, duas ou mais CPUs com um consumo de energia reduzido são utilizadas, diminuindo assim suas frequências de operação e, consequentemente sua capacidade computacional. Porém, graças ao uso de paralelismo, esses sistemas ainda são capazes de realizar tarefas complexas, pois a carga de processamento total pode ser dividida entre todas as unidades do sistema e executar em paralelo, compensando a menor capacidade de cada CPU.

Um cuidado que deve ser tomado ao se utilizar MPSoCs está no fato de que, não apenas as capacidades computacional e de armazenamento do sistema influem no seu desempenho, mas também a capacidade de comunicação. Por exemplo, em um sistema altamente comunicante, se o meio de comunicação escolhido não suportar diversas trocas de mensagens em um tempo aceitável, o desempenho será prejudicado.

Uma tendência de uso, até poucos anos atrás, o uso de um ou mais barramentos como meio de comunicação principal [23], está mudando, pois NoCs já fazem parte de muitos sistemas e seu uso só tende a aumentar com o passar do tempo.

1.1.1 Barramento

Em sistemas que utilizem barramentos, N nodos[1] são interligados por uma ou mais vias que efetuam as trocas de mensagens de maneira individual, ou seja, quando um nodo está utilizando a via, todos os outros devem aguardar pelo término deste uso. O principal motivo do barramento ser largamente usado ainda hoje está na sua simplicidade e eficiência [23].

Uma grande limitação dos barramentos está em sua escalabilidade, ou seja, seu uso em sistemas com muitos nodos. Para situações em que um barramento simples (exemplificado na Figura 1.2) não é capaz de lidar com todo tráfego de informações do sistema, topologias mais complexas podem ser usadas, como por exemplo o barramento hierárquico.

[1] neste caso, um nodo corresponde a um ponto unitário do sistema, podendo representar uma CPU, uma memória ou um DSP, por exemplo.

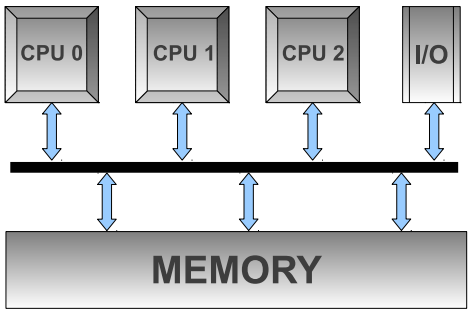

Figura 1.2: Barramento Simples

A Figura 1.3 apresenta um sistema interligado por barramentos hierárquicos. É possível observar diferentes níveis de barramentos. Esta técnica aumenta a escalabilidade de sistemas baseados em barramentos, porém aumenta também a complexidade de implementação do sistema.

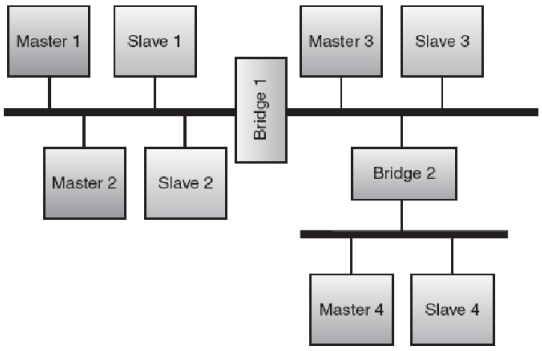

Figura 1.3: Barramento Hierárquico [24]

1.1.2 Network-on-Chip

Conforme descrito anteriormente, barramentos possuem limitações de escalabilidade, o que pode afetar o desempenho do sistema como um todo. A fim de solucionar este problema, algumas soluções foram pesquisadas, como por exemplo, as Redes Intra-chip.

Nesse modelo, a abordagem para comunicação muda em relação àquela adotada pelos barramentos. Diferentemente dos barramentos, em que normalmente todos os elementos são interligados por um meio simples e direto de comunicação, em NoCs roteadores gerenciam todo tráfego e direcionam os pacotes da maneira mais adequada, realizando a troca de informação de maneira paralela e independente.

Dentre as topologias de NoC existentes, é possível ressaltar três: malha, torus e octagonal, descritas a seguir.

O modelo mais usado é o de rede tipo malha 2D regular(do inglês, *mesh*) onde todas as conexões possuem o mesmo comprimento, o que facilita seu projeto. Nesse modelo todos roteadores,

excluindo os nodos externos que possuem apenas duas ou três conexões, estão interligados a quatro roteadores vizinhos, agilizando a troca de pacotes [23].

Um exemplo de rede em malha contendo 16 processadores é mostrado na Figura 1.4.

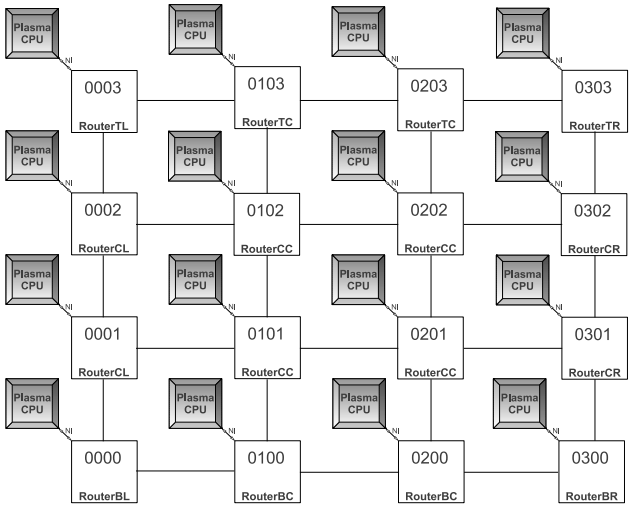

Figura 1.4: Rede em Malha Interligando CPUs

Dentre diferentes modelos encontrados para uso, o modelo de NoC escolhido foi a NoC HERMES-Handshake [22], sendo que esta usa uma topologia do tipo malha e é constituída pelos seguintes módulos:

- roteadores;
- *buffers*, e;
- controladores do fluxo de informações dos roteadores (*switchcontrol*).

A Figura 1.5 apresenta a organização interna de um roteador da NoC HERMES. Na figura é possível observar a presença de 5 portas, sendo 4 de envio e recebimento de pacotes pela rede e uma local. Em cada uma das portas existe um *buffer* de tamanho *n* parametrizável, responsável por armazenar temporariamente as informações que chegam e saem do roteador. Os pacotes trocados pela rede são denominados *flits*, podendo variar seu tamanho de acordo com as necessidades da aplicação. O tamanho usado pela distribuição utilizada neste trabalho é de 16 bits. O controle do tráfego interno de informações do roteador é feito por um componente do roteador chamado *switchcontrol*, sendo esse composto por duas unidades:

- *controle*, encarregado de realizar o chaveamento para troca de informações entre as portas, e;
- *árbitro*, módulo gerenciador de pacotes. Cada uma das filas ao receberem um novo pacote, requisitam chaveamento ao árbitro. O árbitro seleciona a de maior prioridade e solicita chaveamento à lógica de chaveamento (controle). Sendo possível, a conexão é estabelecida e o árbitro é informado. Por sua vez, o árbitro informa a fila que começa a enviar os pacotes armazenados.

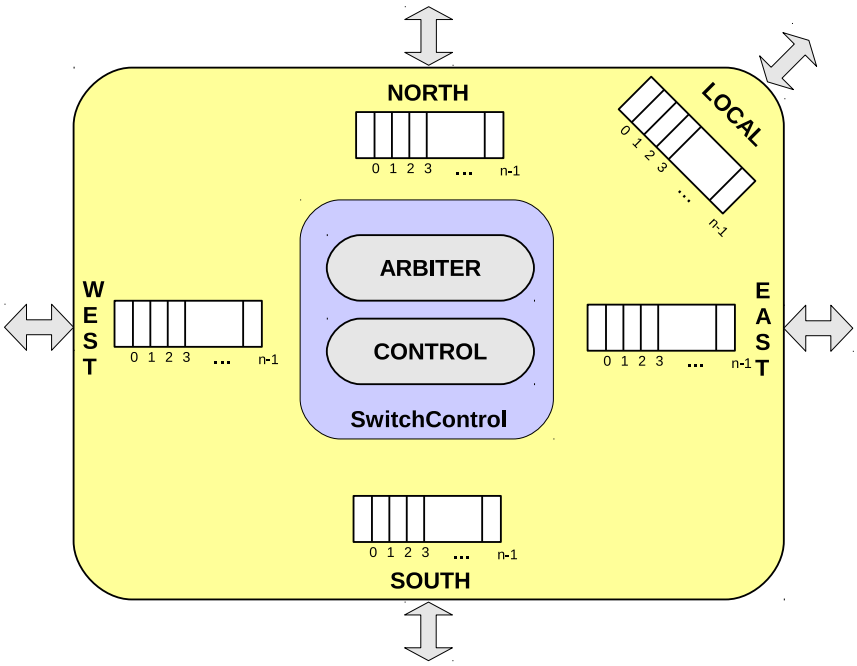

Figura 1.5: Organização do Roteador da HERMES

O escalonamento interno entre as filas é realizado utilizando-se o algoritmo *Priority Round Robin*, onde o escalonador funciona de maneira circular, porém utilizando diferentes prioridades. Já o algoritmo de roteamento de pacotes da NoC, que determina o trajeto que os pacotes irão fazer para ir do ponto A para o ponto B, é o XY [23].

1.2 Cluster

Um *cluster* pode ser definido como um sistema onde dois ou mais sistemas independentes trabalham de maneira cooperativa, a fim de atingir um objetivo comum. Em outras palavras, cada um dos sistemas independentes recebe uma parcela do trabalho, dividindo assim a carga total do sistema em parcelas menores.

Clusters são usados na indústria há muitos anos, com o intuito de processar grandes quantidades de informação em um menor espaço de tempo, sendo que um dos primeiros modelos surgiu na década de 60. Inicialmente eram projetados para operar sobre *mainframes*[2], sendo necessário o uso de um *software* que fizesse a divisão das tarefas entre os diferentes pontos.

Por volta da década de 80 com o aumento da tecnologia no desenvolvimento de processadores, assim como a evolução das redes de computadores, novos modelos de *clusters* começaram a surgir. Máquinas de uso geral puderam, então, ser usadas em uma rede para dividir a carga de processamento.

Dentre os diversos modelos de *clusters* disponíveis, é possível destacar três:

[2]computador dedicado ao processamento de um volume grande de informações

1. **modelo de balanceamento de carga**: controla a distribuição do processamento de forma equilibrada. Para isso, requer um monitoramento constante na sua comunicação e em seus mecanismos de redundância, pois se ocorrer alguma falha, haverá uma interrupção no seu funcionamento;

2. **modelo de alta disponibilidade**: modelado para sistemas onde a estabilidade e disponibilidade são mais importantes do que o desempenho. Normalmente possuem mecânismos de proteção a erros, e;

3. **modelo de alto desempenho**: utilizado principalmente em aplicações que exigem um grande poder computacional, usam computadores comuns e sistemas operacionais gratuitos.

Uma forte tendência nos últimos anos é a utilização de modelos de *clusters* em sistemas embarcados, ampliando assim suas capacidades de comunicação e processamento.

Sendo assim, este trabalho descreve a implementação de uma arquitetura para sistemas embarcados, utilizando os conceitos apresentados de barramentos, NoCs e *clusters*.

1.3 Motivação

A partir do surgimento de sistemas embarcados multiprocessados tarefas que exigiam um grande poder computacional puderam ser executadas a partir de um pequeno chip. Essa capacidade proporcionou o surgimento de novas tecnologias que agrupam em uma única plataforma diversas funções. Ao mesmo tempo, muitas tarefas realizadas por SE são críticas, como por exemplo o controle do sistema de pouso de um avião, ou até mesmo os controles de segurança de uma usina nuclear.

Deste modo, é de grande importância que os SEs apresentem não apenas grande poder computacional e de comunicação, mas também apresentem estabilidade e segurança na sua execução. Arquiteturas do tipo *cluster* apresentam características que contemplam todos os pontos citados. Por exemplo, em um sistema crítico em que a integralidade das informações é crucial, utilizando-se uma arquitetura *cluster* é possível replicar essas informações em diferentes pontos, garantindo assim sua segurança.

Um outro possível cenário de uso de *clusters* é a divisão das aplicações conforme suas características, isolando as tarefas que compõem o sistema entre diferentes domínios. Por exemplo, em um sistema multimídia uma possível divisão poderia ser feita conforme a Figura 1.6, aonde é possível observar quatro domínios, cada qual com uma especialização: vídeo, áudio, entrada/saída e memória.

Sendo assim, este trabalho apresenta uma arquitetura utilizando *clusters* para sistemas embarcados, introduzindo os conceitos de sua implementação e, também, do suporte de alto nível, via sistema operacional, utilizando o HellfireOS [8], arquitetura esta denominada HC-MPSoC (Hellfire Cluster-based MPSoC) .

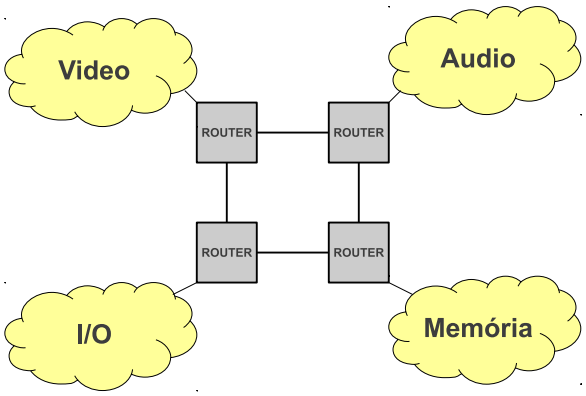

Figura 1.6: Exemplo de Cluster Multimídia

1.4 Objetivos

O objetivo principal deste trabalho é propor uma nova arquitetura para Sistemas Embarcados, explorando as características de barramentos e NoCs. Esta arquitetura será composta por ambos meios de comunicação e por unidades de processamento distribuídas no sistema. A plataforma proposta foi descrita em linguagem de descrição de *hardware* (VHDL), executando em cada instância de um processador, uma imagem[3] do sistema operacional HellfireOS.

A validação dos resultados foi obtida através de simulações em nível RTL, assim como em processos de síntese para tecnologias FPGA e de 65nm da ST.

1.5 Organização do Texto

Este trabalho está organizado da seguinte maneira: o próximo capítulo aborda uma revisão do estado da arte de sistemas embarcados utilizando clusters, seguido pelo Capítulo 3, onde o Sistema Hellfire é apresentado. Já no Capítulo 4 o trabalho desenvolvido é apresentado, com o detalhamento de todos os módulos que compõem o HC-MPSoC, sendo que os resultados obtidos são apresentados no Capítulo 5. Por último, o Capítulo 6 contém uma breve conclusão deste trabalho, assim como uma discussão sobre trabalhos futuros a serem desenvolvidos sobre a plataforma.

[3] no contexto apresentado, imagem corresponde ao arquivo binário gerado após a compilação e que será executado pelo simulador ou pela plataforma de *hardware*.

2. ESTADO DA ARTE

Neste capítulo serão apresentados alguns trabalhos relacionados a arquiteturas de *clusters* embarcados. Será feita uma análise de cada trabalho individualmente e, então uma comparação das metodologias adotadas. Apesar de muitos trabalhos, de outras áreas de pesquisa inclusive, apresentarem algumas características que poderiam entrar nesta comparação, apenas os casos de sistemas *intrachip* foram levados em consideração.

2.1 Performance Evaluation Of Cluster-Based Homogeneous Multiprocessor System-on-Chip Using FPGA Device

O trabalho [20] apresenta um MPSoC formado por 17 processadores NiosII [4] agrupados em quatro grupos de quatro processadores cada, e mais um processador central que controla todo o sistema, dividindo as tarefas entre os grupos de processadores. Cada grupo esta interligado via uma memória compartilhada e cada processador possui uma memória local para execução de suas tarefas. Para realizar a comunicação entre os grupos de processadores e o processador central uma NoC irregular foi utilizada. O acesso a periféricos, assim como a entrada e saída de dados é todo realizado no processador mestre, centralizando o controle de operações do MPSoC em um único ponto. A Figura 2.1 apresenta o modelo de cluster utilizado no trabalho referenciado.

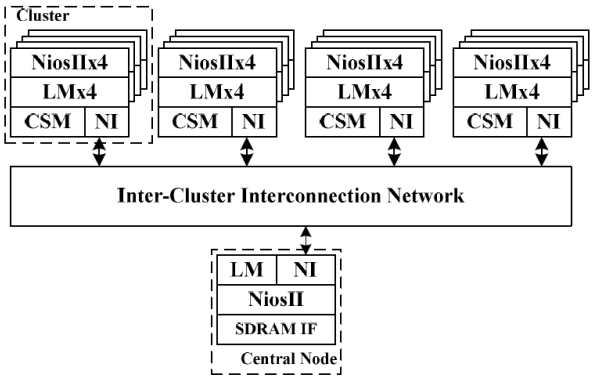

Figura 2.1: Arquitetura de Cluster com controle centralizado - [20]

Algumas considerações podem ser feitas a respeito do modelo adotado pelos autores, como por exemplo a possibilidade de diferentes configurações da arquitetura do modelo. Pelo fato de uma NoC irregular ter sido usada, com roteadores específicos, cada mudança desejada no *layout* da arquitetura implica diretamente em mudanças nos roteadores, que devem ser adaptados à nova configuração. Um outro ponto relevante é o fato de que uma unidade central foi utilizada, controlando o funcionamento de todo o sistema, assim como a entrada e saída de dados. Essa prática pode limitar a capacidade do sistema, funcionando como um gargalo, pois o desempenho da unidade central pode não ser satisfatório, se comparado com o os escravos. Para verificação

do sistema, os autores apresentam dois casos de teste, Multiplicação de Matrizes e Transformada Rápida de Fourier, porém não se tem maiores detalhes de sua implementação, passando a idéia de que nenhum suporte de alto nível foi utilizado.

As principais diferenças entre o trabalho apresentado em [20] e o trabalho aqui proposto está no fato de que este trabalho apresenta uma arquitetura mais facilmente configurável, pois utiliza uma NoC parametrizável. Outro ponto importante é o controle centralizado apresentado por [20], diferente do completo paralelismo empregado pelo trabalho proposto. Uma última consideração reside no fato de que nenhum tipo de suporte de alto-nível foi apresentado por [20], ao contrário do que este trabalho propõem.

2.2 FPGA Prototype Design of the Computation Nodes in a Cluster Based MPSoC

Em [15], os autores introduzem um modelo de *cluster* para sistemas embarcados, formado por processadores ARM agrupados em *clusters* de tamanho variável, que se comunicam por um barramento do tipo AMBA-AHB [1]. A comunicação entre *clusters* é realizada por uma NoC que possui um *cluster* em cada nodo. A Figura 2.2 apresenta um modelo do MPSoC utilizado em uma prototipação, com meios de comunicação, processadores e *IPs*. É possível notar a presença de quatro módulos diferentes: *Ethernet In/Out* responsável pela entrada e saída de dados do MPSoC, *PCC* responsável pela comunicação entre *clusters*, *cluster* formado por três processadores e *DDR II Interface* que realiza a interface para uma memória global do sistema.

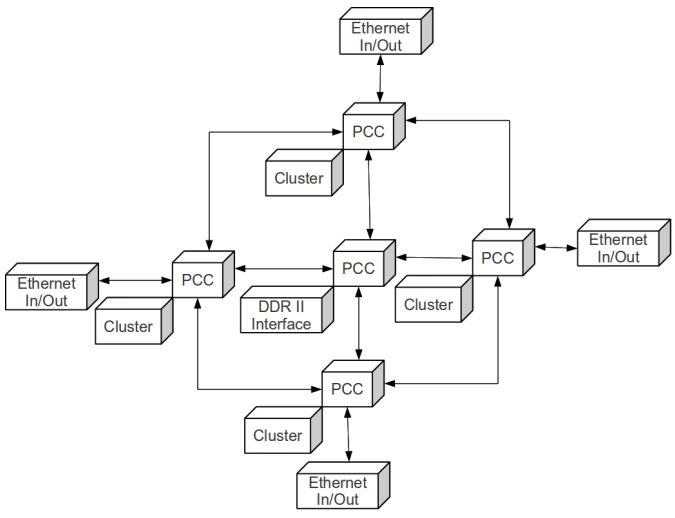

Figura 2.2: Arquitetura de Cluster com *IPs* - [15]

A parte interna do *cluster* é composta por três processadores, uma unidade de controle de memória (DMA) e uma unidade de comunicação com a NoC. Para validar o sistema, um algoritmo de tempo real de processamento de vídeo foi implementado e prototipado em FPGA com o propósito de provar que o sistema garante desempenho de processamento de tempo real.

O sistema introduzido em [15] apresenta uma estrutura muito promissora para uso de *clusters* embarcados, porém, ao contrário do trabalho aqui sugerido, nenhum tipo de suporte alto-nível é apresentado, o que aumenta a complexidade de seu projeto.

2.3 P2012

Um modelo de *cluster* utilizando uma NoC como meio de comunicação entre nodos, e barramentos simples como meio de comunicação interna para cada nodo foi apresentada em [21], nomeado **P2012**. Neste modelo, os nodos são compostos por módulos *IP*, podendo estes serem módulos de *hardware* ou *software*, além de dois módulos para comunicação do nodo com o roteador da NoC. Além da arquitetura de *clusters* introduzida, um algoritmo de decisão entre *hardware* e *software* foi apresentado, para otimização do sistema.

A Figura 2.3 apresenta o modelo de *cluster* introduzido em [21], ilustrando a divisão interna entre módulos de *software* e *hardware*.

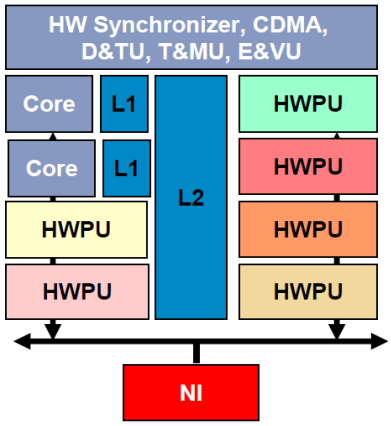

Figura 2.3: Arquitetura de Cluster com módulos de *software* e *hardware* - [21]

O trabalho [21] apresenta um modelo facilmente configurável, escalável para grandes quantidades de núcleos *IP*, apresentando suporte para decisão da divisão das aplicações entre módulos *hardware* e *software*, porém, ao contrário do trabalho sugerido aqui, a plataforma P2012 utiliza núcleos *IP* proprietários, o que pode dificultar o seu uso e integração com projetos que não utilizem tais *IPs*.

2.4 A Clustered NoC-in-Group Communication

Em [26] um sistema de *clusters* utilizando apenas uma NoC com seus roteadores alterados é apresentado. Ao contrário dos trabalhos anteriores citados, o trabalho apresentado em [26] não utiliza barramentos para comunicar os núcleos internos de seus *clusters*, que ficam interligados

diretamente no roteador da NoC. O roteador da NoC utilizada foi alterado, sendo adicionadas mais três portas locais. A Figura 2.4 apresenta a visão do novo roteador utilizado, onde é possível verificar a presença de quatro portas locais, além das portas norte, sul, leste e oeste.

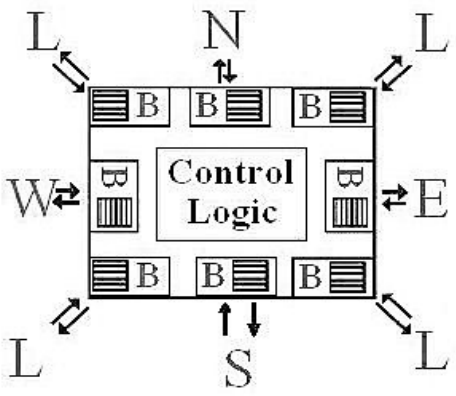

Figura 2.4: Visão do roteador com quatro portas locais - [26]

Esta solução foi adotada visando diminuir a complexidade do desenvolvimento do MPSoC, pois além da estrutura da NoC utilizada, nenhum outro meio de comunicação precisa ser usado. Todavia, para permitir o uso de quatro processadores em um mesmo roteador, a estrutura do mesmo teve de ser alterada, assim como seu algoritmo de escalonamento interno, o que implica em um maior *overhead* de controle, podendo resultar em um menor desempenho.

Uma importante limitação da arquitetura apresentada em [26] está na escalabilidade do modelo. O número de *IPs* em cada *cluster* está limitado a quatro unidades, sempre, o que diminui o espaço explorável de projeto. Outra questão é o fato de que nenhum suporte de alto-nível foi previsto pelo modelo, o que aumenta a complexidade do projeto.

2.5 Embedded Virtualization

O trabalho apresentado em [7] introduz um outro tipo de modelo para utilização de *clusters* embarcados. Enquanto todas as outras soluções ilustradas utilizam implementações físicas dos meios de comunicação e computação, o trabalho apresentado por [7] utiliza técnicas de virtualização para ampliar a capacidade de configuração do MPSoC. Esta solução utiliza processadores MIPS [25] e como meio de comunicação central uma NoC. A Figura 2.5 ilustra um MPSoC formado por quatro processadores (na imagem, cada processador é representado como *PE*) e uma NoC tipo malha central. Em cada nodo, *n* domínios podem ser instanciados, sendo que cada domínio corresponde a um processador virtual.

Para utilizar processadores virtuais, uma camada de *software* foi adicionada ao sistema, sendo esta camada responsável por escalonar estes. Para comunicar os nodos internos, uma me-

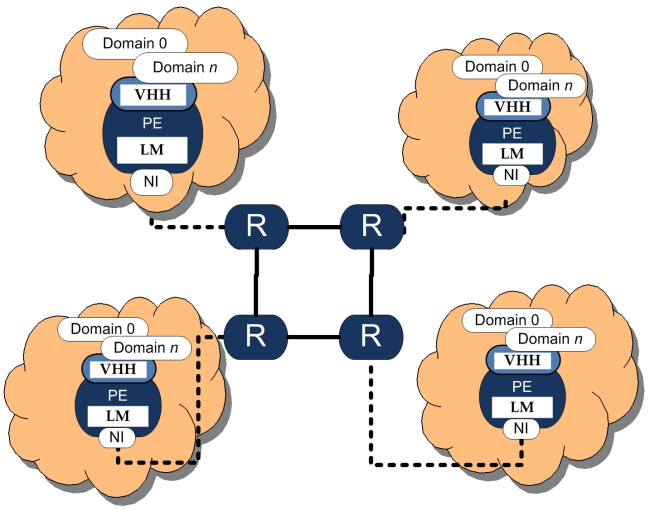

Figura 2.5: Visão geral do MPSoC com clusterização virtual - [7]

mória compartilhada foi utilizada. Como suporte de alto nível, uma API para comunicação foi disponibilizada, via sistema operacional.

Uma importante limitação desta arquitetura reside no fato de que o *overhead* para o uso de processadores virtuais pode ser muito custoso, assim como a escolha do processador utilizado, pois esta escolha esta limitada aos processadores disponibilizados para uso.

2.6 Dynamic SMP Clusters

Em [27] é apresentado um modelo de *clusters* dinâmicos, em que a troca de mensagens internas é realizada por memórias locais. Os módulos internos de cada *cluster* são interligados utilizando-se barramentos e a comunicação externa utiliza conexões ponto-a-ponto. A Figura 2.6 apresenta a visão geral da arquitetura, onde é possível observar a presença de um módulo externo de comunicação, representando as conexões ponto-a-ponto, e três tipos de módulos internos em cada *cluster*: *Local Network*, representando o barramento interno, e os módulos *'P'* e *'M'*, equivalentes aos processadores e memórias, respectivamente.

A grande limitação desta arquitetura está no fato da utilização de uma conexão ponto-a-ponto. Ao mesmo tempo em que este tipo de arquitetura provê uma grande capacidade de comunicação, o seu uso está limitado a poucos nodos, graças a uma grande complexidade de implementação. Um outro ponto que difere o trabalho em [27] do trabalho aqui proposto é o fato que nenhum tipo de suporte de alto nível ao seu uso foi fornecido.

2.7 Hybrid MPSoC Architecture

Uma arquitetura do tipo *cluster* utilizando barramentos internos e uma NoC externa é apresentando em [9]. Em cada nodo três processadores do tipo ARM são utilizados, interligados a

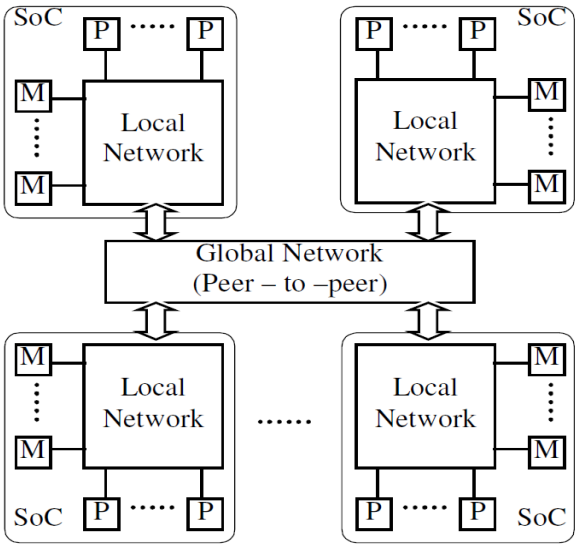

Figura 2.6: Visão geral da Arquitetura [27]

uma interface de comunicação com o roteador da NoC via um barramento. A Figura 2.7 apresenta a arquitetura introduzida, onde é possível observar quatro nodos, contendo três processadores, uma interface e um barramento em cada nodo, assim como uma rede interligando todos os nodos.

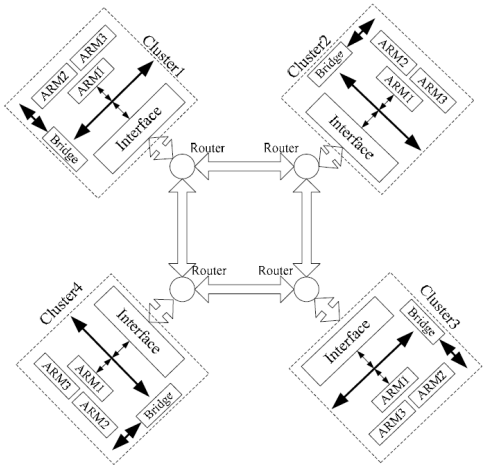

Figura 2.7: Visão geral da Arquitetura [9]

Uma importante diferença da arquitetura apresentada em [9] e do trabalho aqui proposto esta no fato de que o número de processadores internos em cada nodo é fixado em três, além do fato de nenhum tipo de suporte de alto-nível ter sido apresentado.

2.8 Cluster-based Hierarchical NoC

Um modelo de arquitetura irregular é apresentada em [17]. Tal arquitetura é formada por roteadores distribuídos de maneira não linear e, em cada roteador um nodo contendo uma interface de comunicação com o roteador, um barramento e núcleos *IP* podem ser encontrados. A

Figura 2.8 apresenta a visão geral desta arquitetura, onde é possível os elementos identificados como CN* correspondem aos nodos contendo os módulos *IP*. Pode-se ainda visualizar-se três roteadores centrais, nominados CR1, CR2 e CR3, responsáveis por interligar os roteadores que contém nodos. Os roteadores denominados ER* possuem os nodos *clusterizados*.

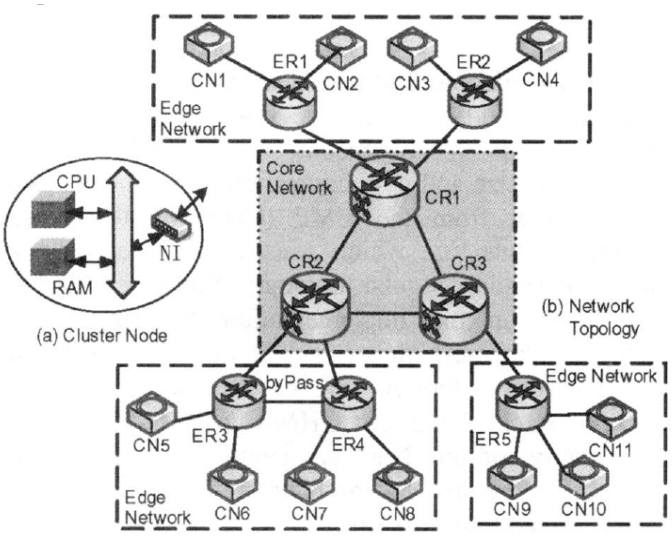

Figura 2.8: Visão geral da Arquitetura [17]

Nenhuma colocação foi feita a respeito de quais núcleos *IP* foram utilizados na arquitetura, ao contrário do trabalho aqui apresentando, que utiliza núcleos do tipo *Plasma*. Uma outra importante consideração está no fato da utilização de uma arquitetura irregular, o que pode acarretar em um maior esforço de fabricação.

2.9 Comparação de Trabalhos

A Tabela 2.1 apresenta uma comparação dos trabalhos apresentados e do trabalho proposto, sendo que seis pontos foram usados como fatores de comparação, conforme segue:

- **A - Meio de comunicação interno do *cluster***: corresponde ao meio de comunicação utilizado pelos nodos internos em cada *cluster*;
- **B - Meio de Comunicação Central entre *clusters***: meio de comunicação utilizado para realizar a comunicação entre diferentes *clusters*;
- **C - Suporte de Alto-nível**: compara os trabalhos, indicando a presença de suporte de desenvolvimento por algum sistema operacional, via API;
- **D - Controle do Sistema**: indica se existe um nodo no sistema que realize a divisão de tarefas, assim como a entrada e saída de dados;

- **E - Complexidade de Configuração da Arquitetura**: compara a complexidade para configurar arquiteturas de diferentes tamanhos e parâmetros e, o quão fácil é feita esta configuração, e;
- **F - Núcleo de Processamento Utilizado**: indica o tipo de núcleo *IP* utilizado. *G-IP* define um núcleo genérico, quando nenhum foi explicitamente definido.

Tabela 2.1: Comparação de Trabalhos

/	**Luo-feng**	**Jin**	**Melpignano**	**Seifi**	**Aguiar**	**Tudruj**	**Chen**	**Leng**	**HC-MPSoC**
A	Bus	Bus	Bus	Roteador	Virtual	Bus	Bus	Bus	Bus
B	NoC	NoC	NoC	NoC	NoC	Ponto-a-ponto	NoC	NoC	NoC
C	Não	Não	Não	Não	Sim	Não	Não	Não	Sim
D	Central	Local	Local	Local	Local	Local	Local	Local	Local
E	Alta	Média	Baixa	Baixa	Baixa	Baixa	Baixa	Baixa	Baixa
F	NiosII	ARM	G-IP	STxP70	Plasma	G-IP	ARM	G-IP	Plasma

A partir da tabela, é possível observar uma grande tendência no uso de barramentos em cada *cluster*, sendo este o meio de comunicação adotado por seis dos oito trabalhos relatados. Um outro ponto em comum na maioria dos casos, com apenas uma exceção, é a adoção de uma NoC como meio de comunicação central, interligando os *clusters* do sistema. A partir disso conclui-se que o trabalho aqui apresentado segue a tendência atual para o uso de *clusters* em sistemas intrachip.

Um ponto importante e de grande relevância é o fato de que em apenas um dos casos citados existe algum tipo de suporte ao desenvolvimento de aplicações nas plataformas discutidas, fato este que o trabalho aqui apresentado suprime, pois disponibiliza uma API (Application programming interface) para uso em conjunto com a plataforma.

3. SISTEMA HELLFIRE

Apesar do tempo para o projeto de sistemas embarcados ser cada vez menor, visando não perder o *time-to-market*, as restrições que o envolvem estão cada mais significativas. Visando auxiliar seu desenvolvimento e, também, diminuir o tempo de projeto, diversas plataformas como, por exemplo, [16] e [32], disponibilizam recursos como ferramentas de simulação e depuração, que agilizam este processo.

Um outro exemplo de ferramenta de auxílio ao desenvolvimento de sistemas embarcados é o Hellfire Framework [8], que será usado como base para o desenvolvimento deste trabalho.

Atualmente, o Sistema Hellfire é constituído por cinco módulos:

- **HellfireOS**: sistema operacional embarcado de tempo real, contendo primitivas para gerenciamento do sistema, de comunicação e migração de tarefas, dentre outras, acessadas via uma API;
- **Plataforma de *Hardware***: composta por *N* processadores MIPS-Plasma [25], interligados por um meio de comunicação, podendo este ser um barramento ou uma NoC;
- **Ambiente de Simulação**: descrito em linguagem 'C', composto por um Instruction Set Simulator (ISS) de um processador MIPS-Plasma e de um modelo em alto nível do meio de comunicação, possuindo precisão de ciclo;
- **Construtor de Arquitetura**: utilizado para especificar a arquitetura destino, assim como configurar todos os parâmetros do HellfireOS, e;
- **Web-Framework**: integra em uma única interface web [5] os módulos HellfireOS, N-MIPS e Construtor de Arquitetura, apresentando um fluxo de projeto introduzido em [8].

Os módulos do HellfireOS, do ambiente de simulação, do construtor da arquitetura e do Web-Framework serão apresentado com mais detalhes nas próximas seções.

3.1 HellfireOS

O HellfireOS (HFOS) [12] é um RTOS desenvolvido no Grupo de Sistemas Embarcados (GSE) que, além de controlar todos os recursos de *hardware* disponíveis [10], provê mecanismos para tratar restrições temporais.

O HFOS foi projetado para permitir a configuração de certos parâmetros do OS, possibilitando assim a customização da plataforma alvo. Para permitir tal customização, o HFOS foi projetado de maneira modular e independente, aonde cada módulo corresponde a uma funcionalidade específica.

A Figura 3.1 apresenta a organização em camadas do HFOS, onde todas funções dependentes de *hardware* estão definidas na primeira camada, denominada Hardware Abstraction Layer (HAL). O uKernel localiza-se acima dessa camada, assim como os *drivers* de comunicação, migração, gerência de memória, ponto flutuante, exclusão mútua e a biblioteca de funções padrão do 'C'. No topo da pilha fica localizada a camada de aplicações do usuário.

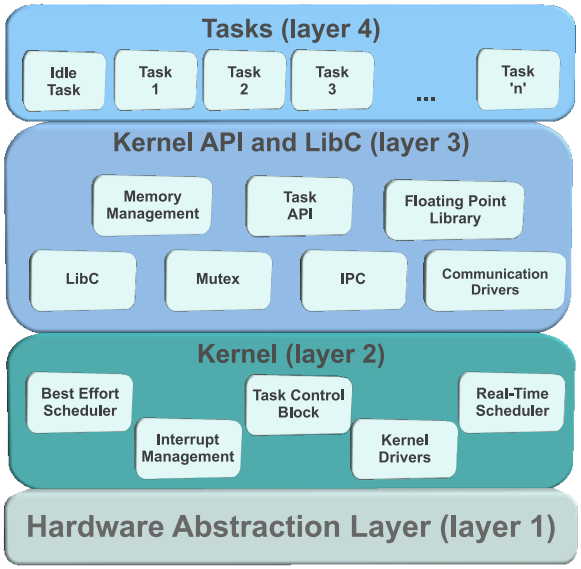

Figura 3.1: Estrutura do HellfireOS

O HellfireOS permite a configuração de diversos parâmetros com propósito de diminuir o tamanho da imagem final, possibilitando assim, seu uso em plataformas com limitações de memória. Em tempo de projeto podem ser definidos o número máximo de tarefas de usuário, o tamanho da pilha de tarefas e do *heap*, a política de escalonamento que será usada e se certos *drivers* serão ou não habilitados, como por exemplo, o *driver* de migração e o de ponto flutuante.

Além disso, é possível ainda definir o tamanho do *tick*, que corresponde a unidade temporal mínima do *kernel*. De uma maneira prática cada *tick* corresponde a um período de execução do sistema, podendo variar de 0,32 *ms* até 83.88 *ms*. É importante notar que quanto maior o valor do *tick*, maior será o tempo de execução de cada período e consequentemente menor será a responsividade do OS. Da mesma maneira, *ticks* muito pequenos reservam pouco tempo para execução dos aplicativos, passando boa parte do tempo apenas trocando o contexto das tarefas a cada novo período. Por esses motivos o valor padrão adotado para uso no HellfireOS é o de 10,48 ms.

Para facilitar seu uso, o HFOS disponibiliza uma API, sendo a mesma dividida em cinco categorias:

1. **manipulação de tarefas**: funções usadas para manipulação, configuração e controle de todas as tarefas do OS;

2. **exclusão mútua**: usadas para limitar regiões críticas utilizando *mutex* e semáforos;

3. **manipulação de memória**: funções de alocação e liberação dinâmica de memória no *heap* do processador;

4. **comunicação**: biblioteca contendo funções para troca de mensagens e migração de tarefas, e;

5. **funções básicas**: como *printf e sleep* de 'C', para uso nas aplicações.

Após compilar os fontes do kernel e da aplicação, uma única imagem é gerada. Esta imagem pode ser simulada em um ISS ou prototipada em *hardware*. Em sistemas multiprocessados cada processador recebe uma imagem do OS, podendo ser, ou não, a mesma imagem, para todos processadores, de acordo com a configuração realizada pelo projetista.

3.2 N-MIPS

N-MIPS [12] é uma ferramenta de simulação desenvolvida para facilitar o teste e depuração de sistemas embarcados, fornecendo diversos relatórios do funcionamento do sistema.

O N-MIPS é um Instrunction Set Simulator (ISS), que executa códigos objeto nativos, de acordo com a implementação de um processador Plasma. Além disso, o mesmo código usado para simulação pode ser usado no *hardware* real. Em conjunto com o simulador de instruções existe uma ferramenta que efetua a análise do consumo de energia [13] do sistema, assim como a contagem de ciclos. Uma emulação temporal e funcional da UART foi implementada de modo a fornecer a simulação da saída de dados dos processadores.

Posteriormente, um modelo de barramento compartilhado foi incorporado, seguido por um modelo em alto-nível de uma NoC [19], permitindo a simulação de sistemas multiprocessados.

Adicionalmente, diversos relatórios de desempenho do sistema são gerados, aumentado a gama de informações fornecidas ao desenvolvedor. Os seguintes relatórios são gerados automaticamente:

1. tipo da aplicação, podendo esta ser classificada como tarefa de tempo real (RT), ou então de melhor esforço (BE);

2. consumo de energia;

3. distribuição da carga dos processadores;

4. taxa de perda de deadlines;

5. taxas de migração e comunicação;

6. taxa de uso de cada conjunto de instruções *assembly*, e;

7. informação ciclo a ciclo do funcionamento de todo sistema.

De posse destas informações o projetista pode avaliar as características do sistema implementado e decidir por validar o modelo atual, ou então voltar à etapa de projeto e refinar o mesmo.

3.3 Construtor de Arquitetura

O terceiro módulo, chamado de Construtor de Arquitetura, é utilizado para especificar a arquitetura destino, assim como configurar todos os parâmetros do HellfireOS. Toda configuração do sistema pode ser feita via uma intuitiva interface *web*, projetada para facilitar o desenvolvimento do projeto. O desenvolvedor pode criar e testar um MPSoC completo apenas utilizando esta ferramenta.

Ao acessar a interface web do HFFW, o desenvolvedor deve criar um projeto e, então fazer a inclusão da aplicação que executará no sistema. Isto pode ser feito de três maneiras: **i)** fazendo *upload* de um arquivo já contendo a descrição da aplicação; **ii)** criando um arquivo novo na interface *web* e descrevendo a aplicação, ou; **iii)** carregando alguns dos exemplos disponibilizados.

O próximo passo do fluxo é a configuração da plataforma a ser usada. Para realizar esta etapa o desenvolvedor utilizará o Construtor de Arquitetura. Na Figura 3.2, é possível notar que o projetista pode escolher entre cinco tipo de processadores MIPS (identificados por uma letra A) e dois tipos de meio de comunicação, barramento ou NoC (identificados por uma letra B). Para configurar todo sistema, o projetista deve apenas arrastar e soltar os módulos desejados para a área de trabalho (identificada, na imagem, por uma letra C). Esta figura ilustra, ainda, um MPSoC formado por nove processadores interconectados por uma NoC 3x3. O projetista pode então configurar cada processador individualmente, ou então, todos de uma só vez.

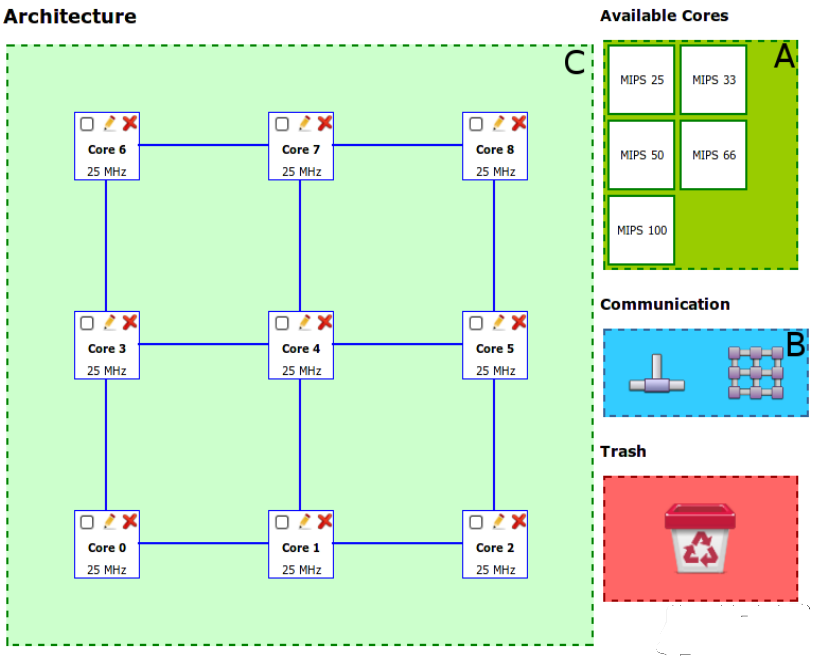

Figura 3.2: Arquitetura Montada

Todos parâmetros podem ser definidos apenas selecionando botões na interface gráfica, e todo sistema é montado automaticamente em *background*. É possível configurar os seguintes parâmetros do HellfireOS, conforme apresentado na Figura 3.3:

- ***User Max Tasks* (A)**: define o número máximo de tarefas que o desenvolvedor pode adicionar ao sistema;
- ***System Heap Size* (B)**: tamanho da memória dinâmica que será disponibilizado para uso;
- ***Tick Size* (C)**: define o tamanho do *tick* do sistema, em *ms*;
- ***NoC Buffer Size* (D)**: define o tamanho dos *buffers* dos roteadores da NoC que serão utilizados pelo simulador;
- ***NI Buffer Size* (E)**: determina o tamanho dos *buffers* intermediários, localizados entre o processador e o meio de comunicação, e;
- ***OS Settings* (F)**: habilita, ou não, o uso de *drivers* do HellfireOS.

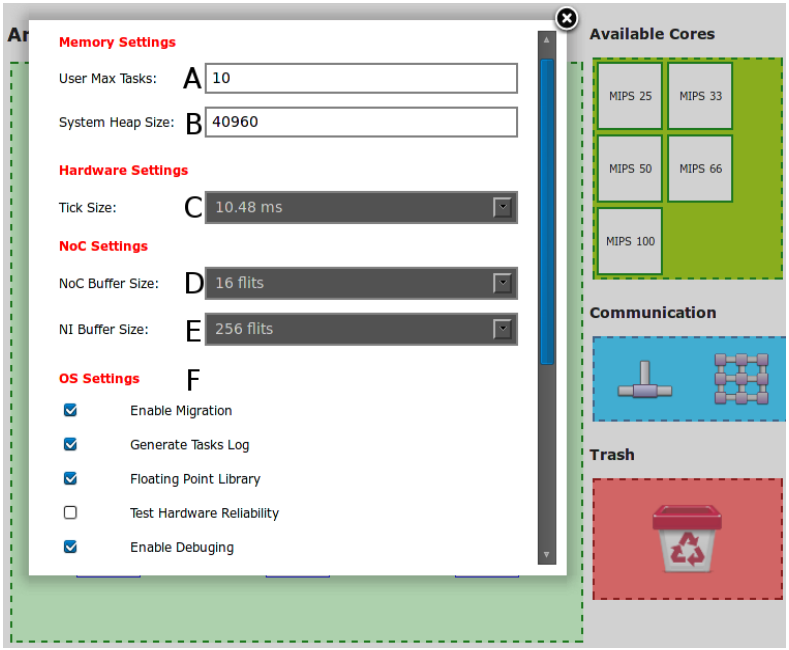

Figura 3.3: Janela de Configuração do HellfireOS

Após a configuração do sistema, o sistema é compilado e, não havendo nenhum erro de compilação pode-se passar para o próximo passo do projeto.

Por último, o sistema é simulado. A Figura 3.4 apresenta o resultado da simulação de um sistema no Hellfire Framework, onde pode-se notar a presença de diversas abas. Cada uma dessas abas corresponde a um relatório específico, gerado pela ferramenta N-MIPS, conforme segue:

- **Application Outputs**: apresenta a saída padrão dos processadores individualmente;

- **Reports**: ilustra as instruções *assembly* utilizadas pela aplicação, assim como a porcentagem de uso das mesmas e também o consumo de energia do processador;
- **Log**: apresenta um *log*, ciclo a ciclo, do funcionamento do HFOS;
- **Graphics**: disponibiliza, graficamente, alguns relatórios do sistema como, por exemplo, consumo de energia e perda de *deadlines*;
- **Briefing**: resumo geral do funcionamento do sistema, agregando em um único relatório algumas informações importantes, como quantidade e momento das comunicações e quais tarefas estão perdendo *deadlines* e quando, por exemplo, e;
- **Scheduler**: ferramenta para avaliação do escalonamento das tarefas onde é possível, graficamente, visualizar o escalonamento *tick a tick* no HFOS.

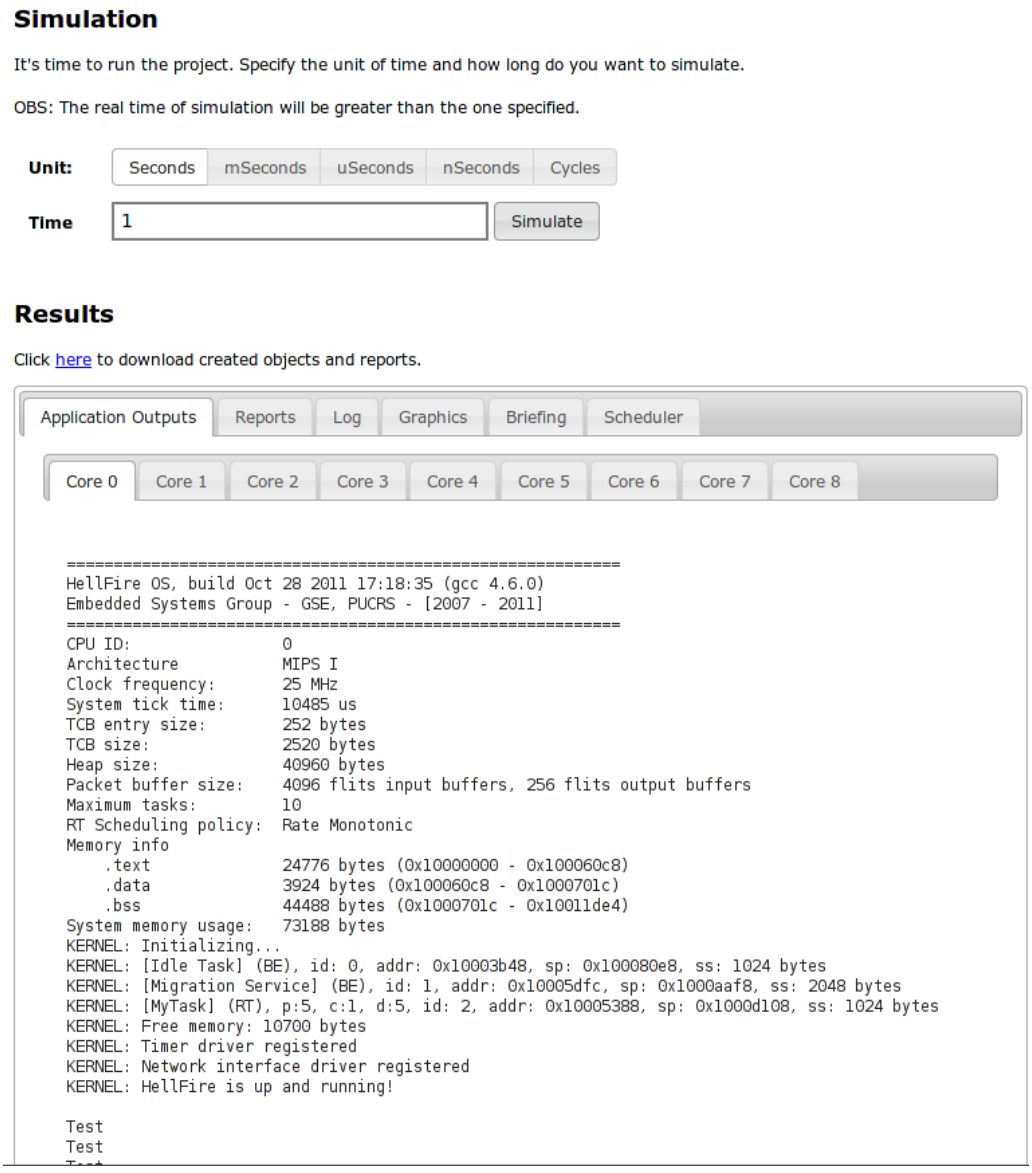

Figura 3.4: Saídas da Simulação no HFFW

Caso todos os resultados sejam satisfatórios o projetista tem a opção de fazer o download das imagens geradas para cada processador, assim como os relatórios das aplicações simuladas. Se o resultado não for o esperado é possível voltar a algum ponto específico do fluxo, refiná-lo e, então, realizar a simulação novamente.

4. TRABALHO DESENVOLVIDO

Neste capítulo são apresentados os detalhes do trabalho desenvolvido visando atingir os objetivos mencionados na Seção 1.4, que teve como base motivadora a necessidade por uma plataforma de alto desempenho e suporte de alto-nível para sistemas embarcados, conforme discutido ao longo do texto.

Trabalho realizado em duas etapas, conforme ordem apresentada:

1. descrição da arquitetura de *hardware*, composta por uma NoC [26], barramentos e *wrappers*[1], assim como unidades de processamento;

2. atualização do driver de comunicação do HFOS, de modo a permitir a troca de mensagens entre tarefas presentes em nodos[2] distintos.

Cada uma destas etapas será detalhada a seguir.

4.1 Arquitetura de *Hardware*

A divisão de tarefas entre diferentes processadores para aumento de desempenho do sistema já é uma pratica comum em sistemas embarcados, porém o meio de comunicação pode ser um gargalo do sistema, por não ter uma grande escalabilidade, limitando o número de processadores que podem ser utilizados (barramento), ou por apresentar situações onde o mapeamento de tarefas no MPSoC pode resultar em um tempo de troca de mensagens proibitivo (NoC), graças a uma má localização das mesmas no sistema. A plataforma desenvolvida busca propor uma solução que minimiza ambas limitações utilizando um sistema híbrido, composto por uma NoC e por diversos barramentos.

A Figura 4.1 apresenta uma visão geral da plataforma desenvolvida. É possível notar a presença de uma NoC centralizada, responsável pela troca de mensagens entre nodos e, em cada nodo, um sub-sistema composto por um barramento simples e *N* unidades internas conectadas a este barramento.

É importante ressaltar que cada um dos nodos da NoC pode ser composto por um *cluster*, ou então por módulos *IP* quaisquer, de acordo com as necessidades do desenvolvedor.

A implementação dos módulos de *hardware* foi realizada em duas etapas: **i**) implementação do barramento e de seus módulos, e; **ii**) implementação dos módulos de comunicação entre o barramento e a NoC.

[1] *wrapper* se refere ao mecanismo de empacotamento, encaminhamento e desempacotamento de pacotes entre nodos que utilizem protocolos de comunicação diferentes.

[2] lembrando que, no contexto de *clusters*, nodo corresponde a um conjunto de unidades agrupadas

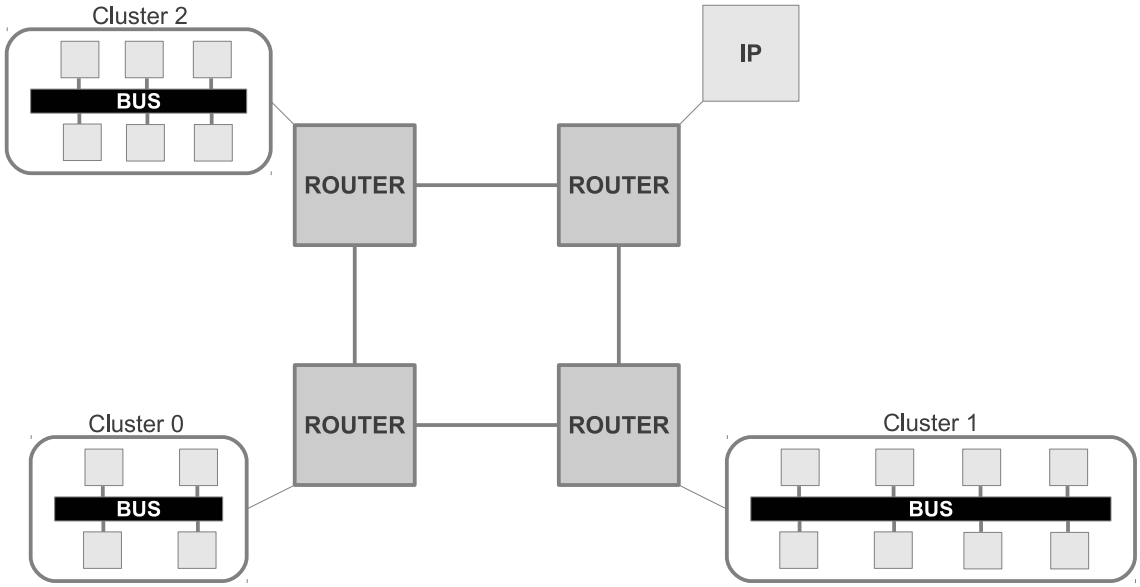

Figura 4.1: Visão Geral da Plataforma

4.1.1 Barramento

Um modelo de barramento foi implementado para servir como meio de comunicação entre os elementos que compõem um *cluster*. Este barramento é composto por um meio de comunicação compartilhado, controlado por um árbitro que decide qual porta terá acesso ao meio e utiliza um algoritmo rotativo. A Figura 4.2 apresenta a estrutura básica do barramento utilizado. Este exemplo é composto por quatro unidades *IP* conectadas ao barramento via *wrappers*, identificadas na figura como **W**. Estes *wrappers* são responsáveis pela conversão dos protocolos de comunicação entre o módulo *IP* e o barramento, e também responsáveis por solicitar o acesso ao meio de comunicação ao árbitro. Assim, o árbitro pode gerenciar os acessos ao barramento, e as unidades de entrada e saída, identificadas na figura por **I/O**. Cada um dos módulos apresentados serão discutidos no decorrer do texto. A validação do barramento foi realizada através simulações, utilizando a ferramenta ModelSim [6] e em prototipações em FPGA.

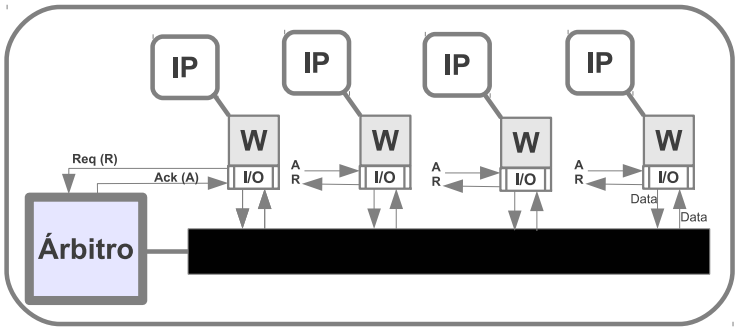

Figura 4.2: Visão Geral do Barramento

Árbitro

O controle central do barramento é realizado pelo árbitro, que recebe solicitações de acesso ao barramento das portas de saída dos *wrappers* e escolhe qual terá acesso. A adoção de um modelo de controle centralizado, como o utilizado neste caso, tem como justificativa a simplificação do processo de controle, buscando diminuir o *overhead* na escolha de um nodo para acesso ao barramento. Desta maneira para qualquer transação realizada por um *IP* no barramento, este *IP* deve fazer uma solicitação ao árbitro e, somente após uma confirmação realizar a ação desejada. A Figura 4.3 apresenta a estrutura do topo do árbitro, aonde é possível observar três sinais de entrada e quatro sinais de saída. Cada um destes sinais será comentado no decorrer do texto, assim como o seu uso no funcionamento interno do árbitro.

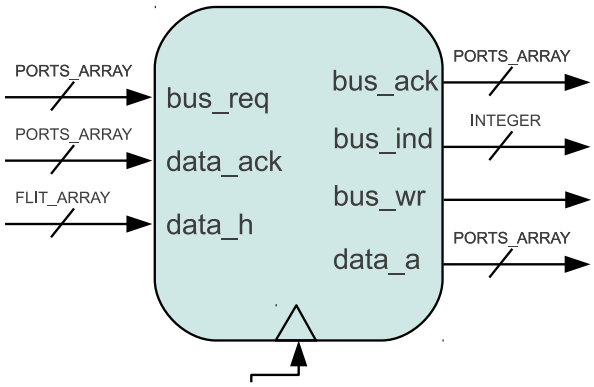

Figura 4.3: Visão Geral do Árbitro do Barramento

Os três sinais de entrada do árbitro são: *bus_req*, *data_ack* e *data_h*, destacados e descritos a seguir.

- ***bus_req***: composto por *y* sinais, sendo *y* igual ao numero de núcleos conectados ao barramento, ou seja, ao número de portas de entrada e saída interligadas ao barramento. Por meio deste sinal o árbitro recebe a solicitação de cada porta para acesso ao barramento;

- ***data_h***: composto por *w* vetores de dados de tamanho *k*, sendo *w* igual ao numero de núcleos conectados ao barramento e *k* o tamanho do *flit*[3] utilizado. Este vetor de entrada é interligado diretamente à saída de todos os nodos conectados ao barramento, e é utilizado pelo árbitro, após uma requisição ter sido recebida, para avaliar se o destinatário da mensagem está disponível, e;

- ***data_ack***: composto por *n* sinais, sendo *n* igual ao número de núcleos conectados ao barramento. Este sinal é recebido pelo árbitro tendo como origem o destinatário, e é utilizado para notificar o árbitro de que a mensagem foi recebida em seu destino.

Como saída o árbitro possuí quatro sinais: *bus_ack*, *bus_ind*, *bus_wr* e *data_a*. Cada um deles terá seus detalhes comentados a seguir.

[3] *flit* corresponde a uma palavra transmitida no meio de comunicação

- ***bus_ack***: vetor de *y* sinais, sendo *y* o número de nodos que compõem o barramento. Utilizado para notificar o solicitante de que o dado foi recebido pelo destinatário;
- ***bus_ind***: um valor integral que contém o índice do solicitante escolhido pelo árbitro para ter acesso ao barramento;
- ***bus_wr***: este sinal binário é utilizado para notificar o módulo de controle de escrita do barramento de que o dado em sua porta de entrada, com índice armazenado em *bus_ind* deve ser escrito no barramento, e;
- **data_a**: composto por *n* sinais, sendo *n* igual ao numero de núcleos conectados ao barramento. Sinal usado pelo árbitro para notificar o destinatário de que uma mensagem endereçada a ele está disponível.

O árbitro é composto por duas máquinas de controle. A primeira é utilizada para o controle da porta que irá ter sua solicitação atendida, e utiliza um algoritmo rotativo, de maneira a tentar evitar *starvation*[4], garantindo justiça entre todas as requisições. Quando uma requisição é recebida e o árbitro decidir por atende-la, o solicitante terá acesso ao barramento até enviar todo seu pacote.

A segunda máquina de controle é utilizada para garantir a sincronia dos dados entre o solicitante e o destinatário. Para realizar isto o árbitro utiliza um protocolo de comunicação baseado no protocolo *credit-based*, aonde, após receber uma sinalização de envio liberado, ou então um "crédito"para envio, o solicitante pode transmitir toda sua mensagem. A Figura 4.4 apresenta um exemplo de uma porta solicitando acesso ao barramento para o árbitro, o envio de seu dado, o recebimento no destinatário e a sinalização ao solicitante de dado recebido. É importante ressaltar que neste exemplo o módulo *wrapper* foi omitido para simplificar o entendimento.

Neste exemplo, supõe-se que o nodo *0000* deseja enviar um *flit* para o nodo *1111*. Os passos para o envio são descritor a seguir:

1. *A na figura*: o nodo solicitante escreve o nodo destino na sua porta de saída e faz uma requisição para o árbitro;
2. *B na figura*: árbitro envia solicitação para o nodo destino a partir do dado de saída do solicitante, e;
3. *C na figura*: recebendo uma confirmação de liberação do destino o solicitante pode enviar toda sua mensagem.

Wrapper

O segundo módulo desenvolvido foi o *wrapper*, sendo este composto por um bloco responsável por interligar o módulo *IP* à via de comunicação e por uma unidade de entrada e saída, para controlar o acesso aos dados do barramento.

[4] *starvation*, ou inanição, é quando um processo nunca é selecionado para efetuar sua tarefa, por exemplo

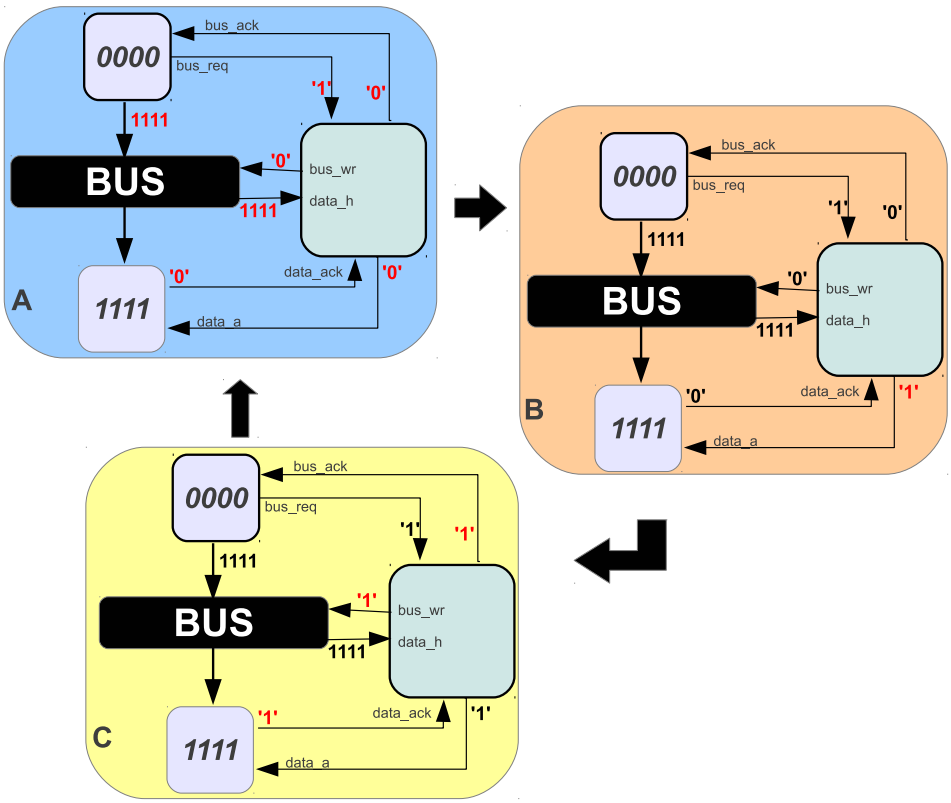

Figura 4.4: Exemplo do Funcionamento do Árbitro

Este módulo, também chamado de *data port* (porta de dados), possui duas máquinas de estado internas, uma para realizar o controle de dados vindos do *IP* com destino ao barramento, e uma para realizar o controle inverso, ou seja, dados vindos do barramento com destino ao *IP*. A Figura 4.5 apresenta a visão do topo do módulo *wrapper*.

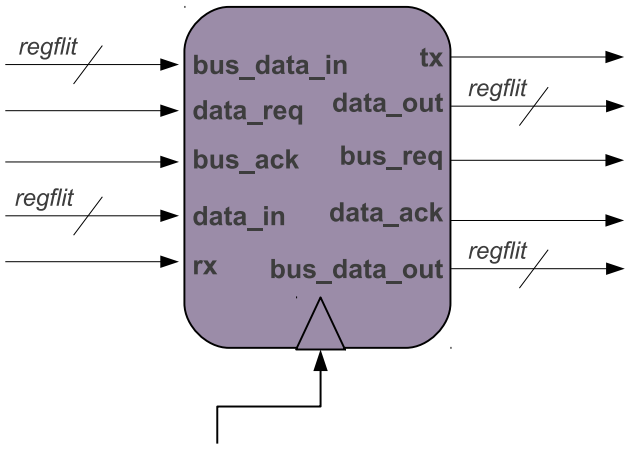

Figura 4.5: Visão Geral do Wrapper do Barramento

Cinco sinais de entrada e cinco sinais de saída compõem a interface externa deste módulo e cada um destes sinais será comentado no decorrer do texto.

Os sinais de entrada do *wrapper* são: *bus_data_in*, *data_req*, *bus_ack*, *data_in*, *ack_tx* e *rx*. Detalhes do uso de cada sinal estão descritos a seguir.

- ***bus_data_in***: vetor de tamanho *Y*, sendo *Y* o tamanho do *flit* utilizado. Recebe o valor de saída do barramento e o escreve na porta de entrada do *IP*;
- ***data_req***: sinal de um *bit* utilizado para sinalizar a presença de um dado no barramento com destino ao nodo deste *wrapper*;
- ***bus_ack***: sinal de um *bit* que serve como resposta positiva a uma solicitação feita ao árbitro para envio no barramento;
- ***data_in***: vetor de *N bits*, sendo *N* o tamanho do *flit* utilizado. Porta de entrada de dados vindo do *IP* com destino ao barramento, e;
- ***rx***: sinal de um *bit* usado para sinalização de novo dado vindo do módulo *IP* com destino ao barramento.

Como saída, o *wrapper* possui os sinais: *tx*, *ack_rx*, *data_out*, *data_ack*, *bus_req* e *bus_data_out*. A seguir um maior detalhamento a respeito de cada sinal.

- ***tx***: sinal de um *bit* usado para sinalizar o *IP* da disponibilidade de um novo dado no barramento;
- ***data_out***: vetor de *X bits*, sendo *X* o tamanho do *flit*, recebe o valor de saída do barramento e o copia para entrada do *IP*;
- ***data_ack***: sinal de um *bit* que contém uma resposta positiva de um dado escrito no barramento;
- ***bus_req***: *bit* utilizado para solicitar acesso do barramento ao árbitro, e;
- ***bus_data_out***: vetor de *K bits*, sendo *K* o tamanho do *flit*, que armazena o pacote vindo do *IP* e o escreve na entrada do barramento.

Importante ressaltar que este módulo funciona apenas como um conversor de protocolos, realizando uma ponte de troca de dados entre um *IP* e o barramento, servindo como interface de acesso ao árbitro.

4.1.2 Módulo de Comunicação Barramento-NoC

O último módulo de *hardware* desenvolvido, foi o *wrapper* responsável por interligar um *cluster*, via a *data port* do barramento, à uma porta da NoC. O módulo criado foi nomeado *Cluster Interface* (CI), sendo composto por duas filas para armazenamento temporário dos dados que trafegam entre um roteador da NoC e o barramento interno do *cluster*.

Este módulo executa em paralelo com as unidades *IP* que estão operando no sistema, ou seja, ele funciona como um *IP* qualquer conectado ao barramento interno do *cluster*. Sendo

assim, para enviar uma mensagem para outro *cluster*, a mensagem deve sair de seu *IP*, passando pelo CI, que está interconectado à NoC via porta local do roteador, que a envia para o *cluster* destino. No destino da mensagem, o roteador recebe a mensagem na porta local, que repassa para o CI do *cluster*. Por fim, o módulo encaminha a mensagem para a unidade *IP* destino. Na versão implementada neste trabalho, o protocolo de comunicação adotado entre os nodos da NoC é o *handshake* e, para uso de outro protocolo, o módulo CI deve ser adaptado.

A Figura 4.6 apresenta uma visão do módulo CI, onde pode-se observar os sinais utilizados para troca de dados com um roteador da NoC com o barramento.

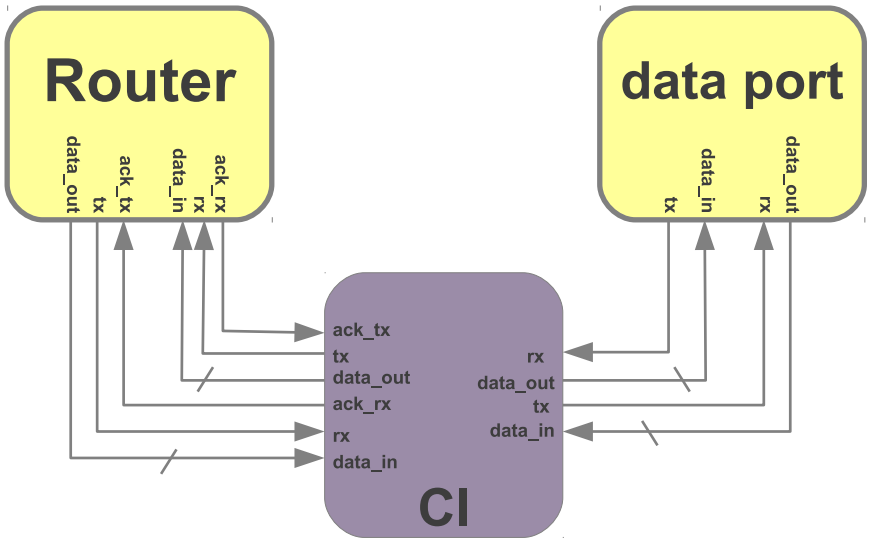

Figura 4.6: Visão Geral do Wrapper NoC-Barramento

É importante ressaltar que no modelo adotado, o módulo de comunicação barramento-NoC concorre com os *IPs* internos do *cluster* pelo barramento, podendo contribuir para um aumento no tráfego de informações.

O fluxo de troca de informações entre o *cluster* e um roteador é realizado em dois momentos distintos: o envio de uma mensagem de dados completa (de tamanho definido em tempo de projeto) da via de comunicação origem para o CI e, somente após o recebimento de toda mensagem, o repasse desta mensagem para a via de comunicação destino. A Figura 4.7 apresenta um exemplo de envio de um mensagem de um nodo em um *cluster* para um nodo em outro *cluster*. Na figura é possível observar a presença de uma NoC externa, contendo em cada porta local de seus roteadores um cluster formado por dois núcleos *IP*, um barramento e o módulo CI. Em um primeiro momento (bloco A da figura) o nodo origem envia sua mensagem para o módulo CI via barramento. Após toda mensagem ser armazenada no *buffer* intermediário do CI, uma solicitação de envio é realizada para a porta local da NoC. Assim que recebe uma liberação da NoC, o módulo CI envia a mensagem para o *cluster* destino, via NoC (bloco B da figura). Por último (bloco C da figura), esta mensagem é recebida pelo módulo CI do *cluster* destino via porta local. Assim que recebe toda, o módulo encaminha esta mensagem, via barramento interno para o nodo destino.

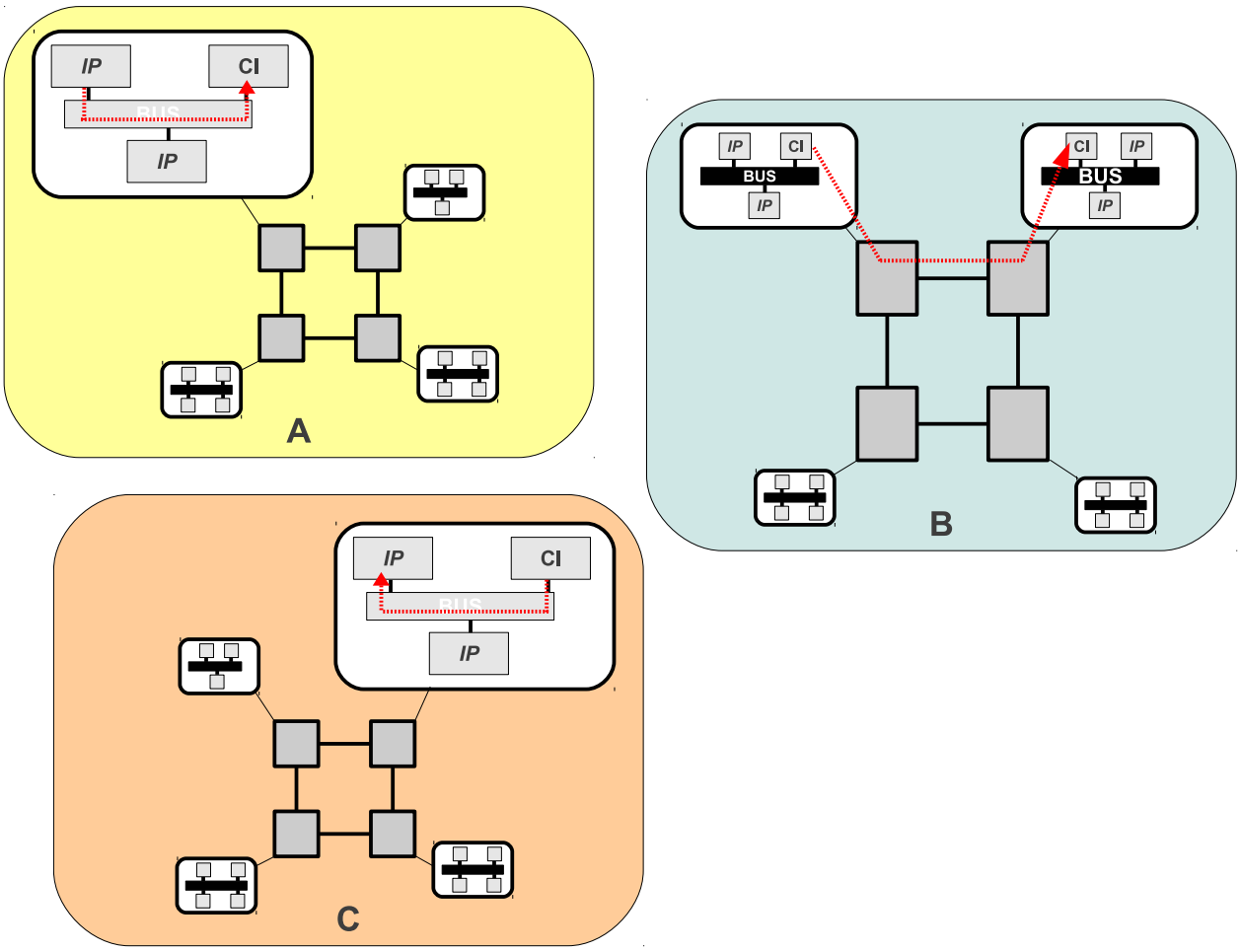

Figura 4.7: Troca de Mensagem entre *Clusters*

4.2 Atualização do Driver do HellfireOS

Finalizada a etapa de desenvolvimento e validação de todos módulos de *hardware*, o suporte de alto-nível via sistema operacional HellfireOS foi implementado.

Um *driver* para comunicação já existia em versões prévias do HFOS, porém apenas com suporte à comunicação direta entre dois nodos. Este *driver* serviu como base para o desenvolvimento da versão destinada à comunicação entre *clusters*. Um detalhe importante é que a comunicação entre componentes presentes em um mesmo *cluster* usará a nova versão do *driver* desenvolvido, ficando o *driver* antigo destinado somente a MPSoCs simples[5].

O *driver* de comunicação já contava com duas funções, **int** *HF_Send(***unsigned short int** *target_cpu,* **unsigned char** *target_id,* **unsigned char** *buf[],* **unsigned short** *size)* utilizada para envio de mensagens e, **int** *HF_Receive(***unsigned short int** **source_cpu*, **unsigned char** **source_id*, **unsigned char** *buf[]*, **unsigned short** **size)* utilizada para recebimento de mensagens, que funcionam da seguinte maneira:

[5] *MPSoCs simples* se refere aos MPSoCs formados por um único meio de comunicação, com apenas um nível de profundidade, ou seja, sem diferentes níveis de comunicação, como em barramentos hierárquicos, por exemplo.

Driver **de envio de mensagens**:

1. o *driver* começa a montar os pacotes que irão ser enviados para a interface de rede, e os armazena em um *buffer* intermediário em *software*, na seguinte ordem: **i**) endereço do roteador destino; **ii**) o tamanho do pacote, para controle do roteador; **iii**) o *id* da CPU que está enviando a mensagem; **iv**) *id* da tarefa destino e também da tarefa origem concatenados; **vi**) o tamanho total da mensagem enviada; **vii**) quantidade de pacotes que compõem a mensagem, e; **viii**) o conteúdo da mensagem;
2. o *driver* verifica se pode escrever no endereço de *hardware* reservado para saída de dados, e;
3. podendo enviar a mensagem, o *driver* apenas copia o conteúdo do *buffer* em *software* para o *buffer* em *hardware*.

Driver **de recebimento de mensagens**:

1. quando uma nova mensagem chega ao processador, uma interrupção é lançada, sendo esta tratada pelo *driver* de recepção de mensagens;
2. o *driver* copia as informações contidas no *buffer* em *hardware* para um *buffer* intermediário em *software*, liberando o *hardware* para um novo recebimento e, também, avaliar a mensagem e encaminhá-la para tarefa destino;
3. uma verificação da integridade dos dados é realizada e, não havendo nenhum problema na mensagem e havendo espaço no *buffer* de recebimento da tarefa, a mensagem é copiada para este *buffer*. Não havendo espaço no *buffer* de recebimento da tarefa, o pacote é descartado, e;
4. a tarefa pode então ler de seu *buffer* a mensagem recebida.

A nova versão do *driver* de comunicação ampliará as informações contidas no corpo das mensagens, apenas adicionando mais dois campos, correspondentes aos endereços de roteador destino, para trânsito entre *clusters*, e *IP* destino, para encaminhamento interno no *cluster* destino.

A estrutura do novo pacote pode ser observada na Figura 4.8 e seus dados serão comentados a seguir.

- ***Módulo_OUT_ID***: contém o endereço do módulo CI no barramento interno do *cluster*, sendo este endereço fixo independentemente do número de *IPs*;
- ***Payload***: tamanho total da mensagem enviada, considerando o cabeçalho;
- ***Cluster_Destino_ID***: endereço do nodo de destino na NoC e utiliza os endereços da NoC empregada, no caso deste trabalho, a NoC Hermes;
- ***Nodo_Destino_ID***: armazena o endereço do nodo destino internamente ao *cluster* que recebeu a mensagem;

- ***CPU_Origem_ID***: contém o endereço do nodo que originou a mensagem;
- ***Task_Origem_ID e Task_Destino_ID***: armazena dois dados concatenados, cada um de oito *bits*, a identificação da tarefa originária da mensagem e a identificação da tarefa destino da mensagem;
- ***Tamanho da Mensagem***: contém o tamanho total da mensagem, utilizado pelo *driver* de recebimento para casos em que a mensagem seja segmentada;
- ***Sequênciamento de Pacote***: campo também utilizado em situações em que a mensagem é segmentada para poder reestrutura-la, e;
- ***Mensagem***: o conteúdo da mensagem.

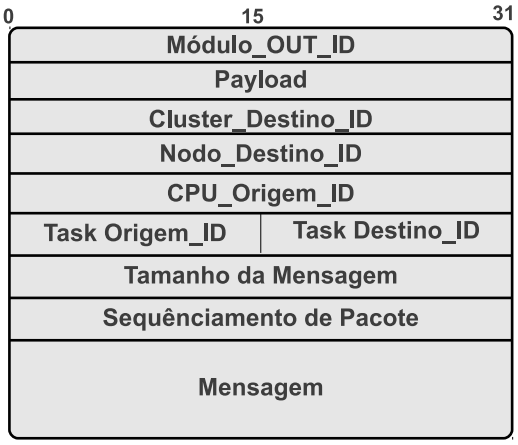

Figura 4.8: Pilha de Informações do Protocolo de Comunicação entre *Clusters*

Para comunicação entre *clusters* as funções de comunicação do HFOS serão utilizadas, porém com o acréscimo de um parâmetro cada, sendo o protótipo das funções apresentado a seguir:

- **int** *HF_Send(***unsigned short int** *target_cpu,* **unsigned short int** *target_node,* **unsigned char** *target_id,* **unsigned char** *buf[],* **unsigned short** *size)* utilizada para envio de mensagens, e;
- **int** *HF_Receive(***unsigned short int** **source_cpu*, **unsigned short int** **source_node*, **unsigned char** **source_id*, **unsigned char** *buf[],* **unsigned short** **size)* utlizada para recebimento de mensagens.

Para enviar mensagem entre os *clusters* o projetista deve ter conhecimento do mapeamento das tarefas realizado, sabendo, deste modo, o *cluster*, o nodo e a tarefa de destino da mensagem. Em casos de comunicação interna em um *cluster* o projetista deve apenas passar *-1* como nodo destino e o mecânismo de controle interno do *driver* realiza a conversão do endereço de envio correto.

Ambas funções possuem duas versões, uma bloqueante e outra não bloqueante. A versão não bloqueante destas funções recebe um parâmetro extra, um valor de *timeout*, que é utilizado

como tempo limite para realizar a operação desejada. Nestes casos um contador interno é utilizado no *driver*, servindo como controle de tempo máximo de espera. Assim que o contador atinge o valor de *timeout*, o *driver* retorna uma mensagem informando falha na operação. Nos modelos bloqueantes o *timeout* é considerado *zero*, ou seja, o *driver* aguarda até finalizar a operação.

4.3 Cluster

Após implementação e validação de todos os módulos de *hardware* que compõem o MPSoC proposto neste trabalho, assim como a atualização e disponibilização de um *driver* para o HellfireOS, uma versão final da arquitetura foi gerada, para fins de simulação, prototipação e debug. Esta arquitetura básica inicial serviu como *template* para geração de casos de teste de validação da arquitetura física, assim como validação do *driver* descrito e é composta dos seguintes módulos: **i)** uma NoC regular de tamanho 2x2, com tamanho de *flit* de 16 *bits* e *buffers* intermediários de 16 posições, utilizando o protocolo *handshake* para controle de fluxo; **ii)** processadores Plasma como núcleos *IP* que compõem cada *cluster*; **iii)** barramentos internos em cada *cluster*, que trabalham com o mesmo tamanho de *flit* que a NoC; **iv)** *wrappers* para comunicar os módulos *IP* com os barramentos, adaptados de [11] e nomeados *Bridge Interface* (BI); **v)** *wrappers* para interligar a NoC aos *clusters*, e; **vi)** imagens do HellfireOS que executam sobre os *IPs*.

As BIs utilizadas neste trabalho possuem *buffers* de armazenamento temporários, assim como no CI. Isto se deve ao fato de que, dessa maneira, o tempo gasto entre a interação *IP*-barramento pode ser reduzido, permitindo um melhor desempenho no processo de saída de dados do *IP*. Um cenário em que é possível verificar a melhor eficiência deste modelo é o seguinte: considerando um processo *p* comunicante qualquer, toda vez que *p* precisar trocar mensagens com outro *IP*, a mensagem será armazenada temporariamente no *buffer* intermediário, sendo enviada quando o barramento estiver livre, liberando a tarefa para seguir sua execução. Não havendo este *buffer*, a tarefa ficaria presa até conseguir enviar a mensagem, o que poderia atrasar sua execução e, por consequência, comprometer o funcionamento do sistema.

A Figura 4.9 apresenta a arquitetura final usada para validação dos módulos. É possível notar que em cada *cluster* existe um número diferente de *IPs*, e que os módulos CI e BI foram ocultados.

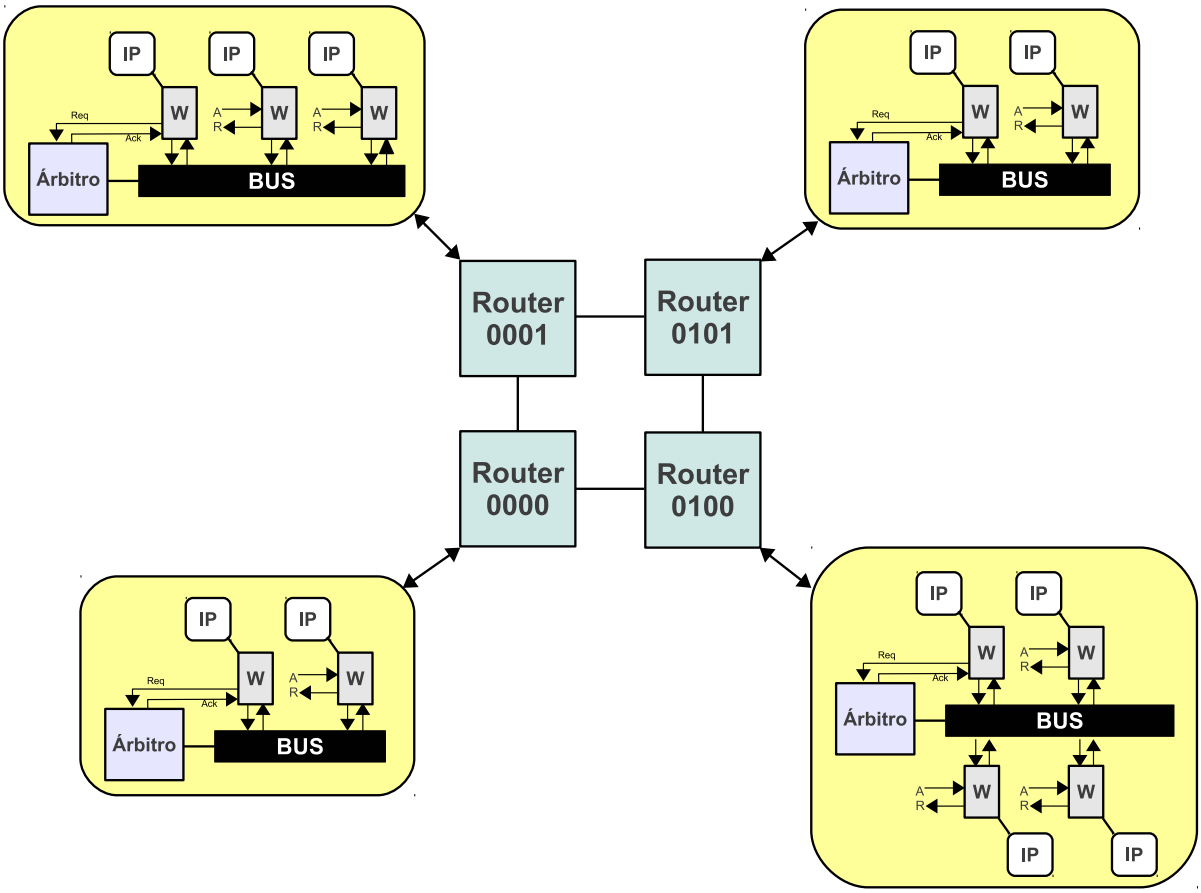

Figura 4.9: Cluster Final Usado para Validação

Ambos, *driver* e arquitetura, foram validados em simulações utilizando Modelsim e por prototipação, utilizando uma FPGA Virtex II Pró [30], e os resultados serão apresentados na Seção 5.

5. RESULTADOS

Após implementação e validação de todos os componentes introduzidos neste trabalho: barramento, módulo de comunicação *cluster*-NoC e *driver* de comunicação para o HellfireOS, três classes de resultados foram gerados. Estas classes foram:

1. comparação de utilização em tecnologias FPGA, onde a área de ocupação e a frequência de operação do MPSoC em dois dispositivos FPGA foram obtidos. Para realizar a geração dos resultados a ferramenta de síntese ISE [29] foi utilizada, tendo como plataformas FPGA alvo uma Virtex V [31] e uma Virtex II [30];

2. resultados para o processo de síntese para tecnologia de 65nm da STMicro em que a área de ocupação e a potência estimada foram comparados, e;

3. comparação de cenários de tráfego utilizando diferentes tamanhos de MPSoC.

Para obtenção de resultados, MPSoCs de tamanho variado, possuindo de quatro pontos[1] até 81 pontos que utilizam tamanhos de *cluster* interno variados, porém com número de pontos internos ao *cluster* nunca maior do que oito, foram gerados.

Para fins de comparação, MPSoCs utilizando apenas uma NoC como meio de comunicação também foram gerados, contendo o mesmo número de pontos dos casos de teste para *clusters*. Os detalhes dos resultados obtidos e a comparação dos resultados para uma solução utilizando um MPSoC que empregue *clusters* e sua versão que utiliza apenas um NoC são apresentados a seguir.

A Figura 5.1 apresenta um exemplo de como foi feita a divisão de nodos de um MPSoC formado unicamente por uma NoC para um MPSoC *clusterizado*. Neste exemplo, um MPSoC contendo 16 pontos de comunicação é apresentado em uma configuração utilizando somente uma NoC e, então este mesmo MPSoC é ilustrado utilizando *clusters*.

5.1 Resultados de síntese para tecnologias FPGA

5.1.1 Resultados para Virtex V

Conforme citado anteriormente, diversos cenários foram utilizados para geração de resultados. No caso dos cenários criados para a Virtex V, dez MPSoCs foram configurados de maneira distinta, utilizando o modelo de *clusters* apresentados no decorrer do texto. Estes MPSoCs são formados por barramentos, *wrappers* interligando cada barramento ao seu respectivo roteador e uma

[1] a seguinte nomeclatura será adotada: ***nodo*** se refere à um *cluster* no MPSoC e ***ponto*** à uma conexão no barramento interno de cada *cluster*, podendo representar um *IP* qualquer

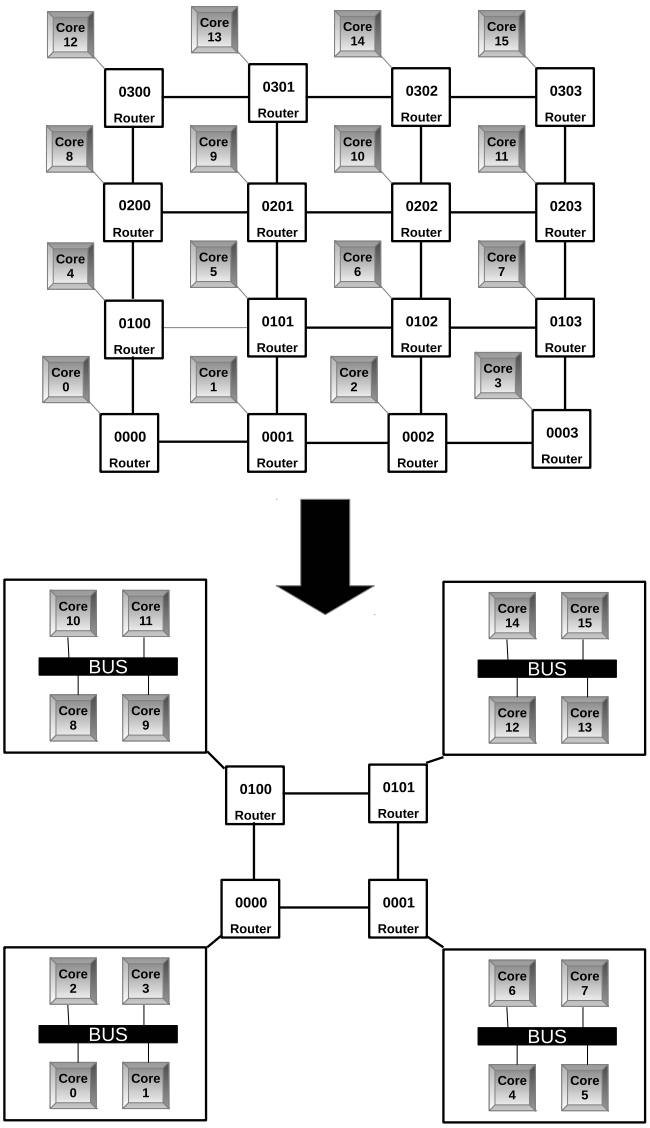

Figura 5.1: Exemplo de mapeamento

NoC central interligando todos os nodos. A lista a seguir apresenta, para cada caso a configuração do número de pontos por *cluster*, assim como o tamanho de NoC central utilizada:

- ***9 pontos***: NoC central de tamanho 2x2, três nodos contendo dois pontos e um nodo contendo três pontos;
- ***12 pontos*** NoC central de tamanho 2x2, quatro nodos de três pontos cada;
- ***16 pontos*** NoC central de tamanho 2x2, quatro nodos com quatro pontos cada;
- ***25 pontos*** NoC central de tamanho 2x2, três nodos com seis pontos e um nodo com sete pontos;
- ***36 pontos*** NoC central de tamanho 3x2, seis nodos contendo seis pontos cada;
- ***42 pontos*** NoC central de tamanho 3x2, seis nodos com sete pontos cada;

- ***49 pontos*** NoC central de tamanho 3x3, cinco nodos contendo cinco pontos cada e quatro nodos com seis pontos cada ;
- ***64 pontos*** NoC central de tamanho 3x3, oito nodos com sete pontos cada e um nodo com oito pontos;
- ***72 pontos*** NoC central de tamanho 3x3, nove nodos com oito pontos cada, e;
- ***81 pontos*** NoC central de tamanho 4x3, dez nodos com sete pontos cada, um nodo com seis pontos e um nodo com cinco pontos.

Além de todos estes MPSoCs gerados, outros dez MPSoCs foram criados, utilizando apenas um NoC central, para fins de comparação. As seguintes dimensões de NoC foram utilizadas: *3x3*, *4x3*, *4x4*, *5x5*, *6x6*, *7x6*, *7x7*, *8x8*, *9x8* e *9x9*.

A Tabela 5.1 apresenta uma comparação dos resultados obtidos para as duas soluções, ilustrando o ganho de área ao se utilizar um MPSoC formado por *clusters*. É possível notar que em todos os casos, houve uma diminuição significativa no uso de área do dispositivo.

Tabela 5.1: Comparação da área ocupada na Virtex V

Pontos	*cluster MPSoC*	*NoC MPSoC*	Ganho
9	2%	3%	50%
12	2%	4%	50%
16	2%	6%	67%
25	2%	10%	80%
36	4%	15%	73%
42	4%	18%	78%
49	7%	22%	68%
64	7%	29%	79%
72	7%	32%	78%
81	9%	36%	75%

A Figura 5.2 apresenta um gráfico apontando o crescimento de área de ambas soluções, *cluster* e MPSoC simples, onde é possível observar o comportamento de crescimento da solução utilizando apenas um NoC como meio de comunicação. Utilizando-se *clusters*, o aumento de área não ocorre nessa mesma taxa de crescimento, com um aumento menos significativo, quando comparado à solução anterior.

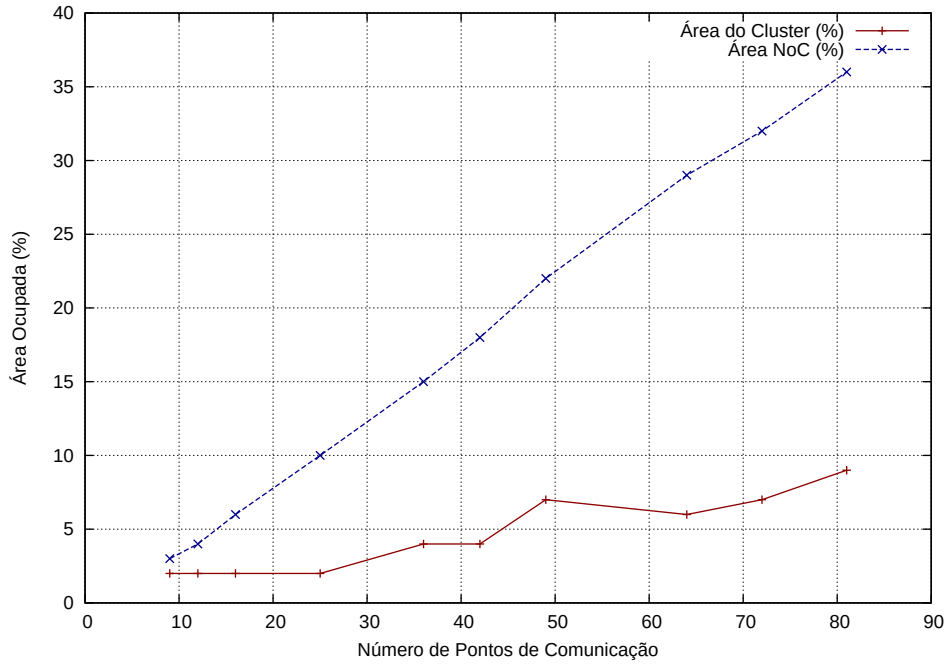

Figura 5.2: Gráfico do aumento da área na Virtex V

A Tabela 5.2 apresenta os dados referentes às velocidades de execução obtidas a partir da síntese para o dispositivo. Na tabela pode-se observar as frequências de operação para ambas soluções, MPSoC simples e *cluster*, assim como o percentual de perda de velocidade da solução utilizando *clusters*. Esta perda já era esperada, e ocorre devido a um aumento na complexidade do *hardware* projetado.

Tabela 5.2: Comparação de frequência na Virtex V

Pontos	*cluster MPSoC*	*NoC MPSoC*	Ganho
9	152.318 MHz	167.805 MHz	-9%
12	152.318 MHz	167.805 MHz	-9%
16	156.014 MHz	167.805 MHz	-7%
25	155.313 MHz	160.844 MHz	-3%
36	155.560 MHz	167.805 MHz	-7%
42	155.313 MHz	167.805 MHz	-7%
49	155.313 MHz	167.805 MHz	-7%
64	155.109 MHz	167.805 MHz	-8%
72	158.546 MHz	164.439 MHz	-4%
81	155.109 MHz	164.439 MHz	-6%

A Figura 5.3 apresenta um gráfico das frequências obtidas. É possível observar certa linearidade em tais frequências em ambos os casos, com uma frequência levemente inferior para solução do tipo *cluster*, o que, conforme comentado anteriormente, era esperado.

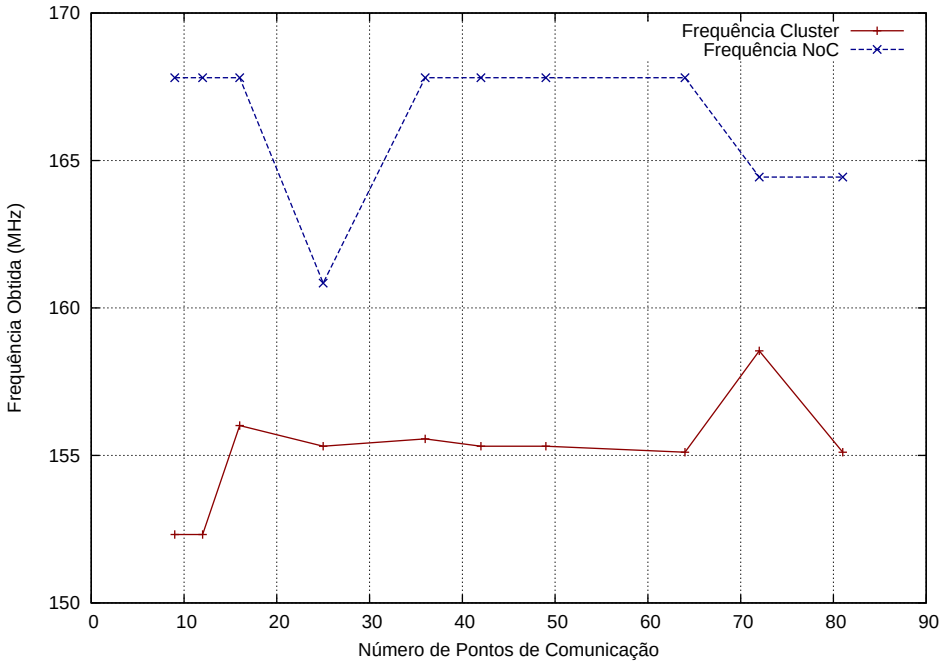

Figura 5.3: Gráfico de frequência para Virtex V

5.1.2 Resultados para Virtex II

Assim como utilizado na obtenção de resultados para a Virtex V, a obtenção de resultados para Virtex II contou com diferentes cenários distintos, porém com um menor número de casos de teste, se comparado aos casos usados na Seção 5.1.1. Isso se deve ao fato de que a Virtex II é um dispositivo com menores recursos, limitando seu uso. Desta maneira, apenas quatro MPSoCs foram configurados, conforme a lista a seguir apresenta:

- ***9 pontos***: NoC central de tamanho 2x2, três nodos contendo dois pontos e um nodo contendo três pontos;
- ***12 pontos*** NoC central de tamanho 2x2, quatro nodos de três pontos cada;
- ***16 pontos*** NoC central de tamanho 2x2, quatro nodos com quatro pontos cada;
- ***25 pontos*** NoC central de tamanho 2x2, três nodos com seis pontos e um nodo com sete pontos;

Para fins de comparação, outros quatro MPSoCs foram criados, utilizando apenas um NoC central. As seguintes dimensões de NoC foram utilizadas: *3x3*, *4x3*, *4x4* e *5x5*.

A Tabela 5.3 apresenta uma comparação dos resultados obtidos para as duas soluções, ilustrando a diferença de ocupação entre cada uma. Um detalhe importante a se ressaltar neste caso é o fato de que a FPGA utilizada contém recursos muito mais limitados em relação à Virtex V. Graças a isso o crescimento apresentado por ambas soluções é muito mais rápido, porém com o mesmo comportamento anterior.

Tabela 5.3: Comparação da área ocupada na Virtex II

Pontos	*cluster MPSoC*	*NoC MPSoC*	Ganho
9	23%	33%	30%
12	23%	47%	51%
16	24%	65%	63%
25	24%	103%	77%

A Figura 5.4 apresenta o gráfico de crescimento de ambas soluções. Novamente, é possível observar o crescimento tendendo à linearidade da solução utilizando apenas NoC, em detrimento ao crescimento inconstante, porém em menor taxa, da solução com *clusters*. Na Tabela 5.4, apresentada

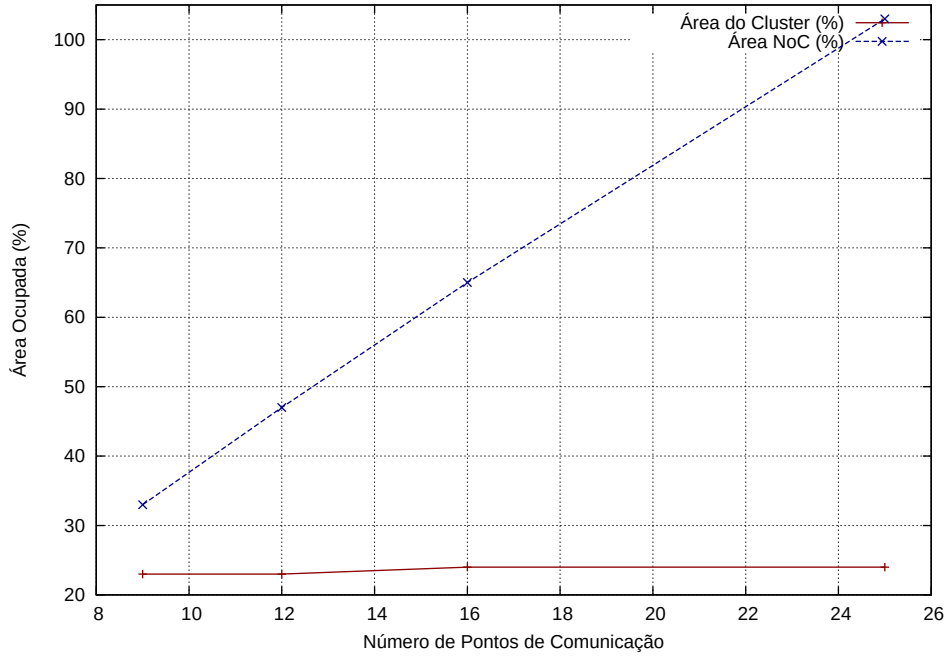

Figura 5.4: Gráfico de área para Virtex II

a seguir, uma comparação entre as velocidades obtidas em ambas soluções é ilustrada. A diferença das velocidades para esta placa é menor quando comparada às diferenças encontradas para a Virtex V. Isto ocorre pois, conforme citado anteriormente, esta FPGA contém menos recursos disponíveis, forçando a implementação de certos módulos do *hardware* de maneira menos otimizada, mascarando possíveis melhorias passiveis de ocorrer na síntese.

Tabela 5.4: Comparação de frequência na Virtex II

Pontos	*cluster MPSoC*	*NoC MPSoC*	Ganho
9	122.318 MHz	110.672 MHz	10%
12	114.699 MHz	110.651 MHz	4%
16	114.788 MHz	109.928 MHz	4%
25	114.788 MHz	103.670 MHz	11%

Finalmente, a Figura 5.5 apresenta o gráfico das velocidades obtidas, onde é possível observar uma queda nas velocidades, de maneira geral quando comparada à Virtex V, graças às limitações da placa já comentadas no texto.

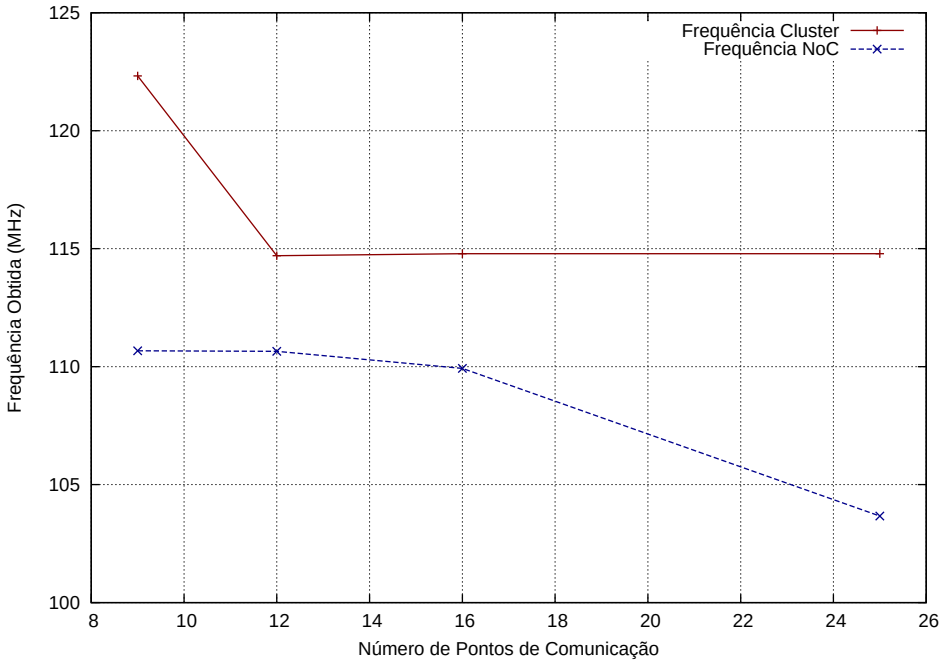

Figura 5.5: Gráfico de frequência para Virtex II

Com posse desses dados foi possível concluir que, utilizando-se a solução de *clusters* para sistemas intrachip multiprocessados, obtém-se uma diminuição na área ocupada do dispositivo em todos casos, porém diminuindo sua frequência de operação também. Para a Virtex V obteve-se um ganho médio relacionado a área de **69.84%**, com uma queda média na frequência de **6.79%**. No caso da Virtex II o ganho médio em área foi de **55.28%** com um ganho médio na frequência de **7.05%**.

5.2 Resultados de síntese para tecnologia 65nm

Para geração de resultados de síntese para tecnologia 65nm, os mesmos dez MPSoCs configurados para síntese em FPGA, na sessão 5.1, foram utilizados. Dois tipos de resultados foram obtidos nesta classe, a de área física ocupada e a potência para ambas soluções.

Primeiramente resultados de área ocupada foram obtidos utilizando-se a ferramenta RC [3]. Nesta geração de resultados, a frequência alvo foi setada para 500MHz para todos MPSoCs. A Tabela 5.5 apresenta uma comparação de ocupação física em micro metros quadrados (um^2) para MPSoCs de nove pontos a 81, utilizando soluções do tipo *cluster* e puramente utilizando NoCs. É possível observar um redução significativa na área ocupada por MPSoCs do tipo *cluster*, chegando a ocupar uma área até 80% menor.

Tabela 5.5: Comparação de área (mm^2)

Pontos	*cluster MPSoC*	*NoC MPSoC*	Ganho
9	0.114	0.149	31%
12	0.114	0.208	45%
16	0.115	0.290	60%
25	0.116	0.477	76%
36	0.184	0.710	74%
42	0.185	0.838	78%
49	0.289	0.988	71%
64	0.290	1.313	78%
72	0.293	1.486	80%
81	0.397	1.682	76%

A Figura 5.6 apresenta o gráfico do crescimento das áreas de ambas soluções. É possível observar um crescimento linear no uso de área quando utiliza-se puramente NoCs, ao contrário do crescimento de uso de área das soluções do tipo *cluster*, e que o crescimento é muito menor.

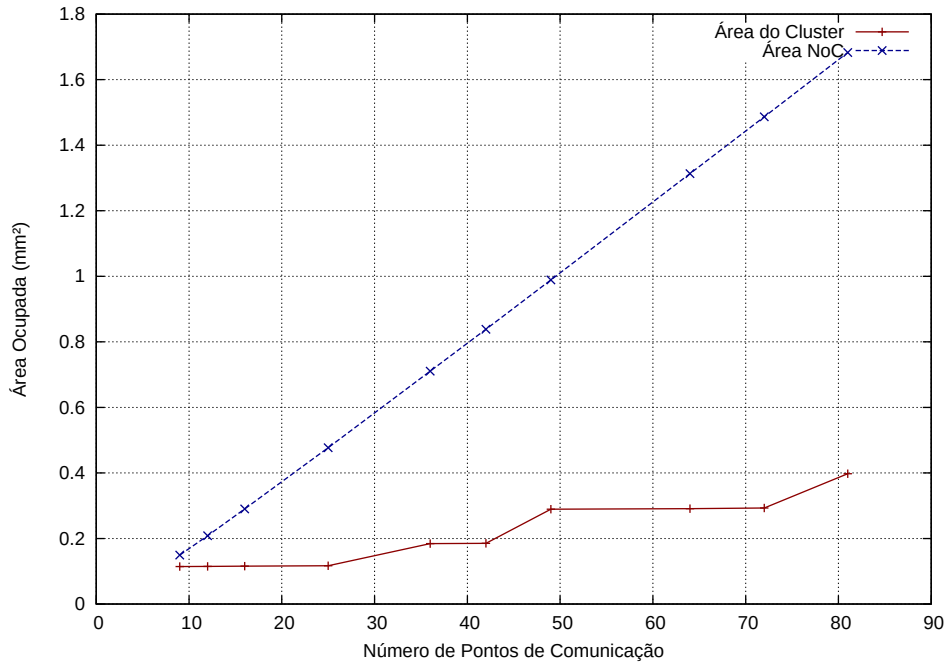

Figura 5.6: Gráfico de área para tecnologia 65nm

Após obtenção das áreas utilizadas para ambas configurações de MPSoC, *cluster* e NoC, os resultados para suas potências foram gerados. Esses resultados são estimativas geradas pela ferramenta RC, da Cadence [2]. A Tabela 5.6 apresenta a comparação dos resultados. Assim como nos resultados de área, os resultados de potência apresentam um menor uso na solução do tipo *cluster*, quando compara à solução puramente com NoC. Neste caso, a potência chega a ser 227% menor no MPSoC *clusterizado*.

Tabela 5.6: Comparação de potência estimada (mW)

Pontos	*cluster MPSoC*	*NoC MPSoC*	Ganho
9	54.535	64.251	18%
12	57.668	90.173	56%
16	62.413	122.773	97%
25	72.740	201.918	178%
36	111.041	299.247	169%
42	117.352	354.028	202%
49	166.192	417.571	151%
64	181.711	552.122	204%
72	190.493	623.421	227%
81	242.746	706.880	191%

Um gráfico representando a taxa de crescimento da potência para ambas as soluções pode ser observado na Figura 5.7, aonde, novamente a solução do tipo NoC apresenta um crescimento linear, ao contrário do tipo *cluster*, em que o crescimento se dá em uma taxa muito menor.

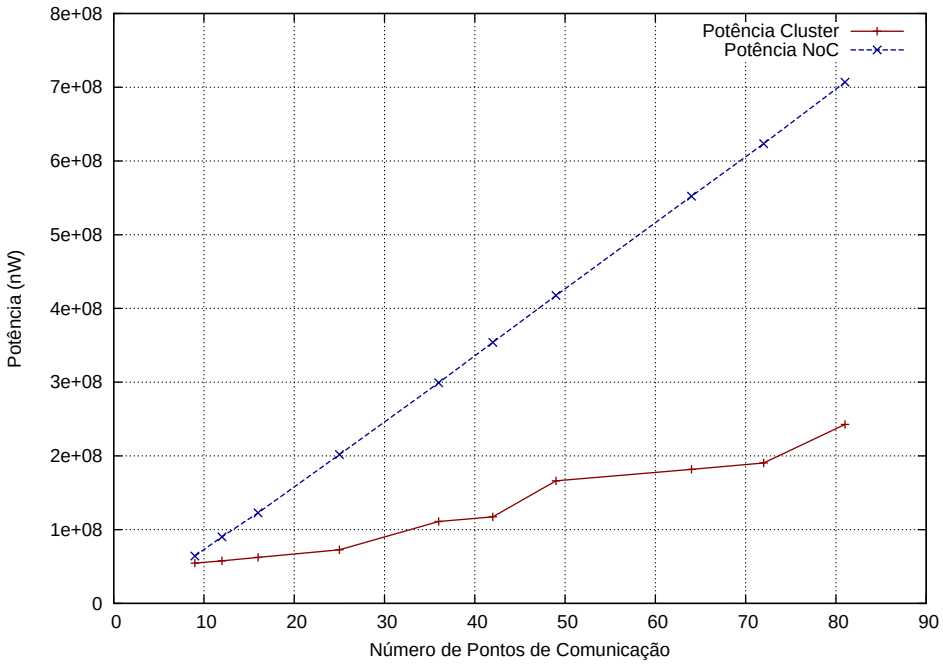

Figura 5.7: Gráfico de potência para tecnologia 65nm

A partir dos dados coletados, pode-se observar que, assim como nos resultados obtidos na síntese para tecnologias FPGA obtém-se uma diminuição na área ocupada pela solução do tipo *cluster*, assim como uma menor potência. Obteve-se um ganho médio relacionado a área de **66.88%**, com uma potência em média **149.32%** menor.

5.3 Resultados de desempenho de comunicação

Para realizar a extração de resultados de desempenho de comunicação, cinco cenários foram gerados para cada tipo de MPSoC, conforme segue:

- ***9 pontos***: NoC central de tamanho 2x2, três nodos contendo dois pontos e um nodo contendo três pontos;
- ***16 pontos*** NoC central de tamanho 2x2, quatro nodos com quatro pontos cada;
- ***25 pontos*** NoC central de tamanho 2x2, três nodos com seis pontos e um nodo com sete pontos;
- ***36 pontos*** NoC central de tamanho 3x2, seis nodos contendo seis pontos cada;
- ***49 pontos*** NoC central de tamanho 3x3, cinco nodos contendo cinco pontos cada e quatro nodos com seis pontos cada, e;
- ***81 pontos*** NoC central de tamanho 4x3, dez nodos com sete pontos cada, um nodo com seis pontos e um nodo com cinco pontos.

Para comparação, cinco MPSoCs formados somente por NoC foram utilizados, contendo os mesmo números de pontos de comunicação citados anteriormente, possuindo as seguintes dimensões: **i)** 9 pontos: NoC 3x3; **ii)** 16 pontos: NoC 4x4; **iii)** 25 pontos: NoC 5x5; **iv)** 49 pontos: NoC 7x7, e; **v)** 81 pontos: NoC 9x9.

Para avaliar a capacidade comunicativa do MPSoC *clusterizado* foi buscado o cenário que mais utilizasse o meio de comunicação, sendo o cenário escolhido um em que todos os pontos se comunicassem com todos, utilizando o meio em sua totalidade. Em cada um dos testes realizados o tamanho das mensagens trocadas variou de 64 *bytes* até 512 *bytes*, sendo os resultados apresentados em número de ciclos de relógio.

A Tabela 5.7 apresenta os valores obtidos em simulações para nove pontos de conexão. Quatro tamanhos de mensagens foram utilizados, 64 *bytes*, 128 *bytes*, 256 *bytes* e 512 *bytes*. É possível observar ainda, dois campos para cada tamanho de mensagem, *Total e Interno*. O campo *Total*, como o nome sugere, refere-se ao tempo total que um ponto leva para enviar mensagens para todos os outros pontos. Já o campo *Interno* refere-se ao tempo que um ponto leva para enviar mensagens para todos os pontos que fazem parte de seu *cluster*.

Na Tabela é possível observar um maior tempo para o envio total das mensagens utilizando-se uma arquitetura do tipo *cluster* quando comparado a arquiteturas exclusivamente formadas por NoCs, fato este já esperado. Ao contrário de MPSoCs compostos somente por NoCs, que valem-se do paralelismo para troca de mensagens, o *cluster* utiliza um barramento que só pode ser utilizado por um ponto de cada vez, gerando uma maior contenção. Porém, o tempo utilizado para a troca

Tabela 5.7: Comparação para 9 pontos

64 *bytes*		128 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
1494 ciclos	98 ciclos	2859 ciclos	184 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
894 ciclos	116 ciclos	1803 ciclos	263 ciclos

256 *bytes*		512 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
5590 ciclos	354 ciclos	11051 ciclos	296 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
3103 ciclos	584 ciclos	7168 ciclos	1126 ciclos

de mensagens internas ao *cluster* foi menor do que o tempo para a NoC trocar estas mesmas mensagens, demonstrando uma vantagem em seu uso.

A Figura 5.8 apresenta um gráfico do tempos médios, em ciclos, dos envios em MPSoCs com nove pontos, neste caso observou-se um aumento de 64.98% em média no tempo Total de troca de mensagens do MPSoC *clusterizado* com uma redução média de 30.79% no tempo de envio Interno.

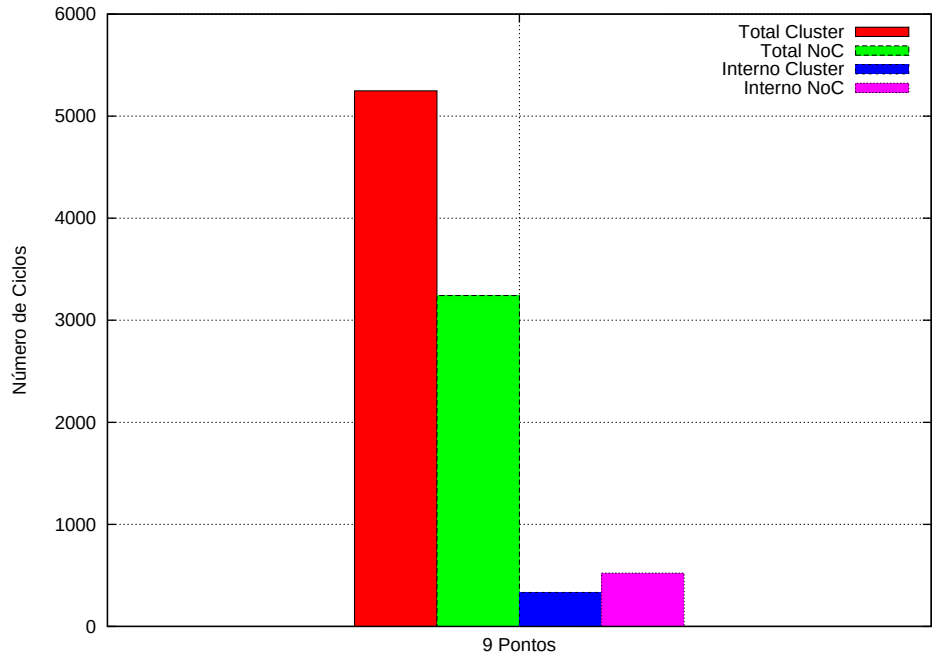

Figura 5.8: Gráfico de comparando ciclos de envio para 9 pontos

A Tabela 5.8 apresenta a comparação dos resultados obtidos para 16 pontos de comunicação, demonstrando mais uma vez uma menor eficiência para o envio Total de mensagens do MPSoC

clusterizado, porém com um menor tempo para troca de mensagens na parte interna do *cluster*, pelos mesmos motivos já comentados anteriormente.

Tabela 5.8: Comparação para 16 pontos

64 *bytes*		128 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
4998 ciclos	391 ciclos	9590 ciclos	728 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
2088 ciclos	476 ciclos	4336 ciclos	852 ciclos

256 *bytes*		512 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
18774 ciclos	1400 ciclos	37142 ciclos	2744 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
8561 ciclos	1865 ciclos	17297 ciclos	3752 ciclos

O gráfico dos tempos médios de envio para MPSoCs com 16 pontos de comunicação é apresentado na Figura 5.9 aonde é possível observar uma queda média de 23.29% no tempo de troca de mensagens internas no *cluster* e um aumento de 123.64% em média para o envio Total.

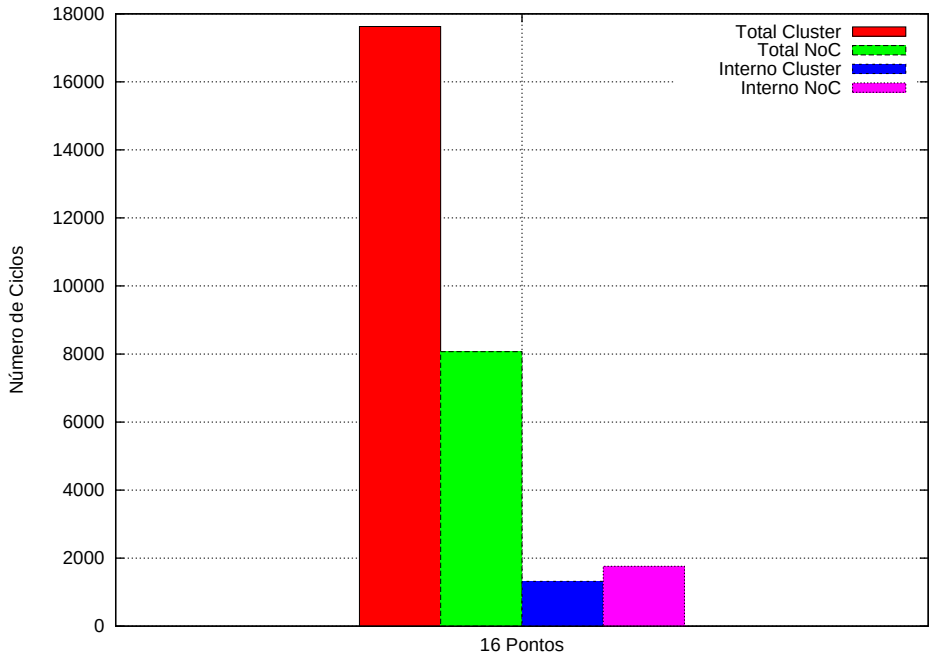

Figura 5.9: Gráfico de comparando ciclos de envio para 16 pontos

Na Tabela 5.9 os resultados apresentados diferem dos demais obtidos. Enquanto o tempo Total manteve o mesmo comportamento dos demais casos, com o MPSoC *clusterizado* levando mais tempo para enviar suas mensagens, neste caso o tempo Interno também foi maior no MPSoC

clusterizado. Acredita-se que este fato ocorreu graças a uma relação entre o número de pontos internos em cada *cluster*, frente ao tamanho total do MPSoC, fato este porém que deverá ser abordado em um trabalho futuro.

Tabela 5.9: Comparação para 25 pontos

64 *bytes*		128 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
12684 ciclos	1144 ciclos	24362 ciclos	2127 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
4080 ciclos	756 ciclos	8427 ciclos	1500 ciclos

256 *bytes*		512 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
47720 ciclos	4093 ciclos	94435 ciclos	8025 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
15586 ciclos	2959 ciclos	33228 ciclos	6076 ciclos

Na Figura 5.10 pode-se notar a perda citada anteriormente no texto em forma de gráfico, onde ambos os casos apresentam um pior rendimento para a arquitetura *clusterizada*. Neste caso o envio Total teve uma média de 197.59% maior para envios utilizando-se MPSoCs *clusterizados* e 40.89% maior nos envios Internos.

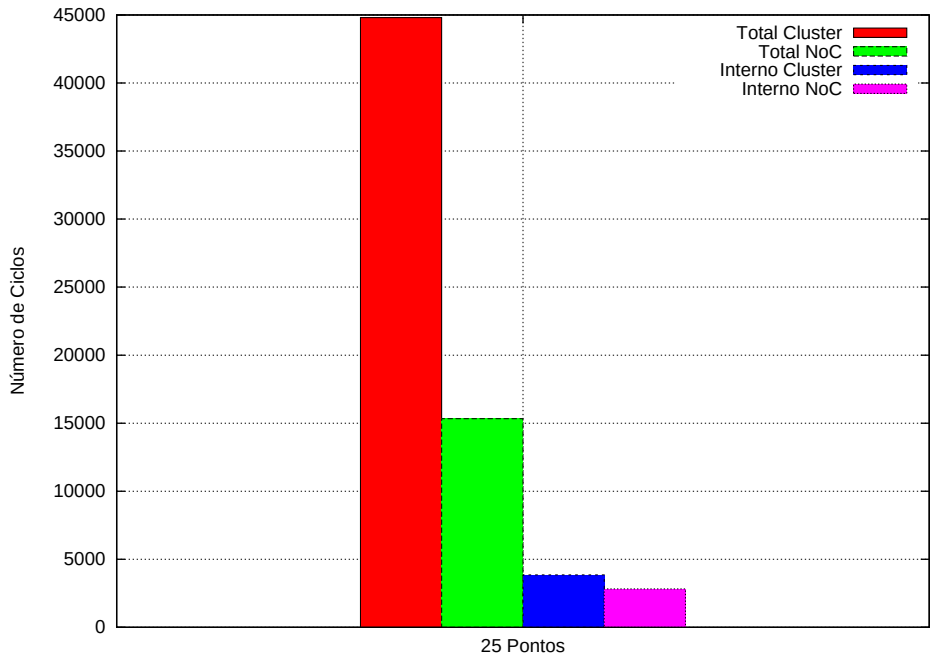

Figura 5.10: Gráfico de comparando ciclos de envio para 25 pontos

O resultados obtidos para a comunicação utilizando 49 pontos podem ser observado na Tabela 5.10 onde, mais uma vez, a comunicação de todos para todos levou mais tempo ao utilizar-se uma arquitetura *clusterizada*, com um tempo menor para os envios internos.

Tabela 5.10: Comparação para 49 pontos

64 *bytes*		128 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
25092 ciclos	849 ciclos	48315 ciclos	1579 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
12859 ciclos	1630 ciclos	24480 ciclos	3316 ciclos

256 *bytes*		512 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
94743 ciclos	3032 ciclos	187648 ciclos	5956 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
59650 ciclos	6152 ciclos	93339 ciclos	11772 ciclos

A Figura 5.11 apresenta um gráfico comparando os tempos de envio de ambas arquiteturas. O MPSoC formado por *cluster* obteve uma média 88.10% maior para envios totais, porém, como nos casos de nove e 16 pontos, observou-se uma redução no tempo de envio para os pontos internos do *cluster*, em uma taxa média de 50.10%.

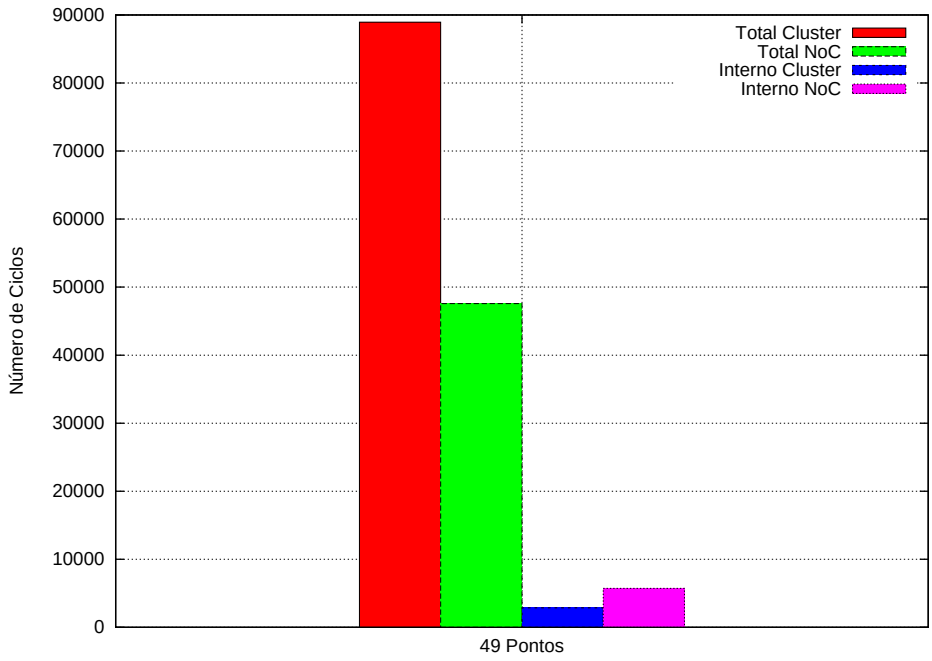

Figura 5.11: Gráfico de comparando ciclos de envio para 49 pontos

O ultimo cenário utilizado foi formado por 81 pontos de comunicação, e os resultados podem ser visualizados na Tabela 5.11, onde é possível notar-se o mesmo comportamento dos casos de nove, 16 e 49 pontos, com um tempo Total maior para arquitetura do tipo *cluster* e um tempo menor para envios internos.

Tabela 5.11: Comparação para 81 pontos

64 *bytes*		512 *bytes*	
Total *cluster*	Interno *cluster*	Total *cluster*	Interno *cluster*
52838 ciclos	1371 ciclos	395735 ciclos	9657 ciclos
Total *NoC*	Interno *NoC*	Total *NoC*	Interno *NoC*
24643 ciclos	2979 ciclos	189998 ciclos	22691 ciclos

Uma queda média de 55.77% no tempo de envio das mensagens internas do *cluster* e um aumento de 111.35% no tempo de envio Total das mensagens foram obtidos para o cenário com 81 pontos de comunicação, tendo seus tempos médios de envio representados no gráfico da Figura 5.12.

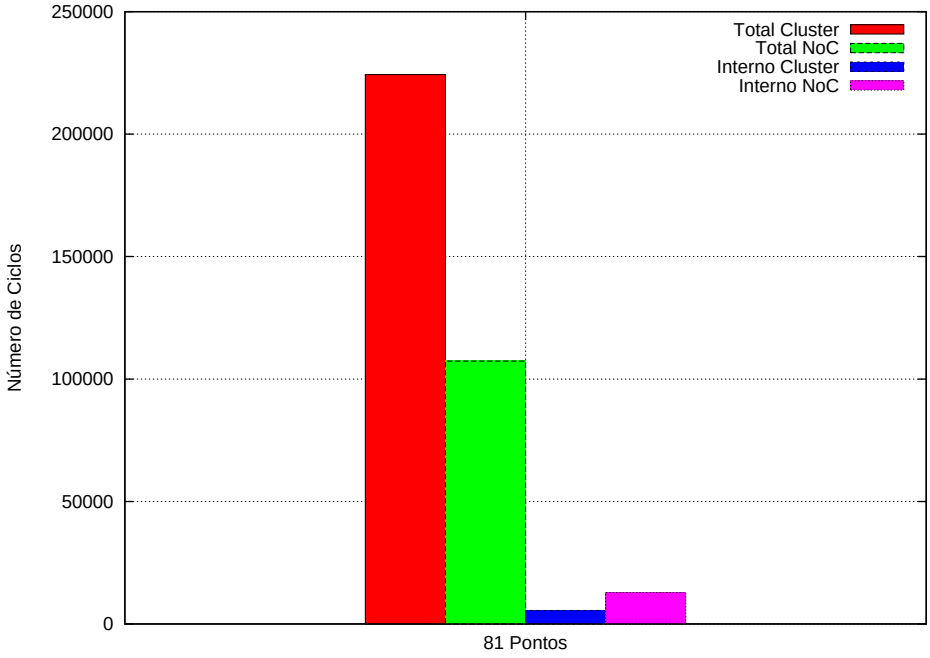

Figura 5.12: Gráfico de comparando ciclos de envio para 81 pontos

Após verificação de todos os cenários, pode-se chegar a conclusão de que o tempo médio para envio de mensagens Total utilizando-se MPSoCs *clusterizados* é 117.77% maior, porém com uma redução no tempo médio de comunicação interna do *cluster* de 20.26%.

6. CONCLUSÃO E TRABALHOS FUTUROS

O trabalho proposto apresentou um modelo de arquitetura *clusterizada* para MPSoCs, denominada HC-MPSoC, utilizando barramentos e NoCs. Esse modelo apresenta uma organização híbrida, constituída por nodos formados por núcleos *IP* interligados por um barramento, e uma rede intrachip central interligando todos estes nodos. Foram apresentados todos os detalhes de sua implementação, explicando o funcionamento de cada módulo que faz parte do sistema, sendo estes os módulos desenvolvidos: barramento, *wrappers* que interligam o barramento ao roteador da NoC, *wrappers* para interligar o processador utilizado ao barramento e a descrição de um *driver* para uso do HellfireOS nesta arquitetura. Esta arquitetura foi projetada de modo a permitir o agrupamento de módulos *IP* com maior afinidade em grupos, denominados *clusters*. Dessa maneira buscou-se permitir ao desenvolvedor projetar seu sistema de forma direcionada à aplicação, utilizando os recursos da maneira mais otimizada possível.

No sentido de validação da arquitetura, três cenários de estudo de caso foram propostos. Em todos os cenários foi feita uma comparação com MPSoCs formados unicamente por uma NoC central, ou seja, com apenas um *IP* em cada roteador. Isso foi feito para ter-se um ponto de referência nos dados obtidos nos resultados para o HC-MPSoC.

O primeiro cenário foi a geração de resultados para sínteses em dispositivos FPGA, onde dois dispositivos, Virtex V e Virtex IIP, foram utilizados. Neste primeiro cenário foram comparadas as áreas de ocupação nos dispositivos e as frequências de operação obtidas. Observou-se que a arquitetura *clusterizada* ocupou, em média, uma área 69.84% menor para o dispositivo Virtex V e 55.28% para Virtex II. Os resultados de frequência apontam uma queda média de 6.79% nos resultados obtidos para Virtex V e um ganho de 7.05% para Virtex IIP.

No segundo cenário, resultados da síntese física para tecnologia de 65nm da STMicro foram obtidos. Neste caso a frequência alvo foi setada para 500MHz e a área física do *chip* e a potência estimada foram avaliadas. Os resultados apontam para um uso médio de área 66.88% menor utilizando-se um MPSoC *clusterizado*, com uma redução média na potência estimada de 149.32%.

O ultimo cenário utilizado para comparação foi o desempenho de comunicação da arquitetura desenvolvida. Neste caso, tráfego era injetado na arquitetura e os tempos, em ciclos de relógio, anotados. Os resultados apontaram um tempo maior para troca de mensagens utilizando-se uma arquitetura do tipo *cluster* de, em média, 117.77%. Porém, quando comparados os tempos de comunicação somente dos pontos internos ao *cluster*, o HC-MPSoC obteve um tempo de comunicação 20.26% menor, quando comparado às mesmas trocas de mensagens utilizando somente uma NoC.

A arquitetura proposta se mostrou válida e foi verificada por simulações e prototipações em FPGA. As alterações realizadas no HellfireOS foram validadas nestes cenários citados anteriormente e mostraram-se funcionais.

Além disso, uma revisão bibliográfica foi realizada com o intuito de explorar alguns dos principais trabalhos relacionados existentes. Uma breve análise crítica de cada um desses trabalhos com relação ao modelo proposto pôde ser apreciada ao longo de sua descrição. Também realizou-se uma pesquisa no sentido de oferecer ao leitor um embasamento teórico mais aprofundado sobre as questões de infraestruturas de comunicação em sistemas embarcados multiprocessados.

6.1 Trabalhos Futuros

É importante destacar que a pesquisa realizada, não obstante, deixa margem para que diversos trabalhos futuros possam ser realizados. Inicialmente, a verificação mais precisa com relação ao impacto da divisão de unidades de processamento em *clusters*, assim como a maneira mais eficiente de fazê-lo. Ainda no sentido da divisão do sistema, deve-se buscar métodos de mapeamento e particionamento de tarefas que levem em consideração este tipo de arquitetura.

A implementação de modelos em alto nível da arquitetura, como simuladores, para uma mais rápida verificação de aplicações executando sobre ela é uma tarefa muito importante a ser realizada, agilizando assim o processo de desenvolvimento.

Finalmente, é interessante a avaliação da arquitetura proposta frente outras plataformas. Deve-se verificar quais componentes arquiteturais beneficiam ou prejudicam o desempenho do sistema como um todo, bem como atualizar e evoluir a arquitetura atual.

6.2 Publicações

Durante o período do mestrado, período este de dois anos, alguns trabalhos científicos foram submetidos para conferencias nacionais e internacionais. Além disso um mini-curso a respeito de sistemas embarcados foi elaborado e publicado em uma revista nacional por dois anos seguidos (2011 e 2012), além de ter sido ministrado pelo autor nos eventos de lançamento de tais revistas.

A lista a seguir apresenta os documentos submetidos e aceitos para publicação durante o período do mestrado do autor:

- Introdução ao desenvolvimento de software embarcado - Curso publicado em revista do CBSEC e ministrado pelo autor nos eventos de lançamento (2011 e 2012);
- Adapting Embedded Systems' Framework to Provide Virtualization: the Hellfire Case Study - Artigo CBSEC 2011;
- Task Model Suitable for Dynamic Load Balancing of Real-Time Applications in NoC-based MPSoCs - Artigo ICCD 2012;
- Noc-based platform for embedded software design: An extension of the Hellfire Framework - Artigo ISQED 2011;

- Embedded Virtualization for the Next Generation of Cluster-based MPSoCs - Artigo RSP 2011, e;
- Communication support at the OS level to enhance design space exploration in multiprocessed embedded systems - Poster SAC 2012.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] "Amba ahb reference". Capturado em: http://alturl.com/88d98, acesso em Abril, 2013.

[2] "Cadence". Capturado em: http://www.cadence.com/, acesso em Janeiro, 2013.

[3] "Cadence rc". Capturado em: http://alturl.com/w4t5h, acesso em Janeiro, 2013.

[4] "Altera ltd, nios ii processor reference". Capturado em: http://www.altera.com, acesso em Junho, 2013.

[5] "Hellfire system - designing embedded systems". Capturado em: http://hellfire.gse.inf.br/, acesso em Junho, 2013.

[6] "Modelsim". Capturado em: http://model.com/, acesso em Junho, 2013.

[7] Aguiar, A.; de Magalhaes, F.; Hessel, F. "Embedded virtualization for the next generation of cluster-based mpsocs". In: Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on, 2011, pp. 113 –119.

[8] Aguiar, A.; Filho, S.; Magalhaes, F.; Casagrande, T.; Hessel, F. "Hellfire: A design framework for critical embedded systems' applications". In: Quality Electronic Design (ISQED), 2010 11th International Symposium on, 2010, pp. 730–737.

[9] Chen, C.; Du, G.; Zhang, D.; Song, Y.; Hou, N. "Communication synchronous scheme for mpsoc". In: Anti-Counterfeiting Security and Identification in Communication (ASID), 2010 International Conference on, 2010, pp. 310 –313.

[10] da Costa, C. M. "Sistemas Operacionais - Programação Concorrente com Pthreads". ediPUCRS, 2010, 1 ed., 212p.

[11] de Magalhães, F. G. "Extensão da plataforma hellfire para utilização de redes intrachip". Trabalho de Conclusão de Curso, Pontificia Universidade Católica do Rio Grande do Sul, 2010, 67p.

[12] Filho, S.; Aguiar, A.; Marcon, C.; Hessel, F. "High-level estimation of execution time and energy consumption for fast homogeneous mpsocs prototyping". In: Rapid System Prototyping, 2008. RSP '08. The 19th IEEE/IFIP International Symposium on, 2008, pp. 27–33.

[13] Filho, S. J. "Estimativa de desempenho de software e consumo de energia em mpsocs", Dissertação de Mestrado, Pontificia Universidade Católica do Rio Grande do Sul, 2008, 160p.

[14] Heath, S. "Embedded Systems Design". Newton, MA, USA: Butterworth-Heinemann, 2002, 430p.

[15] Jin, X.; Song, Y.; Zhang, D. "Fpga prototype design of the computation nodes in a cluster based mpsoc". In: Anti-Counterfeiting Security and Identification in Communication (ASID), 2010 International Conference on, 2010, pp. 71 –74.

[16] Le Moigne, R.; Pasquier, O.; Calvez, J.-P. "A generic rtos model for real-time systems simulation with systemc". In: Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, 2004, pp. 82–87 Vol.3.

[17] Leng, X.; Xu, N.; Dong, F.; Zhou, Z. "Implementation and simulation of a cluster-based hierarchical noc architecture for multi-processor soc". In: Communications and Information Technology, 2005. ISCIT 2005. IEEE International Symposium on, 2005, pp. 1203 – 1206.

[18] Li, Q.; Yao, C. "Real-Time Concepts for Embedded Systems". CMP Books, 2003, 294p.

[19] Longhi, O. B. "Incrementando o simulador do ambiente hellfirefw para suportar sistemas noc". Trabalho de Conclusão de Curso, Pontificia Universidade Católica do Rio Grande do Sul, 2011, 43p.

[20] Luo-feng, G.; Duo-li, Z.; Ming-Lun, G. "Performance evaluation of cluster-based homogeneous multiprocessor system-on-chip using fpga device". In: Computer Engineering and Technology (ICCET), 2010 2nd International Conference on, 2010, pp. V4–144 –V4–147.

[21] Melpignano, D.; Benini, L.; Flamand, E.; Jego, B.; Lepley, T.; Haugou, G.; Clermidy, F.; Dutoit, D. "Platform 2012, a many-core computing accelerator for embedded socs: Performance evaluation of visual analytics applications". In: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, 2012, pp. 1137 –1142.

[22] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. "Hermes: an infrastructure for low area overhead packet-switching networks on chip", *Integr. VLSI J.*, vol. 38–1, 2004, pp. 69–93.

[23] Pasricha, S.; Dutt, N. "On-Chip Communication Architectures: System on Chip Interconnect". San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008, 544p.

[24] Pasricha, S.; Dutt, N. "Presentations about the book: On-chip communication architectures: System on chip interconnect". Capturado em: http://www.engr.colostate.edu/~sudeep/research/book_comm.htm, acesso em Junho, 2013.

[25] Rhoads, S. "Mips plasma". Capturado em: http://opencores.org/project,plasma,overview, acesso em Junho, 2013.

[26] Seifi, M.; Eshghi, M. "A clustered noc in group communication". In: TENCON 2008 - 2008 IEEE Region 10 Conference, 2008, pp. 1 –5.

[27] Tudruj, M.; Masko, L. "Dynamic smp clusters with communication on the fly in soc technology applied for medium-grain parallel matrix multiplication". In: Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on, 2007, pp. 270 –277.

[28] Wolf, W. "Embedded computer architectures in the mpsoc age". In: Proceedings of the 2005 workshop on Computer architecture education: held in conjunction with the 32nd International Symposium on Computer Architecture, 2005, 3p.

[29] Xilinx. "Ise project navigator". Capturado em: http://goo.gl/pXl01, acesso em Maio, 2013.

[30] Xilinx. "Virtex 2 - pró". Capturado em: http://www.xilinx.com/univ/xupv2p.html, acesso em Maio, 2013.

[31] Xilinx. "Virtex v". Capturado em: http://alturl.com/py7nj, acesso em Maio, 2013.

[32] Yoo, S.; Nicolescu, G.; Gauthier, L.; Jerraya, A. "Automatic generation of fast timed simulation models for operating systems in soc design". In: DATE '02: Proceedings of the conference on Design, automation and test in Europe, 2002, pp. 620 – 627.