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

ANDRÉ LUIZ HOFER

MAPEAMENTO DE APLICAÇÕES COM COMUNICAÇÃO A PERIFÉRICOS EM REDES INTRA CHIP COM ZONAS SEGURAS

CHAPECÓ
2022

ANDRÉ LUIZ HOFER

MAPEAMENTO DE APLICAÇÕES COM COMUNICAÇÃO A PERIFÉRICOS EM REDES INTRA CHIP COM ZONAS SEGURAS

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

CHAPECÓ
2022

Hofer, André Luiz

Mapeamento de aplicações com comunicação a periféricos em redes intra chip com zonas seguras / André Luiz Hofer. – 2022.
53 f.: il.

Orientador: Dr. Luciano Lores Caimi.
Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2022.

1. MPSoC. 2. OSZ. 3. Mapeamento de tarefas. 4. Periféricos. 5. NoC. I. Caimi, Dr. Luciano Lores, orientador. II. Universidade Federal da Fronteira Sul.

E-mail: andreluizhofer@gmail.com.br

ANDRÉ LUIZ HOFER

MAPEAMENTO DE APLICAÇÕES COM COMUNICAÇÃO À PERIFÉRICOS EM REDES INTRA CHIP COM ZONAS SEGURAS

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

Orientador: Dr. Luciano Lores Caimi

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 9/8/2022.

BANCA AVALIADORA

Dr. Luciano Lores Caimi – UFFS

Me. Adriano Sanick Padilha – UFFS

Dr. Marco Aurélio Spohn – UFFS

RESUMO

O uso de sistemas de múltiplos processadores em um chip (MPSoC) introduz questões referentes a segurança das aplicações e acesso a periféricos nessas plataformas. Entre as ações para resolver os problemas de segurança estão o uso de criptografia, autenticação de periféricos e a criação de Zonas Seguras Opacas (OSZs). Com a criação de OSZs e a necessidade de acesso a periféricos na execução de aplicações seguras, alguns problemas podem ser observados, como a criação de uma OSZ sobre um elemento de processamento ao qual está conectado um periférico ou a disposição das OSZs impossibilitando o acesso de algum elemento de processamento a um periférico, denominado neste trabalho como problema de bloqueio de periféricos, e ainda o problema causado pelo alinhamento de tarefas que se comunicam com periféricos dentro de uma OSZ, denominado neste trabalho como problema de sombreamento. Tendo em vista que a execução segura é primordial para algumas aplicações, a não resolução desses problemas implica em um possível travamento ou falha na execução de aplicações seguras. Assim, objetivou-se definir um algoritmo de definição do formato e posicionamento de OSZs em um MPSoC e definir um algoritmo de mapeamento de tarefas na OSZ de forma que todo periférico conectado à plataforma esteja acessível para qualquer tarefa. Primeiramente, realizou-se a instalação das ferramentas necessárias para execução da plataforma, bem como a ambientação da mesma, estudando os algoritmos utilizados para definição de formatos de OSZ e mapeamento das tarefas na área da OSZ. Prosseguiu-se com uma revisão de literatura em que foram estudados algoritmos e técnicas para definir o formato e o posicionamento da OSZ evitando o bloqueio no acesso a periféricos por qualquer outra tarefa presente no MPSoC; em seguida, foi implementada uma solução que atende à demanda de não bloqueio do acesso a periféricos. Após a implementação, uma etapa de validação dos algoritmos foi realizada, executando um conjunto de casos de testes de aplicações seguras na plataforma e verificando seu funcionamento para avaliar os algoritmos de definição e posicionamento dos formatos e também de mapeamento de tarefas quanto ao funcionamento, ao desempenho e à disponibilidade de recursos. Os resultados obtidos mostram a correta execução das aplicações levando em consideração as restrições de formato e posicionamento da OSZ e mapeamento das tarefas das aplicações em função da posição dos periféricos presentes no MPSoC. Os resultados mostraram um aumento do tempo de execução dos algoritmos de posicionamento da OSZ e mapeamento das tarefas e eventual aumento no consumo de recursos de processamento na fase de alocação das aplicações. Devido à premissa de que a segurança na execução das aplicações é primordial, esse aumento de tempo e de consumo de recursos se torna um custo aceitável, já que no pior cenário dentre os testes executados o aumento no tempo total de execução foi de 3,51%.

Palavras-chave: MPSoC. OSZ. Mapeamento de tarefas. Periféricos. NoC.

ABSTRACT

Using multi-processor systems-on-chip (MPSoC) introduces issues regarding application security and access to peripherals on these platforms. Among the actions to solve security problems are encryption, authentication of peripherals, and creating Opaque Secure Zones (OSZs). With the creation of OSZs and the requirement to access peripherals in the execution of secure applications, some problems arrise, such as of creating an OSZ on a processing element to which a peripheral is connected or the OSZs disposition preventing the access of some processing element to a peripheral, called in this work as peripheral blocking problem, as well as and the problem caused by the alignment of tasks that communicate with peripherals within an OSZ, called as shading problem in this study. Once secure execution is essential for some applications, not solving these problems implies a possible crash or failure in the execution of secure applications. Thus, this study aimed to create an algorithm for defining the shape and positioning of OSZs in an MPSoC and to create an algorithm for task mapping in the OSZ so that every peripheral connected to the platform is accessible for any task. Firstly, the necessary tools to run the platform were installed, as well as its setting, studying algorithms used to define OSZ formats and mapping tasks in the OSZ area. A literature review was carried out in to study algorithms and techniques to define the format and positioning of the OSZ, avoiding blocking access to peripherals by any other task present in the MPSoC, after implementing a solution that meets the demand for not blocking access to peripherals. Following the implementation, a validation step of the algorithms occurred, executing a set of test cases of secure applications on the platform and verifying its functioning to evaluate the algorithms for defining and positioning the formats and mapping tasks regarding the functioning, performance, and resource availability. The results show the correct execution of the applications taking into account the restrictions of format and positioning of the OSZ and mapping of the tasks of the applications in the function of the position of the peripherals present in the MPSoC. The results shows an increase in the execution time of the OSZ positioning algorithms and task mapping and an eventual increase in the consumption of processing resources in the allocation phase of the applications. Due to the premise that security in the execution of applications is paramount, this increase in time and resource consumption is an acceptable cost since, as in the worst case scenario among the tests performed, the total time of test execution increased 3.51%. The increase in resource consumption is directly related to the number of application tasks with access to peripherals.

Keywords: MPSoC. OSZ. Task mapping. Peripheral. NoC.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

Apesar da Lei de Moore, que afirma que a quantidade de transistores possíveis de serem inseridos em uma mesma área de circuito dobram a cada dois anos, estar caindo em desuso, a evolução a nível de *hardware* das plataformas vem ocorrendo e trazendo um aumento nos componentes inseridos em uma mesma área dos circuitos já há algumas décadas. Existe um vasto campo ao qual essas tecnologias podem ser aplicadas, entre estas os sistemas intra-chip (SoC - *em inglês Systems on Chip*) que consistem em sistemas com vários componentes integrados em um único chip (processadores, aceleradores, memórias, etc). Uma plataforma SoC de múltiplos núcleos com interconexão baseada em redes intra-chip (NoC - em inglês *Network on Chip*) (MPSoC - em inglês *Multi-Processor System on Chip*), tem seu uso crescente relacionado a importantes setores da ampla e desenvolvida cadeia produtiva de equipamentos eletrônicos.

As principais questões em estudo envolvendo SoC estão relacionadas ao consumo de energia, envelhecimento e depreciação por uso ou aquecimento desproporcional dos componentes do circuito e a segurança das aplicações executadas nestes sistemas.

Neste cenário, a segurança é um tópico cada vez mais relevante dado que o paralelismo na execução de aplicações dentro do sistema podem colocar em risco as informações ali contidas. Uma aplicação maliciosa, quando executada, pode ter acesso às mensagens da comunicação entre duas tarefas sendo executadas em elementos de processamento distintos que trocam informações entre si, acessar áreas de memória com informações de outra aplicação, além de fazer ataques à disponibilidade dos recursos de comunicação e computação.

Entre as técnicas usadas para mitigar os ataques de aplicações maliciosas estão o uso de criptografia na comunicação entre as tarefas da aplicação e também entre as tarefas e periféricos conectados à plataforma, a identificação e validação de periféricos conectados à plataforma, bem como o uso de zonas seguras (CAIMI; FOCHI; WACHTER; MUNHOZ et al., 2017). As zonas seguras são definidas como o bloqueio de recursos de processamento e de comunicação em uma área determinada para a execução de apenas uma aplicação, evitando assim que outras aplicações tenham acesso aos elementos de processamento no interior da zona segura ou que mensagens de comunicações de outras aplicações atravessem essa região.

Um MPSoC pode ser construído com diversas arquiteturas distintas. Neste trabalho, é representado um MPSoC constituído de vários elementos de processamento (PE - em inglês *Processor Element*) idênticos, interconectados por uma NoC (Figura 1.a), onde cada elemento de processamento, além de um roteador, contêm CPU, uma memória, um gerador de números pseudo-randômico, uma interface de rede com acesso direto à memória (DMNI - em inglês *Direct Memory Network Interface*), um controlador de empacotadores e empacotadores (em inglês *Wrappers*) em cada porta de ambas as NoCs (Figura 1.b) (RUARO; CAIMI; FOCHI

et al., 2019).

Os elementos de processamento são distribuídos na plataforma em formato de malha 2D, podendo ser divididos em conjuntos chamados de *cluster*. Por exemplo, se o MCSoC possuir um tamanho de 8 x 8 PEs, podem ser gerados quatro *clusters* de tamanho 4 x 4 PEs para gerenciamento local do sistema, como mostra a Figura 1.a. O PE no canto inferior esquerdo da plataforma é o processador de gerenciamento global do sistema (GMP - em inglês *Global Manager Processor*), da mesma forma os PEs nos cantos inferiores esquerdos dos *clusters* são os processadores de gerenciamento local do respectivo *cluster* (LMP - em inglês *Local Manager Processor*). Os demais PEs são os processadores escravos (SP - em inglês *Slave Processors*), onde ocorre a execução das aplicações.

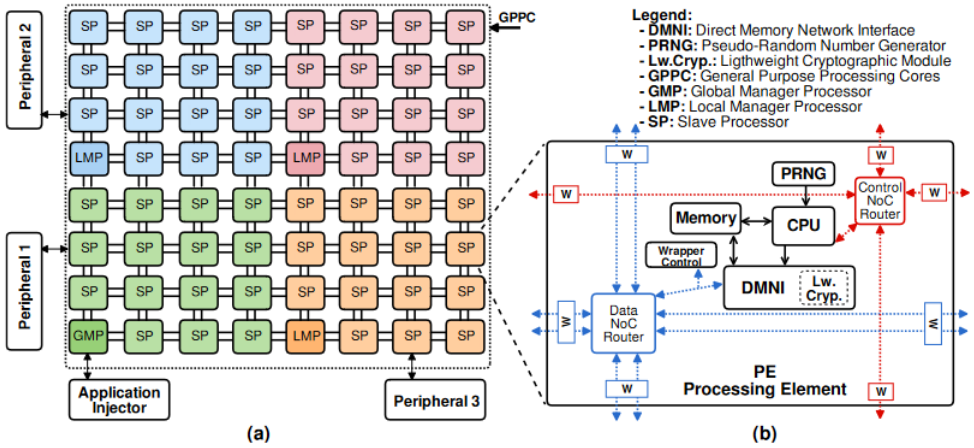


Figura 1 – Arquitetura MPSoC. Fonte: [Caimi, 2019]

A plataforma tem suporte para conexão de periféricos, uso de aceleradores de hardware, memórias auxiliares, dispositivos de entrada e saída como conexões de comunicação externa, como USB ou *Ethernet*. Existe também um periférico específico, o injetor de aplicações, responsável por encaminhar aplicações a serem executadas na plataforma ao processador de gerenciamento global. Os periféricos são conectados às portas livres dos PEs na borda da plataforma, nas portas norte dos PEs na borda superior, oeste dos PEs na borda à esquerda, leste dos PEs na borda à direita e sul dos PEs na borda inferior.

Cada PE têm roteadores com acesso a duas NoCs, sendo uma NoC de dados e uma NoC de gerência/controle, sem nenhuma ligação de *hardware* entre elas. A nível de *software*, apenas o Sistema Operacional de cada PE tem acesso as NoCs, ou seja, não é possível uma aplicação ter acesso aos dados de uma NoC a partir da outra.

Na NoC de controle cada roteador possui uma porta conectada diretamente à CPU (porta local) e quatro portas (leste, oeste, norte e sul) para interconexão da NoC. A NoC de controle é usada para troca de mensagens de controle entre os PEs, como mensagens de busca de caminhos ou mensagens de definição de Zonas Seguras. É usado protocolo *broadcast* na troca de mensagens na NoC de controle (WACHTER et al., 2017).

Na NoC de dados cada roteador, há uma porta conectada a DMNI (RUARO; LAZZAROTTO et al., 2016) e ao controlador de *wrappers* e quatro portas com canais físicos duplicados em cada direção (leste, oeste, norte e sul) para interconexão da NoC. A NoC de dados é usada para troca de mensagens entre as tarefas das aplicações e entre aplicações e periféricos. As portas possuem canais físicos duplicados para garantir redundância nos canais e possibilitar uso de algoritmos de roteamento distintos para cada canal.

Nas três fases das aplicações, que são a alocação de recursos para as aplicações, execução segura das aplicações e acesso seguro a periféricos, emprega-se várias técnicas para garantir a segurança. Tem-se como finalidade reduzir a possibilidade de ataques à execução íntegra das aplicações e sem expor dados sensíveis das mesmas a alguma aplicação maliciosa em qualquer fase de seu ciclo de vida (CAIMI; FOCHI; WACHTER; MUNHOZ et al., 2017).

As aplicações que necessitam de segurança são executadas em Zonas Seguras Opacas (OSZ - em inglês *Opaque Secure Zone*). Uma OSZ é definida como um conjunto de PEs organizados em forma retangular ou retilínea, onde as portas da fronteira da OSZ são fechadas pelos *wrappers* e descartam todas as mensagens da NoC de dados e as mensagens da NoC de controle que não sejam encaminhadas em modo global, evitando assim que o tráfego de qualquer outra aplicação atravesse a zona segura e que as aplicações da zona segura trafeguem informações para fora (CAIMI; FOCHI; WACHTER; MORAES, 2018).

Quando uma aplicação não segura é executada na plataforma, suas tarefas podem ser mapeadas em PEs não adjacentes; dessa forma, caso as tarefas troquem informações o roteamento das mensagens é feito em XY (Figura 2.a), isto é, a mensagem percorre primeiro a direção X até encontrar a coluna do destino e depois percorre a direção Y até o PE de destino. Caso seja alocada uma aplicação segura que sobreponha a rota da comunicação entre as tarefas, quando fechadas as fronteiras da OSZ (Figura 2.b), o *wrapper* que receber uma mensagem tentando atravessar a OSZ retorna uma mensagem de rota quebrada ao PE de origem, que através da NoC de controle encontra um novo caminho para o envio da mensagem, usando roteamento de origem (SR, em inglês *Source Routing*). Este novo caminho torna-se o caminho padrão de comunicação entre os dois PEs, e precisa ser calculado apenas uma vez (Figura 2.c).

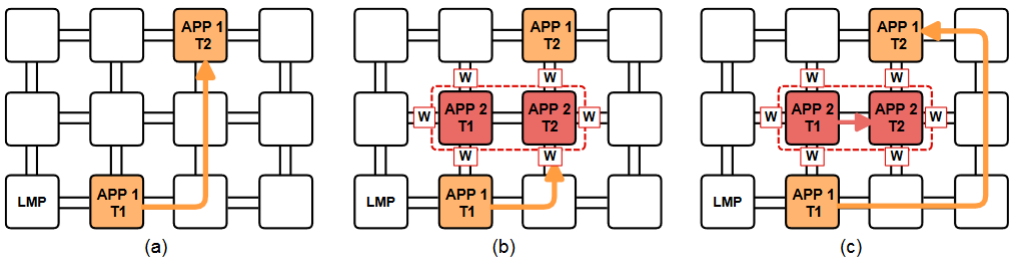


Figura 2 – Definição de novo caminho de mensagem. Fonte: [Caimi, 2019]

Todas as mensagens que atravessam a NoC de dados são marcados com uma *flag* para identificar se o pacote se refere a uma comunicação entre dois PEs, ou entre um PE e um

periférico. Usando essa *flag*, o *wrapper* consegue identificar quando uma mensagem que chega a fronteira da OSZ é destinada a um periférico, então a mensagem é criptografada usando uma chave gerada na inclusão do periférico na plataforma, e os *wrappers* das portas de saída e entrada da OSZ são abertos nos pontos específicos pelo tempo necessário para a transmissão da mensagem. A porta por onde a mensagem do periférico deve retornar é aberta pelo tempo necessário para a entrada da mensagem, que também retorna criptografada. Além disso, para entrada da mensagem de retorno, os *IDs* de origem e destinos são verificados para garantir que a mensagem realmente é vinda do periférico e tem como destino o PE que aguarda a mensagem, caso contrário o pacote é descartado.

Cada PE é controlado por meio de um Sistema Operacional (SO). Baseado na configuração da plataforma na fase de projeto. Os SOs de cada PE assumem papéis diferentes, podendo ser PEs Gerenciadores (MPE - em inglês *Manager Processor Element*) ou PEs Escravos (SPE - em inglês *Slave Processor Element*). Um MPE pode ser um Gerenciador Global (GMP - em inglês *Global Manager Processor*) ou um Gerenciador Local (LMP - em inglês *Local Manager Processor*); em ambos os casos, o PE não executa tarefas das aplicações, sendo seu propósito apenas gerenciar a plataforma, definindo em que SPEs executará cada uma das tarefas das aplicações, formatos e posições das regiões alocadas para as OSZs, mapeando as tarefas nos SPEs, fazendo monitoramento, migração de tarefas quando necessário, gerenciamento das chaves de autenticação dos elementos conectados a plataforma, autenticação dos periféricos e transmissão das tarefas aos SPEs. O GMP é o gerenciador da plataforma enquanto o LMP é o gerenciador do cluster, quando existente. Já um SPE executa as tarefas das aplicações, dessa forma seus recursos de processamento e também sua memória são compartilhados entre as tarefas destinadas a executar nele e algumas tarefas de sistema, como tratamento de interrupções, comunicação com outros PEs quando necessário, fechamento das bordas das OSZ conforme sejam solicitadas pelos MPEs, limpeza e liberação de memória após liberação de recursos da OSZ.

As aplicações executadas na plataforma são inseridas a partir de um injetor de aplicações (AppInj - em inglês *Application Injector*) inserido como periférico, posicionado e identificado em fase de projeto e autenticado na inicialização da plataforma. Cada aplicação é especificada por um descritor e um grafo (Figura 3), em que cada tarefa da aplicação e periférico sejam representados por um vértice. Cada comunicação entre duas tarefas ou entre uma tarefa e um periférico é representada por uma aresta direcionada. Além disso, a aplicação em seu descritor contém uma *flag* para informar se ela deve ser executada de forma segura ou não.

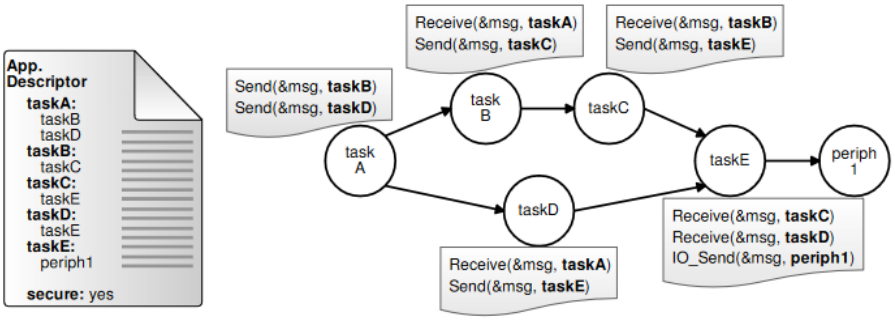


Figura 3 – Descritor e grafo das aplicações. Fonte: [Caimi, 2019]

As mensagens de comunicação são diferenciadas pela API e marcadas como mensagem entre tarefas ou mensagem entre uma tarefa e um periférico. Mensagens entre tarefas utilizam as primitivas *Send*() e *Receive*(), em que a *Send*() não é bloqueante, ou seja, ao passar pela instrução a API armazena o dado em memória e a tarefa segue sua execução normalmente, enquanto a *Receive*() é bloqueante, ou seja, ao passar pela instrução a API paralisa a execução da tarefa até que o dado seja recebido. As mensagens a serem enviadas são armazenadas em uma área de memória do PE, junto às informações de origem e destino da mensagem e consumidas na ordem em que são produzidas. O SO comunica-se com a NoC de dados utilizando pacotes de *data_request* quando encontra alguma instrução *Receive*() na memória de comunicação, e pacote de *data_delivery* após uma comunicação ser solicitada e encontrar um *Send*() na memória do PE à quem foi solicitado pelo *Receive*(). Dessa forma, apenas trafegam pacotes na rede quando alguma tarefa estiver solicitando um dado.

Para comunicação com periféricos de entrada e saída de mensagens (I/O - em inglês *Input/Output*), outra API fornece as primitivas *IO_Receive*() e *IO_Send*(), usando o formato mestre e escravo. Nesse caso, o mestre da comunicação é sempre o PE executando a tarefa e o escravo é sempre o periférico. O SO realiza a comunicação com a NoC de dados usando pacotes *IO_Request* (para solicitar o dado ao periférico) e *IO_Delivery* (para resposta do periférico) quando um *IO_Receive*() é encontrado na memória, e usando pacotes *IO_Delivery* (para enviar o dado ao periférico) e *IO_Ack* (para confirmação de recebimento) quando a instrução *IO_Send*() é encontrada na memória.

Uma aplicação é composta por um conjunto de tarefas, as quais correspondem a um conjunto de trechos de código com objetivos específicos, possuindo dependências e se comunicando entre si. Dessa forma, quando uma aplicação é executada na plataforma, os PEs são alocados para execução das tarefas, podendo ser distribuídas uma ou mais tarefas por PE. Quando a aplicação é executada em modo de segurança são verificadas, através do descritor da aplicação, quantas tarefas podem ser executadas por PE, para definir o tamanho e o formato da OSZ baseado no número de tarefas da aplicação; e posteriormente as tarefas são mapeadas na região da OSZ. O mapeamento das tarefas dentro de uma OSZ pode seguir várias abordagens, como a otimização de recursos de comunicação, deixando lado a lado tarefas que tenham maior latência de comunicação entre si, a distribuição das tarefas baseado na carga computacional, deixando

tarefas com menor carga em PEs mais depreciados, baseado na temperatura, colocando tarefas com maior carga computacional nos PEs menos aquecidos, ou apenas mapeando as tarefas sequencialmente conforme aparecem no descritor.

1.2 PROBLEMATIZAÇÃO

Devido a criação de OSZ na plataforma e ao acesso a periféricos na execução de aplicações seguras, alguns problemas podem ser observados. Um destes problemas é a possibilidade de uma OSZ ser criada em uma região na borda da plataforma onde exista algum periférico conectado. Por exemplo, na Figura 4, os pacotes de dados não pertencentes a aplicação mapeada na *Secure Zone 4* então não podem atravessar sua fronteira, assim, o *Peripheral 2* ficará inacessível a todos os PEs que não pertençam à *Secure Zone 4*.

Outro problema ocorre ao criar OSZs em PEs adjacentes: eventualmente uma ou mais OSZs podem criar uma barreira isolando PEs de um periférico. Por exemplo, na Figura 4, os PEs no canto superior direito do MPSoC e os PEs da *Secure Zone 1* não tem acesso ao *Peripheral 1* pois o posicionamento das OSZs criou uma barreira impedindo os dados de trafegar até o periférico. Estes problemas serão referidos posteriormente no texto como problema de bloqueio de periféricos.

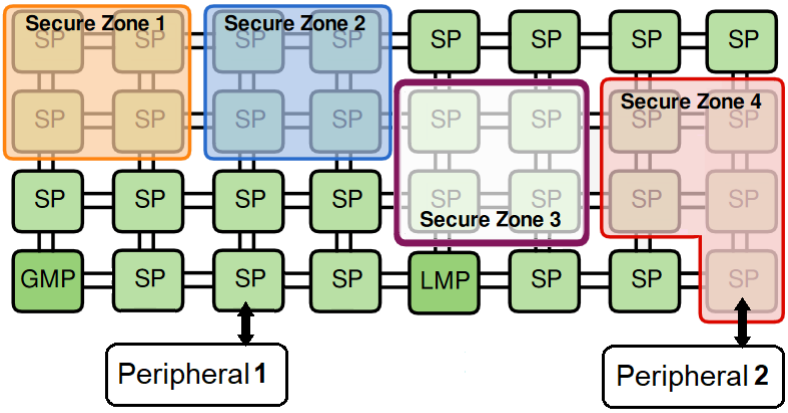


Figura 4 – Problema de bloqueio de periféricos. Fonte: adaptado de [Caimi, 2019]

Outro problema está relacionado à posição em que são mapeadas as tarefas, as quais possuem comunicação com periféricos, dentro da OSZ. Dado que o algoritmo de roteamento é o XY, o mapeamento das tarefas segue a sequência em que as tarefas aparecem no descritor, tendo em vista que as comunicações com os periféricos ocorrem em uma porta para saída e outra porta distinta para retorno da mensagem, sempre usando roteamento XY, e os *wrappers* abrem a porta apenas no período em que a mensagem está sendo trafegada por ela, duas tarefas que se comunicam com periféricos não podem estar alinhadas na linha e nem na coluna de entrada ou saída, pois caso necessitem acesso a algum periférico simultaneamente ocorrerá um conflito e consequentemente uma das tarefas não poderá trafegar sua mensagem através da

fronteira da OSZ, já que a passagem da primeira mensagem faz com o que o *wrapper* bloqueie a comunicação. Por exemplo, na Figura 5, caso as tarefas A e B mapeadas respectivamente nos PEs nas posições (1,2) e (1,1) tentem acessar o *Peripheral* simultaneamente, apenas uma das mensagens de retorno do *Peripheral* será capaz de atravessar a fronteira da OSZ na porta norte do PE na posição (1,2). Este problema será referido posteriormente no texto como problema de sombreamento.

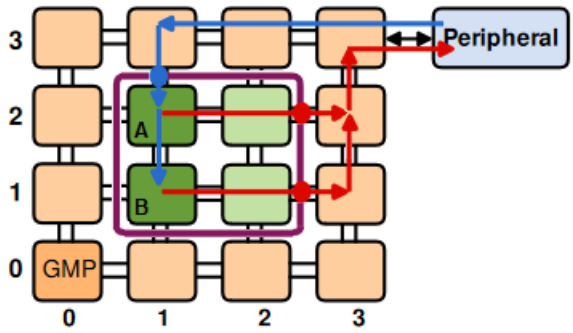


Figura 5 – Problema de sombreamento. Fonte: adaptado de [Caimi, 2019]

1.3 OBJETIVOS

1.3.1 Objetivo geral

O presente trabalho tem como objetivo geral implementar um algoritmo para definir o formato e realizar o posicionamento de OSZs e implementar um algoritmo de mapeamento de tarefas na OSZ de forma que todo periférico conectado à plataforma esteja acessível para qualquer tarefa.

1.3.2 Objetivos específicos

Como objetivos específicos, apresenta-se os seguintes itens:

- Implementar analisador de código fonte para encontrar os periféricos com os quais existe comunicação em cada uma das tarefas de uma aplicação;
- Definir uma estrutura de dados identificando as tarefas da aplicação que comunicam com periféricos durante a criação da plataforma e disponibilizar para *kernel* para utilização nos algoritmos de posicionamento da OSZ e mapeamento das tarefas;
- Definir e disponibilizar uma estrutura de dados para o *kernel* com a posição dos periféricos no MPSoC com vistas posicionamento da OSZ e mapeamento das tarefas;
- Propor algoritmos de definição de formato e posicionamento da OSZ no MPSoC para eliminar o problema de bloqueio de periféricos;

- Propor um algoritmo de mapeamento de tarefas dentro das OSZs para eliminar o problema de sombreamento;
- Criação de casos de teste para validar e obter resultados numéricos sobre as implementações.

1.4 JUSTIFICATIVA

Tendo em vista que a execução segura é primordial para algumas aplicações, a não resolução dos problemas descritos no tópico anterior implica em um possível travamento ou falha na execução de aplicações seguras, principalmente quando a plataforma for de dimensões pequenas ou estejam sendo executadas várias aplicações seguras, o que tende a ser uma realidade baseado no crescente uso desse tipo de sistema. Além disso, a resolução destes problemas deve trazer mais confiabilidade para a plataforma, possibilitando sua difusão, por se tornar mais robusta e confiável.

1.5 METODOLOGIA

Inicialmente instalou-se as ferramentas e dependências necessárias para execução da plataforma. Posteriormente realizou-se a ambientação da mesma e estudo dos algoritmos utilizados atualmente para definição de formato de OSZ, posicionamento da OSZ e mapeamento das tarefas na área da OSZ.

A partir da uma revisão da literatura, foram estudados algoritmos e técnicas para definir o formato e o local onde a OSZ deve ser criada sem que bloqueie o acesso periférico de qualquer outra tarefa. Para tanto, foi necessário extrair dos códigos fonte das aplicações as informações de quais tarefas se comunicam com algum periférico e extrair do arquivo de descrição quais as bordas do MPSoC contam com periféricos conectados. Assim, implementou-se uma solução que atenda a demanda do não bloqueio no acesso a periféricos. Na prática, corresponde a uma solução que garanta que qualquer tarefa executada dentro de uma OSZ tenha um caminho de comunicação com um periférico, tanto para o tráfego de entrada de dados como para o tráfego de saída de dados, não permitindo a criação de OSZs adjacentes ou sobrepondo um elemento de processamento na borda do MPSoc em que e encontre um periférico conectado.

Através dos algoritmos usados para definição do formato e tamanho da OSZ e dos algoritmos de mapeamento, estudou-se de que forma era feito o mapeamento das tarefas dentro da zona segura.

Foi proposta uma solução que resolva o problema de alinhamento de portas de acesso a periféricos através do estudo de técnicas de mapeamento e implementada uma solução que distribua as tarefas dentro da OSZ sem que tarefas que têm acesso a periféricos fiquem alinhadas na mesma linha ou coluna. Essa solução avalia também o melhor formato para a OSZ, dado que essa informação se torna primordial para que a condição de não alinhar as tarefas seja atendida.

Com as soluções implementadas, foi realizada a validação das soluções, executando um conjunto de casos de testes de aplicações seguras na plataforma e verificando seu funcionamento para após avaliar os métodos quanto ao desempenho e disponibilidade de recursos.

2 REVISÃO BIBLIOGRÁFICA

O presente capítulo abordará termos e conceitos, pontuando características importantes para a área de estudo. Iniciando com sistemas multiprocessados, abordando em relação às características das redes intra-chip, à plataforma de base deste trabalho, às questões referentes à segurança, ao uso de Zonas seguras Opacas, à busca por janelas deslizantes e aos algoritmos de mapeamento de tarefas.

2.1 SISTEMAS MULTIPROCESSADOS EM CHIP (MPSOC)

Levando em consideração um cenário geral onde parâmetros como custo, tempo de projeto, desempenho e confiabilidade são delimitantes, os sistemas tendem a ser implantados na forma de um único chip, originando os chamados SoC. Os SoC são construídos como um agrupamento de componentes heterogêneos, incluindo memórias e aceleradores, por exemplo. Porém, considerando uma evolução, complexidade e especialização de um sistema, se faz necessário o uso de um sistema que emprega mais de um elemento de processamento, assim, existem os Sistemas Multiprocessados em Chip (MPSoCs) (FILHO, 2011).

Os Sistemas Multiprocessados em Chip (MPSoCs), com base no seu poder computacional, são relacionados como provável padrão para implantar sistemas futuros. Uma característica desses sistemas está em suas aplicações, que são realizadas a partir de distintas tarefas comunicantes (CARARA, 2011).

Com isso, reconhecemos um MPSoCs como um sistema integrado em um único chip (SoC), ou seja, possui em sua composição vários elementos de processamento que permitem a execução de múltiplas tarefas em paralelo (VIDAL, 2018).

2.2 REDES INTRA-CHIP (NOCS)

NoC refere-se a um modelo de comunicação que é baseado em conceitos de redes de computadores. Os NoCs podem ser aplicados em sistemas multiprocessados em chip, a fim de substituir a comunicação por barramentos por uma comunicação baseada em pacotes (MEDINA, 2019).

A reutilização, a escalabilidade e o paralelismo são alguns dos fatores que amparam o uso desse tipo de comunicação no lugar do barramento. Como o uso dos MPSoCs possuem um amplo número de núcleos internos, é necessário haver comunicação entre eles, por isso a utilização da comunicação NoC se sobrepõe quando comparada a comunicação ponto a ponto e barramento (SILVA OLIVEIRA; KREUTZ, 2019).

2.3 HEMPS

Hermes Multi-Processor System (HeMPS) é um MPSoC homogêneo baseado no processador Plasma e na NoC Hermes, em que os elementos de processamento são interconectados pela infraestrutura de conexão NoC Hermes. O Processador Plasma RISC de 32 bits oferece suporte à linguagem C (CARARA, 2011).

A plataforma de base para este trabalho é desenvolvida pelo Grupo de Apoio ao Projeto de Hardware (GAPH) da PUC-RS, a qual baseia-se no HeMPS (CAIMI, 2019) e está disponível na página do grupo de pesquisa no Github, em: https://github.com/gaph-pucrs/hemps_OSZ/tree/IO.

2.4 SEGURANÇA

Quando um sistema é utilizado, suas inúmeras funcionalidades incluem o processamento e armazenamento de dados. Com isso, podem ocorrer acessos a informações particulares, como dados de usuários e dados de empresas. Assim, surge a segurança dos sistemas, uma temática de extrema importância que visa a contenção e aplicação de recursos para prevenir vazamento de informações (DIORIO et al., 2018).

Existem princípios de segurança que são aceitos como base para uma boa solução de segurança em um determinado sistema, sendo que seguem as seguintes premissas e características: (RAMACHANDRAN, 2002)

- Autenticação, que é o estabelecer da validade de uma identidade reivindicada;
- Autorização, que delimita se uma entidade válida poderá ter o acesso a um recurso seguro;
- Integridade, sendo a prevenção da modificação ou destruição de um recurso ou informação por um não autorizado;
- Disponibilidade, que é a proteção de recurso/informações contra ameaças que possam impactar na disponibilidade do sistema;
- Confiabilidade, que baseia-se na propriedade de não divulgação de informações para não autorizados;
- Auditoria, que é o registro de todas as atividades do sistema, as quais devem ser suficientes para reconstrução de eventos; e,
- Não-repúdio, em que há o impedimento de qualquer participante em uma comunicação ou transação, negando a função na interação, desde que esteja completa (SANT'ANA, 2019).

2.4.1 Zonas Seguras Opacas (OSZ)

Para Sistemas Multiprocessados em Chip, a literatura apresenta diversas possibilidades de mecanismos para proteger a comunicação, dentre eles o uso de Zonas Seguras. Uma Zona Segura usa técnicas como criptografia, esquemas de roteamento ou isolamento lógico para isolar a execução nos PEs da troca de dados (CAIMI; FOCHI; WACHTER; MORAES, 2018).

Conforme descrito por Ruaro, Caimi e Moraes (2020), o método *Opaque Secure Zones* (OSZ), possibilita isolamento temporal e espacial, impedindo o compartilhamento de recursos da comunicação e computação. O isolamento das OSZs se dá por meio de *wrappers*, que são ativos e impedem o tráfego de pacotes na região. Dessa forma, quando um pacote é recebido na fronteira da OSZ, o *wrapper* já fechado impede a passagem, descartando o material e notificando seu descarte através da NoC de controle ao PE de origem do pacote. Através da NoC de controle, o pacote então encontra um novo caminho válido, que não atravesse a OSZ.

A implementação de OSZ exige o uso de mecanismos de roteamento auxiliares para redefinição de caminhos, a fim de evitar a criação de regiões inacessíveis, o que pode gerar um impacto no desempenho das aplicações (RUARO; CAIMI; MORAES, 2020).

2.5 BUSCA POR JANELA DESLIZANTE (SWS)

Conforme descrito por Caimi (2019), a busca pela posição da OSZ é feita através do algoritmo de busca por janela deslizante, (SWS - em inglês *Slide Window Search*). O algoritmo SWS tem seu funcionamento básico definido como a busca do formato desejado da OSZ na malha 2D no *cluster* do MPSoC. Dado o formato da OSZ e a posição do *cluster*, iniciando no canto superior direito do *cluster*, o algoritmo verifica no conjunto de PEs, da janela que tem o formato da OSZ, quantos recursos estão disponíveis. A janela é movida para a esquerda e posteriormente para a linha logo abaixo, repetindo o processo até que o maior número de recursos disponíveis seja encontrado, dentro de todas as possibilidades de posicionamento da janela no *cluster*.

2.6 ALGORITMOS DE MAPEAMENTO

O mapeamento das tarefas pode seguir diversas abordagens baseadas em alguns objetivos, os quais usam informações de tempo de projeto ou tempo de execução. Dentre os objetivos a serem otimizados com o mapeamento destacam-se aqui a melhora do uso dos recursos de comunicação, menor consumo de energia, evitar envelhecimento desigual dos recursos e evitar aquecimento de uma região de elementos de processamento (MARWEDEL et al., 2011); (ALI et al., 2022).

Kaushik et al. (2011) descreve o uso de algoritmos de mapeamento em tempo de execução, baseado no pré-processamento, através dos descritores da aplicação para diminuir os custos

de comunicação e consumo de energia. Nessa abordagem, as tarefas com comunicação são mapeadas no mesmo PE quando possível, e, caso contrário, são mapeadas em PEs adjacentes e conforme a sequência das comunicações definidas no descritor da aplicação.

Das, Kumar e Veeravalli (2014), descreve o uso de técnicas em tempo de projeto para definir o mapeamento de tarefas, com o objetivo de diminuir o envelhecimento dos elementos de processamento avaliando a tensão e temperatura do elemento e de seus vizinhos, bem como a dependência entre elementos de processamento e sua carga de comunicação.

3 DESENVOLVIMENTO

Neste capítulo, apresentamos o desenvolvimento de uma implementação como solução para os problemas de bloqueio de periféricos e de sombreamento no mapeamento de tarefas em Zonas Seguras Opacas.

Como mostra a Figura 6, a plataforma HeMPS é composta de um conjunto de diretórios, subdiretórios e arquivos organizados. Dentre os diretórios, destacam-se aqui: o diretório *applications* que armazena as aplicações que podem ser executadas nas plataforma; o diretório *build_env* e dentro dele outro diretório *scripts*, contendo os *scripts* que interpretam o descritor do caso de teste e geram todo o ambiente de simulação; o diretório *hardware* que contêm os arquivos de descrição e configuração de *hardware* da plataforma; o diretório *software* contendo mais dois diretórios, o *kernel*, onde ficam armazenados os códigos fonte de controle do *kernel* e sistema operacional e o *modules*, onde ficam armazenados os códigos fonte auxiliares para o controle do *kernel* e SO; e o diretório *testcases*, onde ficam os descritores de casos de testes e diretórios criados pelos construtores para execução de cada caso de teste.

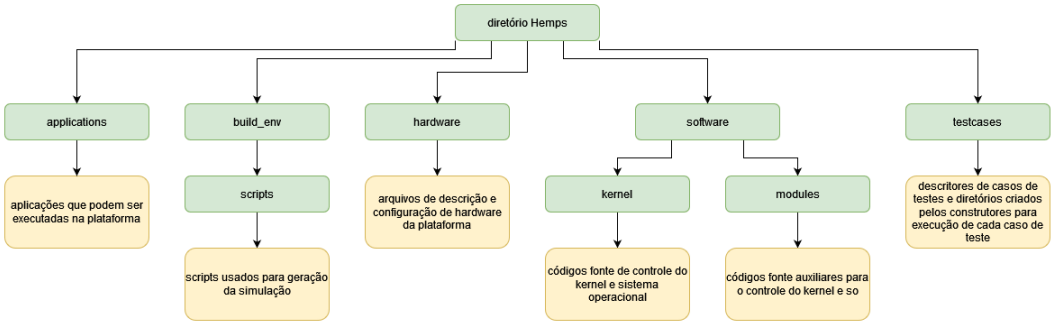


Figura 6 – Visão geral dos diretórios da plataforma HeMPS. [Fonte: elaborado pelo autor]

A plataforma, por se tratar de uma simulação, existe no contexto de uma descrição de *hardware* e uma descrição de *software*. Ela tem um arquivo de configuração inicial, onde são descritos parte do *hardware* e parte do *software*, que são lidos e interpretados por alguns *scripts* e geram a plataforma de forma dinâmica para que após seja simulada, conforme mostra a Figura 7. Este arquivo de descrição de parte do *hardware* e de parte do *software* é um arquivo de extensão ".yaml" e o mesmo deve ser criado na diretório *testcases* e passado como parâmetro no comando de execução da plataforma: *"hemps-run"*.

Após o comando *"hemps-run"*, o arquivo *"testcase_builder.py"* é executado, criando os diretórios do caso de testes no diretório *testcases*. A partir da interpretação do arquivo de descrição do caso de teste, são executados os arquivos *"hw_builder.py"*, *"kernel_builder.py"* e *"app_builder.py"*.

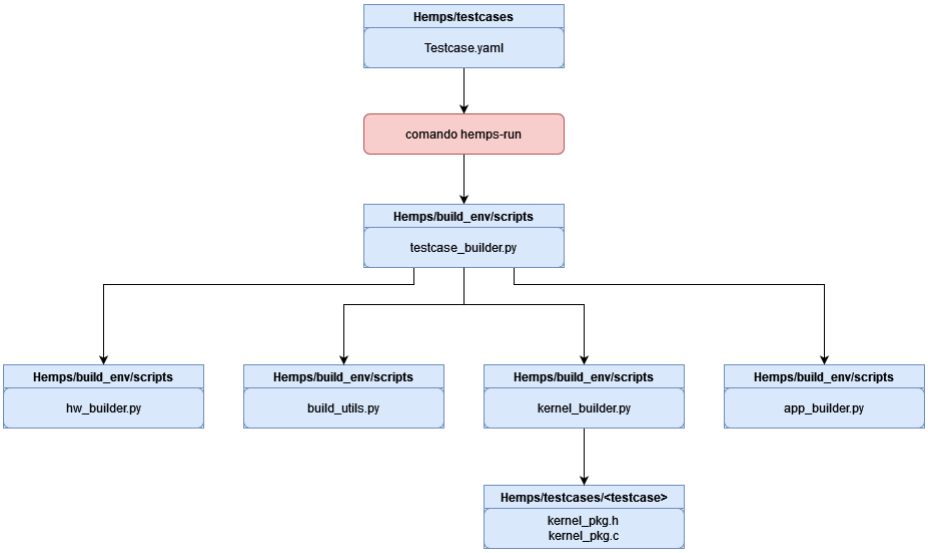


Figura 7 – Visão geral dos arquivos da plataforma HeMPS. [Fonte: elaborado pelo autor]

O arquivo *"hw_builder.py"* gera e compila dinamicamente todo o hardware que será simulado (VHDL e SystemC); o arquivo *"kernel_builder.py"* gera e compila dinamicamente todo o sistema operacional dos PEs que serão simulados (linguagem C), e o arquivo *"app_builder.py"* gera e compila as aplicações a serem executadas na simulação da plataforma (linguagem C).

O arquivo de descrição contém informações como as dimensões do MPSoC, dimensões dos *clusters*, modelo de descrição do hardware, nome, *id* e posições dos periféricos. Neste arquivo também são dadas informações de que aplicações serão executadas, o tempo em que devem ser inseridas na plataforma, se executarão de forma segura ou não, entre outras. Quando é informado que uma aplicação é segura, haverá a criação dinâmica de uma OSZ para a execução da aplicação. A Figura 8 traz um exemplo de arquivo de descrição.

```yaml
hw:
  page_size_KB:  64
  tasks_per_PE: 1
  repository_size_MB:  1
  physical_channels: 2
  model_description: hybrid
  noc_buffer_size: 8
  mpsoc_dimension: [4,4]
  cluster_dimension: [4,4]
  master_location: LB
  open_port:
    - port: [Injector,90,0,0,S,1,0,S]
    - port: [io_peripheral,80,3,0,S]
    - port: [io_peripheral2,70,0,2,W]
apps:
  - name: dtw_IO
    start_time_ms: 0
    secure: yes
    shape: rectangular
  - name: dtw_IO
    start_time_ms: 0
    secure: yes
    shape: rectangular
```

Figura 8 – Exemplo de arquivo de descrição ".yaml".

Na fase de simulação da plataforma, são executados conjuntos de códigos do diretório *software*, que controlam o funcionamento do sistema operacional, onde cabe ao *kernel* controlar o que está acontecendo em cada um dos elementos de processamento e os PEs, conjuntamente, fazem com que a plataforma execute a simulação.

3.1 ALTERAÇÕES NO PRÉ-PROCESSAMENTO DA PLATAFORMA DE SIMULAÇÃO

Para a implementação dos algoritmos de tamanho, formato e posicionamento da OSZ e de mapeamento de tarefas na OSZ, a fim de evitar os problemas de bloqueio de periféricos e de sombreamento, se faz necessário extrair as seguintes informações:

a) Comunicação com periféricos para cada tarefa de cada aplicação a ser executada;

b) Formato da OSZ para cada aplicação a ser executada;

c) Bordas do MPSoC com periféricos conectados.

Dessa forma, foram criadas estruturas de dados para armazenar estas informações. Foi alterado o arquivo *"kernel_builder.py"*(Figura 7), responsável por ler o arquivo de descrição do caso de teste passado como parâmetro na execução da simulação, e, a partir dele, gerar os arquivos *"kernel_pkg.h"* e *"kernel_pkg.c"*, onde são armazenadas as estruturas de dados com as informações como dimensões do MPSoC, dimensões e posição dos *clusters*, número de aplicações as serem executadas, número de tarefas por aplicação, número de periféricos conectados, *ids* e posições de cada periférico, entre outras.

O arquivo *"kernel_buinder.py"* (Figura 7) foi alterado para que, ao ler o arquivo de descrição do caso de teste e criar o arquivo *"kernel_pkg.h"*, adicione a este quatro novas estruturas de dados:

a) uma estrutura contendo um *id* de aplicação, um *id* de tarefa e um *array* de periféricos (linhas 54 à 58 na Figura 9.a);

b) um *array* dessa estrutura para que possa ser criada uma estrutura para cada tarefa, de cada aplicação (linha 60 na Figura 9.a);

c) um *array* para o formato de OSZ para cada aplicação (linha 63 na Figura 9.a);

d) um *array* de quatro posições para sinalizar quais bordas do MPSoC possuem periféricos conectados (linha 66 na Figura 9.a).

O arquivo *"kernel_buinder.py"* (Figura 7) também foi alterado. Desse modo, ao ler o arquivo de descrição do caso de teste e criar o arquivo *"kernel_pkg.c"*, são preenchidas as estruturas criadas baseado na análise de código fonte para obter as tarefas e suas respectivas comunicações com periféricos, e nas informações de formato da OSZ e posições dos periféricos conectados presentes no arquivo de descrição do caso de teste (Figura 9.b).

```c
//This structure stores information about application tasks and their
//communication with peripherals
typedef struct {
    int app_id;                     //ID of the application
    int task_id;                    //ID of the task
    int peripheral_id[IO_NUMBER];   //Array of periferal IDs
} Task_Comm_IO_Info;

extern Task_Comm_IO_Info task_comm_io_info[(APP_NUMBER * MAX_TASKS_APP)];

//This array stores whether the safe zone is rectangular(1) or rectilinear(0)
int OSZ_shape[APP_NUMBER];    //rectangular(1) or rectilinear(0)

//This array stores edges of the MPSOC Who have PEs communicating with IO
int PEs_With_IO[4];    //0-S, 1-W, 2-E, 3-N
```

(a)

```c
Task_Comm_IO_Info task_comm_io_info[APP_NUMBER * MAX_TASKS_APP] = {
    {0, 0, {-1, -1, -1}},
    {0, 1, {-1, -1, -1}},
    {0, 2, {-1, -1, -1}},
    {0, 3, {80, -1, -1}},
    {0, 4, {70, -1, -1}},
    {-1, -1, {-1, -1, -1}},
    {1, 256, {70, -1, -1}},
    {1, 257, {80, -1, -1}},
    {1, 258, {-1, -1, -1}},
    {1, 259, {-1, -1, -1}},
    {1, 260, {-1, -1, -1}},
    {1, 261, {-1, -1, -1}},
};

int OSZ_shape[APP_NUMBER] = {
    0, 0
};

int PEs_With_IO[4] = {
    1, 0, 1, 1
};
```

(b)

Figura 9 – Estruturas de dados gerados para um caso de testes, *"kernel_pkg.h"* e *"kernel_pkg.c"*. [Fonte: elaborado pelo autor]

Essas estruturas de dados devem ser acessadas pelo *kernel* na fase de definição de tamanho e posições das OSZs, bem como na fase de mapeamento das tarefas na OSZ.

3.2 ANALISADOR DE CÓDIGO FONTE DAS APLICAÇÕES

Para extrair as informações de comunicação com os periféricos de cada uma das tarefas, dos códigos fonte das aplicações a serem executadas na plataforma, foi alterado o arquivo *"build_utils.py"* do diretório *build_env/scripts* (Figura 7). A função de leitura de arquivos foi adicionada a ele, além da remoção de comentários e busca de primitivas *"IOSend"* e *"IOReceive"* nos códigos fonte.

A informação de quais tarefas se comunicam com periféricos é importante. Essa compreensão é utilizada na fase de mapeamento das tarefas da aplicação na OSZ, para evitar que duas ou mais tarefas que se comunicam com periféricos sejam mapeadas na mesma coluna ou na mesma linha, evitando o problema de sombreamento. A Figura 10 apresenta o trecho de código adicionado ao arquivo *"build_utils.py"* para extração dessa informação.

Nas linhas 3 a 5, da Figura 10, é aberto o arquivo de código fonte da tarefa, baseado no caminho do diretório do caso de teste, nos nomes da aplicação e da tarefa. Nas linhas 6 a 18, todas as linhas do código fonte são lidas e após a remoção dos comentários na linha 7, é verificado nas linhas 9, 11 e 14 se as primitivas *"IOSend"* e *"IOReceive"* estão presentes, e caso verdadeiro, nas linhas 12, 13, 15 e 16, a primitiva é quebrada extraindo o periférico ao qual se refere para todas as primitivas encontradas na linha. Na linha 17 é verificado se o periférico encontrado não pertence à lista de periféricos, e, caso não pertença, é incluído na linha 18. Após a leitura e interpretação de todas as linhas do código fonte, o arquivo é fechado na linha 19 e é retornada a lista de periféricos com os quais a tarefa se comunica na linha 20.

```
def find_IO_comm_for_each_task(testcase_path, app_name, task_name):
    peripherals_comm = []
    source_file = testcase_path + "/applications/" + app_name + "/" + task_name +".c"
    f = open(source_file)
    source_lines = f.readlines()
    for line in source_lines :
        remove_comments(line)
        line = str.upper(line)
        while ("IOSEND" in line) or ("IORECEIVE" in line):
            line = line.replace(" ","")
            if ("IOSEND" in line):
                peripheral = line.split("IOSEND",1)[1].split(",",1)[1].split(");",1)[0]
                line = line.split("IOSEND",1)[0] + line.split("IOSEND",1)[1].split(");",1)[1]
            else:
                peripheral = line.split("IORECEIVE",1)[1].split(",",1)[1].split(");",1)[0]
                line = line.split("IORECEIVE",1)[0] + line.split("IORECEIVE",1)[1].split(");",1)[1]
            if peripheral not in peripherals_comm:
                peripherals_comm.append(peripheral)
    f.close()
    return peripherals_comm

```

Figura 10 – Analisador de código fonte. [Fonte: elaborado pelo autor]

3.3 DEFINIÇÃO DE FORMATO PARA OSZ

Quando uma aplicação segura é executada na plataforma, recursos de processamento ou de comunicação não são compartilhados. Isto é feito através da criação de uma OSZ dinamicamente na plataforma. Para a criação da OSZ deve-se definir o seu tamanho, formato e a sua localização.

Quando uma OSZ possui um formato em que o número de PEs é maior que o número de PEs necessário para executar a aplicação, alguns PEs excedentes podem ser removidos, sendo retirados ordenadamente iniciando no canto inferior esquerdo em direção ao canto superior esquerdo da OSZ e não excedendo o número de elementos da coluna, dessa forma, a OSZ passa a ser retilínea e não mais retangular (Figura 11).

Os formatos retangular e retilíneo são possíveis, porém, verificou-se que quando uma OSZ retilínea é criada, as comunicações com periféricos que tentam atravessar a parte cortada da OSZ retilínea acabam em erro, pois o sistema tenta fazer a abertura dos *wrappers* de forma retangular. Para evitar a necessidade de alterações em trechos de código que não tratam especificamente dos problemas aqui abordados e para dar uma opção de escolha do formato da OSZ ao descritor da aplicação, foi criado um parâmetro que pode ser adicionado à descrição da aplicação no arquivo de descrição de caso de teste. O parâmetro é *"shape:"* e pode assumir dois valores: a) *rectangular*, que define que a OSZ deve ser criada em formato retangular e b) *rectilinear*, que define o uso de formato retilíneo quando possível e este é o padrão assumido caso não seja inserido o parâmetro ou caso o parâmetro seja criado com algum valor diferente destas opções.

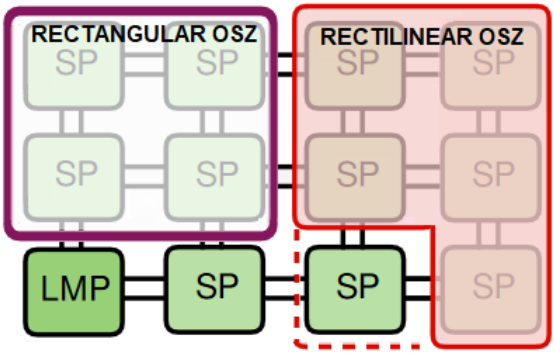


Figura 11 – OSZ Retangular e Retilínea. [Fonte: elaborado pelo autor]

Para criar este parâmetro, foram adicionados ao aquivo *"build_utils.py"* a classe *ApplicationStartTime* e o parâmetro *shape*. Ainda, foi alterada a função *get_app_start_time_list* do arquivo *"yaml_init.py"*, que retorna a lista de aplicações descritas no arquivo de descrição de caso de teste, ordenadas pelo parâmetro *start_time_ms*. A leitura do arquivo é realizada para saber se o parâmetro existe no descritor e se ele é igual a *rectangular*, caso contrário, é definido como *rectilinear*. Após coletada a informação ela é inserida na estrutura de dados *OSZ_shape*, criada no arquivo *"kernel_pkg.h"*(Figura 9).

Já na fase de alocação de recursos para as aplicações, quando é feita a busca pelos formatos válidos para a OSZ na função *create_valid_shapes* no arquivo *"osz_master.c"*, caso o parâmetro *OSZ_shape* seja *rectangular* para a aplicação em questão, o excesso calculado para o formato da OZS será igual a zero. Caso o parâmetro *OSZ_shape* seja diferente de *rectangular*, será definido como ((*X_size* * *Y_size*) - *QPEs*), onde *X_size* e *Y_size*, representam as dimensões da OSZ em X e Y respectivamente e *QPEs* representa a quantidade de PEs necessários para executar a aplicação, para assim posteriormente fazer o corte dessa região caso este formato seja selecionado.

3.4 DEFINIÇÃO DE TAMANHO DA OSZ

Para definição dos tamanhos adequados de OSZ, para que as tarefas possam ser mapeadas sem que ocorra o problema de sombreamento, foi adicionada uma função *cont_tasks_comm*, no arquivo *"kernel_master.c"* que recebe o *id* e a quantidade de tarefas de uma aplicação e retorna a quantidade de tarefas desta aplicação que se comunicam com algum periférico, fazendo uma busca na estrutura *task_comm_io_info* do *"kernel_pkg.h"*. Na função *handle_new_app* a função *cont_tasks_comm* é chamada, e a contagem de tarefas que se comunicam com algum periférico é passada na chamada da função *create_shapes* do arquivo *"osz_master.c"*, responsável por definir o número de PEs necessários para criar a OSZ.

Na função *create_shapes*(Figura 12), a alteração realizada garante nas linhas 97 e 98 da Figura 12 que o número de PEs necessários para criação da OSZ seja maior ou igual a *(tasks_with_comm²)* para formato de OSZ retangular ou maior ou igual a *((tasks_with_comm²)*

- *(tasks_with_comm-1))* (linhas 94 e 95) para formato retilíneo de OSZ, onde *tasks_with_comm* representa o número de tarefas que se comunicam com periféricos. Em seguida, são verificados através da função *create_valid_shapes*, também no arquivo *"osz_master.c"*, quais os formatos válidos dado o número de PEs necessários calculado na função anterior. A alteração feita da conta de invalidar formatos que tenham alguma dimensão menor que o número de tarefas que se comunicam com periféricos. Ou seja, dada uma aplicação com x tarefas, sendo que 4 se comunicam com periféricos, a função *create_shapes* define que serão necessários pelo menos 16 PEs para formato retangular e pelo menos 13 PEs para formato retilíneo da OSZ, e a função *create_valid_shapes* com base na quantidade de PEs necessários calcula as possíveis combinações de formatos e valida apenas os formatos cujas dimensões sejam maiores ou iguais a 4x4, em X e em Y.

```
85  int create_shapes(int task_pe, int tasks_app, int app_id, int tasks_with_comm){
86      int nb_shapes = 0, t, nb_pe;
87      int PEs_remove = 0;
88      for(t=1; t <= task_pe; t++ ){
89          nb_pe = tasks_app/t ;
90
91          if( nb_pe < tasks_app ) nb_pe++;   // ceil
92          if (OSZ_shape[app_id] == 0){
93              PEs_remove = (tasks_with_comm-1);
94          }
95          if (nb_pe < ((tasks_with_comm * tasks_with_comm)-PEs_remove)){
96              nb_pe = ((tasks_with_comm * tasks_with_comm)-PEs_remove);
97          }
98          nb_shapes =  create_valid_shapes(nb_pe, nb_shapes, app_id, tasks_with_comm);
99      }
100     return nb_shapes;
101 }
```

Figura 12 – Alteração na função *create_shapes*. [Fonte: elaborado pelo autor]

3.5 POSICIONAMENTO DA OSZ

Para evitar os problemas de bloqueio de periféricos, foram alteradas as funções que posicionam as zonas seguras dentro da plataforma. Depois de calculados os possíveis formatos das OSZ, verifica-se nos *clusters* se será possível alocar o formato calculado, usando o algoritmo SWS - *Slide Window Search*, para cada formato até encontrar um formato que possa ser alocado nos recursos disponíveis. Para isso é passado um conjunto de posições que formam o formato da OSZ à uma função do arquivo *"osz_master.c"* chamada *"shape_recog"* que verifica que alguma daquelas posições pode ser ocupada. A função *"shape_recog"* incrementa X e Y no intervalo recebido, verificando se a posição é de um gerenciador ou se pertence a uma OSZ, caso nenhuma posição pertença a nenhum dos casos, este posicionamento de OSZ é validado.

Na função *"shape_recog"* foram adicionadas mais duas verificações: a verificação se a posição pertence a área de PEs adjacentes a OSZ (Figura 13, pela função *PE_belong_SZ_shadow*) e a verificação se a posição está em alguma borda do MPSoC a qual tenha algum periférico conectado(Figura 13, pela função *PE_belong_IO_shadow*). Caso nenhuma das verificações seja

verdadeira esse posicionamento de OSZ é válido.

```
537  int PE_belong_SZ_shadow(int PE_x, int PE_y){
538      int i;
539      int xi, yi, xf, yf;
540
541      for(i = 0; i < MAX_SHAPES; i++){
542          if(Secure_Zone[i].occuped == 1){
543              xi =  ((Secure_Zone[i].position >> 8) & 0XFF)-1;
544              yi =   (Secure_Zone[i].position  & 0XFF)-1;
545              xf =  xi + Secure_Zone[i].X_size+1;
546              yf =  yi + Secure_Zone[i].Y_size+1;
547
548              if(((PE_x >= xi) && (PE_x < xf)) && ((PE_y >= yi) && (PE_y < yf))  )
549                  return 1;
550          }
551      }
552      return 0;
553  }
554  int PE_belong_IO_shadow(int PE_x, int PE_y){
555      int i;
556      int xi, yi, xf, yf;
557
558      if(((PE_x == 0) && (PEs_With_IO[1] == 1))||((PE_x == (XDIMENSION-1))&&(PEs_With_IO[2] == 1))||
559         ((PE_y == 0) && (PEs_With_IO[0] == 1))||((PE_y == (YDIMENSION-1))&&(PEs_With_IO[3] == 1))){
560          return 1;
561      }else{
562          return 0;
563      }
564  }
```

Figura 13 – Funções incluídas para o posicionamento da OSZ. [Fonte: elaborado pelo autor]

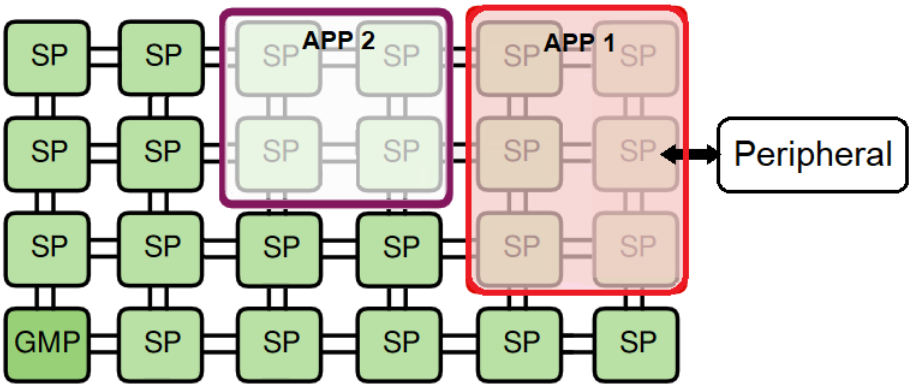


Figura 14 – Posicionamento da OSZ antes das alterações. [Fonte: elaborado pelo autor]

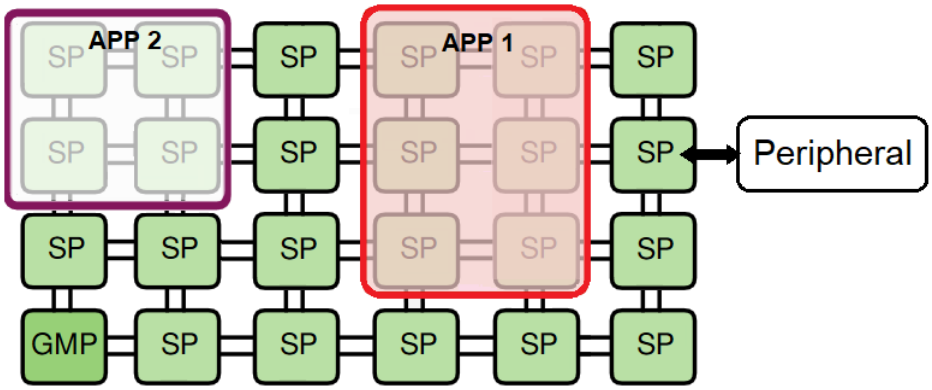


Figura 15 – Posicionamento da OSZ depois das alterações. [Fonte: elaborado pelo autor]

No posicionamento exibido na Figura 14, as alterações previstas no código da Figura 13 ainda não estavam implementadas, logo, a aplicação *APP 1* é posicionada na primeira posição disponível, no canto superior esquerdo. A aplicação *APP 2* é posicionada ao seu lado, pois os algoritmos de posicionamento consideravam apenas a posição de outra OSZ para que não fossem sobrepostas. Na Figura 14, o posicionamento das OSZs gera o problema de bloqueio de periféricos, caso a aplicação *APP 1* tenha alguma tarefa que se comunique com *Peripheral*.

No posicionamento da Figura 15, com as alterações previstas no código da Figura 13 já implementadas, a aplicação *APP 1* não pode ser posicionada na primeira posição devido a posição do *Peripheral*, já que a função *PE_belong_IO_shadow* define que nos PEs da borda onde está conectado um periférico, não pode ser posicionada uma OSZ. Ainda, a aplicação *APP 2* não pode ser posicionada exatamente ao lado da OSZ da *APP 1*, já que a função *PE_belong_SZ_shadow* define que nos PEs adjacentes à OSZ já posicionada não pode ser posicionada outra OSZ. Dessa forma, os problemas de bloqueio de periféricos são mitigados.

3.6 MAPEAMENTO DAS TAREFAS NA OSZ

Após definido o tamanho, formato e posicionamento da OSZ, é feito o mapeamento das tarefas da aplicação na área definida. A função responsável por esse mapeamento é a *"application_mapping"* que se encontra no arquivo *"cluster_scheduler.c"* do diretório *software*.

Anteriormente, as tarefas eram mapeadas sequencialmente nos PEs iniciando do menor X e menor Y, incrementando Y e, por fim, incrementando X. Após a alteração, as tarefas são divididas em dois *arrays*, sendo um *array* de tarefas com comunicação com periféricos e outro com as tarefas sem comunicação com periféricos. Esses *arrays* são separados com base na estrutura de dados *task_comm_io_info* do arquivo *"kernel_pkg.h"*, e a sequência de mapeamento também é alterada, sendo mapeadas primeiro as tarefas com comunicação com periféricos. As tarefas com comunicação com periféricos são mapeadas na diagonal da OSZ, iniciando no canto superior esquerdo e senguingo em direção ao canto inferior direito, usando a função *map_task_with_IO*. As aplicações são mapeadas na diagonal para evitar que tarefas

que se comuniquem com periféricos fiquem alinhadas na mesma linha ou coluna. Após o mapeamento das tarefas com comunicação com periféricos, as tarefas sem comunicação com periféricos são mapeadas da mesma forma que ocorria antes, usando a função *map_task*.

Depois de concluir o mapeamento das tarefas, a OSZ é efetivamente fechada e o *kernel* envia o mapa ao injetor de aplicações que transfere os códigos fonte para cada PE. A partir de então, a aplicação está apta a ser executada.

4 RESULTADOS E CONCLUSÃO

Para avaliar a efetividade dos algoritmos implementados na solução dos problemas existentes e medir o impacto destes algoritmos, foram criados sete cenários de testes com três aplicações para avaliar os tempos de definição dos formatos das OSZs e os tempos de mapeamento das tarefas dentro da OSZ. Cada cenário foi executado duas vezes, sendo a primeira vez com as implementações desabilitadas e a segunda vez com as implementações habilitadas.

Os cenários utilizados foram definidos com o objetivo de mostrar os problemas presentes na plataforma antes das alterações propostas, a execução correta após as alterações e o impacto no tempo de execução devido as alterações realizadas. Todos os dados de configuração e resultados de todos os cenários estão disponíveis na Figura 30.

4.1 APLICAÇÃO *MPEG_IO*

4.1.1 Cenário 1

O Cenário 1 é a execução de uma aplicação *"mpeg_IO"* segura em um MPSoC de dimensões 4x4, com periféricos conectados nas bordas oeste, sul e leste. Nesse caso, a simulação executou até o fim de todas as aplicações apenas após as alterações propostas. Na execução antes das alterações propostas, a tarefa *start* tentou comunicação com o *peripheral2* conectado ao PE (1x0) e, ao tentar cruzar a fronteira da OSZ, os pacotes foram perdidos devido o problema de corte da OSZ dado pelo uso do formato retilíneo, descrito na seção seção 3.3.

Os tempos de busca por formato e posição da OSZ foram de 2607 ciclos de *clock* para as execuções sem as alterações e de 4191 ciclos de *clock* para as execuções após as alterações. Desse modo, calcula-se um aumento de 60,76% do tempo de definição de OSZ.

Os tempos de mapeamento das tarefas na OSZ foram de 3046 de *clock* para as execuções sem as alterações e de 2944 ciclos de *clock* para as execuções após as alterações, apresentando uma redução de 3,35% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 34510 e 36009 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 4,34% do tempo de alocação de recursos.

O tempo total de execução desde o início da simulação até o fim da execução da aplicação foi de 1401173 ciclos de *clock*, representando 0,11% de aumento no tempo total de execução. Este valor evidencia que, embora o aumento no tempo para o posicionamento da OSZ seja de 60,76%, o impacto no tempo total de execução da aplicação é pequeno devido ao baixo custo de tempo que a fase de alocação de recursos tem comparado à execução completa do cenário.

Como pode ser observado na Figura 16 e na Figura 17, o posicionamento da OSZ mudou devido às posições dos periféricos conectados ao MPSoC, e as tarefas foram mapeadas em outras posições devido ao alinhamento de tarefas que se comunicam com periféricos.

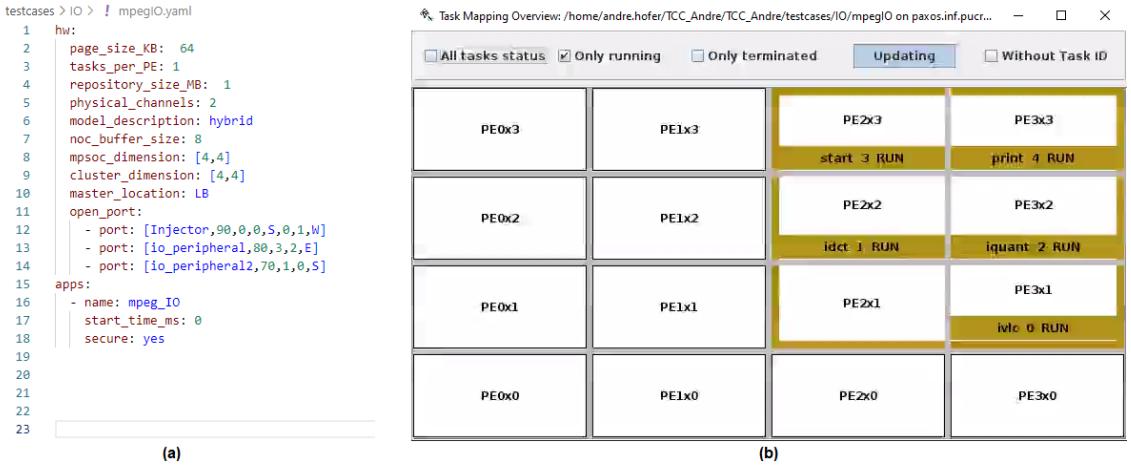


Figura 16 – *Testcase* e janela da simulação do Cenário 1 antes das alterações propostas. [Fonte: elaborado pelo autor.]

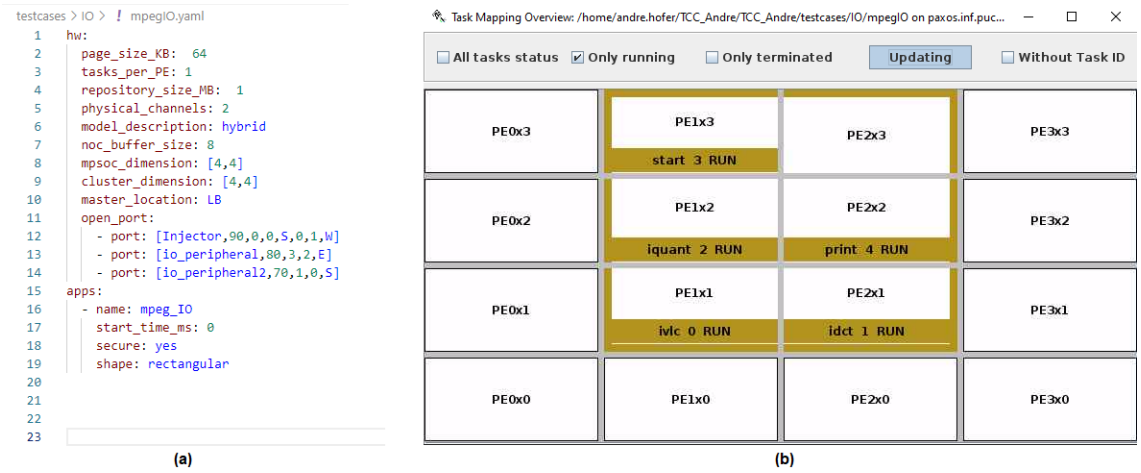


Figura 17 – *Testcase* e janela da simulação do Cenário 1 depois das alterações propostas. [Fonte: elaborado pelo autor.

4.1.2 Cenário 2

O Cenário 2 é a execução de uma aplicação *"mpeg_IO"* segura em um MPSoC de dimensões 4x4 com periféricos conectados nas bordas oeste, sul e leste. Neste caso a simulação executou até o fim de todas as aplicações em ambos os casos. Foi observado o posicionamento dos periféricos para que nenhuma tarefa que se comunique com periféricos tente tentar cruzar a fronteira da OSZ na área de corte da OSZ dado pelo uso do formato retilíneo, descrito na seção seção 3.3.

Os tempos de busca por formato e posição da OSZ foram de 2607 ciclos de *clock* para as execuções sem as alterações e de 4191 ciclos de *clock* para as execuções após as alterações; assim, temos um aumento de 60,76% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 3046 e *clock* para as execuções sem as alterações e de 2944 de

clock para as execuções após as alterações, ocasionando uma redução de 3,35% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 34510 e 36009 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 4,34% do tempo de alocação de recursos. O tempo total de execução desde o início da simulação até o fim da execução da aplicação foi de 1406455 ciclos de *clock*, representando 0,11% de aumento no tempo total de execução.

Podemos comparar, na Figura 18 e na Figura 19, que o posicionamento da OSZ mudou devido as posições dos periféricos conectados ao MPSoC, e as tarefas foram mapeadas em outras posições devido o alinhamento de tarefas que se comunica com periféricos, o que ocorria antes mas não ocorre mais.

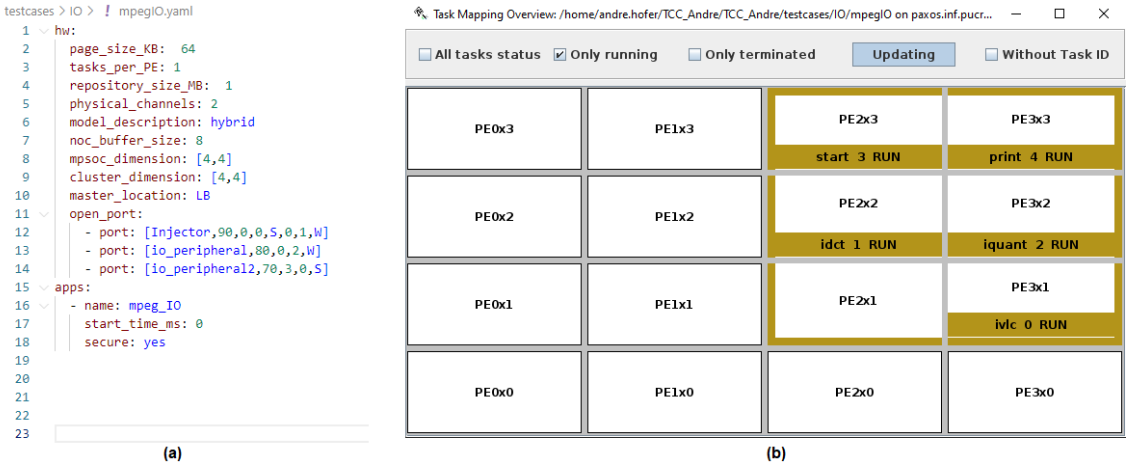


Figura 18 – *Testcase* e janela da simulação do Cenário 2 antes das alterações propostas. [Fonte: elaborado pelo autor.]

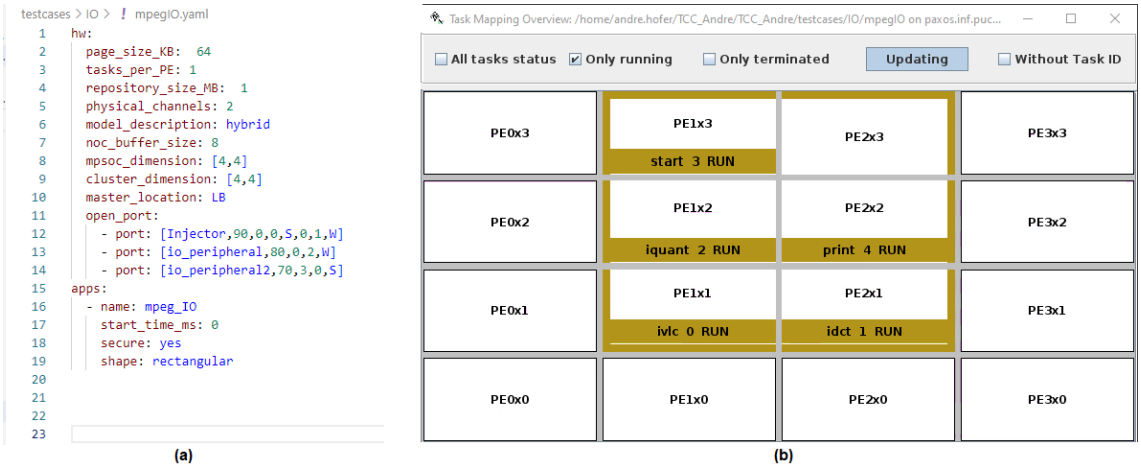


Figura 19 – *Testcase* e janela da simulação do Cenário 2 depois das alterações propostas. [Fonte: elaborado pelo autor.

4.1.3 Cenário 3

No terceiro cenário, temos a execução de uma aplicação *"mpeg_IO"* segura em um MPSoC de dimensões 4x4 com periféricos conectados nas bordas oeste, sul, leste e norte. Aqui, a simulação executou até o fim de todas as aplicações em ambos os casos. Foi observado o posicionamento dos periféricos para que nenhuma tarefa que se comunique com periféricos tente tentar cruzar a fronteira da OSZ na área de corte da OSZ, dado pelo uso do formato retilíneo, descrito na seção 3.3.

Os tempos de busca por formato e posição da OSZ foram de 2817 ciclos de *clock* para as execuções sem as alterações e de 12297 ciclos de *clock* para as execuções após as alterações, indicando um aumento de 336,53% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 3353 ciclos de *clock* para as execuções sem as alterações e de 3245 ciclos de *clock* para as execuções após as alterações, indicando uma redução de 3,22% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 36384 e 45443 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 24,90% do tempo de alocação de recursos. O tempo total de execução desde o início da simulação até o fim da execução da aplicação foi de 1410509 ciclos de *clock*, representando 0,66% de aumento no tempo total de execução.

É possível observar que o Cenário 3 teve um aumento considerável no tempo de busca por formato e posição da OSZ. Isso se dá devido ao posicionamento das regiões em que não podem ser criadas OSZ devido a conexão de periféricos, visto que no Cenário 3, em contraste com o Cenário 2, foi necessário verificar um segundo formato, já que o primeiro formato possível não pôde ser alocado por incompatibilidade de recursos. Além disso, o tempo de simulação deve sofrer algum aumento devido ao tamanho do MPSoC ter aumentado no Cenário 3 em comparação com o Cenário 2.

Como pode ser observado na comparação entre a Figura 20 e a Figura 21, o posicionamento da OSZ mudou devido as posições dos periféricos conectados ao MPSoC, e as tarefas foram mapeadas em outras posições devido o alinhamento de tarefas que se comunica com periféricos, que ocorria antes e agora não ocorre mais.

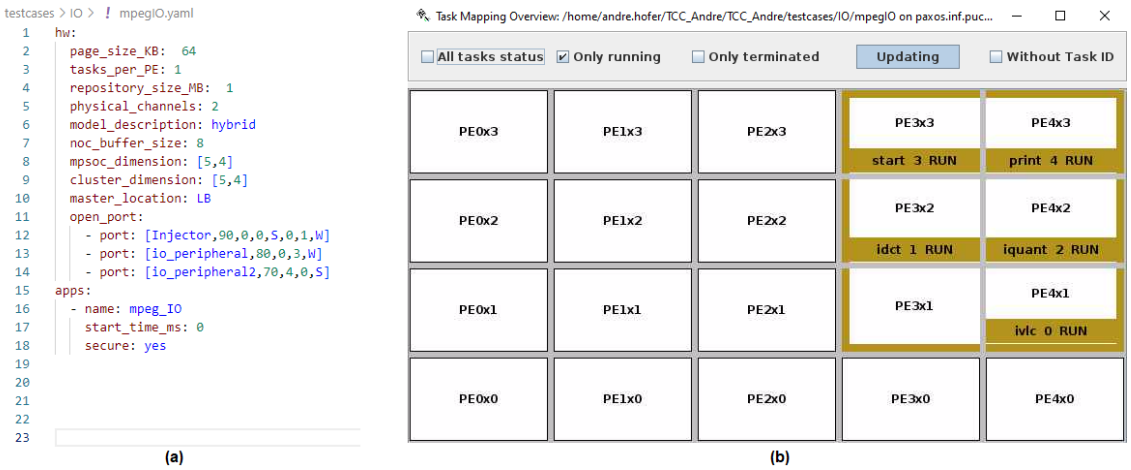


Figura 20 – *Testcase* e janela da simulação do Cenário 3 antes das alterações propostas. [Fonte: elaborado pelo autor.]

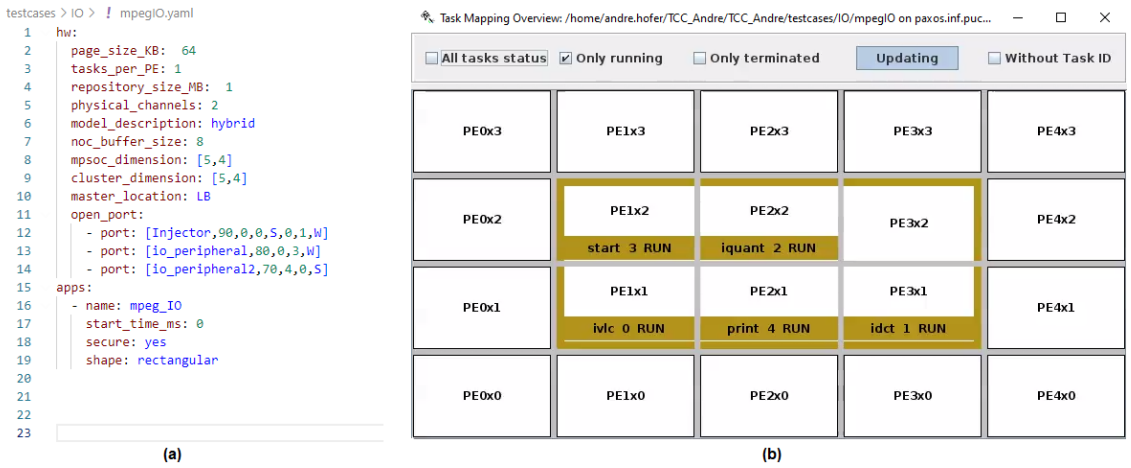


Figura 21 – *Testcase* e janela da simulação do Cenário 3 depois das alterações propostas. [Fonte: elaborado pelo autor.

Tabela 1 – Tempo de execução da busca por formato da OSZ para a aplicação *mpeg_IO* (0), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Aumento	
1	4x4	0	2607	4191	1584	60,76%
2	4x4	0	2607	4191	1584	60,76%
3	5x4	0	2817	12297	9480	336,53%

Fonte: elaborado pelo autor

Tabela 2 – Tempo de execução do mapeamento das tarefas na OSZ para a aplicação *mpeg_IO* (0), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Redução	
1	4x4	0	3046	2944	102	3,35%
2	4x4	0	3046	2944	102	3,35%
3	5x4	0	3353	3245	108	3,22%

Fonte: elaborado pelo autor

Tabela 3 – Tempos totais de execução para a aplicação *mpeg_IO* (0), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução; [2] o tempo de exeucação depois da implementação da solução; [3] o aumento bruto dos ciclos de clock e percentual, relativo a soma do aumento no tempo de busca de formato e a redução no tempo de mapeamento; [4] o percentual de aumento do tempo de início da aplicação, e [5] o percentual de aumento do tempo total da execução do cenário

Cenário	App	Aumento [3]		Start App		Aumento[4]	Fim da	Aumento[5]
				Antes[1]	Depois[2]		execução	
1	0	1482	26,22%	34510	36009	4,34%	1401173	0,11%
2	0	1482	26,22%	34510	36009	4,34%	1406455	0,11%
3	0	9372	151,90%	36384	45443	24,90%	1410509	0,66%

Fonte: elaborado pelo autor

É possível observar, na Tabela 1, que o cenário 3 teve o pior desempenho, uma vez que foi necessário testar o posicionamento do segundo formato da lista de formatos, já que o primeiro formato não encontrou recursos disponíveis para alocação, devido a posição dos periféricos. Essa busca pelo posicionamento do segundo formato impactou o tempo total, conforme observado na Tabela 3, ocasionando um aumento considerável quanto ao tempo na fase de alocação de recursos, porém pouco significativo considerando o tempo total de execução do cenário.

4.2 APLICAÇÃO *SYNTHETIC_IO*

4.2.1 Cenário 4

O Cenário 4 conta com a execução de uma aplicação *"synthetic_IO"* segura em um MPSoC de dimensões 6x6, com periféricos conectados nas bordas oeste e sul. Neste caso, a simulação executou até o fim de todas as aplicações apenas após as alterações propostas. Na execução antes das alterações propostas ocorreram falhas na comunicação com periféricos devido ao alinhamento das tarefas, gerando o problema de sombreamento descrito na seção 1.2.

Os tempos de busca por formato e posição da OSZ foram de 3139 ciclos de *clock* para as execuções sem as alterações e de 9796 ciclos de *clock* após as alterações, representando um aumento de 212,07% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 5237 ciclos de *clock* para as execuções sem as alterações e 4675 cilos de *clock* para as execuções após as alterações, representando uma redução de 10,73% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 44937 e 53215 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 18,42% do tempo de alocação de recursos. O tempo total de execução desde o início da simulação até o fim da execução da aplicação foi de 173647 ciclos de *clock*, representando 3,51% de aumento no tempo total de execução.

Como pode ser comparado na Figura 22 e na Figura 23, o tamanho da OSZ mudou devido à quantidade de tarefas da aplicação que se comunicam com periféricos e, consequentemente, as tarefas foram mapeadas em outras posições devido ao alinhamento de tarefas que se comunicam com periféricos, que ocorria antes e agora não ocorre mais. Nesse caso, pode ser observado que foram alocados recursos de processamento que não serão usados, porém essa é uma perda que definimos como aceitável visto que a preocupação deste trabalho é com o acesso a periféricos e segurança na execução das aplicações. Uma solução que visa diminuir o uso de recursos desnecessários é a criação de OSZs retilíneas quando for solucionado o problema de pacotes que não conseguem atravessar o corte da OSZ. No exemplo dessa aplicação seriam removidos os PEs (2,2), (2,3) e (2,4), passando de 16 PEs para 13 e o corte seria feito nas fases de definição de formato e de posição da OSZ, logo, impactaria no mapeamento das tarefas que não se comunicam com periféricos.

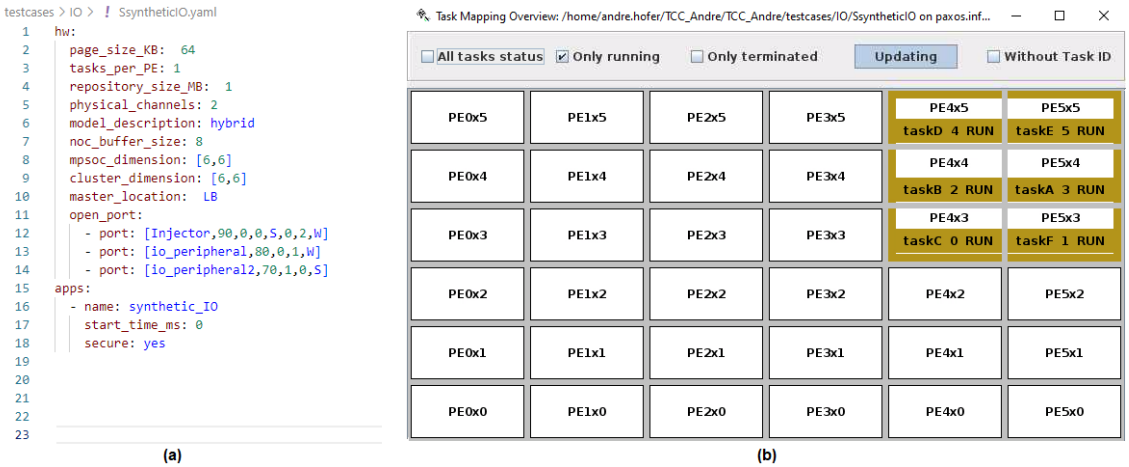


Figura 22 – *Testcase* e janela da simulação do Cenário 4 antes das alterações propostas. [Fonte: elaborado pelo autor.]

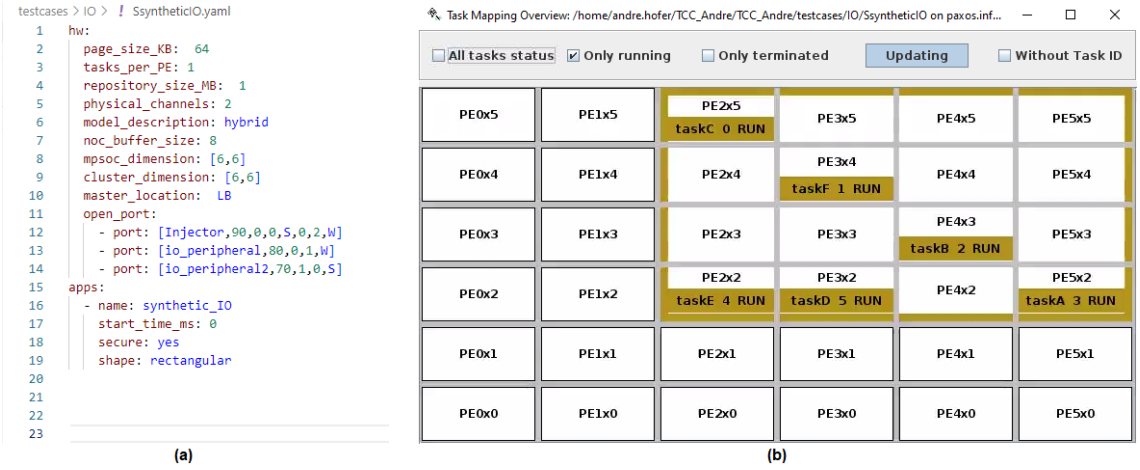


Figura 23 – *Testcase* e janela da simulação do Cenário 4 depois das alterações propostas. [Fonte: elaborado pelo autor.

Tabela 4 – Tempo de execução da busca por formato da OSZ para a aplicação *synthetic_IO* (0), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Aumento	
4	6x6	0	3139	9796	6657	212,07%

Fonte: elaborado pelo autor

Tabela 5 – Tempo de execução do mapeamento das tarefas na OSZ para a aplicação *synthetic_IO* (0), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Redução	
4	6x6	0	5237	4675	562	10,73%

Fonte: elaborado pelo autor

Tabela 6 – Tempos totais de execução para a aplicação *synthetic_IO* (0), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução; [2] o tempo de exeucação depois da implementação da solução; [3] o aumento bruto dos ciclos de clock e percentual, relativo a soma do aumento no tempo de busca de formato e a redução no tempo de mapeamento; [4] o percentual de aumento do tempo de início da aplicação, e [5] o percentual de aumento do tempo total da execução do cenário

Cenário	App	Aumento[3]		Start App		Aumento[4]	Fim da execução	Aumento[5]
				Antes[1]	Depois[2]			
4	0	6095	72,77%	44937	53215	18,42%	173647	3,51%

Fonte: elaborado pelo autor

É possível observar, na Tabela 4, que o Cenário 4 teve um aumento de 212% no tempo de execução nas fases de definição de formato e tamanho da OSZ. Isso se dá pela necessidade de buscar um tamanho que atenda a implementação que considera o mapeamento das tarefas que se comunicam com periféricos na diagonal da OSZ. Pelo fato da implementação do algoritmo de mapeamento das tarefas que se comunicam com periféricos ser mais simples, pode ser observada uma redução considerável no tempo de execução da fase de mapeamento das tarefas(Tabela 5). O aumento de tempo na busca por um tamanho adequado para a OSZ impactou o tempo total na Tabela 6, tendo um aumento considerável quanto ao tempo na fase de alocação de recursos, porém pouco significativo considerando o tempo total de execução do cenário.

4.3 APLICAÇÕES *MPEG_IO* + *DTW_IO*

4.3.1 Cenário 5

O Cenário 5 executa duas aplicações, uma *"mpeg_IO"* e uma *"dtw_IO"*, ambas seguras, em um MPSoC de dimensões 5x5, com periféricos conectados nas bordas sul e norte. Nesse caso, a simulação executou até o fim de todas as aplicações apenas após as alterações propostas. Na execução antes das alterações propostas, ocorreram falhas na comunicação com periféricos devido ao posicionamento das OSZs, gerando o problema de bloqueio de periféricos, e devido ao alinhamento de tarefas de uma aplicação que se comunicam com periféricos, assim como explicado na seção 1.2.

Para a aplicação *mpeg_IO*, os tempos de busca por formato e posição da OSZ foram de 3043 ciclos de *clock* para as execuções sem as alterações e 9502 ciclos de *clock* após as alterações, ocasionando um aumento de 212,26% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 3751 ciclos de *clock* para as execuções sem as alterações e de 3270 ciclos de *clock* após as alterações, ocasionando uma redução de 12,82% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 59514 e 69244 ciclos de *clock* para as execuções sem as alterações e após as alterações, respectivamente, ocasionando um aumento de 16,35% do tempo de alocação de recursos. Para a aplicação *dtw_IO*, os tempos de busca por formato e posição da OSZ foram de 3810 ciclos de *clock* para as execuções sem as alterações e de 7794 ciclos de *clock* após as alterações, representando um aumento de 104,57% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 4074 e 4040 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando uma redução de 0,83% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 71077 e 80551 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 13,33% do tempo de alocação de recursos. O tempo total de execução desde o início da simulação até o fim da execução de todas as aplicação foi de 1900553 ciclos de *clock*, representando 0,52% de aumento no tempo total de execução.

Comparando a Figura 24 e a Figura 25, observamos que o posicionamento das OSZs mudou devido às posições dos periféricos conectados e ao posicionamento da OSZ existente no MPSoC, e as tarefas foram mapeadas em outras posições em razão do alinhamento de tarefas que se comunica com periféricos, que ocorria antes e agora não ocorre mais.

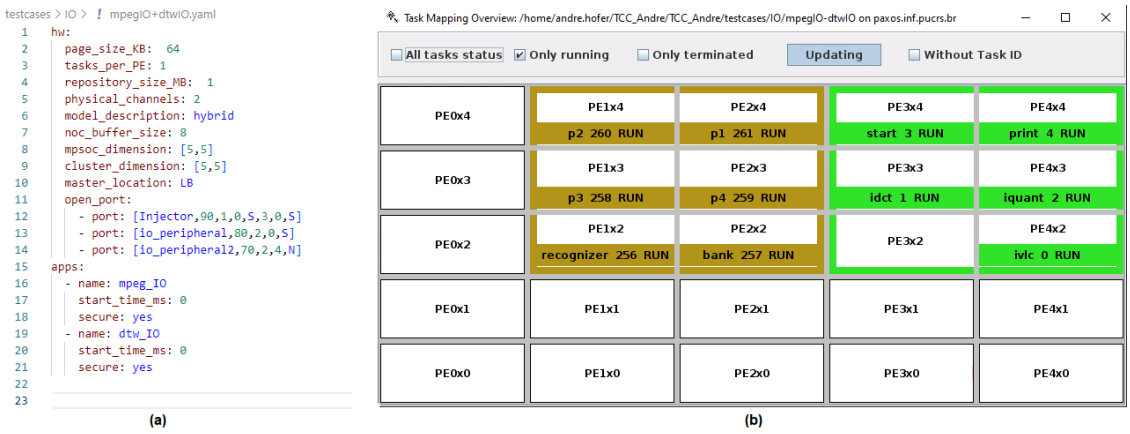


Figura 24 – *Testcase* e janela da simulação do Cenário 5 antes das alterações propostas. [Fonte: elaborado pelo autor.]

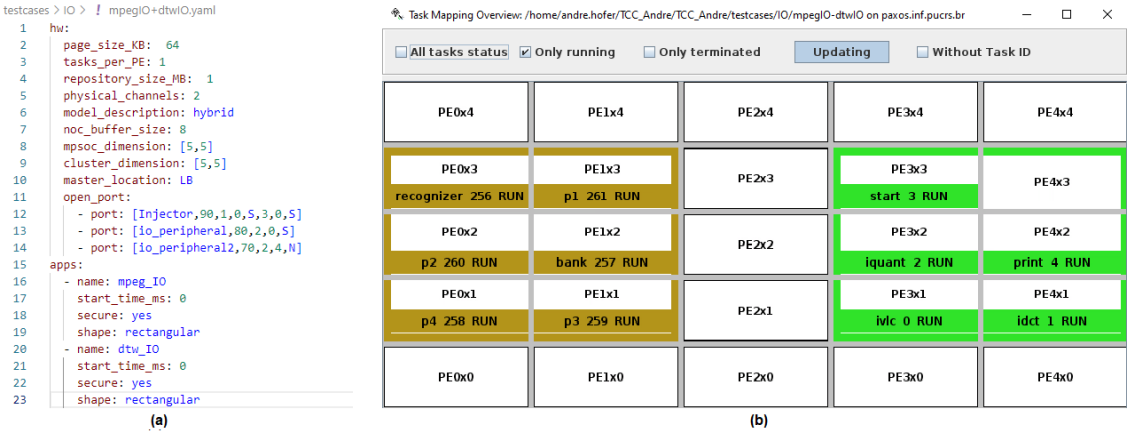


Figura 25 – *Testcase* e janela da simulação do Cenário 5 depois das alterações propostas. [Fonte: elaborado pelo autor.

4.3.2 Cenário 6

O Cenário 6 é a execução de duas aplicações, uma *"mpeg_IO"* e uma *"dtw_IO"*, sendo ambas seguras, em um MPSoC de dimensões 6x5 com periféricos conectados nas bordas sul, leste e norte. Nesse caso, a simulação executou até o fim de todas as aplicações apenas após as alterações propostas. Na execução antes das alterações propostas, ocorreram falhas na comunicação com periféricos devido ao posicionamento das OSZs, gerando o bloqueio de periféricos, e devido ao alinhamento de tarefas de uma aplicação que se comunicam com periféricos, problemas descritos na seção 1.2.

Para a aplicação *mpeg_IO*, os tempos de busca por formato e posição da OSZ foram de 3045 ciclos de *clock* para as execuções sem as alterações e de 10779 ciclos de *clock* para as execuções após as alterações, representando um aumento de 253,99% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 4102 ciclos de *clock* para as execuções sem as alterações e de 3429 ciclos de *clock* para as execuções após as alterações, ocasionando uma redução de 16,41% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 62632 e 73042 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 16,62% do tempo de alocação de recursos. Para a aplicação *dtw_IO*, os tempos de busca por formato e posição da OSZ foram de 4046 ciclos de *clock* para as execuções sem as alterações e de 8162 ciclos de *clock* para as execuções após as alterações, ocasionando um aumento de 101,73% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 4476 e 4290 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando uma redução de 4,16% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 69336 e 84485 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 21,85% do tempo de alocação de recursos. O tempo total de execução desde o início da simulação até o fim da execução de todas as aplicação foi de 1922311 ciclos de *clock*, representando 0,57% de aumento no tempo total de execução.

Comparando a Figura 26 e a Figura 27, observamos que o posicionamento das OSZs mudou devido às posições dos periféricos conectados e ao posicionamento da OSZ existente no MPSoC, e as tarefas foram mapeadas em outras posições devido o alinhamento de tarefas que se comunica com periféricos, que ocorria antes e agora não ocorre mais.

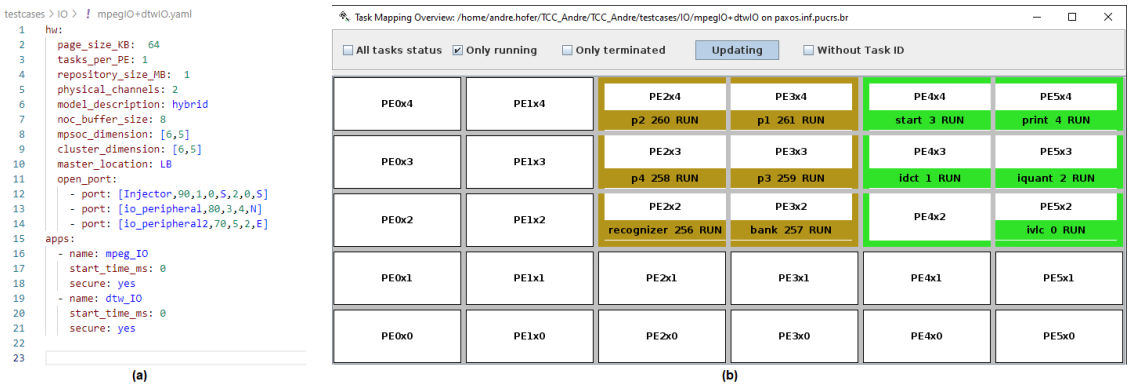


Figura 26 – *Testcase* e janela da simulação do Cenário 6 antes das alterações propostas. [Fonte: elaborado pelo autor.]

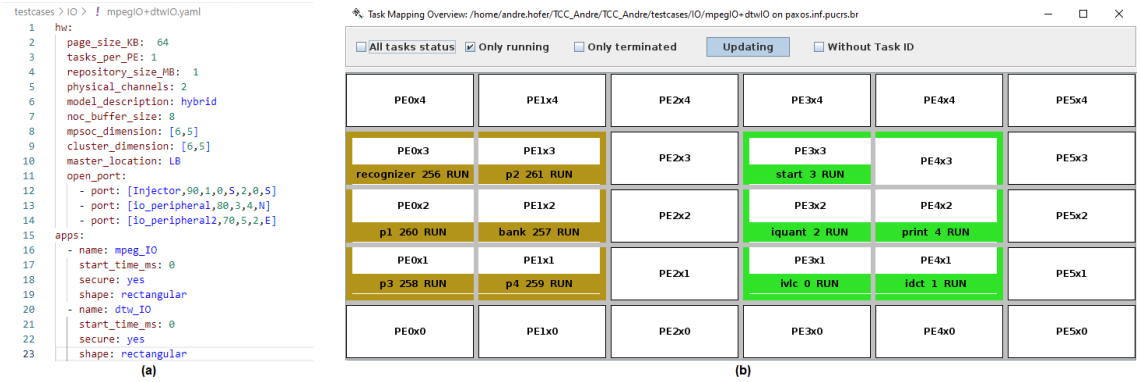


Figura 27 – *Testcase* e janela da simulação do Cenário 6 depois das alterações propostas. [Fonte: elaborado pelo autor.

Tabela 7 – Tempo de execução da busca por formato da OSZ para as aplicações *mpeg_IO* (0) e *dtw_IO* (1), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Aumento	
5	5x5	0	3043	9502	6459	212,26%
		1	3810	7794	3984	104,57%
6	6x5	0	3045	10779	7734	253,99%
		1	4046	8162	4116	101,73%

Fonte: elaborado pelo autor

Tabela 8 – Tempo de execução do mapeamento das tarefas na OSZ para as aplicações *mpeg_IO* (0) e *dtw_IO* (1), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Redução	
5	5x5	0	3751	3270	481	12,82%
		1	4074	4040	34	0,83%
6	6x5	0	4102	3429	673	16,41%
		1	4476	4290	186	4,16%

Fonte: elaborado pelo autor

Tabela 9 – Tempos totais de execução para para as aplicações *mpeg_IO* (0) e *dtw_IO* (1), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução; [2] o tempo de exeucação depois da implementação da solução; [3] o aumento bruto dos ciclos de clock e percentual, relativo a soma do aumento no tempo de busca de formato e a redução no tempo de mapeamento; [4] o percentual de aumento do tempo de início da aplicação, e [5] o percentual de aumento do tempo total da execução do cenário

Cenário	App	Aumento [3]		Start App		Aumento[4]	Fim da execução	Aumento[5]
				Antes[1]	Depois[2]			
5	0	5978	87,99%	59514	69244	16,35%		
	1	3950	50,10%	71077	80551	13,33%	1900553	0,52%
6	0	7061	98,80%	62632	73042	16,62%		
	1	3930	46,12%	69336	84485	21,85%	1922311	0,57%

Fonte: elaborado pelo autor

Todos os cenários tiveram um aumento considerável no tempo de execução nas fases de definição de formato e tamanho da OSZ, como observado na Tabela 7. Na Tabela 8 pode ser observada uma redução considerável no tempo de execução da fase de mapeamento das tarefas *mpeg_IO*. O aumento nos tempos totais nestes casos foi o mais impactante dentre os demais testes, porém continua sendo pouco significativo considerando o tempo total de execução dos cenários (Tabela 9).

4.4 APLICAÇÕES *DTW_IO* + *DTW_IO*

4.4.1 Cenário 7

O Cenário 7 corresponde a execução de duas aplicações *"dtw_IO"*, sendo uma segura e outra não segura, em um MPSoC de dimensões 4x4 com periféricos conectados nas bordas Oeste, sul e leste. Nesse caso, a simulação executou até o fim de todas as aplicações em ambos os casos.

Para a aplicação *mpeg_IO* segura, os tempos de busca por formato e posição da OSZ foram de 2712 ciclos *clock* para as execuções sem as alterações e de 4398 ciclos de *clock* para as execuções após as alterações, resultando em um aumento de 62,17% do tempo de definição de OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 3436 ciclos de *clock* para as execuções sem as alterações e 3888 ciclos de *clock* para as execuções após as alterações, resultando em um aumento de 13,15% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 54656 e 57367 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 4,96% do tempo de alocação de recursos. Para a aplicação *dtw_IO* não segura, não é feita busca por formato e posição da OSZ. Os tempos de mapeamento das tarefas na OSZ foram de 6936 ciclos de *clock* para as execuções

sem as alterações e de 7437 ciclos de *clock* para as execuções após as alterações, representando uma redução de 7,22% do tempo de mapeamento. A aplicação iniciou sua execução nos tempos de 61590 e 64301 ciclos de *clock* para as execuções sem as alterações e após as alterações respectivamente, ocasionando um aumento de 4,40% do tempo de alocação de recursos. O tempo total de execução desde o início da simulação até o fim da execução de todas as aplicação foi de 1896956 ciclos de *clock*, representando 0,14% de aumento no tempo total de execução.

Comparando a Figura 28 e a Figura 29, observamos que o posicionamento das OSZs mudou devido às posições dos periféricos conectados ao MPSoC, e as tarefas da aplicação segura foram mapeadas em outras posições devido o alinhamento de tarefas que se comunica com periféricos, que ocorria antes e agora não ocorre mais. As tarefas da aplicação segura são mapeadas sequencialmente conforme seja encontrado um recurso disponível.

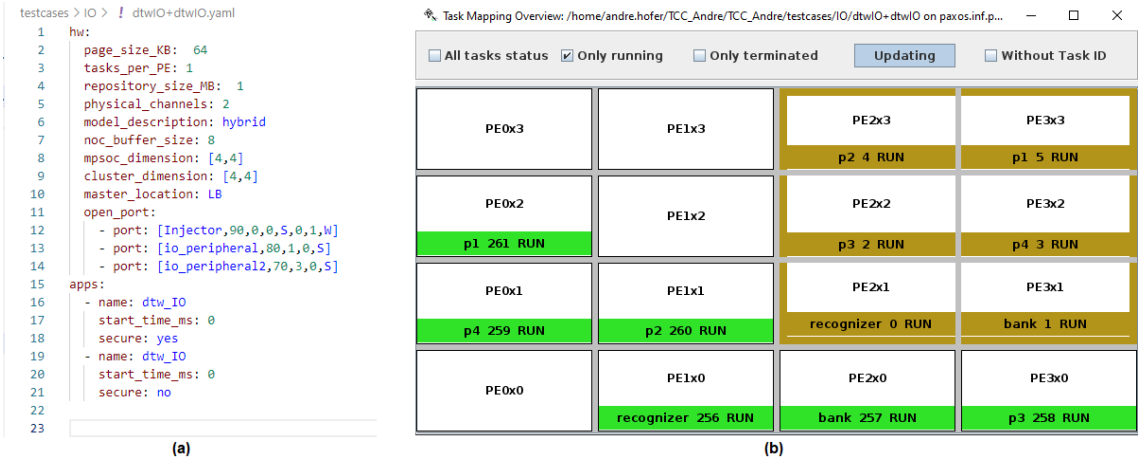


Figura 28 – *Testcase* e janela da simulação do Cenário 7 antes das alterações propostas. [Fonte: elaborado pelo autor.]

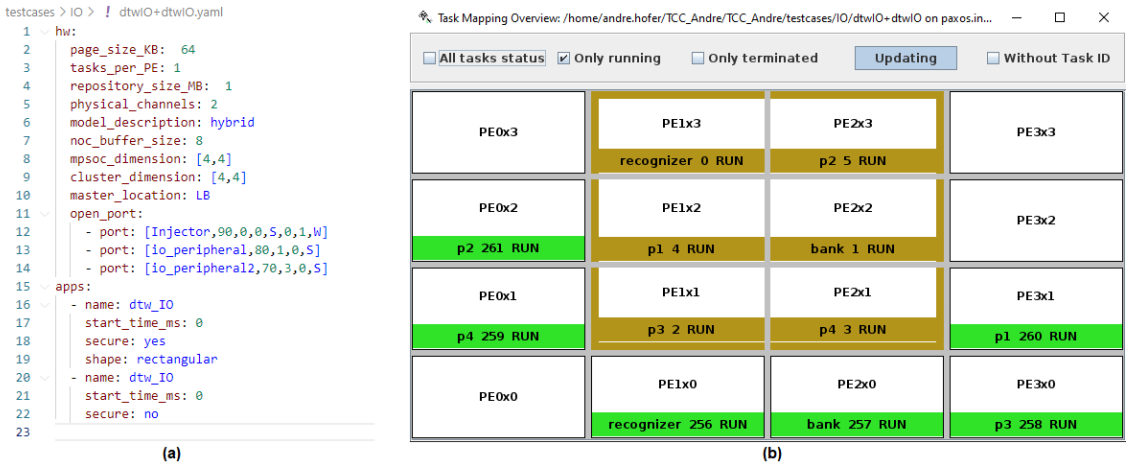


Figura 29 – *Testcase* e janela da simulação do Cenário 7 depois das alterações propostas. [Fonte: elaborado pelo autor.

Tabela 10 – Tempo de execução da busca por formato da OSZ para as aplicações *dtw_IO* segura (0) e não segura (1), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Aumento	
7	4x4	0	2712	4398	1686	62,17%
		1	0	0	0	0%

Fonte: elaborado pelo autor

Tabela 11 – Tempo de execução do mapeamento das tarefas na OSZ para as aplicações *dtw_IO* segura (0) e não segura (1), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução e [2] o tempo de execução depois da implementação da solução

Cenário	MPSoC	App	Antes[1]	Depois[2]	Aumento	
7	4x4	0	3436	3888	452	13,15%
		1	6936	7437	501	7,22%

Fonte: elaborado pelo autor

Tabela 12 – Tempos totais de execução para as aplicações *dtw_IO* segura (0) e não segura (1), em ciclos de *clock*, sendo [1] o tempo de execução antes da implementação da solução; [2] o tempo de exeucação depois da implementação da solução; [3] o aumento bruto dos ciclos de clock e percentual, relativo a soma do aumento no tempo de busca de formato e a redução no tempo de mapeamento; [4] o percentual de aumento do tempo de início da aplicação, e [5] o percentual de aumento do tempo total da execução do cenário

Cenário	App	Aumento [3]		Start App		Aumento[4]	Fim da execução	Aumento[5]
				Antes[1]	Depois[2]			
7	0	2138	34,78%	54656	57367	4,96%		
	1	501	7,22%	61590	64301	4,40%	1896956	0,14%

Fonte: elaborado pelo autor

Observa-se na Tabela 10 que a aplicação não segura não sofreu nenhum impacto nas fases de definição de formato e tamanho da OSZ, pois ela é mapeada sem OSZ. A fase de mapeamento sofre um pequeno impacto devido execução do algoritmo de mapeamento com as alterações implementadas (Tabela 11). O aumento nos tempos totais nesse caso foi considerável para a aplicação segura, porém sendo pouco significativo considerando o tempo total de execução dos cenários (Tabela 12).

4.5 CONCLUSÃO

Apesar de MPSoCs serem um assunto bastante explorado, a questão de acesso a periféricos com a segurança das aplicações através de zonas seguras opacas ainda não é um assunto amplamente explorado. A literatura relaciona métodos de mapeamento das aplicações mas com foco em outros objetivos, como minimizar custo de comunicação, *hotspots*, envelhecimento, dentre outros.

O presente trabalho teve como objetivo implementar uma solução que atenda os problemas de bloqueio a periféricos e de sombreamento, a fim de contribuir para o estudo de sistemas intra chip multiprocessados com interconexão baseada em redes intra chip, e que utilizam zonas seguras opacas como mecanismo de segurança.

A implementação atingiu os objetivos para a resolução dos problemas propostos, podendo ser observado que o tempo de execução das soluções para atender a demanda da definição de tamanho, formato e posição da OSZ, bem como o mapeamento das tarefas na OSZ dependem da disponibilidade e disposição de recursos, dos periféricos conectados e dos casos de teste a serem executados. Todos os resultados estão resumidos em uma tabela no anexo A.

Apesar de que em alguns casos o tempo para definição de tamanho e posição da OSZ mais que triplicar quando comparado à antes das soluções propostas, no pior cenário, considerando como exemplo um processador de 100MHz, que executa 100.000.000 ciclos de *clock* por segundo, o aumento no tempo de alocação de recursos para a execução das aplicações é de 9372 ciclos de *clock*, ou 93,72µs. Dado que o tempo de execução desde mesmo caso de teste foi de 1410509 ciclos de *clock*, ou de 14,10ms, o impacto da implementação é baixo considerando o fator tempo, mas é extremamente significativo considerando o avanço quanto ao acesso a periféricos de aplicações sendo executadas com segurança.

Como trabalhos futuros, seria interessante a execução de casos de testes mais robustos com aplicações com várias tarefas se comunicando com periféricos, com mais de uma tarefa comunicando com o mesmo periférico em situações mais diversas de posicionamento de periféricos e OSZs. Também é necessário ajustar a posição de fechamento dos *wrappers* da OSZ retilínea, respeitando o corte, quando acontecer, já que atualmente não é possível um pacote atravessar a região da fronteira da OSZ no corte gerado pelo formato retilíneo.

REFERÊNCIAS

ALI, Javid et al. Communication and aging aware application mapping for multicore based edge computing servers. **Cluster Computing**, Springer Science e Business Media LLC, mar. 2022. DOI: 10.1007/s10586-022-03588-1.

CAIMI, Luciano L. **Secure Admission and Execution of Applications in NcC-based Many-Cores Systems**. 2019. f. 121. Doutorado em Ciência da Computação – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

CAIMI, Luciano; FOCHI, Vinicius; WACHTER, Eduardo; MORAES, Fernando Gehm. Runtime creation of continuous secure zones in many-core systems for secure applications. In: 2018 IEEE 9TH LATIN AMERICAN SYMPOSIUM ON CIRCUITS & SYSTEMS (LASCAS). [S.l.: s.n.], 2018. p. 1–4. DOI: 10.1109/LASCAS.2018.8399904.

CAIMI, Luciano; FOCHI, Vinicius; WACHTER, Eduardo; MUNHOZ, Daniel et al. Secure Admission and Execution of Applications in Many-core Systems. In: SYMPOSIUM on Integrated Circuits and Systems Design (SBCCI). [S.l.: s.n.], 2017. p. 65–71. DOI: 10.1145/3109984.3110015.

CARARA, Everton Alceu. **Serviços de Comunicação Diferenciados em Sistemas Multiprocessados em Chip Baseados em Redes Intra-Chip**. 2011. f. 107. Doutorado em Ciência da Computação – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

DAS, Anup; KUMAR, Akash; VEERAVALLI, Bharadwaj. Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs. In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). [S.l.: s.n.], 2014. p. 1–6. DOI: 10.7873/DATE.2014.115.

DIORIO, Rafael Fernando et al. Segurança da Informação e de Sistemas Computacionais: Um Estudo Prático sobre Ataques Utilizando Malwares. In: V. 9 (2018): Anais SULCOMP. Criciúma: [s.n.], 2018.

FILHO, Sergio Johann. **Suporte para Aplicações Dinâmicas em Sistemas Multiprocessados Intra-chip Homogêneos**. 2011. f. 160. Doutorado em Ciência da Computação – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

KAUSHIK, Samarth et al. Run-Time Computation and Communication Aware Mapping Heuristic for NoC-Based Heterogeneous MPSoC Platforms. In: 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming. [S.l.: s.n.], 2011. p. 203–207. DOI: 10.1109/PAAP.2011.32.

MARWEDEL, Peter et al. Mapping of applications to MPSoCs. In: 2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). [S.l.: s.n.], 2011. p. 109–118. DOI: 10.1145/2039370.2039390.

MEDINA, Henrique Martins. **Projeto de Roteador para Rede Intra-chip Utilizando a Arquitetura Roundabout**. 2019. f. 33. Monografia (Tcc em Engenharia de Computação) – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

RAMACHANDRAN, Jay. **Designing Security Architecture Solutions**. USA: John Wiley e Sons, Inc., 2002. ISBN 0471430137.

RUARO, Marcelo; CAIMI, Luciano; FOCHI, Vinicius et al. A Framework for Heterogeneous Many-core SoCs Generation. In: 2019 IEEE 10TH LATIN AMERICAN SYMPOSIUM ON CIRCUITS & SYSTEMS (LASCAS). [S.l.: s.n.], 2019. p. 89–92. DOI: 10.1109/LASCAS.2019.8667590.

RUARO, Marcelo; CAIMI, Luciano; MORAES, Fernando Gehm. SDN-Based Secure Application Admission and Execution for Many-Cores. **IEEE Access**, v. 8, p. 177296–177306, 2020. DOI: 10.1109/ACCESS.2020.3025206.

RUARO, Marcelo; LAZZAROTTO, Felipe B. et al. DMNI: A specialized network interface for NoC-based MPSoCs. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS). [S.l.: s.n.], 2016. p. 1202–1205. DOI: 10.1109/ISCAS.2016.7527462.

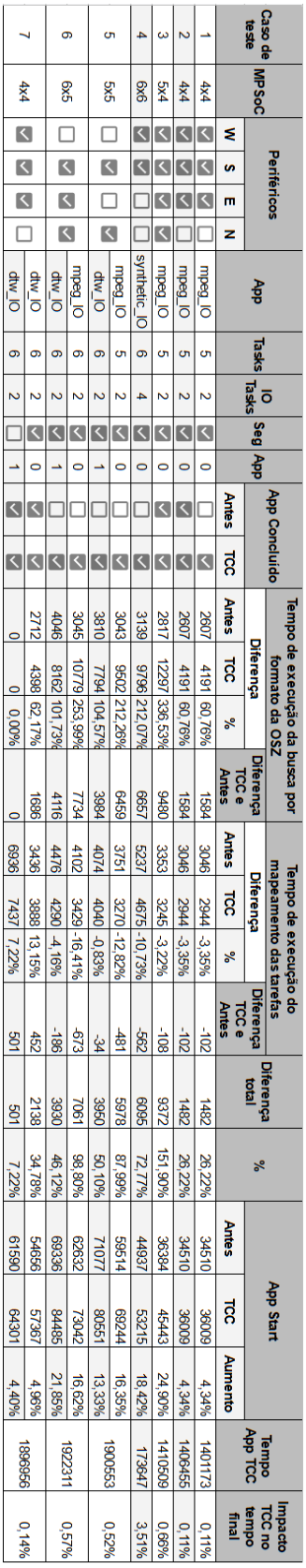
SANT'ANA, Anderson Camargo. **Vulnerabilidades de Segurança eContramedidas em Plataformas MPSoC**. 2019. f. 82. Mestrado em Ciência da Computação – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

SILVA OLIVEIRA, Samuel da; KREUTZ, Márcio Eduardo. Comparação de desempenho entre NoC e WiNoCs usando topologia Mesh. **Revista do CCEI**, 2019.

VIDAL, Ezequiel Luís. **Investigando Heurísticas de Mapeamento de Tarefas sobre a Plataforma HeMPS**. 2018. f. 79. Monografia (TCC em Ciência da Computação) – Universidade Federal do Pampa, Alegrete.

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

ANEXO A – AVALIAÇÃO DOS CASOS DE TESTES



Caso de teste	MPSoC	Periféricos				App	Tasks	IO Tasks	Seg	App	App Concluído		Tempo de execução da busca por formato da OSZ				Tempo de execução do mapeamento das tarefas				Diferença total	%	App Start			Tempo App TCC	Impacto TCC no tempo final
		N	E	S	W						TCC	Antes	Diferença			Diferença TCC e Antes	Antes	Diferença					Antes	TCC	Aumento		
													TCC	%				TCC	%								
1	4x4	☐	☑	☑	☑	mpeg_IO	5	2	☑	0	☑	☐	2807	4191	60,76%	1584	3046	2944	-3,35%	-102	1482	26,22%	34510	36009	4,34%	1401173	0,11%
2	4x4	☐	☑	☑	☑	mpeg_IO	5	2	☑	0	☑	☑	2807	4191	60,76%	1584	3046	2944	-3,35%	-102	1482	26,22%	34510	36009	4,34%	1406455	0,11%
3	5x4	☑	☑	☑	☑	mpeg_IO	5	2	☑	0	☑	☑	2817	12297	336,53%	9480	3353	3245	-3,22%	-108	9372	151,90%	36384	45443	24,90%	1410509	0,68%
4	6x6	☑	☑	☐	☐	synthetic_IO	6	4	☑	0	☐	☐	3139	9796	212,07%	6657	5237	4675	-10,73%	-562	6095	72,77%	44937	53215	18,42%	173647	3,51%
5	5x5	☐	☑	☑	☐	mpeg_IO	5	2	☑	0	☑	☐	3043	9502	212,26%	6459	3751	3270	-12,82%	-481	5978	87,99%	59514	69244	16,35%	1900553	0,52%
						dtw_IO	6	2	☑	1	☐	☑	3810	7794	104,57%	3984	4074	4040	-0,83%	-34	3950	50,10%	71077	80551	13,33%		
6	6x5	☐	☑	☑	☐	mpeg_IO	6	2	☑	0	☐	☑	3045	10779	253,99%	7734	4102	3429	-16,41%	-673	7061	98,80%	62632	73042	16,62%	1922311	0,57%
		☑	☑	☑	☐	dtw_IO	6	2	☑	1	☐	☑	4046	8162	101,73%	4116	4476	4290	-4,16%	-186	3930	46,12%	69336	84485	21,85%		
7	4x4	☐	☑	☑	☑	mpeg_IO	6	2	☑	0	☑	☑	2712	4398	62,17%	1686	3436	3888	13,15%	452	2138	34,78%	54656	57367	4,96%	1896956	0,14%
						dtw_IO	6	2	☑	0	☑	☑	2712	4398	62,17%	1686	3436	3888	13,15%	452	2138	34,78%	54656	57367	4,96%		
													0	0	0,00%	0	6936	7437	7,22%	501	501	7,22%	61590	64301	4,40%		

Figura 30 – Avaliação dos casos de teste executados. [Fonte: elaborado pelo autor.]