

Implementação do RISC-V IOPMP no CVA6 SoC

Francisco Marques , Bruno Sá , Sandro Pinto

Centro ALGORITMI - Universidade do Minho

pg47200@alunos.uminho.pt

bruno.sa@algoritmi.uminho.pt, sandro.pinto@dei.uminho.pt,

Resumo—Atualmente, a segurança é um dos requisitos mais importantes na computação moderna. Com o aparecimento de plataformas heterogêneas compostas por várias unidades de processamento, é necessário desenvolver mecanismos de proteção que garantam um elevado grau de isolamento e segurança do sistema. Uma área proeminente que visa melhorar a segurança de uma plataforma, é o isolamento espacial da memória. É neste contexto que este trabalho apresenta o desenvolvimento de um dispositivo de proteção de memória baseado na mais recente especificação v0.1 do *Input Output Physical Memory Protection* (IOPMP) de RISC-V num *System on Chip* (SoC) CVA6. Este dispositivo tem como objetivo principal a regulação e proteção de acessos à memória efetuados pelos vários *bus masters* que estão ligados ao barramento principal (ex. dispositivos DMA) que podem existir numa plataforma para além do *Central Processing Unit* (CPU).

Keywords—FPGA, RISC-V, Criticidade Mista, Sistemas Embebidos, Isolamento Espacial de Memória, IOPMP.

I. INTRODUÇÃO

As máquinas computacionais dependem cada vez mais de um grande número de dispositivos, sendo a unidade central de processamento (CPU) apenas um deles. Um sistema típico terá pelo menos um *core*, para processamento de dados, múltiplos periféricos (ex. UART, SPI, I2C, Controlador GPIO) para entrada e saída de dados com o ambiente externo, e memória, para armazenar dados. Plataformas mais avançadas podem ainda implementar dispositivos como DMA, *Graphical Processing Unit* (GPU) ou *cores* não regulados, para alcançar um melhor desempenho em tarefas específicas. Dado que estes dispositivos possuem o seu próprio controlador ou unidade de processamento, estes podem funcionar sem a intervenção do CPU. Tais dispositivos são conhecidos como *bus masters*. Este grau de liberdade vem com alguns aspetos negativos. Olhando para o dispositivo *Direct Memory Access* DMA, este oferece a possibilidade de transferir grandes quantidades de dados sem intervenção do CPU, mas isto implica que este tenha de ter acesso à memória do sistema. A divisão da memória do sistema em secções isoladas melhora a segurança do mesmo, uma vez que permite regular os acessos e permissões de diferentes regiões para múltiplos *masters*.

O ISA RISC-V [1] é um *Instruction Set Architecture* (ISA) aberto, modular, altamente extensível que pode ser utilizado em toda a gama computacional e com preocupação ao nível da segurança do sistema. Assim, é necessário desenvolver especificações que prevejam todas as medidas de segurança contra os vários tipos de falhas e para as várias gamas de plataformas. O RISC-V já especifica um dispositivo que permite o isolamento da memória para o processador, o *Physical Memory Protection* (PMP). Isto significa que um *core* RISC-V que implementa um dispositivo PMP, será associado a um conjunto

de posições de memória, as respetivas permissões, e o tipo de acesso. No entanto, esta solução encontra-se apenas ao nível do *core*, ou seja, nenhum mecanismo é formalmente especificado na especificação do RISC-V que defina um mecanismo de proteção ao nível do sistema e dos vários *masters* que podem existir para além do *core*, por exemplo, GPUs, aceleradores de domínio específico, etc.

De acordo com a motivação apresentada e com base no atual estado da arte, este trabalho visa desenvolver uma implementação funcional de um dispositivo de proteção de memória, denominado *Input Output Physical Memory Protection* (IOPMP), no *System on Chip* (SoC) CVA6 [2]. O IOPMP visa controlar o acesso às regiões de memória e a permissão dos *bus masters* não regulados. Recentemente, várias propostas para um IOPMP *standard* foram apresentadas e um *task group* da RISC-V International (TG IOPMP) foi criado com o objetivo de definir um *standard* para o IOPMP. Neste sentido, sendo a especificação um trabalho em desenvolvimento, o objetivo deste trabalho passa por desenvolver uma versão simplificada, modular e expansível da mesma. Esta versão pretende ser facilmente adaptável a futuras revisões da especificação do RISC-V IOPMP. Embora a versão a ser implementada seja uma versão simplificada, esta deverá cumprir a tarefa fundamental de isolamento espacial da memória entre os *bus masters* não regulados no sistema. Para validação do funcionamento do dispositivo desenvolvido, vai ser integrado um dispositivo DMA no sistema. Resumindo, com este trabalho pretende-se fazer as seguintes contribuições:

- Criação de uma especificação do RISC-V IOPMP baseada na versão *draft* v0.1 do *standard*;
- Desenvolvimento e implementação de um dispositivo IOPMP no CVA6 SoC implementado em FPGA;

II. MECANISMOS DE PROTEÇÃO DE MEMÓRIA

A segurança informática deve prestar atenção a todos os detalhes da plataforma do sistema. Uma das partes mais críticas é a memória do sistema. A memória contém todos os dados disponíveis da plataforma, incluindo estruturas de dados sensíveis e, por esse motivo, deve ser acedida de forma controlada. A tarefa de proteção da memória é tipicamente realizada por dispositivos de *hardware*. Isto acontece porque a verificação de cada transação seria demasiado dispendiosa para ser feita em *software*, dado o número elevado de acessos à memória que um programa efetua durante a sua execução. Isto tornaria a tarefa de verificação e validação de uma operação na memória impossível do ponto de vista desempenho do sistema.

Num sistema maior e mais complexo, onde podem existir mais do que um *bus master*, na falta de um mecanismo

para proteger a memória ou periféricos destes dispositivos, tais como dispositivos DMA, *cores* RISC-V não regulados ou GPUs, estes poderão aceder ao resto dos periféricos e memória que compõem o sistema sem qualquer restrição. Por esse motivo, a plataforma necessita de um mecanismo de proteção de memória e periféricos contra estes *bus masters* não regulados, para proteger os dados armazenados.

Os dispositivos existentes para o isolamento espacial da memória podem ser divididos em duas categorias: (1) dispositivos de proteção ao nível do *core* e (2) dispositivos de proteção ao nível da plataforma.

A. Isolamento ao Nível do Core

O conceito de memória virtual é um dos mais importantes na proteção da memória. Neste modelo, a memória principal comporta-se como uma *cache* para o dispositivo de armazenamento secundário. Existem duas grandes motivações para o uso de memória virtual [3]: (1) permitir a partilha eficiente e segura da memória entre vários programas, isto é, a memória virtual implementa a tradução do espaço de endereços de um programa para endereços físicos; e (2) remover os encargos de programação de uma pequena e limitada quantidade de memória principal, desta forma os programadores podem abstrair-se do espaço de memória existente, já que com memória virtual esta pode ser excedida. A tradução de endereços físicos para endereços virtuais é normalmente feita pelo dispositivo *Memory Management Unit* (MMU). Este dispositivo é de elevada complexidade, interessando apenas para este projeto o seu comportamento em termos funcionais.

Assim, podem enumerar-se alguns dispositivos já conhecidos e amplamente utilizados na comunidade como meio de proteção da memória ao nível do *core*.

1) *Memory Management Unit*: O MMU é responsável pela tradução de endereços virtuais em endereços físicos. O sistema operativo pode usá-lo para proteger a plataforma contra programas maliciosos, proibindo o acesso destes a regiões de memória que estes não necessitam de ter acesso ou permissões. Tipicamente, o sistema operativo atribui a cada programa o seu espaço de endereços virtuais, permitindo garantir um isolamento espacial da memória dos vários processos que correm no sistema.

2) *Memory Protection Unit*: Embora o MMU seja um dispositivo bem estabelecido, existem sistemas embebidos de recursos limitados como, por exemplo, plataformas baseadas em microcontroladores. Assim, alguns *trade-offs* precisam de ser feitos, surgindo deste modo o *Memory Protection Unit* (MPU). O MPU não efetua a tradução entre endereços físicos e virtuais, apenas executa parte das funções do MMU como a verificação de permissões e atributos.

Physical Memory Protection (PMP) é um dispositivo especificado no ISA privilegiada do RISC-V, que implementa um MPU. Este tem por objetivo principal de verificação e validação de endereços físicos de acordo com o tipo de acesso e as permissões concedidas a uma determinada região de memória. Note-se que o PMP opera independente do MMU e aplica-se a todas as operações efetuadas na memória. Os *cores* que já implementam o PMP, como é o caso do core CVA6 [2], são chamados de cores regulados. O firmware que corre

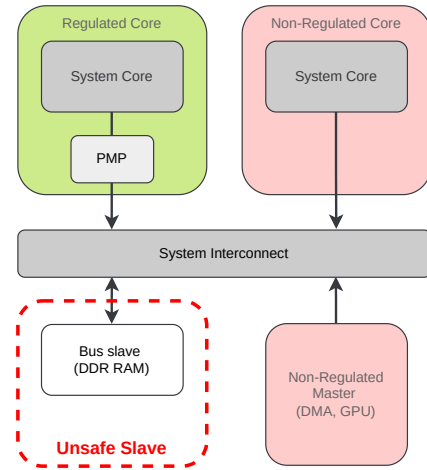


Figura 1. Plataforma com três *bus masters*, dois não regulados que comprometem a integridade dos dados na memória.

em *Machine mode* (M-mode em RISC-V) é responsável pela configuração e manutenção do PMP. Este dispositivo desempenha um papel fundamental no particionamento e isolamento da memória e na construção de Trusted Execution Environments (TEEs). Exemplos podem ser encontrados em trabalhos como o KeyStone [4], MultiZone [5] e nos domínios no *firmware* de RISC-V Opensbi em que este mecanismo é explorado com estes propósitos.

Podemos então concluir que os mecanismos de proteção no core não são suficientes para alcançar um isolamento total da memória numa plataforma em que vários atores de diferentes naturezas operam independentemente. Para uma melhor perceção, veja-se o exemplo hipotético da Figura 1 onde existem três *bus masters*, um deles sendo um core RISC-V regulado, um core RISC-V não regulado e um dispositivo DMA, também não regulado. Estes dispositivos conseguem controlar os canais de comunicação do sistema e por esse motivo podem ler e escrever em periféricos, comprometendo a confidencialidade e integridade dos dados. No caso de um core RISC-V regulado, este emite uma transação, e esta é verificada no PMP, assim esta é segura e válida. Por outro lado, se uma transação for emitida pelo dispositivo DMA, por exemplo, a transação nunca é verificada. Assim, há a possibilidade de *software* malicioso poder tirar partido dos dispositivos DMA para aceder a todos os periféricos, incluindo a memória do sistema.

B. Isolamento ao Nível da Plataforma

A ARM assegura a proteção do sistema através da *TrustZone*. Em sistemas baseados na tecnologia *TrustZone*, o sistema operativo (SO) de confiança funciona com um excesso de privilégios. O SO de confiança implementa gestão de memória, escalonamento e IPC para as aplicações de confiança. A tecnologia *TrustZone* disponibiliza mecanismos que permitem a partição de memória entre dois mundos (normal e seguro) utilizando TZASC e TZPC. O TZASC pode dividir a DRAM em várias regiões de memória, cada uma das quais pode ser configurada para ser utilizada no mundo normal ou seguro.

O TZPC é utilizado principalmente para configurar os periféricos como seguros ou não seguros e negará o acesso

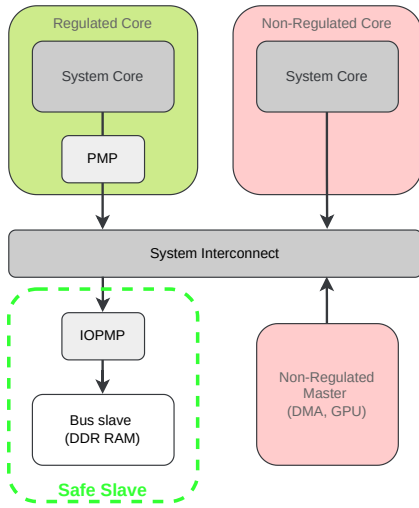


Figura 2. Plataforma com três *bus masters*, dois não regulados, e um periférico protegido pelo IOPMP.

ilegal aos periféricos protegidos. Além disso, a memória ROM ou SRAM também precisa de ser protegida. Isto é feito por um periférico SoC chamado *TrustZone Memory Adapter* (TZMA). Normalmente a ROM interna é definida como segura pelo *hardware designer*. Tanto o TZASC como o TZPC podem ser acedidos e configurados apenas no mundo seguro. Uma violação de acesso de segurança pode leva ao cancelamento da operação e geração de uma *trap* que leva o sistema para o *monitor mode*, por exemplo [6].

Recentemente, a comunidade RISC-V começou a definir dois mecanismos *standard* para proteção de dispositivos ao nível da plataforma: (1) *IO Memory Management Unit* (IOMMU), que fundamentalmente opera como um MMU para os outros *bus masters* do sistema para além do core, e (2) IOPMP, o foco deste trabalho e, por isso, será mais aprofundado no próximo capítulo.

III. RISC-V IOPMP

O IOPMP é o dispositivo responsável por fazer a ponte entre os vários *bus masters* que compõem uma plataforma complexa e o barramento ao qual estes estão ligados. Deste modo, todas as transações de memória efetuados por *bus masters* ao longo do barramento do sistema são validadas e autorizadas pelo IOPMP de acordo com os seus registos de configuração. Tal como o PMP, este dispositivo não efetua tradução de endereços, operando sobre endereços físicos que são acedidos durante uma transação. Este módulo impede o uso indevido de dispositivos DMA que possam existir no sistema, controlando assim todos os acessos à memória. Deste modo, o principal objetivo do IOPMP é agir funcionalmente como um PMP, contudo, neste caso, tendo em mente os dispositivos IO. Atualmente, existem quatro propostas para a especificação *standard* do IOPMP [7][8][9][10], e, tal como dito anteriormente, uma nova está a ser desenvolvida, baseada nas existentes, que tem por objetivo abranger um grande número de casos de uso. A Figura 2 apresenta uma visão geral da configuração de um sistema típico com um dispositivo DMA regulado por um IOPMP.

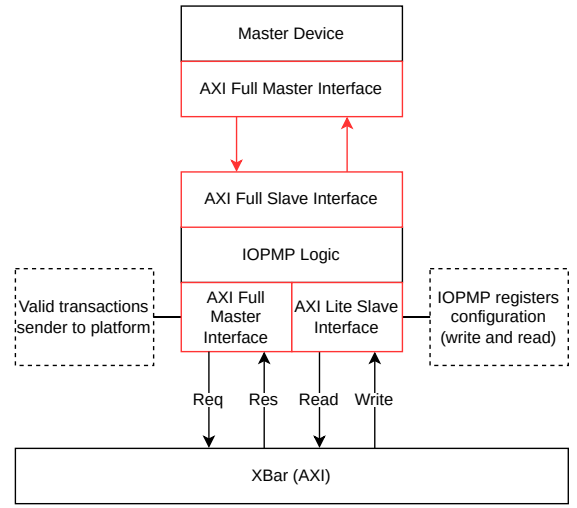


Figura 3. Visão geral da composição modular do IOPMP a desenvolver.

A. Visão Geral do Sistema

Para assegurar que o IOPMP permitisse uma grande modularidade, isto é, fosse facilmente instanciado para regulação dos *bus masters*, definiu-se que este deveria ser composto por quatro módulos essenciais, em que três destes módulos são interfaces de comunicação que implementam o protocolo de comunicação AXI [11]. Como todos os *bus masters* devem possuir uma interface AXI *master* para se poderem conectar ao sistema AXI *interconnect*, o dispositivo IOPMP deve possuir duas interfaces AXI, uma *slave*, interna ao IOPMP, para conexão ao dispositivo a ser regulado, e uma *master* para ser conectada ao AXI *interconnect*. Ambas as interfaces devem ser AXI *full* de forma a oferecer suporte de interface com uma grande variedade de dispositivos. Além destas duas interfaces AXI uma terceira deve ser implementada de forma a permitir a escrita e leitura dos registos de configuração do dispositivo IOPMP. Esta interface não requer um fluxo muito grande de dados e por esse motivo, pode ser implementada como uma interface AXI *lite*. O outro módulo constituinte é a lógica de validação e verificação das transações, aqui chamado de *IOPMP logic*. Esta composição do IOPMP, bem como a sua integração no sistema, pode ser melhor observada na Figura 3.

B. IOPMP logic

O lógica de validação de uma transação de um *bus master* através do IOPMP pode ser dividido em três sub módulos: i) *memory domain discovery*, ii) *rule checker* e iii) *execute response*. Existe uma porta de origem (do inglês *source port*), por onde a transação chega. A transação contém informações como o endereço da posição, ou região de memória a ser acedida, um número que identifica de forma única o dispositivo de origem da transação, *Source ID* (SID) e o tipo de acesso requisitado. Estas informações serão posteriormente usadas para a determinar quais as regiões associadas a cada dispositivo e as suas permissões de acesso à região de memória acedida.

1) *Memory Domain Discovery*: O IOPMP a ser desenvolvido deve implementar o conceito de *Memory Domain* (MD). Este conceito pode ser descrito como o agrupamento de várias regras de entrada do IOPMP. Assim, posições de memória relacionadas como, por exemplo, posições de memória relativas

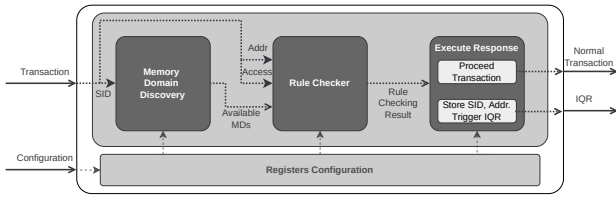


Figura 4. Composição modular do *IOPMP logic* a desenvolver.

a uma UART, podem ser agrupadas num *memory domain*, facilitando o processo de configuração do dispositivo. Deste modo, a primeira etapa do *IOPMP logic* será descobrir quais MDs o dispositivo de origem da transação tem acesso, de forma a saber quais entradas devem ser validadas pelo *rule checker*. Esta descodificação é realizada com o SID que vem na transação. Cada SID tem um registo onde os MDs a que este tem acesso são codificados no formato *bit-mapped*.

2) *Rule Checker*: Nesta etapa, com as entradas descobertas na fase anterior, com o endereço de memória a aceder e o tipo de acesso, escrita ou leitura, são comparadas todas as entradas com o endereço e o tipo de acesso de forma a validar se a transação é válida. Caso exista um *match* em alguma das comparações a transação é dada como válida. Assim, uma transação é considerada válida se:

- 1) O endereço a aceder (ou o conjunto de endereços) é igual ao endereço da *rule entry* X;
- 2) O tipo de acesso que chegou da transação é igual ao acesso especificado na *rule entry* X.

3) *Execute Response*: Por fim, a última fase é a execução de uma resposta apropriada ao tipo de validação que se obteve da etapa anterior. Deste modo, se a transação for válida nenhuma ação é necessária e a transação segue para o barramento normalmente. Por outro lado, caso a transação seja inválida, a transação é barrada, não chegando ao barramento, e mediante a configuração realizada do IOPMP informações sobre a transação ilegal (endereço de memória que tentava aceder, SID do dispositivo fonte, tipo de acesso e um campo de implementação específica) podem ser guardadas e uma interrupção é gerada. Posto isto, é possível verificar na Figura 4 a composição do *IOPMP logic*, bem como a interação entre os seus sub-módulos.

C. Registos de Configuração

Na Tabela I é possível observar todos os registos de configuração do IOPMP, bem como o seu *offset* e descrição. Esta composição forma o mapa de memória do dispositivo IOPMP.

D. Impacto Esperado na Solução Apresentada

Com a introdução do dispositivo IOPMP, é esperada a introdução de alguma latência nos acessos à memória, ou periféricos, por parte dos dispositivos *bus masters* regulados pelo primeiro. O impacto real na latência das transações terá duas fontes: (1) Utilização de *Block Random Access Memory* (BRAM) para guardar as entradas; (2) Utilização do protocolo AXI.

Tabela I. PROPOSTA DE MAPA DE MEMÓRIA PARA O IOPMP.

Registo	Offset	Descrição
iopmp_ctl	0x00	Registo de controlo do IOPMP, i.e, habilitar ou desabilitar o dispositivo
iopmp_rcd	0x04	Registo de controlo/informação dos metadados guardados de uma transação ilegal
iopmp_rcd_addr	0x08	Parte baixa 32-bits do endereço da transação ilegal
iopmp_rcd_addrh	0x0c	Parte alta 32-bits do endereço da transação ilegal
iopmp_mdmsk	0x10	Bloqueia MDs no iopmp_srcmd se iopmp_mdmsk.MD[x]=1
iopmp_mdmskh	0x14	Bloqueia MDs no iopmp_srcmd se iopmp_mdmsk.MD[x]=1
iopmp_mdclk	0x18	Bloqueia <i>bits</i> específicos no registo iopmp_mdcfg.T
iopmp_mdcfg[i]	0x100 + 0x04 × i	Tabela de tradução entre o MD e a tabela de entradas
iopmp_entry_addr[j]	0x104 + OFFMDCGF + 0x08 × j	Endereço da regra de entrada j, parte baixa. OFFMDCGF = 0x04 × i _{max}
iopmp_entry_addrh[j]	0x108 + OFFMDCGF + 0x08 × j	Endereço da regra de entrada j, parte alta. OFFMDCGF = 0x04 × i _{max}
iopmp_entry[j]	0x10C + OFFMDCGF + 0x01 × j	Configuração da interrupção/configuração da regra de entrada j. OFFMDCGF = 0x04 × i _{max}
iopmp_cfg[j/4]	0x10C + OFFADDR + 0x04 × (j/4)	Grupo de regras de configuração de entrada do IOPMP. OFFADDR = OFFMDCGF + 0x08 × j _{max}
iopmp_srcmd[k]	0x110 + OFFCFG + 0x08 × k	Memory Domain bitmap para o SID k. OFFCFG = OFFADDR + 0x04 × (j _{max} /4)
iopmp_srcmdh[k]	0x114 + OFFCFG + 0x04 × k	Memory Domain bitmap para o SID k. OFFCFG = OFFADDR + 0x04 × (j _{max} /4)

1) *BRAMs*: Os ciclos de atraso esperados com a implementação do *IOPMP Logic*, apresentado anteriormente, dependerá da implementação adotada, visto que, caso se utilize a tecnologia BRAM para o armazenamento das entradas — configuração e endereço — este apenas poderá efetuar a validação da transação após a leitura dessas entradas da memória. Desta forma, pode também acrescentar-se que é esperado que a latência causada por este módulo, neste caso, será tanto maior quanto maior for o número da entrada a ser validada, já que estas são verificadas por ordem crescente para garantir a regra de prioridade definida na especificação.

2) *Protocolo AXI*: A utilização do protocolo AXI apesar de conferir ao dispositivo a desenvolver uma maior modularidade e escalabilidade, acrescenta maior *overhead* do que um protocolo personalizado.

Assim sendo, após a implementação e validação do correto funcionamento do IOPMP, serão desenvolvidos testes com e sem o dispositivo, de forma a avaliar o impacto real na latência das transações. Por último, não é previsto que este dispositivo tenha qualquer influência na frequência de relógio utilizada na plataforma desenvolvida em FPGA.

IV. TRABALHO ATUAL

A primeira tarefa de trabalhos passou pelo desenvolvimento e implementação de uma especificação para o IOPMP baseada na versão *draft* v0.1 da especificação num SoC em FPGA baseado no processador RISC-V CVA6. O CVA6 é um CPU RISC-V escrito em System Verilog, de 6 estágios, *single-issue*, que implementa tanto a versão RV32 como RV64 do ISA. As ferramentas utilizadas durante o processo de desenvolvimento

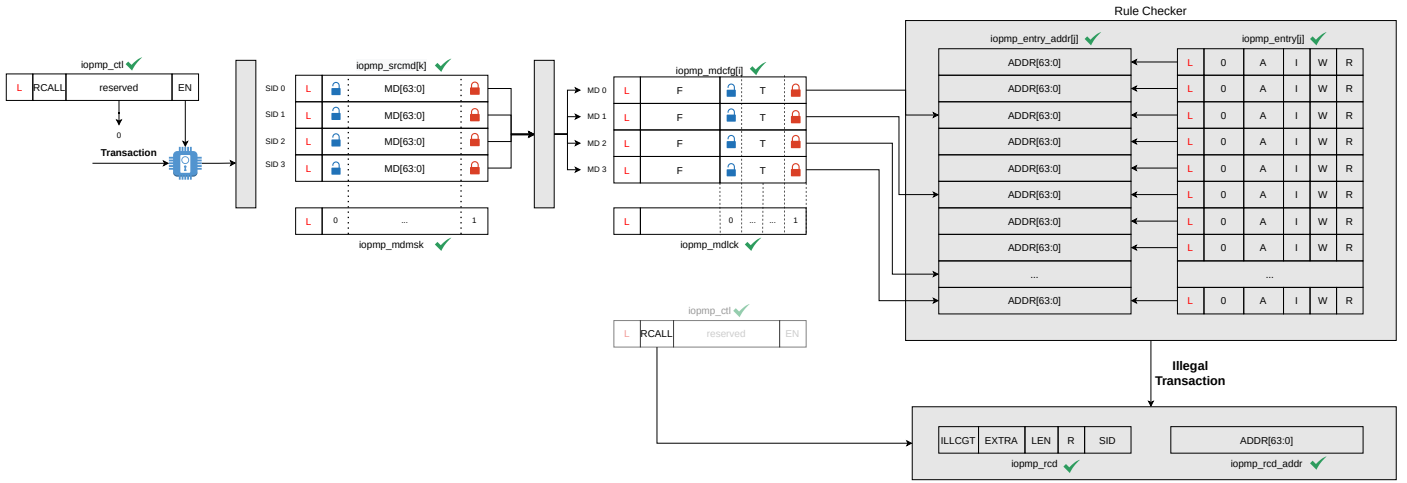


Figura 5. Visão geral da composição funcional do dispositivo IOPMP.

incluem: (1) emulação em FPGA do CVA6 SoC na FPGA Genesys2 e (2) modelo de simulação em Verilator já suportado no ecossistema. De momento, está finalizado o *design* e especificação do IOPMP, que representa um subconjunto mais simples do que já existe no *standard* oficial. Desta forma, e uma vez que a especificação se encontra em estado *draft*, será possível adaptar a implementação de acordo com o que será definido e alterado nas próximas revisões da especificação.

A. Processo de Validação de uma Transação

Na Figura 5 é possível observar-se o processo de validação e interação entre os vários registos de configuração. Assim, caso o IOPMP esteja habilitado será utilizado o SID para determinar a quais MDs este *master* tem acesso através da tabela *srcmd*. É agora necessário saber onde começam e terminam os domínios de memória. Para isso consulta-se a tabela *mdcfg*, que possui o campo T que corresponde ao índice para a última posição daquele MD, obtendo-se quais *entry rules* devem ser usadas na validação da transação. Valida-se agora a transação. Caso se determine que a transação é válida, esta ocorre normalmente. Caso contrário, vários metadados são guardados em *rcd* e *rcd_addr*. De notar que, se o *bit* RCALL do *iopmp_rcd* for um, a transação ilegal guardada é sempre a mais recente. Caso RCALL for zero, e se ILLCGT for um, não guarda informações da transação e mantém a informação da transação ilegal anterior (a primeira).

B. Definição do Módulo IOPMP

Face ao desenho da arquitetura apresentada, mostra-se agora a definição do módulo IOPMP, com as suas entradas e saídas. De forma a torná-lo altamente modular para futuras implementações em outras plataformas, este contém vários parâmetros para a sua configuração na hora da instanciação.

```

1 module iopmp #(
2     // IOPMP block parameters
3     parameter int unsigned PLEN = 34,
4     parameter int unsigned IOPMP_LEN = 32,
5     // Implementation specific parameters
6     parameter int unsigned NR_MD = 2,
7     parameter int unsigned NR_ENTRIES = 8,
8     parameter int unsigned NR_MASTERS = 2,

```

```

9     // AXI interface parameters
10    parameter int unsigned AXI_ADDR_WIDTH = 32,
11    parameter int unsigned AXI_DATA_WIDTH = 32,
12    parameter int unsigned AXI_ID_WIDTH = 10
13 ) (
14    input logic clk_i,
15    input logic rst_ni,
16    output logic irq_o,
17    output ariane_axi::req_t axi_req_to_xbar_o,
18    input ariane_axi::resp_t axi_respto_xbar_i,
19    input ariane_axi::req_t axi_config_req_i,
20    output ariane_axi::resp_t axi_config_resp_o,
21    output ariane_axi::req_t axi_req_to_master_o,
22    input ariane_axi::resp_t axi_resp_to_master_i
23 );
24 ...
25 endmodule;

```

Listing 1. Módulo IOPMP definição de entradas e saídas.

V. CONCLUSÕES E TRABALHO FUTURO

Este trabalho encontra-se em progresso. As próximas fases de desenvolvimento incluem: (1) migração e adaptação do *rule checker* presente no dispositivo PMP para o IOPMP, de forma a que tenha em conta as configurações nos registos do dispositivo IOPMP; e (2) criação de uma interface AXI *full* interna no IOPMP, para comunicação com os dispositivos *master* não regulados, de forma a criar uma interface padronizada para todos os dispositivos. Por fim, os próximos passos incluem a integração e validação do IOPMP no CVA6 SoC e desenvolvimento de um driver para o IOPMP. A validação funcional e avaliação deste módulo será alcançada de duas formas: (1) testes unitários ao módulo IOPMP (ex. leitura e escrita dos registos), usando para isso uma framework testes desenvolvida em trabalhos anteriores [12], e (2) desenvolvimento de um ambiente de testes real, utilizando para isso um dispositivo DMA ligado ao IOPMP, cujos acessos à memória devem ser validados pelo mesmo. Para além disso, todo o sistema deverá ser validado garantindo o correto funcionamento do sistema CVA6 com o novo dispositivo, na totalidade, com o sistema operativo Linux.

AGRADECIMENTOS

Este projeto foi suportado pela FCT - *Fundação para a Ciência e Tecnologia* no âmbito do projeto UIDB/00319/2020.

REFERÊNCIAS

- [1] K. Asanović and D. A. Patterson, "Instruction Sets Should be Free: The case for RISC-V," *EECS Department, Univ. of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.
- [2] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, pp. 2629–2640, 11 2019.
- [3] *Computer organization and design*, author="Patterson, David A and Hennessy, John L.
- [4] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, "Keystone: An Open Framework for Architecting Trusted Execution Environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys'20, 2020.
- [5] C. Garlati and S. Pinto, "A Clean Slate Approach to Linux Security RISC-V Enclaves," in *Embedded World Conference*, 2020.
- [6] Liwenhao. A Technical Report on TEE and ARM TrustZone. Disponível em: <https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/a-technical-report-on-tee-and-arm-trustzone>, Acedido a: 16/02/2022.
- [7] RISC-V. IOPMP Proposal v0.1. Disponível em: <https://github.com/riscv-admin/iopmp/blob/main/specification/IOPMP-poposal-v0-1.docx>, Acedido a: 16/02/2022.
- [8] P. S.-C. Ku. IOPMP Proposal Andes. Disponível em: <https://lists.riscv.org/g/tech-tee/files/IOPMP20Proposal20Andes.pdf>, Acedido a: 16/02/2022.
- [9] J. Xie. RISC-V I/O Physical Memory Protection Unit (IOPMP) Proposal. Disponível em: https://docs.google.com/document/d/15pVt5oG3OybQjMEWlo8U-f59_XD-tYZMYdPqTLqRGHo/edit#heading=h.lu90y4tvml2l, Acedido a: 16/02/2022.
- [10] S. Zhelnio. Syntacore IOPMP proposal. Disponível em: https://s3-us-west-1.amazonaws.com/220124_Syntacore_iopmp_proposal.pdf, Acedido a: 16/02/2022.
- [11] ARM, *AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE and ACE-Lite™*. ARM, 2013.
- [12] B. Sa, J. Martins, and S. Pinto, "A First Look at RISC-V Virtualization from an Embedded Systems Perspective," *IEEE Transactions on Computers*, pp. 1–1, 2021.