UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

ALEXSANDRO CRISTOVÃO BONATTO

Controle Adaptativo para Acesso à Memória Compartilhada em Sistemas em Chip

Tese apresentada como requisito parcial
para a obtenção do grau de
Doutor em Microeletrônica

Prof. Dr. Altamiro Amadeu Susin
Orientador

Porto Alegre, outubro de 2014

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bonatto, Alexsandro Cristovão

Controle Adaptativo para Acesso à Memória Compartilhada em Sistemas em Chip / Alexsandro Cristovão Bonatto. – Porto Alegre: PGMICRO da UFRGS, 2014.

155 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2014. Orientador: Altamiro Amadeu Susin.

1. Subsistemas de Memória. 2. Circuitos Integrados. 3. Hierarquia de Memórias. 4. Sistemas-em-Chip. I. Susin, Altamiro Amadeu. II. Título.

AGRADECIMENTOS

A Deus que por sua presença sempre me abençoa com saúde, força e determinação para seguir com o trabalho ao longo desses anos.

Ao meu orientador, professor Altamiro Amadeu Susin, meu agradecimento pela amizade, apoio e incentivo ao longo desses anos. Obrigado por acreditar no meu trabalho e pelas ideias incentivadoras que motivaram o desenvolvimento desta tese.

À minha querida esposa Camila Funk, que me acompanhou em todos os dias durante o trabalho dessa tese. Muito obrigado pelo carinho, apoio e compreensão que tivestes comigo.

Aos meus pais, Casemiro (*in memorian*) e Mari Stella, que sempre prestaram apoio e enorme incentivo para com meus estudos, desde o início do curso de engenharia, seguindo a carreira acadêmica até chegar ao doutorado, vocês são a minha fonte de inspiração para trabalhar por um mundo melhor.

Aos colegas do Laboratório de Processamento de Sinais e Imagens da UFRGS Marcelo Negreiros, André Borin e Letícia Guimarães. Obrigado pela enriquecedora troca de ideias, pelo apoio nos momentos difíceis, pelas revisões de artigos e textos, pela cooperação na depuração dos códigos e pela ajuda com a configuração dos ambientes de simulação.

Aos colegas da UFRGS Henrique Awoyama Klein, Fábio Pereira, Bruno Policarpo, Gustavo Ilha, Adriano Renner e Guilherme Corrêa pelo auxílio na validação do sistema e análises para o estudo de caso.

Aos meus colegas do Intituto Federal do Rio Grande do Sul Roben Lunardi, João Roberto Gabbardo, Neudy Alexandre Demichei, André Schneider, Jean Carlo Hamerski e Gleison Nascimento; por terem "segurado a barra" durante o meu período de afastamento e por terem incentivado-me a trabalhar na conclusão dessa tese. Fica aqui meu profundo agradecimento e respeito pela grande capacidade destes nobres colegas.

Aos professores Márcio Kreutz e Edgard Correa, por me receberem em Natal e pelas trocas de ideias para o desenvolvimento deste trabalho no grupo do SIGS.

Às agências de fomento FINEP, Capes e ao CNPq.

À todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

RESUMO

Acessos simultâneos gerados por Elementos de Processamento (EP) contidos nos Sistemas em Chip (SoC) para um único canal de memória externa coloca desafios que requerem uma atenção especial por constituírem o gargalo para o desempenho de processamento. No caso em que os EPs são microprocessadores, a questão fica ainda mais evidente, pois a taxa de aumento da velocidade dos microprocessadores excede a taxa de aumento da velocidade da DRAM. Ambas aumentam exponencialmente, mas a expoente dos microprocessadores é maior do que a das memórias. Este efeito é denominado de "muro de memória" (*Memory Wall*) e representa que o gargalo de processamento está relacionado à diferença de velocidade. Neste cenário, novas estratégias de controle de acesso são necessárias para melhorar o desempenho. Plataformas heterogêneas de processamento multimídia são formadas por diversos EPs. Os acessos concorrentes à regiões de memória não contíguas em uma DRAM reduzem a largura de banda e aumentam a latência de acesso aos dados, degradando o desempenho de processamento. Esta tese mostra que a eficiência computacional pode ser melhorada com o uso de um fluxo de projeto centralizado em memória, ou seja, orientado para os aspectos funcionais da DRAM. Neste trabalho é apresentado um subsistema de memória com gerenciamento adaptativo de compartilhamento do canal de memória entre múltiplos clientes. Esta tese apresenta a arquitetura de um controlador de memória com comportamento predizível que faz a avaliação do pior caso de execução para as transações solicitadas pelos clientes em tempo de execução. Um modelo baseado em atrasos é utilizado para prever os piores casos para o conjunto de clientes. O subsistema de memória centraliza a comunicação de dados e gerencia os acessos dos diversos EPs do sistema, de forma que a comunicação seja atendida de acordo com as necessidades de cada aplicação. São apresentadas três contribuições principais: 1) um método de projeto de sistemas integrados centralizado em memória, que orienta o projeto para os aspectos funcionais da memória compartilhada; 2) um modelo baseado em atrasos para estimar o pior caso de execução do sistema, quanto aos tempos de resposta e largura de banda mínima alocada por cliente; 3) um árbitro adaptativo para gerenciamento dos acessos à memória externa com garantias de prazos de execução das transações.

Palavras-chave: Subsistemas de Memória, Circuitos Integrados, Hierarquia de Memórias, Sistemas-em-Chip.

Adaptive Control for Shared Access Memory in System-on-Chip

ABSTRACT

The number of Processing Elements (PE) contained in a System-on-Chip (SoC) follows the growth of the number of transistors per chip. A SoC composed of multiple PEs, in some applications such as multimedia, implements algorithms that handle large volumes of data and justify the use of an external memory with large capacity. External memory accesses are shared by multiple PEs adding challenges that may have special attention because they constitute the bottleneck for performance and relevant factor for power consumption. In the case where the PEs are microprocessors, this issue becomes even more evident as the rate of increase of speed of microprocessors exceeds the rate of increase in speed of DRAM. This effect is called "memory wall" and represents that the bottleneck processing is related to the speed of data access. In this scenario, new access control strategies are needed to improve processing performance. Heterogeneous platforms for multimedia processing are formed by several PEs. The concurrent accesses to DRAM reduce bandwidth and increase latency access to data, degrading the processing performance. This thesis shows that significant improvements in computational efficiency can be obtained using a design methodology oriented to the functional aspects of DRAM through a memory subsystem with adaptive management. It is presented the data communication architecture for integration of PEs system based on an analytical model to reduce latency and guarantee Quality of Service (QoS). The memory subsystem is organized as a hierarchy of memories, with a proposed integration of PEs oriented centered in the main memory. The memory subsystem centralized data communication and manages the access of several PEs system so that communication is served according to the needs of each application. This thesis proposes three major contributions: 1) a methodology for design integrated systems based on the memory-centric design approach, 2) an analytical model based on delays used to evaluate the worst-case performance of the memory subsystem, 3) an arbiter for adaptive management of accesses to the external memory with guaranteed execution times of transactions.

Keywords: Memory Subsystem, Integrated Circuits, Memory Hierarchy, System-on-Chip.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASIC	*Application Specific Integrated Circuit*
ASIP	*Application Specific Instruction Set Processor*
BL	*Burst Length*
CAS	*Column-Address Strobe*
CI	Circuito Integrado
CL	*CAS Latency*
CME	Controlador de Memória Externa
DDR	*Double Data Rate*
DIMM	*Dual In-line Memory Module*
DRAM	*Dynamic Random Access Memory*
DSP	*Digital Signal Processing*
EDA	*Electronic Design Automation*
EP	Elemento de Processamento
ES	Entrada e Saída
FIFO	*First-In First-Out*
FPGA	*Field Programmable Gate Array*
HDL	*Hardware Description Language*
HW	*Hardware*
IP	*Intellectual Property*
ITRS	*International Technology Roadmap for Semiconductors*
JEDEC	*Joint Electron Device Engineering Council*
MEF	Máquina de Estados Finitos

NoC *Network on Chip*

RAM *Random Access Memory*

RAS *Row-Address Strobe*

ROM *Read Only Memory*

RTL *Register-Transfer Level*

SBTVD Sistema Brasileiro de Televisão Digital

SDR *Single Data Rate*

SDRAM *Synchronous Dynamic RAM*

SIA *Semiconductor Industry Association*

SRAM *Static Random Access Memory*

SoC *System-on-a-Chip*

SW *Software*

TIC Tecnologia da Informação e Comunicação

TVD Televisão Digital

VHDL *VHSIC Hardware Description Language*

VHSIC *Very High Speed Integrated Circuit*

VLSI *Very Large Scale Integration*

VSIA *Virtual Socket Interface Alliance*

WCET *Worst-Case Execution Time*

WCRT *Worst-Case Response Time*

LISTA DE SÍMBOLOS

tRC	Tempo de ciclo de acesso completo à linha da DRAM
tCK	Período do relógio do barramento da DRAM
RAS	Tempo de acesso à linha da memória DRAM
tRP	Tempo para fazer a pré-carga de uma linha DRAM
$tRTP$	Tempo mínimo entre o acesso à linha e pré-carga em uma DRAM
$tCCD$	Tempo mínimo entre dois acessos à colunas em uma DRAM
tRL	Tempo para completar a latência de leitura de dados em uma DRAM
$tRFC$	Tempo necessário para completar a operação de auto-refresh em uma DRAM
$tRCD$	Tempo mínimo entre o acesso à linha e o acesso à coluna em uma DRAM
$tCWD$	Tempo mínimo entre o acesso de escrita de linha e a colocação de comandos no barramento da DRAM
tWR	Tempo mínimo de recuperação de uma escrita de dados na DRAM
$tBURST$	Tempo para completar uma rajada de dados na DRAM
t_{w1}	Tempo para escrita de uma rajada
t_{wn}	Tempo para escrita de rajada de tamanho n
t_{r1}	Tempo para escrita de uma rajada
t_{rn}	Tempo para escrita de rajada de tamanho n
e_T	Eficiência das transações para memória
$e_{Tw}(n)$	Eficiência de uma transação de escrita de tamanho n
$e_{Tn}(n)$	Eficiência de uma transação de leitura de tamanho n
g	Granularidade mínima das transações
$l(n)$	Tamanho da transação solicitada pelo cliente n

$tRBC$	Tempo de retenção no buffer de comandos
tED	Tempo para escrita de dados
tLD	Tempo para leitura de dados
tAA	Tempo para completar uma operação de auto-atualização na DRAM
$tR(n)$	Tempo de resposta para o cliente n
$wcrt_E$	Pior caso de tempo de resposta para uma escrita
$wcrt_L$	Pior caso de tempo de resposta para uma leitura

SUMÁRIO

1 INTRODUÇÃO

No último século, a tecnologia de eletrônica e microeletrônica colocou à disposição da sociedade uma grande quantidade de equipamentos que se integraram à vida das pessoas. Os computadores tornaram-se cada vez menores, mais velozes, fáceis de utilizar e portáteis de modo que se incorporaram aos objetos pessoais dos indivíduos, tanto para o trabalho quanto para lazer e cultura. Ao mesmo tempo, servidores gigantescos guardam repositórios de conteúdo disponíveis a um clique do usuário. As pessoas que vivem na sociedade moderna estão cercadas por computadores, equipamentos que deixaram de ser simples máquinas de calcular para tornarem-se elementos importantes do cotidiano. Os computadores vêm tornando-se mais rápidos e acessíveis, ao mesmo passo que usam a energia de forma mais eficiente e ocupam menos espaço. Essa evolução é acompanhada por uma demanda crescente de inovação, guiada pelos consumidores que exigem produtos cada vez mais versáteis para realizar diversas funcionalidades.

A comunicação digital passou a transportar dados, voz e imagem e evoluiu para altas velocidades, em conexões fixas e móveis. A Convergência Digital[1] possibilitada pelas Tecnologias de Informação e Comunicação (TIC) criou a Computação Ubíqua (WEISER, 1999). A era da tecnologia ubíqua refere-se ao fato de que os sistemas computacionais passaram a estar presentes em qualquer lugar e participam da vida das pessoas. Sistemas Computacionais estão presentes nos equipamentos domésticos, industriais e de serviços, como refrigeradores, automóveis, telefones, etc. Um número cada vez maior de equipamentos que podem estar interligados criando o que se chama a Internet da Coisas (ou IoT - *Internet of Things*) (AGRAWAL; DAS, 2011).

A evolução dos sistemas de computação segue acompanhada por uma crescente complexidade no desenvolvimento de produtos e pela redução do tempo de vida útil dos equipamentos. A vida útil de um equipamento de eletrônica de consumo é dada não mais pela durabilidade do produto, mas sim pelo tempo de obsolescência, quando um novo produto ultrapassa o anterior em desempenho ou em funcionalidade. A obsolescência programada faz com que circuitos projetados hoje já não sejam mais úteis em um curto intervalo de tempo. A competição entre as empresas pelo desenvolvimento de novos produtos cresce a cada dia, e o desafio de proje-

[1] A convergência digital refere-se à sinergia de quatro tecnologias em um único equipamento: tecnologias de informação, de telecomunicações, de eletrônica e de entretenimento.

tar um sistema computacional está relacionado principalmente aos prazos limitados de projeto, desenvolvimento e teste de novas arquiteturas e aplicações. A elevação dos custos de projeto, que aumentam exponencialmente com a evolução da tecnologia, é devida principalmente ao projeto, teste e depuração do hardware, do software e da interação entre ambos (KEATING, 2011). O desenvolvimento dos Sistemas Computacionais cresceu proporcionalmente em complexidade e novas metodologias de projeto são necessárias. É nesse contexto que se posiciona a tese apresentada no restante deste documento.

Esta tese de doutorado está voltada para o projeto de sistemas computacionais integrados como os utilizados em aplicações móveis para processamento multimídia, instrumentação, controle, monitoração, medicina, aquisição de dados, etc. O projeto de sistemas cada vez mais complexos é feito de forma hierárquica, guiado por uma metodologia de particionamento em blocos cada vez menores para facilitar sua implementação. A integração dos subsistemas deve garantir o correto funcionamento e o melhor desempenho do sistema final. O foco deste trabalho é o projeto e implementação de um módulo de controle para uma memória externa compartilhada por diversos processos em sistemas com elevadas taxas de acesso a dados. Como estudo de caso é apresentada uma avaliação do desempenho do controlador de acesso à memória para uma aplicação multimídia típica que trata dados de áudio e vídeo. A qualidade e a resolução das aplicações multimídia crescem continuamente exigindo a otimização da banda do canal de acesso à memória. Este Capítulo introdutório detalha o contexto das aplicações no qual está inserido este trabalho, mostra como são extraídos os requisitos de operação dos módulos e lista as contribuições mais relevantes alcançadas.

1.1 Projeto de Sistemas de Computação

Sistemas computacionais são utilizados amplamente em diversos equipamentos que nos cercam, como em dispositivos aplicações portáteis, equipamentos eletroeletrônicos e plataformas industriais. O projeto de tais sistemas envolve desafios de avaliação de possibilidades (análise do espaço de projeto) e de composição de circuitos e sistemas, que resulta em uma ampla gama de decisões a serem tomadas. A avaliação prévia das possibilidades é fundamental para atingir um resultado com melhor desempenho e menor custo final. Esta seção discute alguns aspectos gerais do projeto de sistemas computacionais, destacando os desafios, assim como as tendências passadas e atuais para possibilitar uma extrapolação na definição dos caminhos a seguir.

1.1.1 Evolução dos Sistemas Computacionais

O projeto de sistemas eletrônicos é um desafio crescente. Como previsto por Gordon Moore em 1965, a complexidade dos circuitos integrados duplica a cada 24 meses (MOORE, 1998a)[2], devido ao aumento da quantidade de transistores num único chip. Tal crescimento exponencial do número de transistores nos circuitos integrados, que é possível principalmente devido

[2]Reprodução do artigo original, publicado em *Electronics, pp.114–117, April 19, 1965*

à redução do tamanho dos transistores, permite o aumento da complexidade de integração dos circuitos. O aumento da complexidade é alcançado somente com a qualificação da funcionalidade das ferramentas e das técnicas utilizadas no projeto dos circuitos eletrônicos. A redução do tamanho dos transistores também leva ao aumento da velocidade de operação do circuito. No entanto, o aumento da quantidade e da velocidade dos transistores não implica diretamente em um aumento da capacidade computacional dos sistemas.

A velocidade dos microprocessadores cresce em um ritmo maior do que a velocidade de acesso das memórias e dos discos de armazenamento de dados. Isso cria um gargalo de desempenho computacional do sistema, gerado pela latência de acesso aos dados. Memórias *cache* são utilizadas entre o microprocessador e o sistema de armazenamento de dados, a fim de melhorar a eficiência computacional. Com a introdução de técnicas de paralelismo, os sistemas computacionais multitarefa podem executar mais que uma única tarefa simultaneamente, tendo um ganho computacional importante sem aumentar a frequência de operação do sistema.

O paralelismo da computação é uma das formas encontradas para tirar vantagem do ganho obtido com o aumento da quantidade de transistores. Durante muitos anos o ganho computacional foi obtido com o aumento da velocidade dos microprocessadores e com paralelismo em nível de instruções (arquiteturas VLIW). O paralelismo em nível de tarefas também é explorado em sistemas com um único processador, através do escalonamento de tarefas, e em sistemas multi-processados em um único chip (*Multiprocessor System-on-Chip* ou MPSoC) (FULLER; MILLETT, 2011). Através da estruturação de um programa de computador em diversas tarefas, um núcleo monotarefa de processamento utiliza o escalonamento para paralelizar ao máximo a sua execução melhorando o desempenho. Em circuitos multi-processados, as tarefas são distribuídas entre diversos núcleos, paralelizando a execução das aplicações.

Uma das barreiras encontradas para o aumento da velocidade dos processadores é a elevada dissipação de potência. Atualmente, o incremento de desempenho dos processadores é obtido através do aumento do número de núcleos de processamento dentro do circuito integrado. No entanto, a potência dissipada continua sendo uma barreira também para os chips multi-processados (ESMAEILZADEH et al., 2013). Segundo Esmaeilzadeh et al. (2013), a quantidade de núcleos também tem um limite previsto, devido à quantidade de *dark silicon* (silício apagado) em chips formados a partir da integração de diversos núcleos (*multicores*). *Dark silicon* é o termo usado para descrever a quantidade crescente de transistores que podem ser colocados no chip, devido aos avanços da litografia, mas que não podem ser usados simultaneamente por longos períodos de tempo.

A solução encontrada para minimizar os problemas gerados pelo limite de potência e integração crescente de transistores é a especialização das partes do circuito, que gera o aumento da eficiência computacional (ESMAEILZADEH et al., 2013). Um circuito específico pode ser otimizado para realizar uma tarefa determinada com a maior eficiência energética. O projeto de um sistema complexo é orientado para a integração de módulos heterogêneos de processamento, especializados para cada tarefa, de forma que tenham o melhor aproveitamento de energia para

executar uma aplicação do que um processador de uso geral. Assim, o circuito é mais rápido e eficiente.

O projeto de circuitos integrados orienta-se para a construção de um sistema complexo formado por diferentes unidades independentes de processamento, integradas através de um sistema de comunicação de dados. A comunicação entre essas unidades é um dos pontos fundamentais do projeto do sistema, que determina a eficiência computacional e energética. Com a integração de diversos Elementos de Processamento (EP) em um único chip, o circuito integrado passa a ser denominado de "sistema-em-chip" (*System-on-Chip* - SoC). Os EPs são circuitos projetados de forma independente do sistema, para desempenhar uma tarefa específica. Tais sistemas podem conter partes analógicas, digitais ou ambas, formando um sistema misto de processamento de informação. São utilizados em sistemas embarcados como celulares, câmeras digitais, instrumentos eletrônicos, automóveis, eletrodomésticos, sistemas de controle industrial, dentre outros.

Tipicamente, um sistema-em-chip é formado por um ou mais módulos, blocos de memória, controladores de relógio, periféricos e interfaces de comunicação externa, conversores analógico-digital e digital-analógico, controladores de tensão do núcleo, etc. Todos componentes estão integrados por um sistema de comunicação interno que pode assumir diferentes topologias como: barramento centralizado, barramento estruturado, conexões ponto-a-ponto ou uma rede intrachip (NoC).

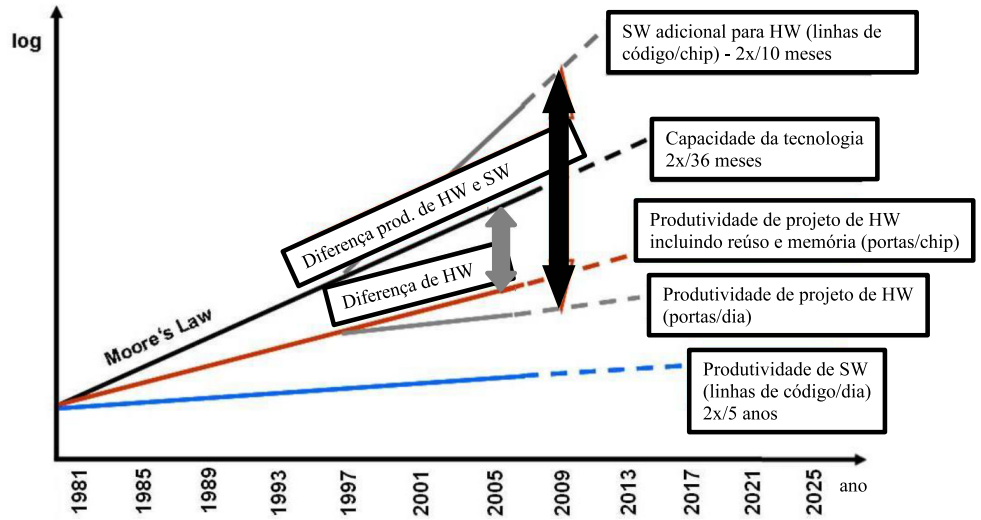
As partes de um SoC projetadas e testadas de forma independente são denominadas de "módulos de propriedade intelectual" (*Intellectual Property* - IP). Essa metodologia de projeto, orientada para a integração de IPs, permite que o SoC seja projetado, implementado e verificado em um tempo menor, através de técnicas de reuso de módulos. Desta forma, um SoC consegue explorar o paralelismo de tarefas no qual as partes do sistema executam suas tarefas de forma paralela e com melhor desempenho. O IP pode ser tanto uma descrição sintetizável de funcionamento encapsulado num módulo (ou objeto no caso de SW), como um desenho de máscara para fabricação, que é capaz de realizar uma tarefa específica de forma autônoma. O projeto de um sistema complexo feito em módulos, com orientação para o reuso de partes, tem como base o desenvolvimento de IPs. Essa metodologia de projeto implica em que a verificação do hardware não dependa da plataforma alvo e seja feita de forma incremental, elevando o nível de confiabilidade do sistema (KEATING; BRICAUD, 1999).

1.1.2 Lacuna de Produtividade

A evolução crescente nos processos de fabricação e o aumento da quantidade de transistores por circuito integrado permitem a evolução em direção à maior integração de diferentes partes no mesmo substrato. Por consequência, o aumento da quantidade de transistores a cada novo nó de tecnologia de fabricação, que disponibiliza mais recursos de hardware, faz com que o projeto de circuitos integrados seja uma tarefa mais trabalhosa. A necessidade de aumento de escalas de integração, de desempenho, de funcionalidade e de redução de consumo de energia

resulta em uma maior complexidade do projeto. A maior complexidade pode demandar um tempo maior de projeto, o aumento do custo em recursos humanos (homens/hora) e de equipamentos (ferramentas de hardware e software). A Figura 1.1 mostra a crescente diferença entre a complexidade dos chips e a produtividade das equipes de projeto, reportada na última previsão do *International Technology Roadmap for Semiconductors* (ITRS) no relatório *Design* (ITRS, 2011a).

Figura 1.1: Lacuna de produtividade no projeto de hardware.



Fonte: adaptado de (ITRS, 2011a).

Esta figura mostra que a necessidade de projeto de software, medida em linhas de código de software por chip, praticamente dobra a cada 10 meses, enquanto que a capacidade da tecnologia, medida em portas lógicas por chip, dobra a cada 36 meses. No entanto, a produtividade do projeto de software (linhas de código por dia) aumenta apenas duas vezes a cada cinco anos. A produtividade no projeto de hardware melhorou através do preenchimento do silício com IPs e memória, proporcionando o acréscimo de funcionalidade pela replicação de partes do sistema. A seta cinza resume a lacuna de projeto referente ao projeto de hardware, enquanto que a seta preta representa uma nova lacuna, inclusa na versão 2011, que mostra a diferença entre a capacidade de projeto de hardware e o software adicional para o hardware gerado[3].

No projeto em nível de sistema, os aspectos metodológicos estão se tornando rapidamente mais complexos do que os aspectos de fabricação. Atualmente, um circuito integrado de enorme complexidade pode ser implementado em um único chip, mas explorar esse potencial de forma confiável e com custo aceitável exigirá um aumento da produtividade de projeto de cerca de 50x sobre o que é possível hoje (ITRS, 2011a). Mesmo com metodologias de reuso desenvolvidas para acelerar a integração de partes do SoC, a diferença de produtividade continua crescendo com a evolução dos processos de fabricação (ITRS, 2011a) (SALEH et al., 2006).

[3]Essa versão do ITRS publicada em 2011 inclui, pela primeira vez, dados adicionais para ilustrar os requisitos de software.

Desta forma, o projeto de sistemas integrados é direcionado para a computação distribuída em partes heterogêneas e confiáveis, integradas através de um sistema de interconexões. A estrutura de comunicação é planejada previamente e as partes funcionais são integradas ao sistema de comunicação. Os custos de projeto de um sistema computacional são devidos na maior parte à escrita, teste e depuração de software (KEATING, 2011). Por outro lado, o custo da pastilha de silício com o circuito integrado manteve-se relativamente estável ao longo das últimas gerações de tecnologia[4].

A lacuna de produtividade de hardware representa a diferença que existe entre o número de transistores teoricamente disponível num chip e a capacidade de utilizar os transistores através de projeto de circuitos. Com o reuso de módulos de hardware pré-projetados, memória embarcada e completando o circuito com lógica dedicada, a curva representada em portas por chip ainda possui uma inclinação menor do que a evolução da tecnologia, indicada pelo número de transistores por chip. A produtividade de hardware é indicada pelo número de portas geradas pelo projetista durante o dia, atualmente estimada em 300 portas por dia (KEATING, 2011), considerando as etapas de especificação, projeto, teste e depuração do circuito. Como consequência engenheiros não conseguem fazer uso adequado dos transistores adicionais que estão disponíveis pela evolução dos processos de fabricação de novas tecnologias, a não ser pela replicação de estruturas regulares como memórias. Reduzir esta lacuna é um dos grandes desafios proposto pelo ITRS.

Como o projeto de sistemas integrados é orientado para o hardware[5], estudos apontam que a diferença de produtividade somente será superada se for modificada a metodologia de projeto. Atualmente circuitos complexos ainda são, em grande parte, projetados a partir de descrições próximas ao hardware, como descrições em nível de transferência de registradores e máquinas de estado. Níveis mais altos de descrição estão em desenvolvimento e a lacuna de produtividade somente será superada com "projetos em nível de sistema" (*system-level design*) sintetizados para hardware com ferramentas de *Electronic Design Automation* (EDA).

A automatização do processo de implementação de circuitos digitais é uma necessidade, como mostram os autores (SUZIM, 1981) e (GAJSKI; KUHN, 1983). E ainda segue sendo tema importante de pesquisa, pois a evolução dos nós de tecnologia continua de forma exponencial. O projeto em nível de sistema deve ser feito a partir de uma descrição comportamental eficiente, que avalie o comportamento da aplicação, com um olhar para as restrições de projeto como plataforma alvo, núcleos de HW pré-projetados disponíveis e descrições de SW em alto nível. Com a capacidade atual de integração de transistores no silício podem ser fabri-

[4] A análise completa é mostrada no livro escrito por Keating (2011), em seu livro intitulado "*The Simple Art of SoC Design*"comparando a evolução que houve entre os nós tecnológicos de 180 nm para 22 nm

[5] O projeto conjunto de hardware e software denominado de HW/SW Codesign é uma metodologia amplamente utilizada que tem como base a implementação arquitetural direcionada para os recursos disponíveis para implementação e fabricação do sistema. Embora seja necessário evoluir bastante o projeto de Software para conseguir aproveitar o hardware disponível, o projeto de sistemas eletrônicos ainda é direcionado para o projeto do hardware, pois ainda não se esgotaram as possibilidades tecnológicas e as previsões de evolução do nó tecnológico avançarão por mais de uma década.

cados sistemas extremamente complexos e o desafio para explorar totalmente esse potencial é enorme. Embora a síntese comportamental[6] tenha evoluído consideravelmente com o uso de descrições em SystemC e modelagem em nível de transações (TLM), ainda não é realizada de forma eficiente ao ponto de se ter uma "descrição sintetizável".

Os avanços atuais no projeto de sistemas complexos apontam que a solução para reduzir a lacuna de produtividade entre hardware e software está em dividir o sistema em componentes de tamanho apropriado e projetar interfaces eficientes entre eles. Segundo Keating (2011), estamos entrando na terceira revolução no projeto de hardware. A primeira ocorreu com a introdução da síntese a partir de uma descrição RTL, iniciada em 1986. A segunda ocorreu com a adoção de técnicas de reuso e IPs, em 1996. Atualmente tem-se a terceira revolução no projeto de hardware, que é proposta para reduzir a lacuna de projeto de sistemas de computação. Esta terceira revolução direciona o projeto de um sistema complexo para uma avaliação de funcionamento e de complexidade da aplicação, através da observação do seu funcionamento usando modelos de descrição em alto nível e combinados. Após é feita a especificação da plataforma utilizada e os modelos em níveis elevados de abstração com ferramentas que permitam a exploração do espaço de projeto para a definição da melhor arquitetura para aquela aplicação. Como é proposto por Gajski et al. (2009), o sistema de computação é visto como um sistema eletrônico, feito como uma simbiose de hardware e software, que deve ser especificado em diferentes níveis de abstração. A validação do seu funcionamento é feita em diferentes níveis de abstração e combinados se necessário.

1.1.3 Níveis de Abstração

Tradicionalmente um sistema integrado numa pastilha de silício é desenvolvido de forma hierárquica, com subpartes ou módulos projetados separadamente. A automatização do processo de síntese de circuitos integrados é proposta como a principal direção para reduzir a lacuna de produtividade que existe. Mas esse processo de automatização requer também que a descrição do sistema seja simplificada, de forma a permitir que sistemas cada dia mais complexos possam ser descritos. As técnicas de automatização de processos são aplicadas em todas as etapas de projeto: modelagem, simulação, síntese e verificação.

A automatização das etapas de projeto usando linguagens e ferramentas específicas requer que existam diferentes níveis de abstração ou níveis de descrição. Como proposto por Suzim (1981), os níveis de descrição de um microprocessador formam um fluxo de projeto que inicia desde uma descrição em alto nível das unidades operativas e de controle até chegar à descrição do desenho das células que formam a arquitetura. Uma biblioteca de células, com dimensões e conexões padronizadas, é utilizada no nível mais baixo da descrição para gerar o desenho do silício. Cada nível de descrição envolve uma linguagem que traduz um algoritmo, e a mudança de níveis sempre provoca a alteração do algoritmo originalmente descrito em um nível mais

[6]Referimos aqui a Síntese Comportamental como a síntese para hardware a partir de uma descrição do comportamento do sistema, ou seja, a partir da descrição da aplicação e de seus requerimentos.

elevado, mas nem sempre uma mudança de linguagem (SUZIM, 1989). O projeto de circuitos integrados é concebido, desta forma, a partir de uma sequência de refinamentos sucessivos, partindo de uma descrição mais abstrata até chegar ao nível de implementação.

Regras de projeto e de simulação devem ser criadas a partir da especificação dos níveis de abstração que descrevem um circuito complexo (GAJSKI et al., 2009). A abordagem mais comum no projeto de uma plataforma processada é a realização de co-simulação de software e componentes de hardware (HW/SW Codesign). Processadores de aplicações específicas e padrão são simulados em nível de instrução usando um *Instruction Set Simulator* (ISS). Os componentes de HW dedicados ou de propriedade intelectual (IP) são descritos a partir de uma descrição funcional e integrado à dos processadores, formando um Modelo em Nível de Transação (TLM), que é usado para avaliar a comunicação entre os componentes do sistema. Linguagens utilizadas nessas etapas são baseadas na linguagem C, como SystemC (ACCELLERA, 2013) e SpecC (GAJSKI et al., 2000).

O diagrama Y proposto por Gajski e Kuhn (1983) foi desenvolvido como uma orientação à prática de projeto de sistemas integrados. Ele descreve a relação existente entre os níveis de abstração do sistema e suas três formas de representação: funcional, estrutural e geométrica. Cada espaço de representação do sistema é formado por diferentes níveis de abstração. A representação funcional é usada para descrever o funcionamento do sistema. Nesse espaço de representação ainda não se sabe como o chip será construído, mas somente é feita uma análise da complexidade computacional. A representação estrutural é a ponte entre a representação funcional e a representação geométrica. É o mapeamento de uma representação funcional para um conjunto de componentes e conexões seguindo as restrições de projeto, tais como custo, área e desempenho. A representação geométrica é a finalização do projeto que resulta em um circuito físico, que se concentra na concepção espacial e geométrica do silício e ignora os aspectos de funcionamento do circuito.

Os níveis de abstração são usados para projetar o sistema a partir de diferentes pontos de vista, preservando os aspectos e particularidades de cada nível. No nível de processador o projeto orienta-se para a descrição e implementação das tarefas para uma arquitetura específica de processamento. Já no nível de sistema o projeto orienta-se para a descrição e implementação das tarefas do sistema distribuídas em EPs. Um EP pode ser um processador dedicado ou genérico, ou também um IP concebido para uma tarefa. O modelo comportamental do sistema deve representar a execução dos múltiplos processos em paralelo, tanto SW como HW, e a comunicação entre eles. O modelo estrutural em nível de sistema contém uma descrição em diagrama de blocos ou *netlist* de todos os componentes usados para computação, armazenamento e comunicação. Os elementos de processamento são processadores específicos ou de uso geral, IPs ou componentes de hardware. Os elementos de armazenamento podem ser memórias locais ou compartilhadas. Os elementos de comunicação podem ser barramentos ou roteadores conectados a uma NoC.

A síntese desse sistema para uma plataforma de computação parte do modelo comporta-

mental em nível de sistema, passando pelas etapas de estimação e análise do comportamento, alocação de componentes e conexões, definição das partes de HW e de SW, até o refinamento do modelo. Esta é a metodologia atual de projeto de sistemas, que se baseia em especificar o sistema, explorar as possibilidades de síntese e refinar o resultado. O nível de abstração RTL é ampliado para um nível de abstração de sistema (SL - *System Level*), que permite incluir no modelo diferentes detalhes para avaliar ao máximo as decisões de projeto. Os modelos em nível de sistema são usados para avaliar as propriedades: funcionalidade, aplicação de algoritmos, conectividade, comunicação, sincronização, coerência, roteamento, desempenho, potência e área. Essa abordagem permite que o modelo do sistema seja refinado, tanto globalmente como em suas partes menores individuais. Além disso, permite também que um modelo seja constituído a partir de diferentes linguagens e níveis de descrição, aliando a verificação de partes da plataforma em um nível maior de detalhamento do que outras partes.

1.1.4 Projeto Baseado em Plataforma

O projeto baseado em plataforma é orientado pela definição de fronteiras bem delimitadas do desenvolvimento arquitetural e das aplicações que irão rodar na plataforma alvo. A plataforma é desenvolvida para uma classe de aplicações, podendo ser adaptada para gerar diversos produtos ou variações de um produto final. Atualmente, com o nível de integração de transistores em um circuito integrado, uma plataforma pode conter diversos processadores homogêneos ou heterogêneos. A especificação do produto gera as definições de projeto para instanciar (ou customizar) a plataforma para uma aplicação específica. Em uma segunda etapa as especificações da aplicação são usadas para guiar a implementação da aplicação sobre a plataforma alvo. A definição da plataforma envolve a escolha de um ou mais processadores, definição de um sistema de comunicação, organização da memória e escolha de periféricos. Por exemplo, para um sistema multimídia deve-se definir os *codecs* (codificadores e decodificadores) utilizados e se serão implementados em hardware ou em um processador digital de sinais (DSP). Uma vez que a plataforma é definida, engenheiros de hardware trabalham a sua implementação, usando modelos em linguagem de descrição de hardware (HDL) para avaliar o seu funcionamento. A implementação das aplicações que irão rodar sobre a plataforma é baseada no mapeando das partes da aplicação para os componentes da plataforma selecionados para formar a arquitetura. Para acelerar o projeto do sistema desenvolveu-se uma nova metodologia baseada no projeto conjunto de hardware e software denominada de Hardware/Software Codesign. Desta forma, tanto o software como o hardware são projetados de forma paralela, por equipes diferentes, seguindo os requisitos definidos inicialmente no modelo da plataforma. O modelo pode ser uma descrição em alto nível do funcionamento do sistema, mas também pode ser um modelo mais completo que possibilite a execução e avaliação do funcionamento da aplicação. Para esses modelos foram criadas Plataformas Virtuais (PV) descritas em linguagem de alto nível baseadas em C/C++, como SystemC ou SpecC. A plataforma virtual é um modelo em alto nível dos componentes do sistema, capaz de executar o software a partir de um simulador do conjunto de

instruções. A desvantagem dessa abordagem é que alterações na PV devem ser feitas manualmente e, assim como no projeto baseado em plataforma, a definição da plataforma utilizada é uma premissa para o projeto do SoC.

A evolução no desenvolvimento de sistemas computacionais é a descrição em nível de sistema, seguindo uma metodologia baseada em modelos de descrição de elementos de processamento e elementos de comunicação. A aplicação é o ponto de partida do projeto do sistema, que gera uma descrição em alto nível orientada para os processos e elementos de comunicação. As interfaces entre esses elementos são bem definidas e seguem protocolos de comunicação de dados. Posteriormente, os processos são mapeados para elementos de software ou hardware e os elementos de comunicação são mapeados para barramentos, memórias, NoCs ou *crossbars*[7].

Keating (2011) faz uma discussão sobre novos paradigmas de projeto de sistemas eletrônicos, incluindo projeto e síntese em alto nível. Atualmente existe uma grande lacuna entre a descrição em alto nível e a síntese para silício. Isso também está reportado pelo ITRS, que afirma em seu último *roadmap* de semicondutores que o projeto em alto nível não fez avanços significativos na última década (ITRS, 2011a). Os principais aspectos que devem evoluir são as etapas de verificação e a escrita do software, que atualmente tem a maior parte do custo no projeto de circuitos integrados. Para muitos projetos baseados em integração de IPs, o esforço de verificação dos módulos de hardware é de 80% ou mais do esforço total do projeto (KEATING, 2011).

A síntese e projeto em alto nível têm como principais vantagens permitir que o projeto seja avaliado para diferentes arquiteturas, antes de ser implementado, e também simplifica o projeto, pois a quantidade de código é menor. Além disso, permite que a simulação seja feita com ou sem informações de temporização, reduzindo o tempo de simulação. A abordagem utilizada atualmente para a síntese em alto nível usando linguagens baseadas em C, como o SystemC, baseia-se na premissa de que é possível sintetizar um circuito RTL a partir de uma descrição C com informações de temporização. No entanto, isso nem sempre é possível para circuitos de alto desempenho pois para que a síntese ocorra de forma correta é necessária uma quantidade maior de informações de baixo nível do que o código C pode fornecer (KEATING, 2011). Em contrapartida, a descrição RTL apresenta uma quantidade de informações sobre temporização maior do que seria necessário para projetar e verificar o comportamento de um sistema complexo, mas necessárias para a sua implementação. O ponto ideal ainda não foi alcançado, que seria balancear as informações de temporização para conseguir implementar com eficiência circuitos operativos e de controle de alto desempenho. É preciso prover um ambiente de desenvolvimento que permita passar de uma descrição de alto nível (C não temporizado) para uma de baixo nível (como uma descrição em RTL) à medida que seja necessário para resolver um problema específico do projeto[8].

[7]Metodologia em nível de transações (TLM) está descrita por Gajski no livro "*Embedded System Design: Modeling, Synthesis and Verification*"(GAJSKI et al., 2009)

[8]Na obra "Closing the Abstraction Gap", cap. 10, onde o autor relata a partir de experiências de projetistas de hardware os obstáculos encontrados para sintetizar uma descrição a partir de linguagens de alto nível.

A avaliação das possibilidades de projeto para o sistema integrado deve partir do conhecimento prévio da aplicação alvo. Embora tal prática tenha como finalidade avaliar os requisitos funcionais e de implementação da aplicação, as limitações devem ser conhecidas para que a aplicação possa ser realizada em uma plataforma física. O projeto baseado em plataforma foi desenvolvido como uma solução para promover o reuso (SALEH et al., 2006). IPs devem ser verificados antes de serem integrados ao sistema. Com o processo de reuso, um IP, depois de usado e fabricado num projeto, poderá ser qualificado como "Silicon Proved", aumentando o grau de precisão sobre suas características. De qualquer forma, sempre será necessário incluir um conjunto de vetores de teste para validar a fabricação do circuito. Usando esse conceito, o projeto deve ser segmentado em partes menores (subsistemas) para reduzir o nível de complexidade de cada módulo do projeto em fase de desenvolvimento.

1.1.5 Plataformas de Processamento Multimídias

O aumento do número de elementos de processamento que compõem o SoC beneficiou diretamente a implementação de aplicações multimídia. A crescente integração de funcionalidades em um sistema de computação tira proveito da replicação das arquiteturas de processamento, obtendo uma maior paralelização em nível de tarefas. O número de processadores que formam um sistema multimídia tende a aumentar, como se pode observar pela evolução das arquiteturas (KOLLIG; OSBORNE; HENRIKSSON, 2009), (WOLF; HENRIKSSON, 2008), (BORKAR; CHIEN, 2011). Além disso, plataformas contendo um menor número de processadores e diversos aceleradores de hardware tem melhor compromisso de desempenho, área e potência (FULLER; MILLETT, 2011), (HENKEL, 2003). Os elementos de processamento são aceleradores de hardware (ou IPs) projetados para uma função específica que são integrados à plataforma. Aceleradores de hardware tem melhor desempenho para uma tarefa específica do que um processador de uso geral. Como uma previsão, o poder de processamento dos SoCs portáteis tende a aumentar em torno de 1.000 vezes nos próximos 10 anos e o número de elementos de processamento irá alcançar 2.000 elementos de processamento de dados (ITRS, 2011b)[9].

Aplicações multimídia são caracterizadas por processar informações em partições ou blocos de dados, realizando iterações sobre uma mesma região de dados até passar para a região seguinte. Um exemplo disso é o processamento de imagem, onde cada figura é segmentada em regiões retangulares de pixels e as regiões são processadas quase que de forma independente. No processamento de áudio, como em um decodificador AAC, o processamento é feito sobre cada trecho de áudio. Após o processamento nas regiões, cada parte é armazenada formando uma sequência de áudio armazenada em uma memória antes de ser transmitida ou enviada para um receptor. Esses dados armazenados em memória podem ser utilizados novamente como referência para o processamento de outros blocos de dados.

[9]Nesse relatório, o ITRS refere-se à elementos de processamento de dados (DPE – Data Processing Elements) como macro-células de computação de dados. Tais dados foram retirados do relatório *System Drivers*, na seção "*SoC consumer portable processing performance trends*", edição de 2011, do ITRS.

Plataformas multimídia de alto desempenho são construídas utilizando uma combinação de memórias locais e memórias compartilhadas. Memórias locais são distribuídas pelos elementos de processamento, para o processamento em blocos, e memórias compartilhadas são acessadas por diversos elementos de processamento simultaneamente. Memórias SRAM são utilizadas como memórias locais com capacidade e dimensões (número de endereços e número de bits de dados) suficientes para suportar o processamento do módulo ao qual está conectada. As memórias SRAM são usadas como memórias *scratchpad* ou memórias *cache* de uma CPU. As memórias compartilhadas têm maior capacidade do que as memórias locais dos elementos de processamento e são implementadas utilizando SRAMs ou DRAMs. A última é preferida nos sistemas multimídia, pois possui maior capacidade de armazenamento com menor custo em comparação com memórias estáticas.

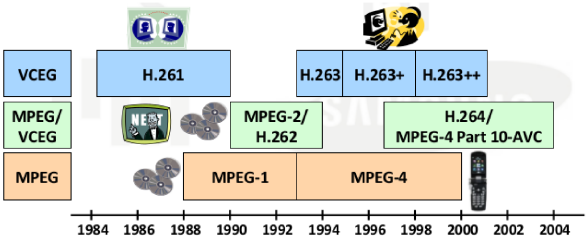
Como exemplo de aplicações multimídia, podemos considerar a estrutura de um Codec de vídeo nos padrões MPEG-2, MPEG-4, H.264/AVC e HEVC. No padrão MPEG-2, os quais as tarefas de maior demanda computacional são a estimação de movimento e a transformada discreta de cossenos (DCT). A estimação de movimento utiliza a correlação em duas dimensões com partes de imagens codificadas anteriormente, armazenadas em uma memória de quadros. O cálculo da correlação é relativamente simples, embora seja de grande intensidade computacional, pois esta operação é repetida inúmeras vezes até que seja encontrada a melhor posição. A transformada de cossenos é composta por uma série de multiplicações e somas. Nos algoritmos de processamento de vídeo, os requisitos de acesso à memória são críticos para o funcionamento em tempo real da aplicação. As sequências de vídeo formadas com imagens em alta definição aumentam a complexidade computacional dos algoritmos de codificação e decodificação de vídeo.

Percebe-se que a complexidade dos algoritmos aumenta a cada nova geração de padrão de codificação de vídeo. O incremento da eficiência de compressão é obtido às custas da maior complexidade computacional dos algoritmos. A complexidade computacional no padrão H.264/AVC é duas a três vezes maior do que no padrão antecessor, o H.263 (HOROWITZ et al., 2003). Para o padrão H.264, atualmente o padrão de imagem utilizado no sistema de televisão digital brasileiro, a tarefa de maior complexidade é a estimação de movimento (HOROWITZ et al., 2003). A evolução entre os padrões de codificação de vídeo tende a uma capacidade maior de compressão dos dados, mantendo boa qualidade perceptual da imagem. O padrão H.264/AVC foi desenvolvido entre 1999 e 2003, quando foi publicada a primeira versão da sua norma pela ITU-T (ITU-T, 2003). Esse padrão é amplamente utilizado para transmissão de vídeo em alta definição (HD), transmissão de sinais de TV via satélite, cabo e sistemas de transmissão terrestre. Além disso, é usado na aquisição de conteúdo de vídeo e sistemas de edição, câmeras de vídeo, aplicações de segurança, Internet e rede móvel de vídeo, discos Blu-ray e aplicações de conversação em tempo real, tais como chat de vídeo, videoconferência e sistemas de tele-conferências. A Figura 1.2 mostra a evolução histórica dos padrões de codificação de vídeo e uma comparação entre a capacidade de compressão da informação e a qualidade da

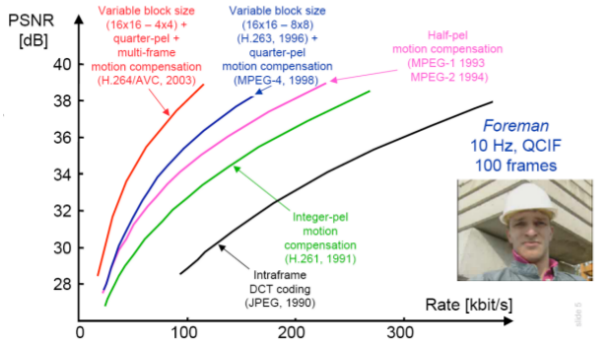
imagem (PSNR).

Figura 1.2: Evolução da qualidade subjetiva dos codecs de video comparada a capacidade de compressão da informação.

(a) Evolução histórica dos codecs de vídeo.



(b) Qualidade subjetiva da imagem para codecs de vídeo.



Fonte: (SZE; BUDAGAVI, 2014).

O padrão de codificação de vídeo de alta eficiência, denominado de (*High Efficiency Video Coding*) (HEVC), é o mais recente padrão de codificação de vídeo gerado pelas organizações de normatização *Video Coding Experts Group* (VCEG) da ITU-T e *Moving Picture Experts Group* (MPEG) da ISO/IEC. As duas equipes trabalharam em conjunto para formar o grupo *Joint Video Coding* (JCT-VC). A primeira edição da norma HEVC foi publicada em abril de 2013 (ITU-T, 2013). Esse novo padrão prevê uma série de aplicativos adicionais com maior precisão de codificação e suporte para diferentes formatos de cor, codificação de vídeo escalável e codificação de vídeo 3-D, estéreo e multi-vistas (SULLIVAN et al., 2012). O padrão HEVC foi projetado para, além de atender todas as aplicações existentes para o H.264/AVC, permitir o aumento da resolução de vídeo e aumento do uso de arquiteturas de processamento paralelo (SULLIVAN et al., 2012).

A complexidade computacional do novo padrão HEVC é maior em relação ao seu predecessor, o padrão H.264. O padrão de codificação de vídeo HEVC tem como objetivo proporcionar o dobro da eficiência de codificação em relação ao padrão H.264/AVC, oferecendo a mesma qualidade de vídeo com metade da taxa de bits (BOSSEN et al., 2012). Bossen et al. (2012) referem que a complexidade dos codificadores HEVC não parece ser significativamente diferente em relação aos codificadores H.264/AVC. No entanto, a complexidade computacional do codificador HEVC é superior à de um codificador H.264/AVC. A complexidade de computação para codificar imagens de alta definição em tempo-real leva à implementações de arquiteturas dedicadas, contendo diversos elementos de processamento implementados em Hardware e diversos níveis de memória para gerenciamento dos dados enquanto ocorre o processamento. Um exemplo de implementação para silício de um codificador HEVC é apresentado por Tsai et al. (2013). Esse chip foi fabricado usando processo CMOS TSMC 28 nm 1P10M (1 polisilício

e 10 camadas de metal), com 8350 mil portas lógicas ocupando uma área de 25 mm^2. Para este chip foi medido um consumo de potência de 708 mW rodando em 312 MHz para codificar vídeos progressivos de tamanho 8192x4320 em 30 quadros por segundo. Uma hierarquia de memória interna ao chip foi implementada contendo três níveis de memórias do tipo SRAM, com capacidade total de armazenamento interno ao chip igual a 7,14 Megabytes e largura de banda total das memórias externas igual a 2,97 GB/s (giga-bytes por segundo), em dois canais de memória independentes.

Aplicações multimídia são de grande complexidade, geralmente empregando um ou mais processadores, elementos de processamento dedicado, memórias locais e distribuídas pelo SoC. As aplicações multimídia são executadas sobre plataformas avançadas, como se pode observar através de implementações industriais baseadas em plataformas heterogêneas e multiprocessadas. Um exemplo de MPSoC concebido para aplicações é a plataforma Phillips Viper Nexperia (DUTTA; JENSEN; RIECKMANN, 2001), que inclui um processador MIPS e um processador VLIW Trimedia. Outro exemplo de plataforma MPSoC para multimídia, com aplicação principal em telefones móveis, é a plataforma de hardware-software OMAP da Texas (CUMMING, 2003). A plataforma OMAP contém diversas implementações, mas todas são baseadas na integração de um processador ARM, atuando como mestre da plataforma, e um DSP para realizar as tarefas de processamento de sinais. Seguindo a linha de telefones móveis, outra plataforma multiprocessada desenvolvida para esse tipo de aplicação é a Nomadik, da STMicroelectronics (PAGANINI, 2007). Nomadik é uma família de sistemas em silício desenvolvidos para processamento de aplicações multimídias. Existem diferentes modelos para essa plataforma, mas genericamente é baseada em um processador ARM rodando à 334 MHz e com 1.000 MIPS (*Mega Instructions per Second*) e aceleradores de áudio e vídeo. Da parte de vídeo, a plataforma suporta processamento gráfico em duas e três dimensões, decodificação de vídeo no padrão H.264/AVC no formato 640x480 em 30 quadros por segundo para televisão portátil. Além disso, faz a captura e codificação de sequências de imagens no formato JPEG, na taxa de 30 imagens por segundo. O módulo de aceleração de áudio suporta padrões MP3, AAC, AAC+, WMA, além de realizar a redução de ruído e cancelamento de eco. As aplicações multimídias orientadas para jogos demandam plataformas de alto desempenho, como o processador Cell da IBM (KAHLE et al., 2005) e o XBox 360 (ANDREWS; BAKER, 2006).

A comunicação entre os elementos de processamento de uma plataforma multimídia é implementada geralmente utilizando-se barramentos ou redes intra-chip, denominadas de NoC (*Network on Chip*). Uma especificação importante no projeto de plataformas multimídia que influencia na definição do sistema de comunicação é o sincronismo entre os elementos da plataforma. Geralmente, uma plataforma multimídia multiprocessada e heterogênea contém diversas regiões de relógio, pois ela contém elementos de processamento intensivo e elementos de menor capacidade. A comunicação entre módulos com diferentes relógios pode ser feita utilizando-se FIFOs ou um protocolo assíncrono de comunicação de dados. Os periféricos utilizados em plataformas multimídia são os mais variados e dependem da finalidade para qual a plataforma

foi criada. Podemos encontrar controladores de E/S como entradas e saídas de vídeo em alta resolução como DVI e HDMI, controladores de comunicação como USB, Ethernet, WIFI, Bluetooth, além de controladores de cartões de memória Flash e SSD e controladores de áudio SPDIF, etc.

1.1.6 Fluxo de Projeto Centralizado em Memória

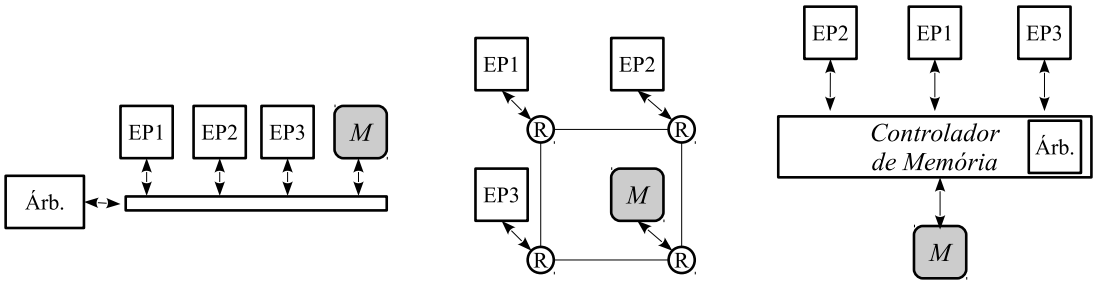
Os sistemas computacionais de elevada capacidade de processamento paralelo, como as plataformas para processamento multimídias citadas na seção anterior, utilizam-se de um canal de acesso aos dados compartilhado com elevada largura de banda. Nesses sistemas, o projeto dedicado de interfaceamento e controle de acessos é importante para obter-se uma melhoria da capacidade de processamento. O fluxo de projeto centralizado em memória, denominado de *memory-centric desing*, é orientado para a implementação de melhorias ao projeto do sistema orientadas ao uso de memória pela plataforma alvo. Tais melhorias podem ser implementadas tanto em nível de modificações em alto nível na descrição algorítmica da aplicação quanto em nível de implementação da arquitetura da plataforma. A organização e otimização da memória são descritas amplamente por Catthoor et al. (1998), definindo a metodologia de *memory-centric design*, que tende à uma especialização do uso da memória em sistemas computacionais.

O projetistas de arquiteturas e aplicações para uma plataforma são motivados a considerar como os dados são organizados nas aplicações, permitindo melhoras significativas de potência e desempenho (NEWMAN, 2014). Portanto, quando se trata da integração de EPs dentro de um SoC, é preferível que o projetista deste EP leve em conta considerações sobre a interface de acesso com a memória principal e considere as transações que são realizadas. Estruturas como memórias *cache* são projetadas com interfaces otimizadas para acessos em rajadas de dados, levando em conta o acesso de bancos, linhas e colunas existentes nas memórias DRAM externas. É conveniente que os dados sejam organizados em linhas e páginas de memória *cache* armazenadas de forma adequada às características da memória externa (JACOB; NG; WANG, 2007) (CUPPU; JACOB, 2001). O cuidado no ajuste dos parâmetros de funcionamento de um sistema feito pelo projetista influencia significativamente no seu desempenho.

Uma análise da eficiência de estruturas de comunicação como barramentos ou NoCs é apresentada em (SEICULESCU et al., 2011), mostrando que tais estruturas podem não ser eficientes o suficiente para integrar EPs heterogêneos com a memória principal. Como a sua eficiência é limitada em termos de latência, largura de banda, área e potência dissipada, o fluxo de projeto centralizado em memória é uma alternativa para atingir a qualidade de serviço desejada. A qualidade de serviço oferecida pelo sistema de comunicação deve considerar diferentes aspectos funcionais dos elementos que constituem o SoC, e isso deve ser feito a partir de uma avaliação de funcionamento da memória compartilhada pelos EPs. No estudo apresentado por Seiculescu et al. (2011), o projeto de uma NoC com topologia de interconexões centralizadas em uma DRAM traz benefícios para o desempenho da plataforma. Considerando-se diferentes topologias de interconexão entre EPs e memória, a Figura 1.3 mostra uma comparação entre

topologias utilizadas para interconexão de elementos do SoC.

Figura 1.3: Comparativo entre estruturas de interconexão: a) barramento, b) NoC e c) múltiplos clientes.



a) Barramento b) Rede em Chip c) Múltiplos clientes

Fonte: elaborado pelo autor.

A estrutura dedicada para interconexão com compartilhamento da memória principal é feita através de um sistema com múltiplas interfaces de acesso, denominado de multi-cliente. Nesta proposta, a interface entre a memória externa e os EPs é feita através de um controlador de memória externa (CME) com interfaces de acesso independentes para cada EP. O CME é implementado como um elemento ativo do sistema, não somente como um elemento passivo que interpreta comandos gerados pelos EPs e realiza as transações de escrita e leitura para a memória.

1.2 Objetivos

Esta tese de doutorado tem como objetivo principal propor um método de gerenciamento das transações entre os elementos que compõem o SoC e a memória externa de forma que a execução das transações seja otimizada em termos de largura de banda e de prazo de conclusão. Tais elementos devem fornecer informações sobre os tipos de acesso aos dados que são usadas pelo sistema de gerenciamento de memória de forma a melhor organizar as transações. O método proposto tem como finalidade melhorar o desempenho de um sistema computacional através do controle da latência do acesso aos dados. Para tal fim, é apresentado um fluxo de projeto centralizado em memória, denominado de *memory-centric design*, que tem como princípio fundamental o casamento de projeto da interface de comunicação dos elementos do SoC com a interface de acesso aos dados armazenados em memória. Esta tese propõe uma arquitetura de controlador de memória multi-clientes implementado a partir de um fluxo de projeto centralizado em memória.

O fluxo de projeto centralizado em memória proposto baseia-se na construção de sistemas de processamento que gerem acessos ao subsistema de memória passando informações complementares de acesso, além do endereço alvo e tipo de requisição (se escrita ou leitura). A eficiência dos acessos é gerenciada pelo controlador do subsistema de memória, garantindo requisitos de tempo-real mesmo com uma implementação de sistema formado por módulos he-

terogêneos. A comunicação de dados entre os elementos de processamento de uma mesma plataforma e a memória externa é uma operação que interfere mutuamente no desempenho e na potência dissipada.

Os objetivos deste trabalho são descritos abaixo:

- Desenvolver um fluxo de projeto centralizado em memória para implementar o gerenciamento dos acessos considerando as características de funcionamento dos EPs do sistema e as características de funcionamento da memória externa;
- Garantir a escalabilidade das interfaces de clientes, para os EPs do SoC, em tempo de projeto;
- Implementar um mecanismo de gerenciamento dos acessos gerados pelos EPs a partir de um árbitro com função adaptativa baseado nas características de acesso nas interfaces de clientes;
- Garantir que clientes com comportamento heterogêneo (entre sí e ao longo do tempo) tenham seus acessos à memória realizados dentro dos prazos solicitados, para satisfazer requisitos de tempo-real.

Portanto, a estrutura de comunicação deve ser projetada de forma escalável, com número de interfaces e conexões que possa ser alterado de acordo com o número de EPs. O sistema de memória deve ser capaz de integrar elementos de processamento com características diferentes, com suporte para aplicações com diferentes padrões de acesso aos dados. A plataforma alvo para esta implementação é composta de elementos de processamento heterogêneos. Neste contexto, a heterogeneidade significa que transações à memória são geradas pelos elementos de processamento em intervalos de tempo diferentes e com tamanhos e prazos diferentes.

O CME avalia a característica dos acessos aos dados e das transações que são solicitadas pelos EPs para gerenciar da melhor forma as transações com a memória. O controle da latência, o ajuste de granularidade dos acessos e a organização das transações de dados para a memória são avaliados em tempo de execução de forma a satisfazer a Qualidade de Serviço (QoS).

Por fim, como a memória é um recurso compartilhado por diversos elementos do sistema, diferentes aplicações competem por largura de banda do canal de memória, gerando interferência entre os acessos simultâneos (concorrência). Além disso, a memória tem capacidade e largura de banda limitadas. A previsibilidade de comportamento dos acessos gerados pelas aplicações que disputam o recurso compartilhado é fundamental para garantir que o controle dos acessos melhore o desempenho da memória, e portanto, de todo o sistema. Portanto, o sistema de memória deverá ter características funcionais previsíveis, como latência e largura de banda, para que os acessos sejam gerenciados de forma previsível, evitando a interferência entre os clientes concorrentes. Garantir a previsibilidade também permite que seja possível implementar modelos que determinem qual é o melhor ajuste para o subsistema de memória, permitindo avaliar corretamente o comportamento do subsistema de memória para um número

crescente de elementos de processamento que acessam a memória. A partir do conhecimento do funcionamento do subsistema de memória, é possível administrar os casos de latência e de largura de banda para cada uma das transações geradas por EPs, permitindo atendê-las dentro das restrições críticas de tempo-real.

1.3 Contribuições e Inovação

A proposta inovadora apresentada aqui é a implementação de um subsistema de memória com características de adaptação que avalie em tempo de execução as características de acesso dos EPs conectados ao subsistema de memória e determine a melhor forma de implementar as transações. O subsistema de memória adapta-se às diferentes condições de funcionamento dos EPs usando informações de cálculo dos piores casos de execução das transações e de prazos de conclusão. Esta proposta supõe que o comportamento do sistema seja previsível, dentro dos limites da memória e dos elementos internos do controlador de memória. O gerenciamento dos acessos e a regulação das transações para a memória são feitos de forma adaptativa, baseado em funções de estimativa de desempenho calculadas em tempo de execução. A latência de início das transações é a variável gerenciada pelo árbitro para otimizar os acessos à memória: a latência é flexibilizada para permitir a finalização de acessos em curso desde que que os deadlines sejam garantidos. A referida flexibilização permite minimizar a latência global dos acessos e melhorar a largura de banda disponível para os clientes.

A principal contribuição é a descrição de um fluxo de projeto centralizado em memória, que usa informações da memória alvo e das transferências solicitadas pelos clientes para garantir os requisitos de acesso dos EPs do sistema. O modelo de funcionamento do subsistema de memória é implementado neste trabalho, servindo como uma base de análise do comportamento do sistema a partir dos aspectos funcionais do conjunto de EPs que compõem o sistema e da memória principal. É proposto um modelo matemático baseado em atrasos que gera uma estimativa do pior caso de latência para os elementos de processamento do SoC, conectados à uma memória compartilhada do tipo DRAM. O tempo de resposta no pior caso (WCRT – *Worst-Case Response Time*) é usado como um parâmetro de avaliação do projeto e o atendimento aos "deadlines" é o controle de desempenho para o sistema.

Este trabalho também desenvolve a análise e implementação de um sistema de gerenciamento dos acessos à memória através de um árbitro adaptativo. Os diversos elementos de processamento interconectados ao subsistema de memória fornecem informações sobre os requisitos de tempo e tamanho da transferência de dados em cada acesso. Estes parâmetros são avaliados continuamente em tempo de execução do sistema gerando uma adaptação que garanta os prazos mínimos para completar as transferências solicitadas pelos clientes. É apresentada uma comparação entre árbitros tipicamente utilizados para gerenciar acessos a DRAM e o árbitro adaptativo proposto. Neste trabalho, diferentes algoritmos de arbitragem são avaliados para gerenciar o fluxo de dados pela estrutura de comunicação interna do controlador de memória.

Além disso, o dimensionamento das memórias internas e bufferização de dados são usados para controlar o fluxo de dados e garantir a qualidade de serviço para os clientes do controlador de memória. A taxa de acessos à memória principal é avaliada em cada interface de clientes e o controlador é capaz de ajustar em tempo de execução o melhor dimensionamento dos buffers internos e a repetição dos acessos, para garantir os requisitos de acesso de cada cliente. A configuração é feita de forma adaptativa, baseada na classificação prévia dos clientes em categorias de acordo com seus requisitos de funcionamento. Ao final deste trabalho é feita uma avaliação das características de previsibilidade, heterogeneidade, escalabilidade e adaptatividade do controlador de memória que é usado para validar o conceito do projeto centralizado em memória.

1.4 Estrutura da Monografia

Essa monografia de tese de doutorado está organizada da seguinte forma. O Capítulo 2 apresenta um detalhamento do problema de compartilhamento do canal de memória e os trabalhos relacionados. O Capítulo 3 apresenta os detalhes técnicos referentes às características das memórias utilizadas em SoCs, para servir como base técnica e teórica do trabalho. O Capítulo 4 apresenta a metodologia de trabalho utilizada para verificar a validade das ideias apresentadas neste trabalho de tese. Esse capítulo descreve a proposta do fluxo de projeto centralizado em memória, a implementação da arquitetura do controlador de memória com árbitro adaptativo e a implementação árbitro adaptativo que usa informações de prazo e tamanho para ordenar as transações. No Capítulo 5 é apresentada a análise dos resultados e o estudo de caso utilizado para validar as ideias a partir de uma implementação real de sistema multimídia. Conclusões finais, comentários e proposta de futuros trabalhos são apresentados no Capítulo 6.

2 DESCRIÇÃO DO PROBLEMA

Este Capítulo descreve o problema relacionado ao compartilhamento de acessos à memória aprofundado neste trabalho e que guia a busca por uma solução para integração de um SoC. É avaliada a hierarquia de memórias de um sistema computacional com diversos módulos heterogêneos integrados no mesmo chip. Primeiramente, é apresentado o estudo do compartilhamento do canal de memória em um sistema com concorrência de acessos e as possíveis formas de arbitragem. Em seguida é analisada a forma de integração de memórias em SoCs, que utilizam uma hierarquia de memória para reduzir o tempo de acesso aos dados armazenados em memória. Além disso, a localidade das informações e o uso de memória distribuída são estudadas com vistas a avaliar sua influência nos processos de acesso à memória. Ao final do Capítulo são apresentados os trabalhos relacionados.

2.1 Visão Geral do Problema

A otimização do canal de acesso à memória é um dos fatores chave para o aumento da eficiência de um sistema computacional. Nos SoCs multimídia esse fator é ainda mais preponderante, visto que grandes quantidades de informação são armazenadas em memória para serem processadas pelos módulos. O problema abordado nesta tese é a implementação de um subsistema eficiente de acesso à memória capaz de satisfazer os requisitos de acesso em tempo real das aplicações rodando sobre uma plataforma SoC. A eficiência dos acessos à memória é o que determina o desempenho computacional de um sistema e pode tornar-se o gargalo de processamento. A cada geração de produtos de eletrônica de consumo, novas funcionalidades são agregadas, resultando em um aumento da demanda por capacidade de processamento e de armazenamento de informações. A qualidade dos gráficos aumenta a cada nova geração de produtos, exigindo uma taxa de dados cada vez maior para gerenciar a quantidade de informação necessária em tempo-real. Além disso, com o aumento da demanda por produtos portáteis e também por produtos ecologicamente corretos (*green engineering*), a eficiência energética é outro ponto chave para o projeto de um sistema de memória eficiente.

2.1.1 Memória em Sistemas Digitais

Como o aumento na densidade de integração de vários EPs para um SoC, o sistema de memória torna-se preponderante para determinar o desempenho final, área e consumo de energia. O projeto do subsistema de memória envolve vários aspectos como o projeto dos níveis de memória locais e compartilhadas, o controle de memórias externas, a melhoria da eficiência dos acessos, do aproveitamento da largura de banda e o gerenciamento de dados.

Sistemas de processamento de dados de alto desempenho necessitam de uma larga banda de acesso à memória. A evolução dos SoCs tende a aplicações cada vez mais complexas computacionalmente, com uma maior diversificação de aplicações e maior qualidade de imagem e áudio. Isso tende ao aumento da capacidade de memória utilizada e aumento da velocidade de acesso aos dados. Nos sistemas multimídia, a capacidade do canal memória tende a ser um limitante para o desempenho de processamento devido às taxas elevadas de acesso aos dados armazenados durante as etapas de codificação e decodificação, fazendo do acesso aos dados armazenados o gargalo de desempenho. Um sistema com um processador acessando a memória pode tirar vantagem da propriedade da localidade de dados e instruções: uma pequena memória de acesso mais rápido pode conter os elementos que têm maior probabilidade de serem utilizados por estarem próximos ao contexto atual de processamento. Para tirar benefício da localidade, cria-se uma hierarquia de memórias a partir de níveis de memória de diferentes tamanhos e velocidades. As memórias mais rápidas podem ter um sistema de endereçamento especial como as memórias do tipo *cache*, com interface rápida e menor tempo de acesso. O desempenho dos processadores rápidos é limitado pela velocidade de acesso aos dados armazenados em memória. A hierarquia de memória reduz esse problema pela disponibilidade de uma memória mais rápida (e de maior custo por bit) conectada por um caminho rápido ao processador, utilizada para armazenar dados e instruções utilizados com maior frequência, e de memórias com maior tempo de acesso e de menor custo para armazenar quantidades maiores de dados.

Duas tecnologias de memória são utilizadas amplamente no projeto de sistemas de armazenamento de dados para formar uma hierarquia de memória: a memória dinâmica de acesso aleatório (do inglês *Dynamic Random Access Memory* ou DRAM) e a memória estática de acesso aleatório (*Static Random Access Memory* ou SRAM). O projeto de SRAMs é orientado para maior velocidade, enquanto que o projeto de DRAMs objetiva menor custo por bit e maior capacidade. A Tabela 2.1 mostra um comparativo entre capacidade de armazenamento, custo e energia dispendida para memórias estáticas e dinâmicas (JACOB; NG; WANG, 2007). Uma hierarquia de memórias é utilizada para aproveitar da melhor forma as diferentes características das memórias existentes em circuitos de computação. A criação de hierarquia de memórias para sistemas de processamento ganhou importância com a evolução dos processadores. A Figura 2.1 ilustra uma projeção histórica do desempenho de processadores comparado ao tempo gasto no acesso à memória principal.

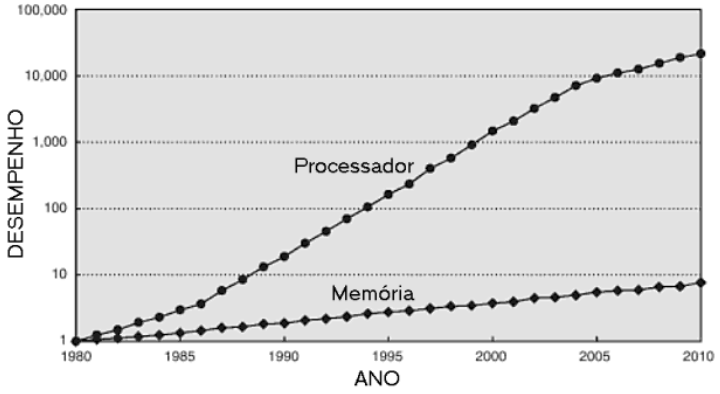
Nas sucessivas gerações de DRAM, a largura de banda do barramento de dados tem aumentado significativamente, resultando em taxas elevadas de transferência de dados por segundo.

Tabela 2.1: Custo e desempenho para diferentes tecnologias de memórias.

Tecnologia	*Latência por Acesso (ns)*	*Custo por Megabyte (U$)*	*Energia por Acesso*
SRAM on-chip	0,1	1 – 100	1 nJ
DRAM externa	10 – 100	0,1	1 – 100 nJ
Disco	> 1000	< 0,001	100 – 1000 mJ

Fonte: (JACOB; NG; WANG, 2007).

Figura 2.1: Diferença de velocidade entre processador e memória.



Fonte: (HENNESSY; PATTERSON, 2006).

Recentes tecnologias de memórias, como DDR2 e DDR3, apresentam um aumento da largura de banda médio igual a 5% ao ano, como reporta o estudo feito por Wolf e Geuzebroek (2011). No entanto, o mesmo estudo referencia que a latência de acesso aos dados reduziu a uma taxa média de 3% ao ano. A pequena melhora no tempo de acesso (para linha e coluna) é de alguns nanossegundos, reduzindo de 54 ns nas DDR2 para 46,13 ns nas DDR3 vendidas comercialmente nos dias atuais. Considerando o uso de memórias cache e DRAM, a velocidade de acesso tem aumentado cerca de 7% ao ano. Enquanto isso, o desempenho dos processadores tem evoluído 25% ao ano até 1986, 52% ao ano até 2004 e 20% ao ano nos dias atuais (HENNESSY; PATTERSON, 2006). A taxa de aumento da velocidade dos microprocessadores excede a taxa de aumento da velocidade das memórias DRAM. Ambas aumentam exponencialmente, mas a expoente dos microprocessadores é maior do que a das memórias, efeito que causa a degradação de desempenho do sistema sendo denominado de “Memory Wall” (WULF; MCKEE, 1995) ou “Memory Gap” (WILKES, 2001). Essa diferença faz com que o gargalo de processamento de um sistema digital esteja relacionado à memória. Uma solução para esse problema seria desenvolver uma tecnologia de memória que tivesse uma escala de velocidade comparável à dos microprocessadores. Como ainda não é possível tal tecnologia, a alternativa seria reduzir ao máximo o número de perdas da memória *cache* e maximizar a probabilidade dos acertos para assim reduzir a latência no acesso aos dados. No entanto, nem todos os processos em um sistema têm amplo paralelismo ou localidade dos dados para esconder ou minimizar a latência de acesso.

As demandas de aplicações para o espaço de memória continuam a crescer, enquanto a diferença de capacidade entre caches de último nível e da memória principal ainda é grande. Consequentemente, reduzir a latência da memória principal ainda é o fator primordial para melhorar o desempenho de processamento. Sistemas computacionais modernos exigem um maior desempenho e capacidade das memórias dinâmicas de acesso aleatório para atender aos requisitos de desempenho do sistema. Como a Lei de Moore (MOORE, 1998b) prevê, a tecnologia CMOS caminha para o regime de nanoescala profundo, as memórias DRAM enfrentam sérios desafios de velocidade, largura de banda, capacidade e custo. A evolução das memórias DRAMs está relacionada à capacidade de armazenamento por pastilha de silício e à velocidade de acesso aos dados na interface da memória. Além disso, as latências de acesso aleatório e os tempos de ciclo interno não estão diminuindo à mesma taxa que o tempo de ciclo do microprocessador.

A diferença entre o desempenho de processadores e memórias é resolvida atualmente com o uso de uma hierarquia de memória para explorar a localidade da informação. Assim, uma parte dos dados usados pelo processador está armazenada na memória mais próxima, com velocidade maior de acesso. A hierarquia de memória é formada por diversos níveis de memória, formando uma estrutura de armazenamento que parece ser redundante, mas provê ganho em desempenho ao custo de área de silício. Borkar e Chien (2011) explicam que a razão para o lento desenvolvimento de memórias DRAM não é um fato tecnológico, mas sim prático, pois o avanço em hierarquias de memórias tiraram o foco no desenvolvimento da velocidade das memórias. O mercado de DRAMs busca o aumento da capacidade de armazenamento e redução do custo, em vez de melhorar o desempenho das memórias. Esse fato certamente pode ter comprometido, e continuará frustrando, grande parte do progresso obtido de desempenho dos processadores (BORKAR; CHIEN, 2011). O aumento do número de transistores por pastilha de silício é direcionado para memórias *cache* de grande capacidade. A quantidade de memória, assim como a quantidade de lógica, dobrará a cada novo nó tecnológico (ITRS, 2011b).

O uso de memórias externas (*off-chip*), além de permitir escalabilidade, faz-se necessário em sistemas onde a capacidade de memória interna é insuficiente. O acesso à memória externa também é um ponto crítico no desempenho do sistema rápido de processamento de dados. O projeto de uma interface de memória externa deve explorar o melhor desempenho possível para reduzir o tempo de acesso aos dados. Nesse aspecto, algumas técnicas conhecidas para melhorar o acesso aos dados são: o aumento da largura do barramento de dados e o aumento da velocidade da interface de dados; aumento da eficiência de algoritmos de memórias *cache*; aumento do tamanho de memória *cache on-chip* e reordenamento dos dados.

2.1.2 Integração de Memórias em Circuitos Digitais

O desempenho de um sistema computacional diretamente depende do desempenho do sistema de memória. A integração de memórias em circuitos digitais tem evoluído continuamente buscando melhorar o desempenho. As inovações em integração de memórias tem foco no aumento da capacidade de armazenamento e redução da potência e da latência.

Um microprocessador necessita de uma memória de acesso aleatório (RAM) para realizar o processamento de informações. O princípio de acesso aleatório é fundamental para o funcionamento dos computadores: tanto o código quanto os dados estão localizados em posições da memória que dependem dos próprios dados e do algoritmo em execução e, portanto, imprevisíveis em tempo de projeto. Arquiteturas de computadores podem ser classificadas de duas formas: Harvard e Von Newmann. Ambas requerem memórias de acesso aleatório para armazenar instruções e dados. Os computadores modernos são construídos seguindo a arquitetura Von Newmann, a qual define que instruções e dados podem compartilhar o mesmo espaço físico e, portanto, residem na mesma memória física. Os acessos são direcionados para essa mesma memória, que em um momento pode armazenar um dado e, no momento seguinte, uma instrução. O sistema de memória deve ser capaz de gerenciar o conteúdo armazenado em um espaço de endereçamento definido pela aplicação que está sendo executada. O programa de computador segue passos pré-definidos e ordenados, constituídos por instruções de programa que manipulam dados. Os dados podem interferir na ordem de execução do programa, assim como na definição dos próprios dados a serem processados.[1].

No entanto, o processo de computação de uma informação tende a ser feito de forma linear e com alto nível de predição. O princípio da localidade define que um programa tende a reutilizar a maior parte das instruções e dados usados recentemente, comprovado pelos estudos realizados por Denning e Schwartz (1972). Uma implicação da localidade é que é possível prever com uma boa aproximação quais instruções e dados um programa irá acessar no futuro próximo, com base em seus acessos em um intervalo de tempo passado recente. Um programa que ocupa diversas páginas na memória principal pode rodar de forma eficiente se apenas um subconjunto de suas páginas estiver localizado em uma memória local de acesso rápido. Esse comportamento observado da execução de aplicações sugere que uma memória de tamanho reduzido pode ser usada para um processamento eficiente no lugar de uma memória maior. A localidade da referência ocorre nas formas temporal, devido aos laços e execução em módulos com dados privados, e espacial, devido aos valores relacionados que são agrupados em vetores, matrizes, sequências e estruturas de dados (DENNING, 2005). A localidade temporal é definida como a tendência de um programa utilizar novamente itens do seu conjunto de instruções e dados durante a sua execução em um tempo futuro próximo. Já a localidade espacial refere-se à proximidade que os dados e instruções utilizados recentemente, ou futuramente, estão no entorno dos dados ou instruções acessadas no momento. Ou seja, durante a execução de um programa, os endereços acessados futuramente estarão localizados em uma região próxima do endereço acessado atualmente. Portanto, é possível reduzir o número de acessos à memória de instruções e dados, armazenando temporariamente a região de memória que o programa está acessando em uma memória de acesso rápido e de tamanho menor. Dessa forma, uma hierarquia de memórias

[1] Aqui utilizaremos o termo código para referenciar o código de programa armazenado em memória, que é composto por dados e por instruções para o processador. O termo programa será usado para definir o código que está sendo executado pelo processador, que realiza uma tarefa determinada por uma sequência de instruções e que acessa e processa dados.

é utilizada para atender da melhor forma a execução do programa, explorando adequadamente a localidade da referência para execução das aplicações em uma CPU. Para tanto é fundamental que a memória que contém dados e instruções seja rápida o suficiente para conseguir atender a velocidade esperada pelo processador. Um sistema de computação eficiente apresenta diversos níveis de memória, tipicamente um ou mais níveis de cache, DRAM e memória de massa (como discos magnéticos ou de estado sólido).

Observando esses aspectos, da localidade da referência, da velocidade de acesso e da capacidade de armazenamento, são projetadas as hierarquias de memórias. A hierarquia de memórias em um sistema de computação representa diferentes níveis de armazenamento da informação, os quais possuem características específicas para atender os requisitos de processamento. A escolha correta desses níveis define o desempenho de processamento do sistema. Sistemas de memória modernos são compostos por uma hierarquia de memórias que compreende, do nível inferior até o superior, memórias *cache* (SRAM), memórias DRAM e disco rígido. Sistemas computacionais de alto desempenho utilizam uma hierarquia de memórias para poder tirar benefícios de velocidade e de capacidade de armazenamento do sistema de memória, não de um único tipo de memória. Tipicamente, uma hierarquia de memória é formada por memórias rápidas e menores até memórias lentas e maiores (HENNESSY; PATTERSON, 2006), (JACOB; NG; WANG, 2007).

A memória *cache* (SRAM) provê acesso aos dados e instruções com latência muito baixa (maior velocidade) e elevada largura de banda. Memórias DRAM são o nível intermediário de memória, provendo acesso rápido aos dados e instruções e uma capacidade maior de armazenamento que memórias SRAM. O custo de implementação desses dois tipos de memórias é diferente. O custo por bit uma memória DRAM é menor do que o custo de uma memória SRAM. O disco rígido possui maior capacidade de armazenamento com custo relativamente menor, se comparado às DRAMs e SRAMs, mas o tempo de acesso aos dados é elevado e a largura de banda é menor.

Memórias feitas com tecnologia DRAM são fabricadas de forma diferente de portas lógicas e diferente de uma memória SRAM, portanto é feita em outra pastilha de silício, separada do circuito. A integração de múltiplas pastilhas de silício no mesmo encapsulamento é uma alternativa eficiente para integração memória-circuito em larga escala. O processo de fabricação de uma memória DRAM é diferente dos processos utilizados para fabricar uma memória SRAM e um circuito integrado. As memórias SRAM são fabricadas diretamente na mesma pastilha de silício que o circuito integrado, sendo usadas como memórias *cache* e *scratchpad*. Já memórias DRAM são utilizadas como memórias externas, pois são fabricadas em uma pastilha diferente, mas atingem uma escala maior de capacidade que uma SRAM com menor custo.

Atualmente têm-se explorado a integração de diferentes pastilhas de silício em um único encapsulamento, possibilitando interconexões de comprimento reduzido e baixa capacitância entre uma pastilha de memória e o SoC. As técnicas de integração de múltiplas pastilhas no mesmo encapsulamento conhecidas atualmente são: *3D stacking* e *System in Package* (SiP). A

técnica SiP não integra pastilhas em um único circuito, mas as pastilhas são colocadas dentro de um único encapsulamento e conexões externas ao chip são utilizadas para ligar as pastilhas. Já a técnica do *3D stacking* utiliza vias através do silício (*Through Silicon Vias* TSV) (XIE, 2013), (BORKAR; CHIEN, 2011), formando um chip em três dimensões.

O custo da integração de memórias externas ao circuito depende do número de pinos existentes entre os dois encapsulamentos, portanto, a redução do número de conexões entre memória e o SoC representa reduzir o custo de fabricação e de teste. Com a integração das pastilhas no mesmo encapsulamento, o custo de integração e teste foi reduzido e o número de conexões entre memória e circuito tende a aumentar. Esse aumento tem limite no número máximo de TSVs que podem ser inseridas na memória. O empilhamento 3D começa a ser visto como uma solução avançada para aumentar o desempenho de arquiteturas multi-processadas. Espera-se resolver com essa técnica alguns dos principais problemas existentes nos sistemas de processamento: elevada carga de trabalho na memória externa e latência; gargalo de E/S de dados; consumo de energia e comunicação. 3D-DRAM é a tecnologia estado da arte que está sendo proposta recentemente para reduzir a potência dissipada nos acessos aos dados pelo sistema de memória (MENG; KAWAKAMI; COSKUN, 2012).

WIDE-I/O é provavelmente a aplicação mais avançada de fabricação em 3-D. Apresenta elevada largura de banda de comunicação e demonstra melhorias em termos de eficiência de energia, escalabilidade e da tolerância para variabilidade. Sua adoção depende da introdução de novas interfaces DDR que são mais tradicionais em termos de modelo de negócio, pois utilizam um modelo conhecido e consagrado de acesso à memória. O aumento da largura de banda da memória é devido à integração de vários controladores de memória com um grande número de E/S e técnicas de sinalização de alta velocidade. No entanto, mesmo para projetos sofisticados de circuitos em larga escala, existe um limite para o número máximo de pinos disponíveis no encapsulamento, que é reduzido pela necessidade de alimentação dedicada e pinos de terra. Já no caso de memórias DRAM embarcadas (eDRAM), o limite de integração é dado pelo número de TSVs que a técnica de empilhamento permite. Com eDRAMs, o nível L3 de memória *cache* dos processadores modernos terá um aumento na capacidade de armazenamento e será colocada em outro "andar" do silício (ITRS, 2011b).

Olhando para tendências futuras de integração de memórias, o "muro de memória" pode ser minimizado através da capacidade de reunir grandes quantidades de memória no mesmo chip, utilizando para isso as técnicas de empilhamento, formando um cubo de memória monolítico, integrado ao sistema de processamento. Olhando para esse cenário, o grande problema do sistema será o gerenciamento dessa enorme quantidade de memória acessada por uma grande quantidade de elementos de processamento, através de uma rede complexa de interconexões. Além disso, haverá uma mudança de cenário de projeto, de sistemas com um número limitado de pinos para uma ou duas portas de memória DRAM externas para sistemas com diversos controladores de memória distribuídos pelo chip em conexão com amplos barramentos de dados para memórias externas.

2.1.3 Multiplexação da Informação

A multiplexação de dados na interface de memória é um problema que se tornará mais complexo a cada nova geração de tecnologia. Observando os apontamentos feitos na Seção 1.1, percebe-se que o número de elementos de processamento integrados no mesmo chip irá aumentar exponencialmente com a evolução da tecnologia. Já na Subseção 2.1.1, pode-se avaliar que o "muro de memória" é um problema que não está sendo resolvido de forma eficiente, com o aumento de tamanho das memórias *cache* e o aumento de níveis de *cache*. A Subseção 2.1.2 mostra as inovações existentes em formas de integração entre processador e memória. É necessário melhorar o desempenho dos sistemas de memória nos sistemas computacionais para tirar proveito completo da evolução dos processadores e elementos de processamento. O problema da multiplexação da informação é um problema que terá complexidade crescente com o aumento de elementos de processamento nos sistemas multimídia.

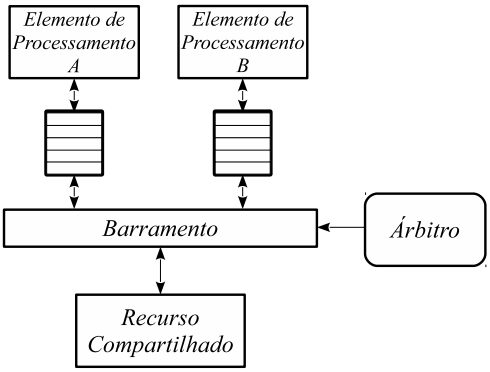
Os conflitos de acesso determinam a eficiência do canal de memória, e por consequência, a eficiência de processamento do sistema. Os conflitos no acesso são responsáveis pelo aumento da latência dos dados e pela redução da largura de banda do canal de memória. A latência excessiva na transferência de dados da memória é responsável pela redução da capacidade de processamento dos módulos do SoC. Uma hierarquia de memória é necessária para otimizar os acessos à memória principal do sistema e para garantir que todos os processos tenham suas transferências de dados finalizadas em tempo hábil para que possam executar o processamento das informações em tempo real.

Similar ao "muro de memória" e "muro de potência", os sistemas multimídia atuais encontram o problema do "muro de largura de banda". O crescimento do número de núcleos no mesmo circuito encontra como principal barreira a largura de banda da memória externa ao chip. Rogers et al. (2009) mostram a partir de um modelo analítico um estudo do "*bandwidth wall*" e apontam como solução o uso de *caches* feitas com memória eDRAM, uso de técnicas de compressão de dados e redução do tamanho dos núcleos de processamento. Somente dessa forma será possível aumentar o número de núcleos e acompanhar a evolução tecnológica.

Recursos compartilhados em um sistema de computação, como memórias e periféricos, são acessados por múltiplos elementos de processamento de forma concorrente. Quando há compartilhamento da informação, os pedidos de acesso aos dados são processados em tempo de chegada ao elemento compartilhado, respeitando algum critério de classificação ou prioridade. Os demais elementos que esperam pela permissão permanecem bloqueados, e as informações de acesso ficam retidas em memórias temporárias (denominadas de *buffer*). O compartilhamento das informações é gerenciado por um árbitro de barramento, e os dados e comandos trafegam através de uma estrutura de comunicação compartilhada entre todos os elementos do sistema. No caso de uma rede em chip, o recurso compartilhado é conectado à um dos nós da rede, que gerencia a comunicação de pacotes entre o recurso local e a rede, conectando aos demais elementos. Para ambos sistemas de comunicação, diferentes modelos de arbitragem podem ser utilizados, dependendo do tipo do sistema e da classificação dos acessos, como a multiplexação

por divisão de tempo, fila por ordem de chegada, Round-Robin ou prioridade com preempção. O árbitro deve ser justo para suprir as necessidades de acesso ao elemento compartilhado pelos elementos de processamento. A Figura 2.2 mostra a estrutura de compartilhamento de um recurso por elementos de processamento, controlada por um árbitro do barramento.

Figura 2.2: Diagrama de blocos que ilustra a estrutura de compartilhamento de um recurso em um sistema com múltiplos elementos de processamento.



Fonte: Elaborado pelo autor.

A velocidade de multiplexação do barramento entre diferentes elementos de processamento do sistema depende da velocidade do árbitro e do recurso compartilhado. Geralmente, a implementação de um árbitro de barramento permite que se tenha velocidade de ciclo para a troca no barramento. Dessa forma, a velocidade de troca no barramento depende da velocidade do recurso compartilhado. Quando esse recurso é uma memória DRAM, a velocidade está diretamente relacionada à largura de banda da memória e inversamente relacionada à latência dos dados.

2.2 Trabalhos Relacionados

Essa seção apresenta uma revisão dos trabalhos encontrados na literatura que representam atualmente o estado da arte na área de projeto de subsistemas de memória. O desenvolvimento de modelos de sistemas de memória e a implementação de arquiteturas mais eficientes são amplamente explorados na literatura atual. Em sistemas de processamento multimídias, questões relacionadas à redução da latência de acesso à memória principal, à otimização dos acessos, ao aumento da largura de banda da porta de memória e avaliação da eficiência energética são tópicos atuais de pesquisa. Essa seção primeiramente faz uma análise do estado da arte no desenvolvimento de sistemas de memória para SoCs. Após é feita uma análise sobre os tópicos relacionados que ainda podem ser explorados em novas pesquisas.

No momento em que os sistemas de computação migraram para plataformas multiprocessadas embarcadas no mesmo chip, com o surgimento dos MPSoCs, os projetistas perceberam a necessidade de controlar a latência dos acessos de uma forma mais elaborada do que simplesmente utilizar estruturas de memória cache. Wolf (2004) destacou que uma estratégia de

projeto sutil e que poderia levar mais plataformas multiprocessadas a ter um melhor desempenho é o gerenciamento de buffers e memórias. Sistemas heterogêneos de memória, projetados especificamente para atender de forma eficiente o conjunto de elementos que forma o MPSoC, atenderiam de forma mais eficiente os requisitos de desempenho e potência. Um problema no projeto de sistemas de computação embarcada é encontrar arquiteturas de memória flexíveis o suficiente para suportar requisitos heterogêneos de tempo real (WOLF, 2004).

Arquiteturas heterogêneas de processamento realizam as tarefas de forma mais eficiente do que arquiteturas homogêneas. Dessa forma, as plataformas multiprocessadas tendem a ser implementadas usando uma combinação de diferentes módulos, que leva a uma grande diversificação dos requisitos de processamento nas diferentes regiões da plataforma. Considerando isso, os acessos à memória têm diferentes características nas diversas partes da plataforma. Wolf e Henriksson (2008) apontam que uma plataforma composta de sistemas de memórias irregulares têm menor consumo de energia, além de atender de uma melhor forma os requisitos de tempo real da aplicação. Portanto, um sistema de memória heterogêneo possui melhor desempenho do que um sistema de memória homogêneo, baseado em estruturas de memória cache. A distribuição do sistema de memória permite que métodos de software encontrem e eliminem conflitos de memória, permitindo que os programadores determinem quais as tarefas que estão acessando os locais de memória para garantir o processamento em tempo real (WOLF; HENRIKSSON, 2008). Uma possível solução é apresentada por Seiculescu et al. (SEICULESCU et al., 2011), que propõem o uso de uma rede-em-chip dedicada para o tráfego de acessos à DRAM.

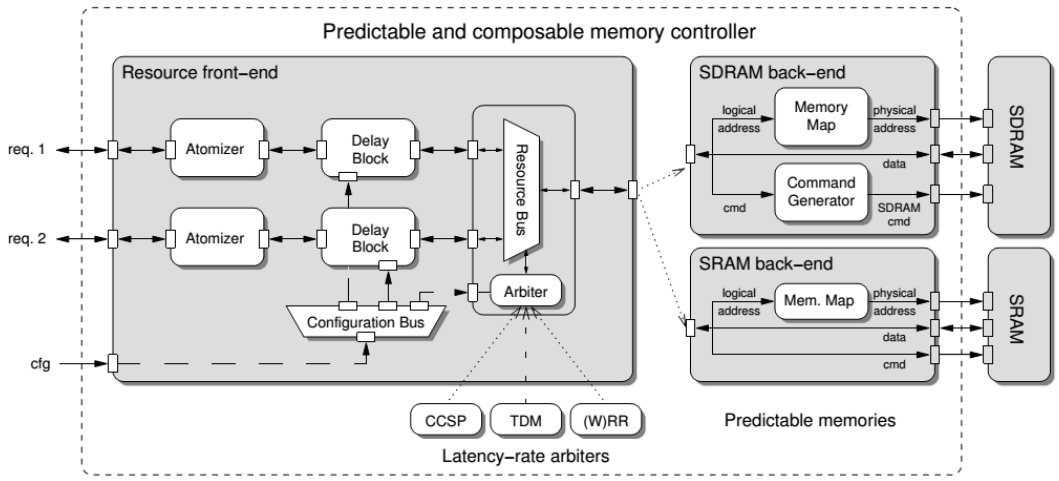
2.2.1 Controladores de Memória

Diversos estudos encontrados na literatura referenciam que o controladores de memória com comportamento predizível são necessários para minimizar os efeitos indesejados de variabilidade dos acessos em sistemas multi-processados. Sistemas previsíveis e combináveis têm sido estudados para resolver o problema da complexidade de comunicação em sistemas heterogêneos com elevado número de elementos de processamento.

O projeto de um controlador de memória que oferece uma largura de banda mínima garantida e uma latência máxima vinculado ao IPs é apresentado em (AKESSON; GOOSSENS; RINGHOFER, 2007). Os autores propõem o uma verificação formal dos requisitos de acesso em tempo-real através de simulação para definir os grupos de acesso à memória, o que corresponde a sequências pré-computadas de comandos SDRAM, com eficiência conhecida e latência. Nesse trabalho, o controlador de memória é segmentado em uma parte frontal e uma traseira (*front-end* e *back-end*), onde a parte traseira comunica com a memória e a parte dianteira com os solicitadores de acesso (descritos como *Front-end e Back-end Ressources* na Figura 2.3). A parte traseira é implementada como um recurso com comportamento conhecido de modelamento simples, pois implementa cinco tipos conhecidos de padrões de acesso a memória: leitura, escrita, transição escrita para leitura, transição leitura para escrita e atualização. Cada um desses padrões de acesso é caracterizado previamente e tem comportamento conhecido em

termos de ciclos de execução e latência. A parte dianteira faz interface com os solicitadores de acesso[2] e contém um árbitro necessário para gerenciar os acessos para a memória. O árbitro processa as solicitações recebidas geradas pelos clientes e gera padrões de acesso para a parte traseira. Na análise proposta, o árbitro é um recurso com comportamento previsível, pois possui um número máximo conhecido de solicitações que são possíveis antes de que uma determinada solicitação de acesso seja efetuada. O modo de arbitragem implementado é feito através de um árbitro próprio, que usa prioridades estáticas e controle de créditos, baseados na categoria de servidores de latência, chamado de *Credit-Controlled Static-Priority Arbiter* (AKESSON; STEFFENS; GOOSSENS, 2009). Os parâmetros de funcionamento do árbitro são ajustados em tempo de implementação e permanecem estáticos durante a execução do controlador. O árbitro controla o acesso dos clientes através de um contador de créditos, que implementa prioridades estáticas e limites de largura de banda e latência estabelecidos em tempo de projeto. É composto por um regulador de taxa de acesso e um escalonador com prioridade fixa. O regulador de taxa é usado para garantir que cada cliente tenha uma taxa de largura de banda mínima, como estipulado em tempo de projeto para cada cliente. O elemento atomizador é usado para criar acessos em granularidades pré-estabelecidas em tempo de projeto, dependendo da aplicação alvo. Outras abordagens semelhantes desenvolvidas a partir dessa arquitetura, descrevendo a integração de controladores de memória predizíveis, foram apresentadas em (AKESSON; GOOSSENS, 2011), (GOMONY; AKESSON; GOOSSENS, 2013) e (GOMONY; AKESSON; GOOSSENS, 2014).

Figura 2.3: Modelo de arquitetura para um controlador de memória predizível e combinável, com suporte para dois clientes.



Fonte: (AKESSON; GOOSSENS, 2011).

Outra técnica para implementar uma integração predizível de elementos em um SoC é baseada na definição de técnicas de modelagem e análise do pior caso de tempo de execução que

[2]Os autores denominam de solicitadores de acesso os clientes do controlador de memória, que são as interfaces com os EPs do SoC.

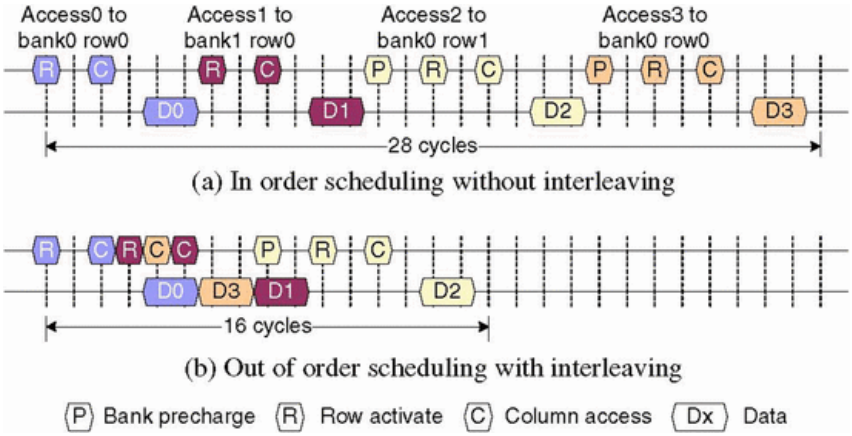
pode ocorrer na arquitetura. Os trabalhos anteriores apresentados por Henriksson et al. (2007) e Vink et al. (2008) propõem o uso de cálculo de redes para fazer uma verificação estática da latência no acesso à memória compartilhada em um SoC. Henriksson et al. (2007) descrevem a inserção de dois elementos de rede de comunicação internamente ao subsistema de memória: o compressor e descompressor de pacotes. A técnica denominada de cálculo de redes define o conceito de fluxo de dados através de elementos de comunicação e tem sido utilizada para analisar o desempenho das redes de comunicação. O cálculo de rede é aplicado para a verificação da latência no acesso à memória, e utilizado para modelar as interconexões entre o SoC e o controlador de memória. Com a metodologia proposta, os autores implementam o cálculo de redes de comunicações para permitir uma verificação estática para a latência nos acessos à memória. Seguindo a mesma direção do modelamento de latência través do cálculo de redes, os autores Wolf e Geuzebroek (2011) detalham a integração predizível de sistemas baseada na análise e projeto da infraestrutura de comunicação com garantias de Qualidade de Serviço (*Quality of Service* – QoS) para atingir o processamento em tempo real de aplicações multimídia. Os autores detalham a proposta de um árbitro baseado em contagem de tempo de espera (tipo *latency-rate server*), usando um escalonador de prioridades fixas estabelecidas durante a avaliação prévia das características de acesso dos clientes. Os autores propõem a classificação de QoS em três níveis: tempo-real, baixa latência e melhor esforço possível. Tal classificação é baseada no estudo anterior de Lee e Chang (2006).

Controladores de memória com escalonamento dinâmico de acessos propostos na literatura fazem o escalonamento de comandos para a DRAM externa em tempo de execução baseado no conjunto de pedidos que está sendo processado. Tais controladores são implementados a partir de algoritmos complexos de escalonamento de comandos com alvo em sistemas de alto desempenho computacional. Exemplos de escalonamento dinâmico é o aumento de prioridade para pedidos direcionados à análise das páginas abertas[3] na memória, como apresentado por Mutlu e Moscibroda (2009) e por Hur e Lin. (2004), ou ordenar as sequências de comandos de escrita ou de leitura melhorando o aproveitamento do barramento de dados da memória, como é apresentado por Shao e Davis (2007). Esse tipo de escalonamento tem como benefício a redução do tempo de execução das escritas e leituras na memória, como pode ser visto no exemplo de intercalamento de comandos apresentado na Figura 2.4. Nesse exemplo, a sequência de comandos gerada sem intercalamento faz um acesso de leitura para o banco 1 (Access 1) seguido de um acesso de leitura para o banco 0 (Access 2), que estão abertos, seguidos de um acesso de leitura para a linha 1 do banco 0 (Access 2), que requer uma pré-carga do banco. No final da sequência, um acesso de leitura é feito novamente para a linha 0 do banco 0 (Access 3), necessitando de uma nova pré-carga do banco. O intercalamento de comandos gerados pelo controlador de memória ordena a sequência anterior para dois acessos consecutivos para o banco 0 (Access 0 e 3), um acesso para o banco 1 (Access 1) e um acesso para a linha 1 do banco 0 (Access

[3]Página aberta significa quando uma linha de memória em um determinado banco está carregada para o buffer de linha, de forma a estar diretamente acessível para escrita e leitura.

2). Esse tipo de escalonamento dinâmico reduz o tempo de execução dos acessos maximizando a largura de banda em alguns casos. No entanto, o intercalamento de comandos gerenciado pelo controlador de memória não é desejável para sistemas predizíveis, pois torna impossível a determinação dos tempos gastos pelo controlador para realizar uma transação gerada por um cliente.

Figura 2.4: Escalonamento de comandos na memória para minimizar os tempos de acesso.



Fonte: (SHAO; DAVIS, 2007).

Uma proposta de escalonamento dinâmico que considera uma avaliação de tempo de acesso e de largura de banda por cliente é proposta por Zhang et al. (2011). Neste trabalho os autores propõem um método de escalonamento adaptativo de acessos à memória baseado no tempo de relaxação de um acesso a memória que é gerado por um cliente. O método proposto ajusta a prioridade do acesso de cada cliente a partir da sua tolerância à latência. O cliente que tiver o menor valor de relaxação terá a maior prioridade para ocupar o canal de memória. Na definição dos autores, relaxação é a diferença entre o prazo de conclusão de uma tarefa (*time deadline*) e o tempo necessário para processar esse acesso. Os autores propõem a um algoritmo complexo que monitora a o tempo decorrido de uma solicitação de acesso por um cliente e calcula o tempo de relaxação a partir da determinação de um prazo de conclusão para o completar uma sequência de acessos. A partir desses prazos, o algoritmo proposto é usado para calcular a largura de banda que está sendo servida para cada cliente a cada instante de tempo. Com tais informações e a informação de largura de banda que deve ser servida para cada cliente, o árbitro gerencia prioridades para clientes que tem menor relaxação e também para aqueles que tem menor largura de banda atendida. O algoritmo de escalonamento é implementado em um árbitro de prioridade que leva em conta a classificação de cliente em três níveis diferentes: clientes com pequena largura de banda com acessos de elevada frequência, clientes com elevada largura de banda e tolerantes à latência e clientes com largura de banda baixa e acessos de baixa frequência tolerantes à latência.

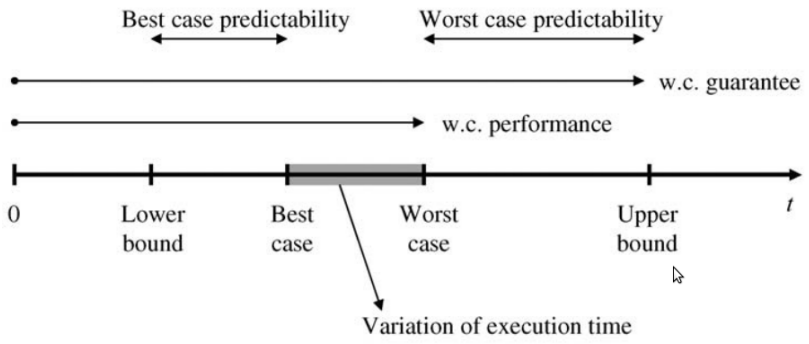
Outra abordagem para o escalonamento dinâmico de acessos é monitorar a execução dos clientes conectados ao controlador de memória e modificar as prioridades de acesso conforme

a largura de banda necessária durante toda a execução do sistema. Essa forma de adaptação dinâmica é estudada por Jeong et al. (2012) com o gerenciamento de prioridade de acessos à memória realizados por uma CPU e uma GPU com base em um novo mecanismo que acompanha o progresso de cargas de trabalho de GPU. Nesta avaliação, os autores mostram que o mecanismo proposto de adaptação dinâmica de prioridades melhora significativamente o desempenho GPU com apenas um impacto mínimo sobre a CPU.

2.2.2 Análise do Tempo de Resposta

O conhecimento dos limites superiores e inferiores precisos dos tempos de execução é importante para a concepção de sistemas de elevado desempenho. Para tal, é necessário estabelecer princípios de projeto de sistemas que sejam predizíveis em seu comportamento temporal e suportem a sua descrição através de um modelo. Thiele e Wilhelm (2004) definem uma semântica apropriada que define os termos concisos relacionados aos tempos de execução dentro de um sistema. O tempo de execução é delimitado entre a ocorrência de dois eventos, o início de o final de uma ação no sistema. Define-se que entre esses dois intervalos existe um limite superior e um limite inferior, sendo que o limite superior é o maior prazo de execução que o sistema admite. Além disso, definem o melhor e o pior caso de execução, sendo que o pior caso é o maior dos tempos possíveis para que a tarefa seja executada. O pior caso deve ser inferior ao limite superior, como mostrado na Figura 2.5.

Figura 2.5: Representação das medidas relacionadas à previsibilidade das arquiteturas de sistema.



Fonte: (THIELE; WILHELM, 2004).

A análise do tempo de resposta no pior caso em um sistema de memória é importante para poder-se determinar se uma transferência de dados pode ser finalizada antes dos prazos limites de execução de uma tarefa. Dessa forma, é preciso estabelecer quais são os prazos limites da aplicação que solicita dados para a memória e quais são os prazos limites de conclusão da transferência. Uma comparação entre dois métodos para estabelecer o WCET é apresentada em (SHAH; KNOLL; AKESSON, 2013). Este trabalho compara um método analítico detalhado e outro baseado em abstração, para avaliar a latência de acesso a um recurso compartilhado (me-

mória) por diferentes processadores em um sistema de memória. O método analítico é baseado no cálculo aproximado de atrasos na execução das operações na memória e na propagação dos dados e comandos internamente através dos elementos do controlador de memória. O segundo método comparado, usa um modelo matemático baseado na análise de servidores de latência (VINK; BERKEL; WOLF, 2008). Árbitro implementado faz o controle a partir de uma técnica de controle de créditos. O tempo de atualização da memória é considerado pelo árbitro para gerar créditos para os mestres em espera. O algoritmo do árbitro calcula o WCRT (worst-case response time) para cada pedido e recalcula os créditos para os mestres. Os autores fazem a avaliação de uma implementação em FPGA contendo seis mestres conectados a um controlador de memória, implementados como geradores de tráfico para recriar os traços de execução obtidos a partir da execução de um benchmark CHStone rodando sobre um simulator de conjunto de instruções de um processador Simplescalar. Foram estudadas duas aplicações do CHStone: JPEG e compensação de movimento. O artigo mostra que o método analítico resulta em latências menores (em torno de 3,8x) do que o método de servidores de latência. Os resultados de alocação de largura de banda para os clientes igualam-se entre os dois métodos analisados.

Ao calcular uma estimativa WCET, é necessário levar em conta o maior tempo de latência possível gerado para cada solicitação de acesso à memória. Para tanto, é necessário considerar as interferências geradas por outros acessos concorrentes e também os atrasos de operação da memória externa, como as operações de atualização e trocas de bancos e linhas da memória. Os controladores de memória devem ser implementados de forma predizível, devendo ser modelados com segurança para poder-se avaliar os piores casos. Internamente ao controlador de memória, o árbitro que gerencia os acessos deve ter um comportamento previsível. Controladores de memória predizíveis ou analisáveis são propostos por diversos trabalhos anteriores, como as propostas inicialmente apresentadas por Akesson et al. (2007), Paolieri et al. (2009), Wolf e Geuzebroek (2011), Henriksson et al. (2007) e Vink et al.(2008) e Zhang et al. (2011). Os trabalhos que foram citados tem foco na implementação de controladores de memória predizíveis com parâmetros de configuração estipulados em tempo de projeto, que se mantêm estáticos ao longo do tempo de execução do sistema. No entanto, a adaptação desses sistemas ao longo do tempo é uma funcionalidade que pode trazer vantagens para o funcionamento de um sistema, principalmente em sistemas de computação formados por diversos núcleos de processamento heterogêneos. Com a especialização das partes e a diversificação de elementos de processamento que compõe os SoCs, os acessos gerados para o sistema de gerenciamento de memória podem ter características heterogêneas com requisitos variantes ao longo do tempo, como tamanho das transferências solicitadas para a memória e prazos de conclusão. Além disso, partes do SoC podem permanecer inoperantes por um determinado período de tempo, quando não utilizadas, estando em modo de baixo consumo. Essa diversificação de situações nem sempre pode ser prevista completamente em tempo de projeto a fim de determinar com precisão qual é será a concorrência de acessos para o sistema de controle de memória compartilhada. Por esses motivos, o controladores de memória adaptativos podem conhecer algumas características de

funcionamento do sistema para a partir disso adaptar-se às características de acessos gerados no SoC em tempo de execução do sistema.

2.2.3 Controladores de Memória Comerciais

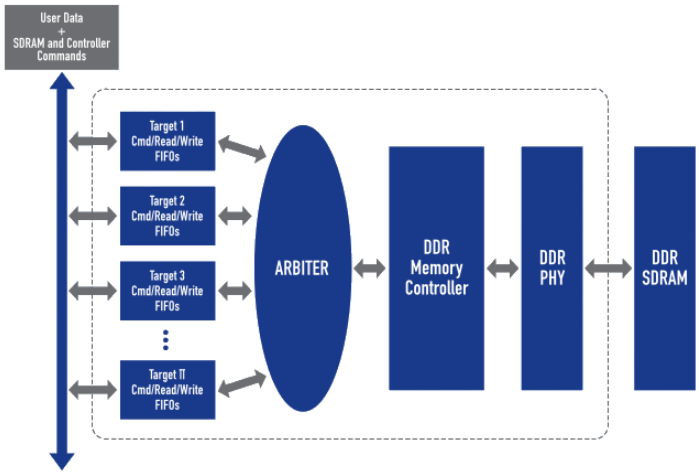
Soluções propostas pela indústria para implementar controladores de memória com múltiplos pontos de acesso também foram avaliadas. Para implementações utilizando dispositivos de lógica programável, o fabricante Xilinx disponibiliza através da sua biblioteca de IPs o *LogiCORE^{TM} IP Multi-Port Memory Controller* (MPMC) (XILINX, 2012). É um controlador de memória externa parametrizável, capaz de implementar diversas interfaces de acesso (denominadas de portas) para uma única memória externa, na forma de um barramento multiplexado. Essa implementação de controlador de memória suporta até oito portas, todas no padrão CoreConnect PLB (*Processor Local Bus*), sendo uma delas dedicada para processamento de vídeo (*frame buffer*). O controlador de memória MPMC gerencia os acessos utilizando um árbitro que implementa os modos de arbitragem com prioridade fixa, Round Robin e arbitragem customizada pelo usuário. O controlador multi-portas garante que, em cada porta de acesso, as transações de dados são executadas na ordem em que foram solicitadas. No entanto, não há garantia de que as transações das diferentes portas sejam feitas na ordem em que foram solicitadas, devido ao modo de arbitragem implementado. O MPMC não verifica a coerência dos dados, portanto, uma aplicação que escreve um dado em um endereço compartilhado, deve garantir uma forma de verificação para que esse endereço não seja lido antes de ser escrito, já que o controlador não faz a verificação de escrita antes da leitura.

A fabricante ARM criou o *CoreLink^{TM} Static and Dynamic Memory Controllers* (DMC) (ARM, 2013). É uma solução inovadora para integração entre processadores ARM, processadores de mídias e IPs. A solução proposta aborda diretamente o problema das garantias de qualidade de serviço para um sistema de memória complexo, formado por diversos elementos de processamento e um sistema de arbitragem com algoritmos de escalonamento. O DMC é concebido para maximizar o uso da largura de banda disponível da memória DRAM externa. Além disso, essa implementação propõe o uso de conexões hierárquicas de barramentos, para lidar com problemas de latência e gerenciar os acessos em diferentes níveis da rede de interconexões.

A empresa Uniquify comercializa um IP de subsistema para controlar memórias externas do tipo DDR2/3/4 e LPDDR2/3/4 composto por um controlador de memória, uma PHY (interface física para memória) e circuitos de E/S nas tecnologias 28nm, 40nm, and 65nm para diferentes foundries, suportando DDR3 até 2.133 Gbps e DDR4 até 2.4 Gbps. Tal controlador de memória pode operar com a mesma ou a metade da frequência de relógio da DDR externa. Pode ser configurado para suportar até 32 clientes de acesso (denominados de “targets”). O diagrama de blocos do controlador de memória é mostrado na Figura 2.6, que mostra a interconexão entre as entradas de clientes com FIFOs de armazenamento de entrada de comandos e um árbitro para gerenciamento dos acessos. O árbitro faz a seleção dos comandos e gerencia os acessos para o

controlador de memória DDR, que por sua vez controla a PHY e a DDR externa.

Figura 2.6: Diagrama de blocos do controlador de memória da empresa Uniquify.



Fonte: (UNIQUIFY, 2014).

2.3 Resumo do Capítulo

Este Capítulo apresentou a descrição do problema de integração de memórias em sistemas digitais complexos e o gerenciamento dos acessos aos dados, que fundamenta o desenvolvimento desse trabalho. Em seguida foram apresentados os trabalhos relacionados que embasam as ideias apresentadas e desenvolvidas nessa tese, bem como os trabalhos que servem de comparação para os resultados apresentados. No próximo capítulo será discutido o funcionamento de um subsistema de memória e será apresentada uma avaliação de desempenho de controladores de memória e memórias DDR externas.

3 SUBSISTEMAS DE MEMÓRIA

Memória é um dos elementos essenciais para o funcionamento de um sistema de computação. A hierarquia de memórias de um sistema é parte fundamental do projeto de uma arquitetura de processamento, visto que através da hierarquia de memórias é possível satisfazer os requisitos de desempenho de todas as partes que compõem o SoC. Embora uma arquitetura plana e regular é mais simples de ser implementada, somente através de uma hierarquia de memórias é possível explorar circuitos de alto desempenho, combinado ao melhor aproveitamento do custo por bit armazenado e ter a melhor eficiência energética possível. Convenientemente, a hierarquia de memórias é projetada seguindo uma estrutura de memórias *cache* (SRAM), DRAM e Disco. Embora essa seja a hierarquia mais utilizada, não é eficiente em termos de velocidade ao ponto de acompanhar a evolução dos processadores atuais.

Os sistemas computacionais estão migrando para plataformas complexas com uma grande diversidade de elementos de processamento. Um SoC composto por diversos subsistemas isolados, interagindo através de interfaces bem definidas de comunicação, utiliza uma hierarquia de memória para gerenciamento da informação. Novas tendências no projeto de hierarquias de memórias devem considerar o suporte para concorrência, a granularidade dos acessos, bufferização de dados, algoritmos de escalonamento e escalabilidade das interconexões. Um subsistema de memória eficiente deve considerar essas características na integração de diferentes tecnologias de memórias. Basicamente, existem três tipos de memória que são fabricadas utilizando tecnologia CMOS semelhante à tecnologia utilizada na fabricação de semicondutores: SRAM, DRAM e Flash. Os processos de fabricação utilizados nesses três tipos de memórias é diferente e, portanto, cada tipo de memória possui características de diferentes de funcionamento. Todo elemento de memória é responsável por armazenar informação durante um período de tempo mínimo necessário para que um sistema de computação possa processar a informação. Embora essas memórias sejam usadas para a mesma finalidade elas diferem nos aspectos de volatilidade da informação, velocidade de acesso, área de silício, potência dissipada e custos de fabricação. Essas características, se utilizadas corretamente, permitem a construção de uma hierarquia de memória que permita explorar ao máximo todos esses aspectos para melhorar o desempenho do sistema.

O princípio que justifica a criação de hierarquias de memória em sistemas de computação é a

"localidade da referência", denominado assim pelo motivo de que uma informação em memória está localizada em tempo e espaço. A partir disso, denominam-se os princípios da "localidade espacial" e "localidade temporal". Na hierarquia de memórias, diversos níveis de memória são utilizados para armazenar a mesma informação, em quantidades diferentes e com velocidades de acesso diferentes.

Este Capítulo trata das tecnologias envolvidas no projeto de subsistemas de memória, que são utilizadas no projeto de subsistemas de memória para SoCs modernos. A Seção 3.1 mostra o funcionamento das memórias DRAM e os aspectos relacionados ao seu desempenho e capacidade de armazenamento. A Seção 3.3 descreve a organização dos subsistemas de memória, que envolve o acesso aos dados armazenados em DRAM e a hierarquias de níveis de memórias.

3.1 Memórias DRAM

Esta Seção trata sobre as características e funcionamento das memórias DRAM, apresentando os conceitos básicos de funcionamento e de construção de memórias DRAM.

3.1.1 Evolução da Memória DRAM

Foi patenteada em 1968 por Dennard, chamada de memória dinâmica de acesso aleatório (*Dynamic Random Access Memory* – DRAM). Muito mais barata que uma memória SRAM, que é formada a partir de 6 transistores, a célula de memória DRAM é formada somente por um transistor e um capacitor. O bit é representado como '0' ou '1' pelo valor da carga no capacitor, carregado ou descarregado. Devido ao capacitor de armazenamento de carga, que deve ser carregado ou descarregado sempre que uma informação é escrita, a memória DRAM é mais lenta do que uma memória SRAM. O tempo de acesso na memória DRAM é maior que na memória SRAM, sendo que esse tempo de acesso não tem reduzido com o avanço dos processos de fabricação de semicondutores, pois o atraso é dominado pelas interconexões. Portanto, não são preferidas para uso como memórias locais de processamento rápido, como caches ou scratchpads, mas sim como memórias de armazenamento em grande quantidade.

As memórias DRAM têm evoluído nos últimos anos formando uma série de gerações que apresentam um aumento significativo da capacidade de armazenamento e da velocidade de barramento. Primeiramente, adicionou-se um sinal de sincronismo (relógio) ao barramento de acesso, criando as memórias SDRAM (Synchorous DRAM). Em seguida, as transferências de dados foram aproveitadas nas duas bordas desse relógio, tanto na de subida como na de descida, formando as memórias DDR (double data rate). A sequência evolutiva foi obtida com o aumento da velocidade do barramento de dados, gerando as memórias DDR2, DDR3 e DDR4. Além dessa evolução, houve melhorias quanto a potência dissipada, nas memórias de baixa potência (Low Power) LPDDR2 e LPDDR3. Todos os padrões de memória são padronizados pelo JEDEC *Solid State Technology Association*, órgão internacional de padronização de interfaces.

O padrão DDR3 foi publicado pelo JEDEC no ano de 2007 e o padrão DDR4 no ano 2012.

O padrão DDR4 apresenta como principais modificações o aumento da velocidade do barramento e taxas de transferência de dados maiores. Enquanto dispositivos de memória da geração DDR3 conseguem transferir dados nas taxas entre 800 to 2.133 Mega transferências por segundo (MTps), os dispositivos no padrão DDR4 estão sendo fabricados inicialmente com taxa de 2.133 MTps, mas com previsão de atingir 4.266 MTps. O número de bancos foi aumentado para 16 e a tensão de alimentação e operação dos dispositivos foi reduzida para 1,05 a 1,2 volts[1] (enquanto que na DDR3 é de 1,2 a 1,65 volts). Outra modificação importante está relacionada à topologia dos pentes de memória, que serão implementados na forma de ponto-a-ponto com o controlador de memória, fazendo um único dispositivo por canal de memória.

Os fabricantes de semicondutores Hynix, Micron e Samsung foram os primeiros a iniciarem a produção de dispositivos de memória DDR4. A Samsung anunciou a produção de dispositivos 4 Gigabits DDR4 1,2V em agosto de 2013, utilizando a tecnologia de processo de fabricação *20nm-class*. A Hynix atualmente está disponibilizando amostras de engenharia de dispositivos 4 Gigabits DDR4 1,2V, embora no página do fabricante já estejam disponíveis os *datasheets* das memórias[2]. A Micron disponibiliza amostras de dispositivos de 4 e 8 Gigabits DDR4 1,2V fabricados em tecnologia TwinDie. Atualmente, essa recente geração de memórias DDR4 não está sendo fabricada em massa por nenhum dos fabricantes citados e está prevista para produção em 2014.

Exitem também memórias variantes das DDR como as memórias gráficas síncronas GDDR SGRAM (*Synchronous Graphics* RAM), utilizadas principalmente em conjunto com processadores gráficos, e memórias de baixa potência LPDDR. A memória GDDR5 é baseada na DDR3, mas com capacidade de transferência de dados maior atingindo 7 Gigabits por segundo. Uma GDDR opera em velocidades entre duas a cinco vezes a largura de banda de uma memória DDR. São memórias utilizadas preferencialmente por unidades de processamento gráfico (GPU). Devido à baixa localidade das transações à memória feitas pela GPU, modos de rajadas longos são menos utilizados. Nessas memórias, a vantagem de velocidade é alcançada mantendo-se múltiplos bancos abertos simultaneamente.

O empilhamento em três dimensões é uma tendência promissora de integração de novas tecnologias de memórias DRAM. O JEDEC publicou a especificação de Wide-I/O DRAM no final do ano de 2011 (JEDEC, 2011). A empresa Samsung demonstrou uma memória SDRAM empilhada seguindo o padrão Wide-I/O (KIM et al., 2011). Essas memórias serão utilizadas como memória principal do sistema, e em casos que se requer maior escalabilidade, usadas como scratchpad ou último nível de cache. Essas implementações demonstraram que empilhamento 3D é uma tecnologia que está se consolidando como promissora forma de fabricação de circuitos integrados. A TSMC propõe a sua Tecnologia *Chip-On-Wafer-On-Substrate* (CoWoS), que utiliza a tecnologia de TSVs para integrar múltiplos chips em um substrato que contém as

[1] Uma entrada de tensão de alimentação de 2,5 V foi adicionada ao novo padrão, denominada de VPP. É usada para melhorar a velocidade de carga e descarga da linha de palavras internamente ao banco de memória.

[2] Os datasheets foram publicados com data de março de 2013, mas estão sofrendo alterações e ainda não contém todos os dados necessários para uma avaliação completa da memória.

conexões entre os chips e os pinos do encapsulamento (WOO et al., 2010). Esta arquitetura proporciona maior densidade de interligações, diminui o comprimento de interconexão global e reduz a carga RC associada, resultando em melhor desempenho e menor consumo de energia em um fator de forma menor.

3.1.2 Dispositivos de memória DRAM

Uma DRAM é formada a partir de um conjunto de bancos de memória, que por sua vez são indexados a partir de informações de linha e coluna. Dessa forma, para acessar uma palavra armazenada na memória (palavra é o conjunto de bits que é da mesma largura do barramento de dados da memória), deve-se gerar um endereço de banco, linha e coluna. Diversos chips são combinados para formar um pente de memória, aumentando a largura do barramento de dados. O acesso aos dados armazenados em memória é feito de uma forma particular, seguindo uma sequência de comandos em intervalos de tempo definidos. Primeiramente, deve-se selecionar o banco e linha da memória e em seguida é selecionada a coluna dentro dessa linha. Cada linha de memória em um determinado banco é copiada para um buffer correspondente, que fica acessível para ser lido ou escrito. Dessa forma, o buffer de linha de um banco fica acessível ao circuito de multiplexação para o barramento de dados. Essa operação é chamada de ativação do banco/linha (activate). Quando uma linha está "ativa", ela está acessível para o barramento de dados, então são feitas leituras ou escritas em rajadas consecutivas de dados. Uma é uma seqüência de acessos de escrita ou leitura feitas em endereços subseqüentes, dentro de uma mesma linha. As rajadas podem ser encadeadas em seqüência.

Cada um dos bancos contém um buffer de linha independente, de forma que todos podem estar ativos simultaneamente. A linha é pré-carregada do buffer e salva novamente na memória DRAM, antes de que uma nova linha possa ser aberta, usando a operação de pré-carga. Além dos comandos de ativação, pré-carga, leitura e escrita, o comando de auto-atualização (*auto-refresh*) é utilizado para atualizar o conteúdo armazenado em memória, devido à perda que ocorre pela descarga dos capacitores que foram cada célula. A recarga dos capacitores é uma operação periódica, que é controlada internamente pela memória DRAM. Cada tecnologia de memória tem um período característico de atualização de, por exemplo, na DDR2, 64 ms.

Os dispositivos de memória DRAM são organizados na forma de bancos, linhas e colunas de endereçamento. Em uma mesma geração de memórias DRAM, os dispositivos podem ser organizados com diferentes tamanhos de barramento de dados. A matriz de memória é organizada em bancos, que por sua vez contém linhas e colunas de células, com largura igual ao barramento de dados do dispositivo. Memórias de 1Gbit de capacidade de armazenamento com largura de barramento de 8 bits são denominadas de 128Meg x8. A Tabela 3.1 mostra diferentes organizações internas de memórias 4 Gigabit DDR3 (MICRON, 2009) e 8 Gigabit DDR4 (MICRON, 2013).

A estrutura interna de uma memória DRAM está representada na Figura 3.1. Existem dois decodificadores de endereços: linha e coluna. O decodificador de endereço de linha seleciona

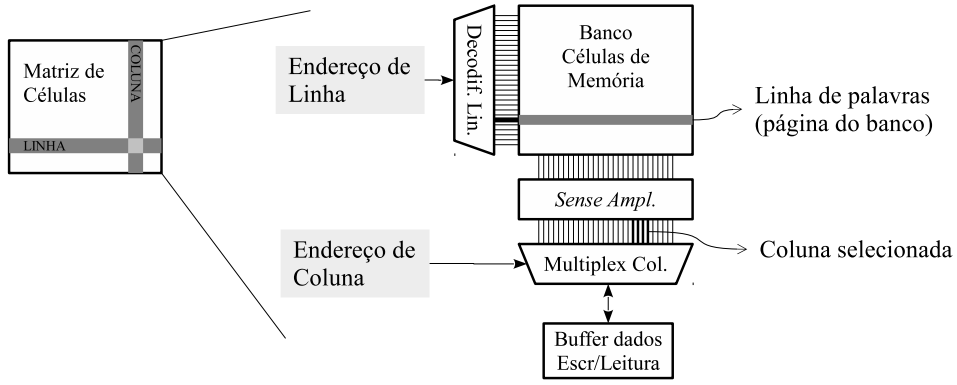
Tabela 3.1: Organização interna de memórias 4 Gigabit DDR3 e 8 Gigabit DDR4.

Geração	***DDR3***			***DDR4***	
Configuração	*1 Giga x 4*	*512 Mega x 8*	*256 Mega x 16*	*2048 Mega x 4*	*1024 Mega x 8*
Barramento de dados	4 bits	8 bits	16 bits	4 bits	8 bits
Número de linhas	65.536	65.536	32.768	65.536	32.768
Número de colunas	2.048	1.024	1.024	1.024	1.024
Número de bancos	8	8	8	16	16
Número de fileiras	1	1	1	2	2

Fonte: elaborado pelo autor.

uma linha de um determinado banco de memória, fazendo com que essa linha seja colocada na entrada do *sense-amplifier*. A linha de memória então está no estado "ativo" e pronta para ser lida ou modificada. O decodificador de endereço de coluna seleciona um segmento de endereços para a saída de dados ao mesmo tempo em que realiza a operação de escrita ou de leitura sobre esse segmento. A estrutura interna das memórias DRAM suporta a organização de dados em múltiplos bancos de memória (Figura 3.2). Essa estrutura interna permite que a ativação de uma linha em um determinado banco seja feita enquanto outros bancos estiverem abertos. Na prática, a memória DRAM pode trabalhar com todos os bancos abertos simultaneamente, ou seja, com todos os *sense-amplifiers* carregados com alguma linha de seu banco correspondente.

Figura 3.1: Organização lógica de dados interna à uma memória DRAM.

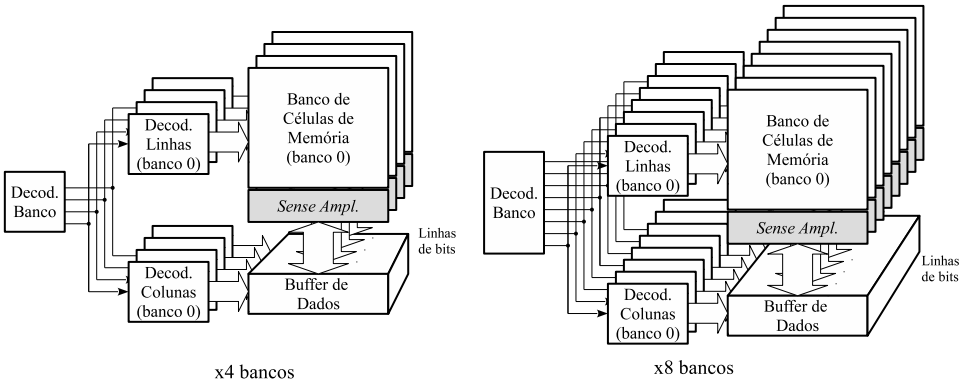


Fonte: elaborado pelo autor.

3.1.3 Arquitetura Interna

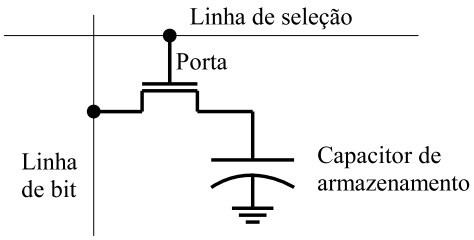
A célula de memória DRAM é formada por um transistor e um capacitor (1T1C), como mostra a Figura 3.3. O transistor é controlado através da porta por uma linha da palavra selecionada (endereçamento), o terminal fonte é conectado à linha de bits e o terminal dreno é conectado ao capacitor. A pequena carga elétrica que é armazenada no capacitor define o valor binário armazenado. Essa pequena carga é amplificada para um nível de tensão aceitável através de um circuito denominado *sense-amplifier*.

Figura 3.2: Organização lógica de memórias DRAM com 4 e 8 bancos. Cada banco contém um conjunto de decodificadores de linha e de coluna, além de um conjunto de amplificadores de entrada e saída de dados. O amplificador de dados é do tamanho de uma linha do banco de memória.



Fonte: elaborado pelo autor.

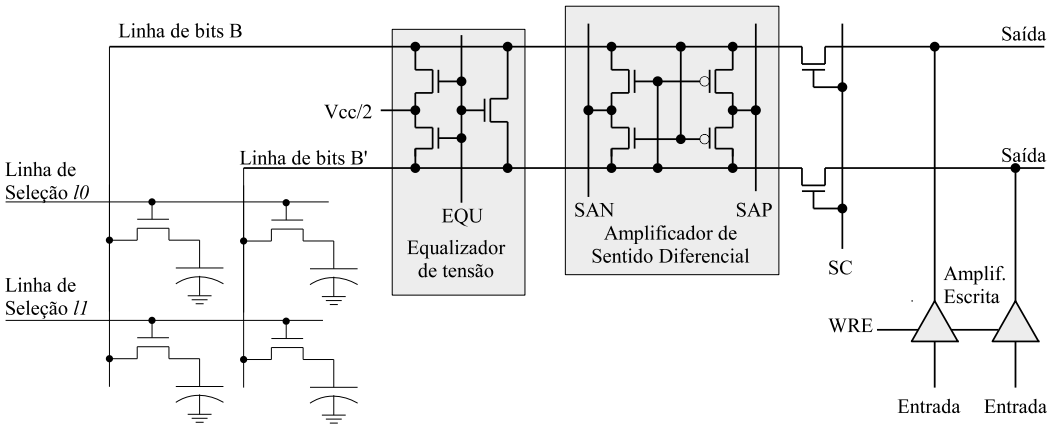
Figura 3.3: Estrutura básica de uma célula de memória DRAM formada por um transistor e um capacitor (1T1C).



Fonte: elaborado pelo autor.

A leitura de um valor armazenado em uma célula de memória DRAM é feita através de um amplificador diferencial chamado de *sense-amplifier*, mostrado na Figura 3.4. O acesso aos bits armazenados na célula DRAM é feito em quatro etapas: pré-carga, acesso, sente e restaura.

Figura 3.4: Diagrama elétrico de um amplificador diferencial para duas linhas de bits de uma memória DRAM.

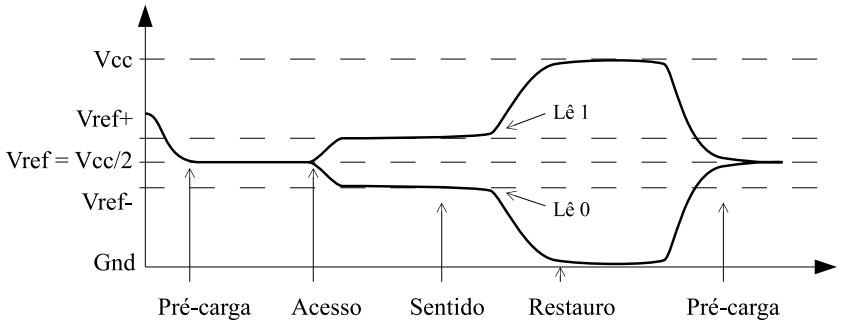


Fonte: elaborado pelo autor.

E etapa de pré-carga é a primeira das etapas para acessar o conteúdo de uma célula de memória, mas ela é feita na etapa anterior de acesso, deixando a linha de bits pronta para o acesso seguinte. A etapa de pré-carga é responsável por levar o nível de tensão da linha de bits a uma tensão de referência (Vref), que equivale à metade do valor de tensão de alimentação (Vref = Vcc/2). A segunda fase é denominada de acesso, ocorre quando as linhas de controle das células são ativadas, colocando a carga dos capacitores em contato com a linha de bits. Se o capacitor estiver com carga, a linha de bits passará a ter uma carga maior que Vref (Vref+), caso contrário, se o capacitor estiver descarregado, a linha de bits conterá o valor de tensão inferior à Vref (Vref-). A terceira etapa é denominada de sentido, quando os amplificadores diferenciais detectam a diferença de potencial existente na linha de bit para o nível de tensão superior ou inferior, dependendo se a tensão da linha de bit estava acima ou abaixo de Vref. A última etapa é denominada de restauro e ocorre após a estabilização do valor de tensão na linha de bit. Nessa última etapa, o valor é escrito novamente no capacitor da célula DRAM é também é disponibilizado para o circuito de saída da matriz de memória. No caso de uma escrita à uma célula de memória, o valor escrito é gravado no capacitor durante a etapa de restauro, após realizar a pré-carga e a ativação. Durante uma escrita, o tempo de restauro é um pouco maior do que o restauro de uma operação de leitura. A Figura 3.5 mostra em um diagrama de tempos a diferença de potencial na linha de bits durante as etapas de pré-carga, acesso, sentido e restauro em uma memória DRAM.

Cada uma das etapas descritas deve respeitar um determinado intervalo de tempo, para que os níveis de tensão sejam estabilizados antes de iniciar uma nova operação na célula de memória. Nas sucessivas gerações de memória DRAM, os intervalos de tempo são definidos por

Figura 3.5: Diagrama de tempo para as etapas de acesso aos dados em memória através do amplificador diferencial.



Fonte: elaborado pelo autor.

parâmetros de acesso que devem ser respeitados pelo circuito que faz interface com a memória, sendo denominados de tempos de acesso. O tempo de acesso de uma memória DRAM é dado pela combinação de quatro intervalos de tempo necessários para efetivar uma operação de escrita ou leitura. O tempo de acesso é determinado pela combinação em sequência dos parâmetros tCAS-tRCD-tRP-tRAS. As etapas de acesso e sentido formam o *row column delay time* (tRCD). As etapas de acesso, sentido e restauro formam o *row adress strobe time* (tRAS). A etapa de pré-carga é definida pelo *row precharge time* (tRP). Por último, o tCAS representa o *column address strobe delay*, que não está representado nos tempo de acesso descritos anteriormente mas pelo tempo necessário para decodificação dos dados carregados no buffer de linha para a saída do barramento de dados da memória. Outro parâmetro importante faz respeito ao tempo de restauro durante uma escrita de um bit na célula de uma DRAM, denominado de *write recovery time* (tWR).

3.1.4 Interface de Barramento

Para acessar o conteúdo das células de memória, cada dispositivo DRAM contém uma interface de acesso composta por três barramentos: dados, endereços e comandos. O barramento de comandos e de endereços opera na mesma frequência de relógio da memória. Já o barramento de dados pode operar com o dobro da frequência do barramento, devido à transferência de dados nas duas bordas do relógio de sincronização, de subida e de descida. Essa capacidade de duplicar a taxa de dados em relação à frequência do barramento é encontrada nas memórias DDR (*Double Data Rate*). O barramento de dados é composto por uma interface bidirecional de dados (geralmente denominado de DQ) e uma interface bidirecional de sinalização (denominada de *strobe*, ou DQS). O sinal DQS é usado como referência de amostragem para o sinal de dados DQ, sendo utilizado um sinal DQS para cada oito sinais de dados DQ. Internamente à SDRAM, existe um circuito DLL (*Delay-Locked Loop*) usado para controlar o atraso entre o sinal de relógio recebido pela memória e os sinais do barramento de dados. Durante uma leitura, a memória gera os sinais de dados DQ e o sinal de referência DQS com a mesma fase. No entanto, durante uma escrita de dados, o circuito que controla o barramento da memória

(controlador de memória) deve gerar o sinal de amostragem DQS com a defasagem de 90º em relação aos dados. O projeto de um controlador de memória DDR SDRAM deve garantir que a fase dos sinais de referencia e dos sinais de dados seja mantida com a maior exatidão possível, evitando desvios de fase e atrasos excessivos, a fim de amostrar corretamente o dado transferido.

A latência de leitura (CAS Latency) pode ser configurada em todas as gerações de memória SDRAM. A latência de CAS é o tempo que a memória DRAM leva para retornar os dados após receber um comando de leitura em uma determinada coluna. A latência de leitura é usada para ajustar a velocidade de operação do barramento de acesso à DRAM. Como o núcleo de memória DRAM tem limitações severas de freqüência de operação, o ajuste da latência de leitura é usado para adaptar a freqüência de operação do barramento para uma determinada geração de memória.

O tamanho da rajada pode ser configurado. A rajada representa uma seqüência de acessos pré-configurados em endereços consecutivos dentro de uma região de endereçamento, iniciando pelo endereço que é passado juntamente com o comando de escrita ou leitura. Caso a memória seja configurada para operar com tamanho de rajada igual a 4 acessos (BL4), então ao receber um comando de escrita ou leitura a DRAM inicia o acesso ao número de colunas indexadas pelo endereço inicial. Isso elimina a necessidade de repetir o comando de coluna. Isso permite que o barramento de comandos da memória seja usado para efetivar outros comandos em bancos diferentes do que foi inicialmente acessado, aumentando o paralelismo das operações.

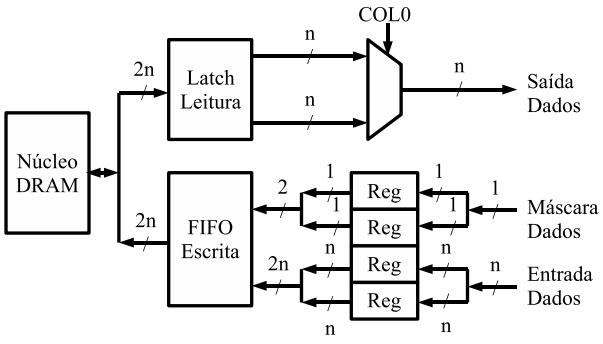
3.1.5 Entrada e Saída de Dados

Em uma memória SDRAM, um comando de escrita ou leitura é usado para transferir um bloco de dados com tamanho configurado pelo número de sequências de acessos, denominado de *burst-length* (BL). A sequência de acessos é disparada por um único comando e o número de repetições é configurado previamente, durante a inicialização da memória. A sequência de endereços acessados é dada a partir do endereço inicial passado para a memória, durante o comando de escrita ou leitura. O número de colunas acessado é contido dentro de um segmento de endereços que tem limiares contados a partir do primeiro endereço de coluna da linha de memória que está aberta. No caso de um acesso ao endereço de coluna 5, para uma memória configurada como BL4 (rajadas de tamanho igual a quatro), a sequência de endereços é 5-6-7-4. Se essa mesma memória estiver configurada como BL8 (rajadas de tamanho igual a oito), a sequência de endereços iniciando na 5ª coluna seria 5-6-7-4-1-2-3-0. Ou seja, a rajada sempre acessa um número predeterminado de endereços, mas sempre contidos dentro de um mesmo segmento.

A arquitetura interna das memórias SDRAM é denominada de *n-bit prefetch* (pré-carga de n bits). Para cada bit de dados do barramento de dados da memória existem *n* bits de dados no barramento interno que faz interface com a matriz de memória. Dessa forma, a memória SDRAM em suas diferentes gerações, permite que a transferência de dados que ocorre no barramento de dados externo à memória seja n vezes mais rápido do que a transferência que ocorre

no barramento interno para a matriz de memória. Isso foi desenvolvido para que as interfaces de acesso à memória pudessem evoluir em termos de velocidade de acesso, e de largura de banda, em relação ao núcleo da memória, que não apresenta uma evolução significativa ao longo das gerações de memória. Nos dispositivos DDR, uma linha de dados acessada, que fica gravada no *buffer* de linha, é demultiplexada para o barramento de saída pelo decodificador de colunas na forma de um pipeline de colunas de dados. A Figura 3.6 mostra a arquitetura interna da interface de dados de uma memória DDR SDRAM exemplificando o funcionamento da arquitetura *2-bit prefetch*.

Figura 3.6: Entrada e saída de dados em um dispositivo de memória DDR-SDRAM ilustrando a arquitetura *2n-prefetch*.



Fonte: elaborado pelo autor.

Durante os acessos aos dados armazenados em uma memória DDR SDRAM, a transferência de 1 bit do barramento de dados resulta em uma transferência de 2 bits pra o núcleo de memória (n=2). Na geração seguinte, de memórias DDR2 SDRAM, a arquitetura de interface de entrada e saída de dados foi expandida para “4-bit prefetch”, ou seja, o barramento interno da matriz de memória é 4x maior que o barramento de dados. Nas memórias DDR3 SDRAM a arquitetura foi expandida para “8-bit prefetch”. Na DDR3, a fila interna de tamanho de 8 bits permite que um dispositivo de frequência de barramento igual a 533 MHz, como uma DDR3-1066, tenha um núcleo de memória operando a somente 66,6 MHz e uma interface de barramento de dados que faça 1066 Mega transferências por segundo.

Essas alterações referentes ao barramento permitem que o circuito interno da matriz de memória permaneça praticamente inalterado ao longo das gerações DDR. As modificações ficam concentradas na paralelização das interfaces de entrada e saída de dados dos circuitos de armazenamento temporário de linha, matriz de memória e amplificadores diferenciais. Dessa forma, é possível que a matriz de memória e a própria célula que armazena a informação permaneça com praticamente as mesmas características. O ganho dessa arquitetura está na velocidade do barramento de dados, que aumenta significativamente durante as gerações de memória, aumentando a largura de banda máxima de transferência de dados.

3.1.6 Acesso aos Dados na DRAM

O acesso aos dados armazenados na DRAM é feito através de uma combinação de comandos colocados para a memória através do barramento de comandos em combinação com o barramento de endereços. Para realizar uma operação de escrita ou leitura, uma sequência de comandos é necessária e entre cada comando é preciso respeitar certo número de ciclos de relógio, devido aos atrasos do núcleo da memória. Esses atrasos são necessários pela arquitetura interna da memória, que é mais lenta do que a interface de barramento. A eficiência da memória depende dos intervalos de tempo necessários para realizar as operações na memória. A escrita ou leitura de dados na memória é feita em ciclos, formados a partir de diferentes etapas para garantir o acesso correto aos dados. O ciclo de escrita é composto pelas etapas de acesso de linha, leitura de coluna, restauro de coluna e pré-carga. O ciclo de leitura é composto pelas etapas de acesso de linha, leitura de coluna e pré-carga.

3.1.6.1 Comando de Acesso de Linha

O comando para acesso de linha é usado para ativar uma determinada linha de memória em um determinado banco. A linha ativada é copiada para o buffer de linha (formado pelo *sense-amplifier*), e permanece armazenada para as operações de escrita ou leitura. Também faz parte desse comando o restauro do conteúdo das células de memória, a fim de não perder a informação armazenada nos capacitores. Dois parâmetros são utilizados para definir os intervalos de tempo associados a esse comando: tRCD e tRAS. O tempo necessário para que o conteúdo das células seja movido da matriz de memória até o buffer de linha é denominado de *row to column delay time* (tRCD). Após esse intervalo de tempo, os dados de linha estão disponíveis para futuras operações de escrita ou leitura solicitadas através do barramento. O segundo parâmetro associado a essa tarefa é denominado de *row access strobe time* (tRAS), que representa o tempo restante necessário para finalizar o restauro do conteúdo do buffer de linha para as células de memória. Ou seja, é o intervalo de tempo entre a ativação de um banco e o seu restauro interno feito pelos *sense-amplifiers*. Um comando de pré-carga não pode ser realizado até que esse intervalo de tempo seja finalizado.

Uma sequencia de comandos de acesso de linha podem ser realizados para bancos diferentes, o que melhora o desempenho final da passagem de comandos para a memória. Dois comandos active podem ser gerados para a memória respeitando o intervalo mínimo de tempo *row to row delay time* (tRRD). Nas memórias DDR2 SDRAM e DDR3 SDRAM todos os bancos de memória podem permanecer abertos simultaneamente. Existe um limite para o número máximo de bancos que poder estar ativos em uma janela de tempo, devido à elevação da corrente drenada pelo dispositivo de memória durante o comando de ativação de linha. A duração da janela é determinada pelo parâmetro *four-banck activation window* (tFAW).

3.1.6.2 Comando de Leitura de Coluna

O comando de leitura de coluna é usado para mover um segmento de dados armazenado no buffer de linha para o barramento de entrada e saída de dados da memória. Três parâmetros são utilizados para definir os intervalos de tempo associados à esse comando: tCAS, tCCD e tBURST. O parâmetro *column access strobe latency* (tCAS), também conhecido por tCL (*column latency time*) é o tempo necessário para que o dispositivo DRAM coloque os dados no barramento após receber o comando de leitura de coluna. O parâmetro *burst time* (tBURST) corresponde ao número de ciclos necessários para completar a leitura, em forma de rajada, do conjunto de dados que foi solicitado com o comando de leitura de coluna. Internamente à DRAM, a rajada de dados tem um intervalo de duração que depende do parâmetro *column to column command delay* (tCCD), que determina o intervalo mínimo de tempo entre pedidos consecutivos de leitura. O tempo que a operação de leitura leva para colocar os dados completamente no barramento para o controlador de memória é dado por: tCAS + tAL + tBURST. A pré-carga da linha pode ser feita após o intervalo de tempo igual ao parâmetro *read to precharge time* (tRTP).

3.1.6.3 Comando de Escrita de Coluna

O comando de escrita de coluna move um segmento de dados do barramento de interface da memória para uma região do amplificador diferencial de um determinado banco da memória DRAM. A rajada de dados para ser escrita na memória é colocada no barramento após o comando de escrita, respeitando o intervalo de tempo denominado de *column write latency time* (tCWL). Esse parâmetro é programado através do registrador de configuração das DDR3 e na família das DDR2 ele é igual a um ciclo de relógio. Os dados colocados no barramento são gravados na matriz de memória na etapa de restauro da linha, que tem duração igual ao parâmetro *write recovery time* (tWR). O tempo total entre as etapas de escrita de coluna e restauro é dada por: tCWL + tBURST + tWR.

3.1.6.4 Comando de Pré-Carga

A etapa de pré-carga marca o encerramento do ciclo de acesso á uma linha de dados da memória. Essa etapa é necessária para acessar uma nova linha de memória no mesmo banco. Sucessivas operações de escrita e leitura podem ser feitas sobre uma linha de memória que está aberta, sem a necessidade de realizar a pré-carga. A etapa de pré-carga reinicia o estado dos sense-amplifiers e prepara as linhas de bits da matriz de memória para um novo acesso de linha. O parâmetro *row precharge* (tRP) define o tempo necessário para que o comando de pré-carga seja realizado, de forma que outros comandos não podem ser realizados durante esse intervalo de tempo. Os intervalos de tempo definidos por tRAS e tRP marcam o tempo de ciclo de linha (*row cycle time* - tRC), que representa o intervalo de tempo mínimo entre acessar uma linha de memória é fechá-la para um próximo acesso. O tempo de ciclo de linha é a maior limitação encontrada atualmente quanto ao desempenho de velocidade dos circuitos de memória DRAM.

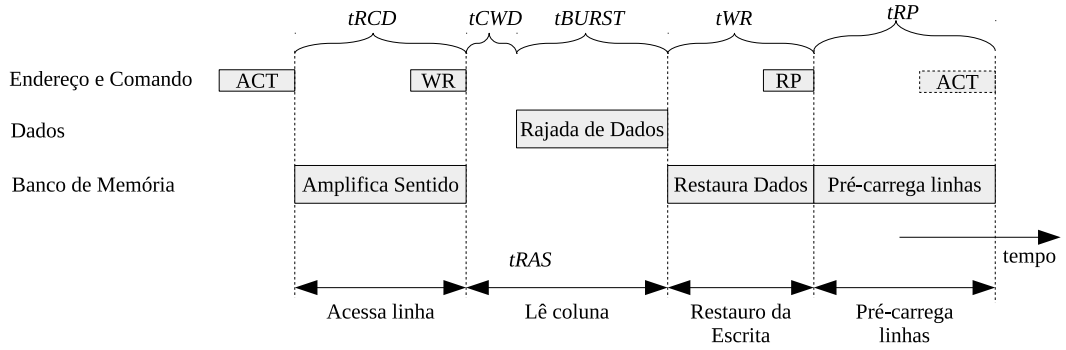
3.1.6.5 Comando de Auto-Atualização (Auto-refresh)

A fuga de carga dos capacitores da matriz de memória é compensada pela atualização automática da memória, operação que é controlada pelo circuito interno da DRAM. Periodicamente, o controlador de memória DRAM deve enviar para o dispositivo de memória o comando para realizar a atualização da informação. O parâmetro *refresh cycle time* (tRFC) determina o tempo necessário para que a memória realize a operação de atualização para uma linha inteira. Quanto maior é o número de linhas de um dispositivo DRAM, maior é o tempo necessário para realizar a operação de atualização da memória. A atualização completa de todos as linhas e bancos também pode ser feita por um comando através do controlador de memória, que controla o período de realização através do parâmetro tRFC.

3.1.6.6 Interação entre Comandos

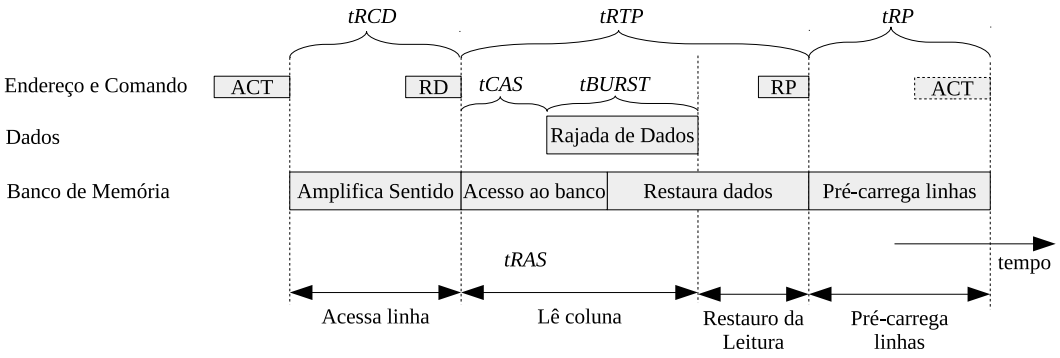
A interação entre comandos é formada pela sequência de comandos passados para a memória, que também deve respeitar limites de tempo mínimos e máximos a fim de proteger a informação que está sendo manipulada. Uma leitura ou escrita na memória é formada por uma sequencia de comandos, definida como um ciclo de acesso. Um ciclo de leitura deve iniciar com o comando de acesso de linha, o comando de leitura de coluna e finalizar com o comando de pré-carga. Um ciclo de escrita na memória deve iniciar com o comando de acesso de linha, o comando de escrita de coluna e finalizar com o comando de pré-carga. Sucessivas operações podem ser feitas dentro de uma mesma linha de um banco de memória que está aberto. É possível realizar uma sequencia de rajadas de leitura ou uma sequencia de rajadas de escrita. Da mesma forma, uma sequencia de comandos como leitura após escrita, escrita após leitura pode ser feita sobre uma mesma linha de memória, sem a necessidade de um comando de pré-carga da linha. As Figuras 3.8 e 3.7 mostram o diagrama de tempo para as etapas que formam os ciclos de escrita e leitura, e os intervalos de tempo associados, representados através dos parâmetros descritos nas subseções anteriores.

Figura 3.7: Diagrama de tempo para as etapas que formam o ciclo completo de escrita de dados na memória DRAM.



Fonte: elaborado pelo autor.

Figura 3.8: Diagrama de tempo para as etapas que formam o ciclo completo de leitura de dados na memória DRAM.



Fonte: elaborado pelo autor.

As sequências de comandos devem obedecer a limites de tempo do núcleo de memória DRAM. Isso impacta na eficiência da memória. A interação entre comandos é dependente dos dados. Dessa forma, os ciclos de acesso devem respeitar os limites de tempo estabelecidos pela estrutura interna da memória. O ciclo de linha (*tRC*) é dado por (3.1), que determina o tempo mínimo para uma escrita ou leitura de rajada simples em um dos bancos da memória DRAM.

$$tRC = tRAS + tRP \tag{3.1}$$

O ciclo completo de escrita para uma rajada simples (t_{w1}) é dado por (3.2).

$$t_{w1} = tRCD + tCWD + tBURST + tWR + tRP \tag{3.2}$$

O ciclo completo de leitura para uma rajada simples (t_{r1}) é dado por (3.3).

$$t_{r1} = max(tRC, tRCD + tRTP + tRP) \tag{3.3}$$

De uma forma genérica, o tempo de duração do ciclo de escrita e de leitura de uma sequência de rajadas de tamanho igual a n é calculado usando as Equações 3.4 e 3.5.

$$t_w(n) = tRCD + tCWD + n \cdot tBURST + tWR + tRP \tag{3.4}$$

$$t_r(n) = max(tRC, tRCD + (n-1) \cdot tCCD + tRTP + tRP) \tag{3.5}$$

Em uma leitura, o tempo necessário para que os dados estejam no barramento após a ativação de linha e inserção do comando de leitura é dado por $tRCD + tCAS$. No caso de uma escrita, os dados somente podem ser colocados no barramento após o tempo igual a $tRCD + tCWD$.

Dentro de uma mesma linha de um banco, sucessivos comandos de escrita ou leitura podem ser realizados, caracterizando um ciclo de acesso com sequencia de rajadas. Uma sequência

de escritas ou de leituras melhora o desempenho da transferência de dados para a memória, caracterizando um aumento da largura de banda. A Tabela 3.2 mostra os valores dos principais parâmetros de tempo utilizados nos ciclos de escrita e leitura, fazendo uma comparação entre as diferentes gerações de memória SDRAM. Os valores estão descritos em ciclos de relógio, com referência o relógio de sincronismo da memória. A Tabela 3.2 mostra os atrasos existentes entre diferentes comandos de uma memória DDR, em diversas gerações, e a nomenclatura que é utilizada nos padrões de memória normatizados pelo JEDEC. Dados obtidos dos manuais dos dispositivos DDR2 e DDR3 da Micron (MICRON, 2007) e (MICRON, 2009) e DDR4 da Hynix (HYNIX, 2013).

Tabela 3.2: Comparativo dos valores de atrasos entre diferentes comandos nas gerações de memória SDRAM.

Memória	***DDR2-1066***		***DDR3-1333***		***DDR3-2133***		***DDR4-2400***	
Lat. CAS	***CL = 7***		***CL = 9***		***CL = 14***		***CL = 16***	
Capacidade	2 Gb		4 Gb		4 Gb		4 Gb	
Fabricante	Micron		Micron		Micron		Hynix	
Modelo	47H256M8-187E		41J512M8-15E		41J512M8-093		H5AN4G8NMFR	
Velocidade	533 MHz		667 MHz		1066 MHz		1200 MHz	
Taxa de dados	1066 Mtps		1333 Mtps		2133 Mtps		2400 Mtps	
Parâmetro	ciclos	ns	ciclos	ns	ciclos	ns	ciclos	ns
tCK	1	1,875	1	1,500	1	0,938	1	0,833
tRC	29	54	33	49,5	50	46,13	55	45,32
tRCD	7	13,125	9	13,5	14	13,09	16	13,32
tRAS	22	40	24	36	36	33	39	32
tRP	7	13,125	9	13,5	14	13,09	16	13,32
tRRD	4	7,5	4	6	6	5	–	–
tRTP	4	7,5	5	7,5	8	7,5	–	–
tWR	8	15	10	15	16	15	–	–
tWTR	4	7,5	5	7,5	8	7,5	–	–
tCCD	4	3,25	4	6	4	3.752	–	–
tCL	7	13,13	9	13,5	14	13,132	–	–
tBURST	4	7,5	4	6	4	3,75	–	–
tFAW	19	35	20	30	27	25	–	–
tCWD	6	11,25	7	10,50	10	9,38	–	–
tRFC		195		260		260	–	–
tREFI		7800		7800		7800		3900
CL			5 – 10		5 – 14		11 – 18	

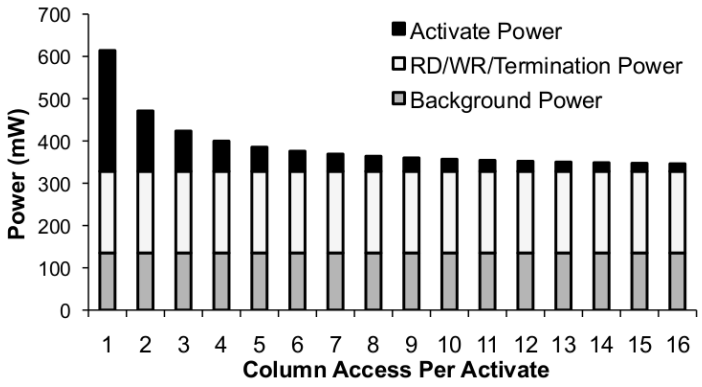
Fonte: elaborado pelo autor.

A evolução das memórias DRAM tem sido suportada principalmente pela redução da potência dissipada, redução da tensão de alimentação e aumento da largura de banda do barramento de interface com a memória. No entanto, o núcleo de memória DRAM ainda sofre de limitações severas de temporização e velocidade de acesso aos dados. Tais limitações estão relacionadas ao fato de que a capacitância das linhas de bits de dados não escala da mesma forma que a tecnologia utilizada. Dessa forma, o núcleo de memória DRAM sofre de limitações relacionadas à dois fatores principais: 1) a latência de acesso de linha e 2) a atualização interna das células de memória.

O tempo necessário para abrir uma linha de memória em um banco específico é representado pelo parâmetro *tRC*, denominado de tempo de ciclo de linha (*Row Cycle time*). O tempo necessário para completar a pré-carga da linha de memória que está aberta no núcleo DRAM é representado pelo parâmetro *tRP*, denominado de tempo de pré-carga de linha (*Row Precharge time*). O tempo de acesso de linha de memória, representado pelo parâmetro *tRC*, é o resultado da soma do tempo de acesso de linha *tRAS* e do tempo de pré-carga da linha *tRP*. Além disso, a DRAM permanece inacessível durante todo o tempo de atualização periódica, descrito pelo parâmetro *tRFC* que se repete a cada *tREFI* segundos.

Explorar a localidade espacial dos dados armazenados na memória DRAM além de trazer benefícios de desempenho reduz a potência total dissipada pelo dispositivo de memória. A Figura 3.9 mostra a redução da potência dissipada em um dispositivo de memória durante uma sequência de acessos de escrita e leitura em diferentes sequências de rajadas para a mesma página de memória. A organização dos dados em páginas de memória reduz a potência dissipada devido à redução da ocorrência de comandos de ativação de página, associado ao gasto de armazenar o conteúdo das células DRAM no buffer de linha. No exemplo mostrado é possível alcançar uma redução de potência de aproximadamente 40% acessando 8 colunas por página se comparado com o acesso de uma única coluna por página.

Figura 3.9: Potência de um chip 2Gb DDR3-1333 acessado em diferentes tamanhos de rajada para uma utilização de 40% para leituras e 20% para escritas.



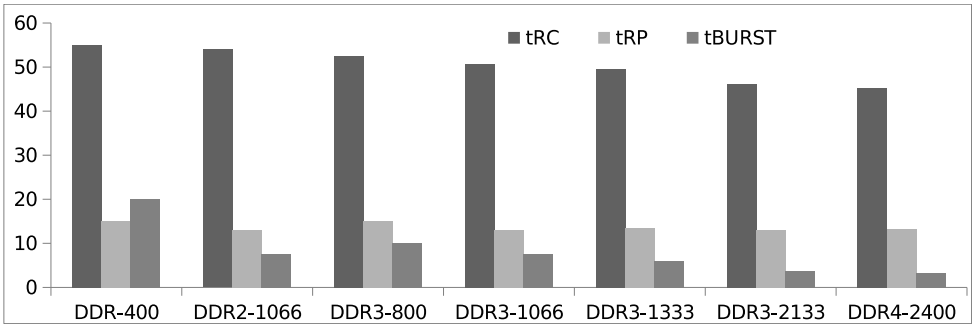
Fonte: (KASERIDIS; STUECHELI; JOHN, 2011).

3.1.7 O Problema da Latência

É possível observar que a evolução dos parâmetros *tRC* e *tRP* nas gerações DDR2, DDR3 e DDR4 é pequena. No comparativo apresentado na Tabela 3.2 mostra os parâmetros referentes à velocidade do barramento e do núcleo de memória para o modelo mais rápido de memória DDR, DDR2 e DDR4 e para alguns modelos de memória DDR3. A frequência do barramento passou de 200 MHz na DDR para 1200 MHz na DDR4 e, por consequência, o tempo necessário para transferir uma rajada de dados (*tBURST*) reduziu de 20ns para um pouco mais de 3,3ns. No entanto, a redução do tempo de ciclo de linha (*tRC*) foi de aproximadamente 17,6% e a

redução do tempo de pré-carga de linha (*tRP*) foi de aproximadamente 11,2%. A Figura 3.10 mostra um comparativo gráfico desses valores.

Figura 3.10: Atrasos dos núcleos de memória para alguns modelos de DDR2 e DDR3.



Fonte: elaborado pelo autor.

Através desse comparativo fica claro que o desempenho de uma DDR está limitado pela velocidade interna de funcionamento do seu núcleo de memória, limitado principalmente pelas capacitâncias internas de longas linhas de bits. Essa análise mostra que o ganho obtido com o aumento da largura de banda do barramento de memória, relacionado ao aumento da frequência, não é completamente aproveitado devido aos atrasos internos do núcleo de memória.

3.2 Análise da Eficiência da DRAM

Os principais fatores que influenciam na eficiência da memória são:

- Eficiência do acesso aos dados: uma transferência de dados para a memória DRAM ocorre em rajadas, que acessam um segmento inteiro de endereços mantendo uma seqüência pré-definida. O acesso em uma região de endereços menor do que o tamanho da rajada sofre problemas de desalinhamento, quando o endereço que se deseja acessar não encontra-se no início do segmento. Existe uma redução da eficiência quando o tamanho dos dados transferidos é menor do que o tamanho da rajada. Além disso, a perda pode ser maior quando existe o desalinhamento dos endereços, fazendo com que duas rajadas sejam necessárias para transferir uma determinada quantidade de dados.

- Eficiência de acesso aos bancos: a eficiência de acesso aos bancos é limitada pelo intervalo de tempo entre dois comandos de ativação em bancos diferentes. Quando um banco é aberto e é feita uma operação de escrita ou leitura nesse banco, deve-se esperar por tRRD para poder abrir outro banco diferente. O conflito de bancos é um fator que reduz significativamente a eficiência da memória e depende do tipo de tráfego de dados.

- Eficiência de troca entre escrita e leitura: a troca de operações de escrita e leitura no barramento da memória também exige um determinado número de ciclos extras, reduzindo a eficiência da transferência de dados. Esse fator de redução depende do tráfego de dados.

- Eficiência da auto-atualização da memória: durante os intervalos de tempo em que a memória está sendo atualizada nenhum outro comando pode ser realizado. Esse intervalo de atualização independe do tamanho do núcleo de memória e é controlado pela própria memória.

A eficiência do acesso aos dados depende do modo como o sistema faz um acesso de escrita ou leitura com a memória a partir da razão entre a duração da rajada de dados (em ciclos de relógio do barramento) e a duração total do acesso. Como o acesso aos dados depende de uma sequência de comandos além dos comandos de escrita ou leitura, denominaremos de transação a operação completa de acesso aos dados, que inclui a ativação e pré-carga de banco/linha (quando necessário) e os comandos de escrita ou leitura. A transação então define a operação de acesso aos dados na memória, formada a partir de uma sequência de comandos. Dessa forma, definiremos por e_T o parâmetro que descreve a eficiência de uma transação de dados entre o sistema e a memória externa (3.6).

$$e_T = \frac{ciclos\_rajada}{ciclos\_rajada + overhead} \tag{3.6}$$

O número de ciclos de uma rajada é medido pelo tempo de duração da transferência de dados no barramento de memória, representado pelo parâmetro tBURST. O tempo de acesso é medido pelo tempo de duração das etapas de ativação, acesso, restauro e pré-carga da memória, necessários para realizar uma ou mais rajadas de transferência de dados. O tempo de acesso é representado pelas equações (3.4) e (3.5), para escrita e leitura de uma sequencia de rajadas de tamanho n. Dessa forma, a eficiência da transferência de dados entre o sistema e a memória depende diretamente do uso de sequência de rajadas.

Como exemplo de eficiência da transferência, pode-se analisar a razão entre o tempo de uma única rajada e o tempo de ciclo de linha (tBURST/tRC) para memórias DDR. Essa razão é de 36,4% para DDR-400, 13,9% para DDR2-1066, 8,1% para DDR3-2133 e de 7,3% para DDR4-2400. Isso representa que a fração de tempo utilizada para efetivamente transferir dados reduz a cada nova geração de memória DDR. Dessa forma, os controladores de memória devem lidar com essa limitação de tempo através do acesso aos dados em rajadas maiores. O aumento do número de repetições das rajadas de escrita e de leitura permite melhorar a eficiência da transferência de dados entre memória e sistema.

Os gráficos apresentados na Figura 3.11 mostram um comparativo para memórias DDR2 e DDR3 sobre a largura de banda efetiva utilizada considerando diferentes modos de acesso em rajada. Essa análise já considera os ciclos gastos com a atualização automática.

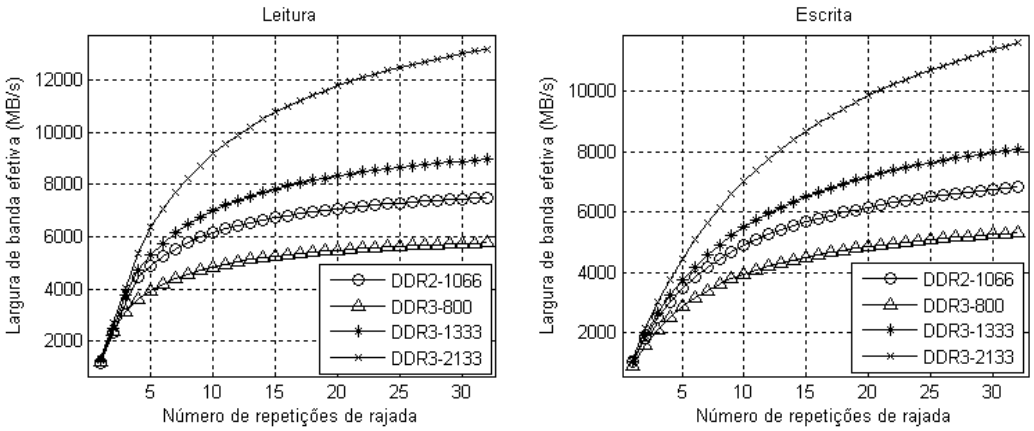
O aumento da eficiência é obtido com o aumento do tamanho das transferências entre o sistema e a memória externa. Combinando as equações (3.4), (3.5) e (3.6) tem-se as funçao de eficiência da transferência em função dos parâmetros da memória DDR, como mostrado em (3.7) e (3.8). Com essa análise pode-se verificar que a tanto a eficiência de escrita quanto a de leitura tendem a 90% com o aumento dos parâmetros n e tBURST, como mostrado na Figura

3.12. Essa análise que deve ser feita para prever o desempenho de um sistema de memória, pois a largura de banda efetiva durante escritas ou leituras na memória externa depende da sequência de rajadas que é utilizada. Tanto a eficiência de escrita como a de leitura são parâmetros que dependem do número de rajadas consecutivas durante o acesso aos dados, já que os parâmetros da memória são constantes, para uma mesma tecnologia de memória.

$$e_{Tw}(n) = \frac{tBURST}{tBURST + \frac{tRCD+tCWD+tWR+tRP}{n}} \tag{3.7}$$

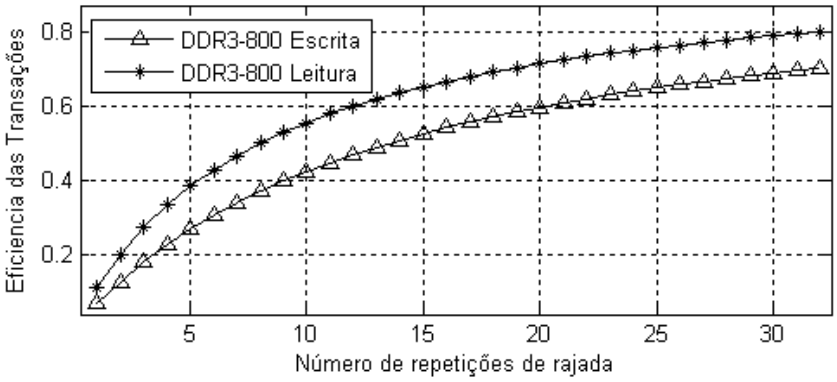
$$e_{Tr}(n) = \frac{tBURST}{tCCD + \frac{tRCD+tRTP+tRP-tCCD}{n}} \tag{3.8}$$

Figura 3.11: Largura de banda para escrita e leitura de dados variando a granularidade dos acessos.



Fonte: elaborado pelo autor.

Figura 3.12: Eficiência das transações de escrita e leitura para a DDR3-2133 em função do número de repetições de rajada.



Fonte: elaborado pelo autor.

Percebe-se em 3.7 e 3.8 que o *overhead* de acesso em uma transação diminui com o aumento do parâmetro n. Interessante é mostrar que para todas gerações de memória, fazer acessos simples significa não ter ganho de desempenho em largura de banda. Isso está relacionado à latência do núcleo de memória. Essa análise considera duas formas de eficiência descritas anteriormente: a eficiência de acesso aos dados e a eficiência de acesso aos bancos. Nessa análise que resulta na Figura 3.12 cada acesso é direcionado para uma nova linha da memória, significando que é necessário fazer a troca entre linhas completando a operação de acesso com um comando de pré-carga. A eficiência de acesso aos bancos está relacionada à capacidade de paralelização do acesso à memória, de forma que um ou mais bancos podem estar abertos simulaneamente e uma sequência de rajadas de escrita ou leitura pode ser concatenada para diferentes bancos abertos na mesma memória. Para exemplificar, uma memória DDR3 possui 8 bancos que podem estar abertos simultaneamente, então uma transação de escrita ou leitura com sequência de 8 rajadas pode ser feita 1 por banco de memória, mas somente nas linhas que estão abertas em cada banco.

O último fator que interfere no desempenho é a eficiência de auto-atualização do núcleo DRAM, que depende do número de bancos de memória e do número de colunas por linha. O intervalo entre atualizações de todas as linhas de memória em todos os bancos é dado pelo parâmetro tREFI, que tipicamente em dispositivos DDR2 e DDR3 é de 64 ms. O controle da auto-atualização é feito internamente à DDR, através de um contador de atualização de linhas, que realiza uma operação de auto-atualização a cada 7,8 μs. Nas gerações anteriores de DRAM, os fabricantes mantiveram o valor do contador utilizado para atualização igual a 8.192, que era igual ao número de linhas de uma memória nas gerações SDR e DDR. No entanto o número de linhas e bancos e o tamanho das linhas aumentou nas gerações DDR2 e DDR3 e o contador foi mantido em 8.192, mesmo em dispositivos com 2, 4 e 8 bancos e com 16k e 64k de tamanho das linhas de memória. Dessa forma, o tempo de auto-atualização das memórias modernas aumentou significativamente (JACOB; NG; WANG, 2007). A fração de tempo que um dispositivo DRAM leva para fazer a auto-atualização das linhas de memória pode ser calculada pela relação $tRFC/tREFI$. A Tabela 3.3 mostra a relação entre o tempo de auto-atualização da memória nos dispositivos de DDR de diferentes capacidades de armazenamento e gerações[3].

A eficiência de acesso aos bancos e de troca de escrita e leitura também são dois fatores que podem ser controlados pelo subsistema de memória, embora sejam dependente dos dados. A eficiência de acesso aos bancos é melhorada se o controlador de memória mantiver um registro dos bancos abertos, fazendo o controle de bancos, reduzindo o tempo de ativação de banco e linha para os acessos. A eficiência de troca entre escrita e leitura também pode ser melhorada com políticas de ordenamento dos acessos pelo controlador de memória.

[3]Para os valores de tempo para a duração do tempo de auto-atualização para cada linha (tREF) para DDR4 foram usados os valores da DDR3, já que os datasheet atuais ainda não apresentam esses valores.

Tabela 3.3: Aumento do tempo de atualização em um comparativo entre famílias e capacidade de armazenamento:

	DDR		***DDR2***		***DDR3***		***DDR4***	
	tRFC	*% tempo*	*tRFC*	*% tempo*	*tRFC*	*% tempo*	*tRFC*	*% tempo*
256 Mb	75 ns	0,9 %	75 ns	0,9 %				
512 Mb	75 ns	0,9 %	105 ns	1,3 %				
1 Gb	120 ns	1,5 %	128 ns	1,6 %	110 ns	1,4 %	110 ns	2,8 %
2 Gb			195 ns	2,5 %	160 ns	2 %	160 ns	4,1 %
4 Gb					260 ns	3,3 %	260 ns	6,7 %
8 Gb					350 ns	4,5 %	350 ns	9,0 %

Fonte: elaborado pelo autor.

3.2.1 Soluções Propostas

Como o projeto de novos sistemas, aplicativos e tecnologias tendem a exigir mais capacidade e largura de banda, a eficiência e a previsibilidade do sistema de memória torna-o um gargalo do sistema ainda mais significativo. Ao mesmo tempo, a tecnologia DRAM está enfrentando desafios de escala da tecnologia difíceis que fazem a manutenção e reforço da sua capacidade, eficiência energética e confiabilidade significativamente mais caros utilizando as técnicas convencionais de projeto. Diferentes abordagens de pesquisa e projetos promissores para superar os desafios os colocados pela escala de memória podem ser classificadas em três direções (MUTLU, 2013): 1) novas arquiteturas e funcionalidades de DRAM, que permitam uma melhor integração com o resto do sistema; 2) criação de um sistema de memória que emprega tecnologias de memória emergentes e tirando proveito das diferentes tecnologias; 3) proporcionar um desempenho previsível e QoS para aplicações que compartilham o sistema de memória.Algumas soluções são propostas para melhorar o desempenho dos subsistemas de memória e são descritas nas sub-seções seguintes[4]:

3.2.1.1 Redução do Tempo de Auto-Atualização

Relativo à arquitetura interna da memória, são apontadas como necessidades de melhoria para o funcionamento da DRAM a redução do impacto do tempo de auto-atualização da memória. Como discutido na seção 3.2, o tempo gasto com a auto-atualização do conteúdo de memória vem aumentando significativamente com o aumento da densidade dos dispositivos DRAM. É necessário desenvolver novas tecnologias para a redução do impacto do tempo de atualização interna da memória. Para um dispositivo de memória hipotético com densidade de 64 Giga-bits, o tempo de atualização corresponde à 47% do tempo de uso da memória e a energia gasta nessa operação corresponde à 47% da energia total da memória (LIU et al., 2012).

3.2.1.2 Redução da Latência do Núcleo de DRAM

Memórias DRAM especializadas para baixa latência podem ser fabricadas com linhas de bits menores, contendo um número menor de células de armazenamento. No entanto, o custo

[4]Nessa subseção não trataremos sobre o uso de tecnologias emergentes de memórias.

desse tipo de memória é maior pois necessita de um número maior de amplificadores diferenciais que gera um aumento de área. LEE et al. (2013) propõem o projeto de uma DRAM com estrutura interna organizada em níveis de latência, combinando em um único dispositivo as características de custo baixo por bit e de baixa latência. O uso de linhas de bits longas reduz o custo por bit e no mecanismo proposto por LEE et al. (2013), transistores de isolamento são utilizados para dividir em dois segmentos mais curtos. Outra abordagem para reduzir a latência da DRAM é proposta por Son et al. (2013), modificando a organização interna de bancos do núcleo de memória. Nesse artigo, os autores propõem uma arquitetura assimétrica de organização de bancos de um dispositivo DRAM com foco na redução da latência média nos acessos à memória. O artigo mostra um exemplo de aplicação com dispositivos de memória com um quarto dos bancos modificados para uma razão de aspecto 2x maior. Nesse exemplo, o aumento da área do chip é de 3% mas a melhora IPC (*Instructions per Cycle*) e do EDP (*Energy-Delay Product*) são de 21% e 32%, respectivamente, em aplicações mono-tarefa, se comparado à uma organização regular.

3.2.1.3 Uso de Operações Gerenciadas Internamente à DRAM

Mover dados para fora da DRAM é uma operação que emprega um gasto elevado de energia, devido ao uso de transferência de dados pelos pinos de entrada e saída em frequências elevadas. Algumas operações podem ser feitas internamente à DRAM sem que seja necessário transferir externamente os dados. Operações como inicialização de uma página de memória ou cópia de uma página de um endereço para outro podem ser feitas internamente à memória, com algumas modificações arquiteturais internas. Como a DRAM opera a partir de comandos, a inserção de novos comandos seria uma tarefas que diria respeito somente à arquitetura interna da memória, sem a necessidade de modificar interfaces. Se os dados copiados ou inicializados não são necessários imediatamente pelo processador, tais operações poderiam ser realizadas internamente à DRAM com uma grande redução de energia, tempo e largura de banda.

A ideia principal desse método é capturar o conteúdo de uma linha de origem, gravado no conjunto de amplificadores diferencias do seu respectivo banco após a operação de ativação, e movê-la para uma nova linha no mesmo banco. Seshadri et al. apresentam uma implementação inicial dessa proposta, onde os autores verificam que esse mecanismo de cópia reduz a latência e consumo de energia da cópia de uma linha de 4 kB em 11,6 e 74,4 vezes, respectivamente (SESHADRI et al., 2013).

3.2.1.4 Projeto Orientado à Memória

O projeto de sistemas de computação deve ser orientado para o funcionamento e características do subsistema de memória. Memória é um recurso compartilhado entre múltiplos elementos de processamento do sistema, que possuem diferentes características de acesso e formatos de transferência de dados. O projeto do subsistema de memória deve prover a largura de banda e capacidade de alocação de clientes para o sistema à qual serve os dados. A fim de satisfazer

essas características, uma linha de pesquisa se dedica ao desenvolvimento de controladores de memória projetados para prover qualidade de serviço. O desenvolvimento de mecanismos de controle de acessos mais eficientes e sistemas com comportamento predizível no tempo premitem que os controladores de memória possam minimizar os problemas de latência de acesso aos dados e também maximizar a largura de banda máxima efetiva da memória. Um controlador de memória externa, tipicamente uma DRAM, é um elemento do subsistema de memória que implementa uma interface rígida com a memória externa, traduzindo comandos para a memória e controlando a transferência de dados. Com o advento do projeto conjunto do sistema e DRAM, o controlador de memória deixa de ser um elemento passivo do sistema de computação para tornar-se um elemento ativo que avalia o comportamento do sistema e controla os acessos aos dados. O projeto de controladores de memória que considerem os requisitos de qualidade de serviço é uma proposta promissora que modifica a forma de concepção de um sistema integrado pois determina que algumas diretivas de projeto devem estar orientadas para a integração dos módulos do sistema com o subsistema de memória.

3.3 Organização de Subsistemas de Memória

O número de transistores duplica a cada dois anos, assim como essa tendência ocorre em circuitos lógicos, a capacidade de armazenamento das memórias DRAM também duplica em poucos anos. As memórias DRAM são usadas externamente ao chip do circuito lógico, pois são fabricadas utilizando um processo diferente. Chips dedicados para memórias DRAM são encapsulados separadamente do chip do circuito lógico e conectadas a ele através das conexões de uma placa de circuito impresso.

Com a crescente complexidade dos sistemas computacionais, o projeto de um sistema de computação é feito em partes separadas agrupadas na forma de subsistemas. No entanto, o projeto da hierarquia de memórias não pode ser projetado e otimizado isoladamente. Podemos definir um subsistema de memória como memórias cache, DRAM e disco. No entanto, o subsistema de memória engloba mecanismos de transação e de sinalização, de gerenciamento de dados, de filas para armazenamento temporário e de troca de domínios de relógio. Para cada um desses itens, que geralmente envolvem grande complexidade relacionada à composição e funcionamento do sistema, é necessária uma investigação para otimizar o projeto. Quando esses subsistemas são combinados em um único sistema de elevada complexidade, o problema de otimização resultante é simplesmente extraordinário.

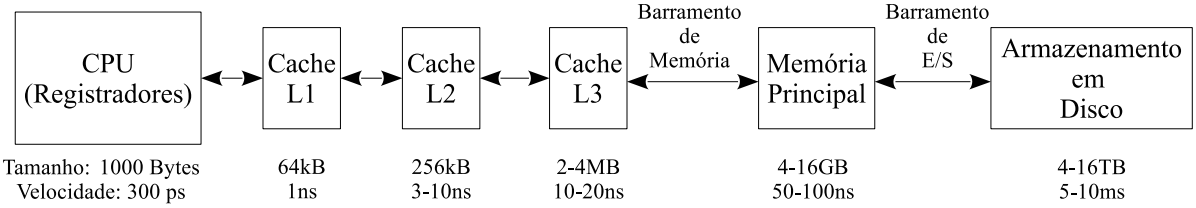
Recentemente, novas tecnologias de integração de circuitos integrados permitiram a evolução dos encapsulamentos para comportar mais de uma pastilha de silício dentro do mesmo encapsulamento. Dessa forma, os chips de memória DRAM são colocados mais próximos ao circuito lógico, através de conexões com dimensões físicas menores. Em ambas as abordagens, a interface entre o chip de memória e o chip do circuito lógico é feita através de um Controlador de Memória Externa (CME). Múltiplos chips de memória são conectados a um circuito lógico

para aumentar a capacidade de armazenamento de dados. Os dispositivos de memória podem ser conectados em paralelo, aumentando a largura do barramento de dados e aumentando o espaço de endereçamento disponível.

O subsistema de memória é formado pelo conjunto de memórias internas e externas ao chip, distribuídas através dos elementos de processamento do sistema como memórias locais ou compartilhadas. O subsistema de memória é parte integrante de um sistema de processamento, sendo formado por uma estrutura hierárquica de memórias de diferentes tamanhos, aspectos e tecnologias de fabricação. O funcionamento do sistema de computação depende do funcionamento do subsistema de memória, que por sua vez depende do funcionamento de cada parte de memória que compõe a sua hierarquia.

Denomina-se de hierarquia de memórias o conjunto de heterogêneo de elementos de memória que forma uma estrutura redundante de armazenamento da informação. O princípio da localidade da referência é o guia para a criação de uma hierarquia de memórias. Cada nível da hierarquia de memória tem características próprias de fabricação, dimensão e funcionalidade que o difere dos demais níveis. A memória mais próxima ao elemento de processamento tem tamanho menor mas velocidade maior, portanto é acessada mais vezes para uma troca rápida de informações. Já a memória que está mais distante do elemento de processamento é de maior capacidade porém mais lenta, de forma que é acessada um número menor de vezes para transferir grandes quantidades de informação. Intermediariamente existem outros tipos de memória que formam a hierarquia, dimensionadas para cada sistema de computação específico. A Figura 3.13 mostra um diagrama simplificados de hierarquia de memórias em processadores modernos, contendo três níveis de memória cache, DRAM e disco, utilizada em sistemas de computação de alto desempenho.

Figura 3.13: Diagrama simplificado dos níveis que formam uma hierarquia de memória em um computador.



Fonte: elaborado pelo autor.

Um processador composto por quatro núcleos rodando a 3.2 GHz precisa buscar cerca de 25,6 bilhões de referências de dados de tamanho 64 bits por segundo, mais 12,8 bilhões de referências de instruções de 128 bits por segundo. Isso gera um total de 409,6 GB/s (HENNESSY; PATTERSON, 2011). No entanto, a DRAM utilizada por esse processador consegue prover apenas 6% dessa largura de banda, que é igual a 25 GB/s. Portanto micro-arquiteturas modernas de processadores utilizam caches em dois níveis para cada núcleo com um terceiro

nível de cache compartilhada entre todos os núcleos. Além disso, as caches internas dos processadores tem aumentado de tamanho e atualmente ultrapassam os 10 MB para a maioria dos processadores.

Em Multi-Processor Chips (MPC), como a micro-arquitetura Intel i7, o último nível de cache é utilizado para compartilhar informações entre as tarefas dos núcleos do processador. Esse último nível de cache faz interface com o nível intermediário de cache e também com o controlador de memória externa. Na presença da falta de referência no último nível de cache(*cache miss*) a busca é feita na memória externa com acessos de tamanho que dependem tamanho do bloco da cache de último nível e do número de caminhos de associatividade. Em geral, as arquiteturas recentes de processadores utilizam caches com blocos de tamanho igual a 64B (também denominado de *cache line*). Para aumentar o paralelismo da estrutura de cache, a organização de blocos de dados é mapeada para bancos de cache. Essa configuração de acesso buscas na memória externa em forma de sequências de rajadas para bancos que estão abertos na memória, aumentando o paralelismo na memória externa e aproveitando a localidade espacial de organização de dados nas linhas da memória.

Com a microarquitetura Nehalem (45 nm), Intel introduziu um nível extra de cache on-chip, a cache L3, com um controlador de memória externa integrado ao chip do processador. A versão seguinte da microarquitetura Intel Core i7 denominada Westmere (32 nm) possui uma hierarquia de memória formada por três níveis de cache e um controlador de memória integrado conectado ao 3o nível de cache através de uma fila global com 96 posições de 16 bytes. As buscas feitas na cache L3, tanto as que resultam em acertos ou faltas, resultam em transferências de dados através de uma fila global. O controlador de memória integrado ao processador possui três canais de memória DDR3-1333 com 8 bytes de largura por canal, resultando em uma largura de banda máxima teórica de 31,992 GB/s. Teoricamente cada um dos seis núcleos pode acessar a memória externa em uma taxa média de 5,332 GB/s. O controlador de memória integrado possui tecnologias *Just-in-Time Command Scheduling*, *Command Overlap* e *Out-of-Order Scheduling* (INTEL, 2014).

Memórias caches com associação em conjuntos (*set-associative*) usam múltiplos bancos da DRAM que são acessados em paralelo para buscar um item desejado armazenado em memória. Uso de múltiplos bancos (2-8) para cada dispositivo DRAM melhora a gestão de energia e permite intercalar acessos entre bancos, pois múltiplos bancos podem permanecer abertos simultaneamente, reduzindo penalidades de acesso e melhorando o desempenho (HENNESSY; PATTERSON, 2011). A associação de conjuntos em uma cache L3 de um processador Intel Core i7 é feita em 16 vias, com latência de acesso no caso de uma falha igual a 35 ciclos de relógio do processador, para detectar da falha, mais a latência da DRAM para instruções críticas (com maior prioridade de busca) (HENNESSY; PATTERSON, 2011). O tamanho do bloco (ou linha) de cache é igual a 64 bytes, que é a quantidade de dados transferida quando ocorre uma cache miss em um dos níveis de cache. Para um único banco de uma DDR3-1600, a latência da DRAM é de 35ns ou 100 ciclos de relógio do processador para buscar os primeiros 16 by-

tes, resultando em uma latência total de 135 ciclos de relógio do processador. o controlador de memória consegue completar o restante do bloco de cachê de 64B em uma taxa de 16 bytes por ciclo de relógio da memória, o que leva outros 15ns ou 45 ciclos de relógio do processador. O Core i7 contém um buffer de escritas pendentes de 10 entradas, usado para escrever novamente nas linhas de cache modificadas quando o nível seguinte de cache não está sendo usado para leitura.

Com o advento de memórias DRAM embarcadas no mesmo encapsulamento, denominadas de eDRAM, existe a possibilidade de criar uma hierarquia de memória com um novo nível de cache. Explorando essa nova possibilidade a Intel fabrica o Core i7 de microarquitetura denominada Haswell-H, um novo multichip de quatro núcleos fabricado em tecnologia 22 nm tri-gate (KURD et al., 2014). O Haswell-H que integra no mesmo encapsulamento uma pastilha contendo a CPU, unidade de gráficos e multimídias e em outra pastilha uma eDRAM com capacidade de 128 MB fabricada em tecnologia 22 nm tri-gate CMOS atuando como uma cache L4 (4o nível) com largura de banda máxima igual a 102 GB/s (HAMZAOGLU et al., 2014). O controlador de memória utilizado nessa micro-arquitetura também está embarcado dentro da pastilha da CPU e suporta 2 canais de DDR3-SDRAM ou LPDDR3, com alimentação de 1.2 até 1.5V, com 2 canais de memória chegando a 25,6 GB/s de largura de banda máxima.

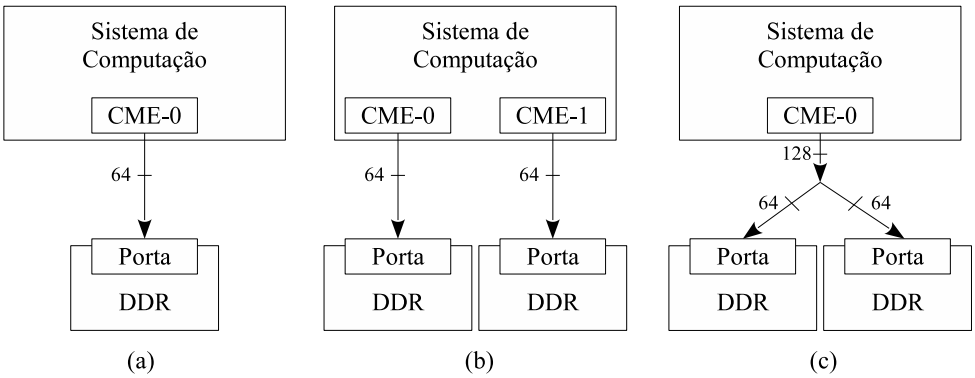
3.3.1 Controladores de Memória Externa

Um controlador de memória é responsável por interpretar os comandos de escrita ou leitura gerados pelo sistema e gerar uma sequência adequada de comandos para a memória DRAM. Os endereços do sistema (espaço linear de endereçamento) são traduzidos para os endereços da memória DRAM, gerando o acesso aos dados organizados em bancos, linhas e colunas no espaço de memória. Além disso, o controlador de memória é responsável por manipular os dados de forma adequada durante escritas e leituras. O controlador de memória faz interface com a memória externa através de um "canal de comunicação". O termo canal é usado para denominar a ligação entre o controlador de memória e a porta da memória DRAM. Porta da memória é o termo usado para denominar o conjunto de pinos de entrada e de saída do dispositivo de memória. A largura da porta de dispositivos de memória DDR pode ter largura de 4, 8 ou 16 bits, tamanhos padronizados pelas normas JEDEC. Um pente de memória DRAM pode ser formado a partir de dois conjuntos de dispositivos DRAM, formando duas fileiras de memória. Cada um dos conjuntos permanece conectado aos mesmos barramentos de comando, dados e endereço, mas com seletores de chip diferentes. Dessa forma, o espaço de endereçamento é duplicado.

Em processadores de propósito geral para servidores e notebooks cada canal físico de memória geralmente tem largura de 8 bytes, agrupando vários dispositivos de memória em paralelo. Já em sistemas computação embarcada esse valor pode variar dependendo das necessidades da aplicação para o qual a plataforma alvo foi projetada. Um controlador de memória também pode ser usado para controlar dois pentes de memória em paralelo, através de dois canais físicos. Nesse caso, o controlador de memória tem largura de dados igual a 128 bits e

cada pente de memória tem largura do canal físico igual a 64 bits. A Figura 3.14 mostra as configurações para canal simples. Um sistema com canal duplo de acesso à diferentes memórias é formado usando dois controladores de memória independentes, conectados à duas ou mais memórias independentes.

Figura 3.14: Subsistemas de memória com: a) canal de largura 64 bits e uma porta de dados de 64 bits, b) dois controladores de memória independentes e c) canal de largura 128 bits e duas portas de dados de 64 bits cada.

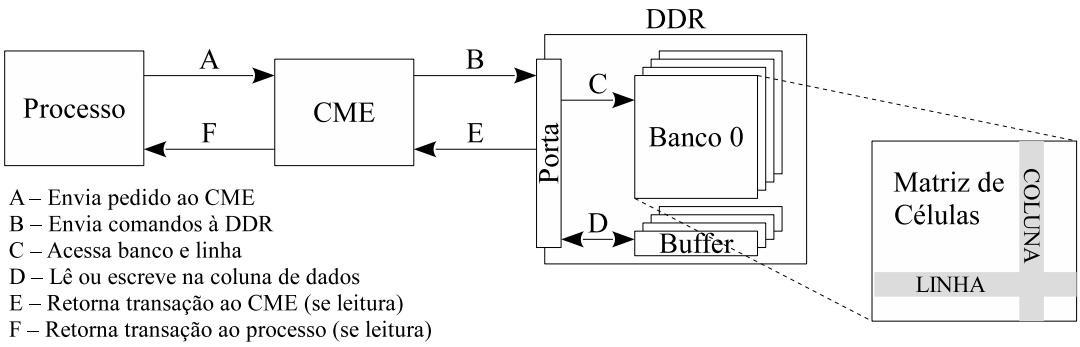


Fonte: elaborado pelo autor.

O acesso aos dados armazenados em uma memória DRAM feito através de um CME tem tempos de acesso maior do que o acesso direto à uma memória local, próxima ao elemento de processamento. A Figura 3.15 mostra as etapas que compõem uma transação de dados entre um processo requisitante e a memória DRAM, através de um CME. Quando o processo inicia uma transação de dados à memória externa (etapa A), ele envia um pedido ao CME que faz a tradução de endereços e também de comandos. A tradução de endereços (etapa B) é necessária pois o espaço de endereçamento visto pelo processo é linear, enquanto que o espaço de endereçamento na memória DRAM é na forma banco, linha e coluna. O CME deve conter um registro dos bancos e linhas que estão ativos (carregados no buffer), a fim de controlar a abertura e fechamento de linhas para cada nova transação solicitada pelo processo. Na Figura 3.15, a etapa C é necessária somente se o banco e linha solicitados pela transação atual não estiverem abertos. O acesso aos dados é feito na etapa D, que completa a transação de escrita na memória. As etapas E e F completam a transação de leitura de dados, no qual a memória DRAM passa o conjunto de dados lidos para o CME, que por sua vez passa os dados para o processo.

Uma transação de dados para a memória externa depende de uma sequência de etapas que podem ou não ocorrer, dependendo do tipo de acesso (escrita ou leitura) e da região de memória que está sendo acessada. Dessa forma, existe uma diferença de tempo entre o instante da requisição da transação até que ela seja completada inteiramente. Essa diferença de tempo é conhecida por latência da memória. A latência que o subsistema de memória gera ao gerenciar as transações é implícita ao seu funcionamento, de forma que o sistema deve poder lidar com esse problema da melhor forma para melhorar a eficiência das transações. O sistema não

Figura 3.15: Etapas de uma transação de dados entre processo e memória DRAM.



Fonte: elaborado pelo autor.

é capaz de prever com exatidão o tempo necessário para completar cada transação, a não ser que ele seja capaz de verificar o tipo de acesso e a faixa de endereços acessada. Além disso, a latência implica proporcionalmente na eficiência das transações, e por consequência, na largura de banda máxima de memória disponível para o sistema. Quanto maior é a latência em cada transação, menor é a largura de banda que o sistema consegue utilizar do canal de memória.

3.3.2 Controlador Multi-Cliente

O subsistema de memória apresentado e discutido na seção anterior considera somente um único processo acessando a memória externa. Em sistemas modernos de computação, como as arquiteturas de processamento de uso geral de computadores pessoais IBM86 e AMD64, o controlador de memória externa está inserido internamente ao encapsulamento do processador. Mesmo que a unidade de processamento seja formada por dois ou mais núcleos, um único controlador de memória é utilizado para controlar memórias externas e a comunicação entre processadores e o controlador é feita através de memórias *cache*.

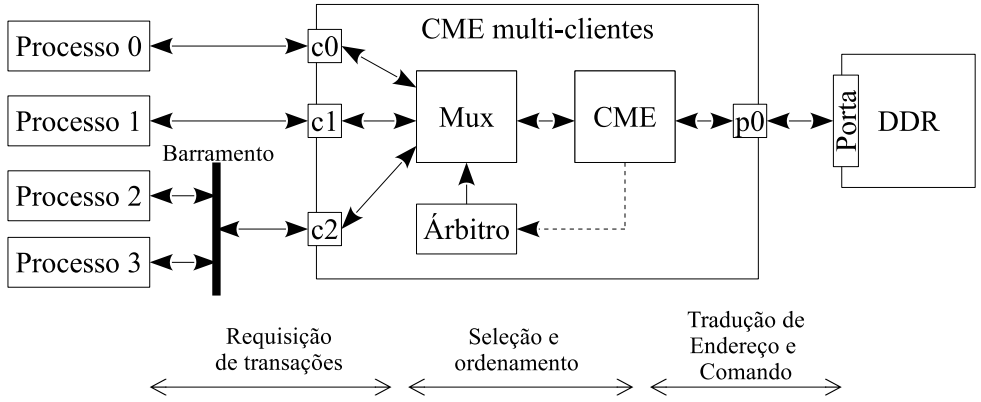
A expansão de um número maior de processos integrados à um único controlador de memória externa permite a integração direta de novas arquiteturas multi-núcleo à memória principal. Anteriormente, essa expansão era possível através de arquiteturas de comunicação como barramentos e redes-em-chip, de forma que o controlador de memória é integrado como um outro elemento do sistema. O termo cliente é utilizado aqui para definir uma interface de acesso ao controlador de memória, ou seja, uma interface que seja capaz de gerar os pedidos de transações à memória externa.

A proposta de uma arquitetura de controlador de memória externa multi-cliente tem como principal objetivo resolver dois dos problemas mais importantes relacionado ao desempenho do sistema de memória. Em primeiro lugar, como o controle das transações à memória externa é feito pelo controlador de memória, a escolha da sequência de clientes feito pelo controlador de memória apresenta vantagens em relação à escolha feita pelo mestre de barramento do sistema (ou pelo roteador da rede-em-chip). Dessa forma, é possível escolher o algoritmo de escalonamento e ordenamento mais adequado para o controle das transações. Em segundo lugar,

como o controlador de memória mantém um registro de bancos e linhas que estão abertos, o controlador de memória pode reordenar as transações dos clientes, de forma a fazer o melhor aproveitamento dos acessos na memória externa.

Controladores de memória multi-cliente possuem mais de uma interface de entrada de comandos e uma ou mais canais de acesso à memória externa. A Figura 3.16 mostra um exemplo de arquitetura de controlador de memória multi-cliente contendo três interfaces de acesso de clientes (c0, c1 e c2) e uma interface para canal de memória (p0). O multiplexador-demultiplexador (Mux) é utilizado para fazer o chaveamento de comandos, endereços e dados entre os clientes e o CME. O árbitro é utilizado para gerenciar as transações que são feitas para o CME, implementando um algoritmo de escalonamento. O CME retorna para o árbitro as informações sobre o estado atual da memória externa, como a lista de bancos e linhas que estão abertas e se a memória está em estado de auto-atualização.

Figura 3.16: Arquitetura de um controlador multi-cliente de memória externa.



Fonte: elaborado pelo autor.

Diferentes algoritmos de escalonamento podem ser empregados no gerenciamento do fluxo de dados no subsistema de memória, caracterizados pelo tipo de prioridade empregada e pelo tamanho das transferências. Os modos de escalonamento podem ser classificados em algoritmos de arbitragem por fila de ordem de chegada, por fatia de tempo ou por classificação de prioridades. A arbitragem por fatia de tempo é denominada de Round-Robin e faz a escolha do cliente seguindo uma ordem de rodízio, sem obedecer a qualquer critério de seleção. Da mesma forma a arbitragem por fatia de tempo escolhe o cliente durante um intervalo de tempo e após isso faz a troca para o seguinte. Esse modo de arbitragem pode ser programado para transferências simples, com tamanho fixo ou com tamanho variável. Os modos de arbitragem por prioridade são usados em algoritmos que estabelecem certos critérios para escolha do cliente que acessará o canal de memória. A arbitragem por prioridade pode ser classificada em prioridade fixa ou dinâmica, onde o pedido de transferência é acompanhado de um valor de prioridade da transferência. Os modos de arbitragem por prioridade também podem ser de transferência simples, múltipla ou com tamanho variável. A preempção é quando um cliente de maior prioridade consegue interromper temporariamente a transação que está sendo feita por outro cliente de menor

prioridade. A preempção deve ser empregada com alguns cuidados no subsistema de memória, a fim de não degradar o desempenho das transferências. O excesso de ocorrências de preempção provoca o aumento da troca de clientes que estão acessando a memória, podendo também provocar o aumento de trocas de linhas e bancos na memória DRAM.

3.4 Resumo do Capítulo

Este Capítulo apresentou uma descrição do funcionamento de memórias DRAM, explorando os aspectos do comportamento temporal dos dispositivos de memória e relacionando aspectos desempenho e latência de acesso. Além disso, este Capítulo apresentou uma análise do desempenho e latência para alguns tipos de memória DDR SDRAM mais utilizadas atualmente, salientando como a latência de acesso ao núcleo da memória interfere no desempenho. Ao final, o Capítulo faz uma descrição de hierarquia de memórias para sistemas computacionais e descreve conceitos importantes para o funcionamento dos controladores de memória externa. O Capítulo seguinte trata da proposta de um método de projeto de subsistemas de memória, a partir do fluxo de projeto centralizado em memória, apresentando o trabalho que foi desenvolvido até o momento.

4 METODOLOGIA DE PROJETO

A metodologia usada para o projeto e a implementação do controle adaptativo de gerenciamento dos acessos à memória externa é apresentada neste Capítulo. O método proposto é descrito através de um fluxo de projeto centralizado em memória, descrevendo seus princípios fundamentais usados na integração de elementos de processamento heterogêneos em um SoC. O objetivo principal é projetar um sistema de gerenciamento de transações com a memória principal de armazenamento de informações que tenha características de heterogeneidade, previsibilidade, adaptação e escalabilidade, descritas anteriormente na Seção 1.2. O método proposto e os resultados apresentados são direcionados ao desenvolvimento de plataformas de processamento multimídia, compostas a partir de uma quantidade crescente de elementos heterogêneos de processamento como processadores e aceleradores em hardware.

Ao longo deste Capítulo é descrita a arquitetura de um controlador de memória externa com suporte para múltiplos clientes, com capacidade de gerenciamento das transações entre clientes e memória prevendo piores casos de acesso aos dados. A análise do pior caso de tempo de resposta para um cliente é feita calculando o tempo necessário para que dados e comandos sejam transferidos entre memória e clientes através dos elementos do subsistema de memória. É proposto um modelo para estimar os piores casos, que é implementado pela arquitetura do controlador de memória e faz análise em tempo de execução. O cálculo do pior caso para conclusão das transações com a memória é usado pelo árbitro do controlador como parâmetro de entrada da função que classifica a prioridade dos acessos. Este trabalho descreve a proposta de um algoritmo de gerenciamento das transações baseado na classificação dos acessos gerados pelos clientes e requisitos de prazo para conclusão das transferências.

A arquitetura contém três elementos principais: 1) árbitro de controle de acessos, 2) módulo de interface com os clientes e 3) módulo de interface com a memória externa. A interface com os clientes contém um elemento de memória temporária do tipo *buffer*, que faz o armazenamento temporário de comandos e dados durante as transferências entre a interface de clientes e a memória externa. O elemento de controle é um árbitro de gerenciamento de acessos e seleção de clientes, que implementa um algoritmo de decisão e ordenamento para gerenciar os pedidos de acesso gerados nas interfaces de clientes. Além disso, o elemento de controle é responsável por atualizar as informações de tráfego e calcular os parâmetros para ajuste dinâmico do sistema

de memória. O árbitro contém um elemento de multiplexação é usado para controlar o fluxo de informações através do subsistema de memória. Podem-se ter múltiplos níveis de arbitragem e bufferização, de acordo com a topologia de subsistema de memória utilizada. Essa estrutura regular de integração de elementos é necessária para ter previsibilidade do funcionamento do controlador, que deve ser conhecida para gerenciar os acessos com segurança dentro dos tempos de execução possíveis de uma aplicação (AKESSON; GOOSSENS, 2011).

O fluxo de projeto proposto neste trabalho descreve a implementação do subsistema de memória com controle adaptativo de acesso dos clientes baseado nas informações de tamanho e de prazo de conclusão das transações. O gerenciamento dos acessos no controlador de memória é feito através de um controle dinâmico, que regula o acesso dos clientes prioritários e ajusta a granularidade mínima das transações. A granularidade impacta na largura de banda sustentada pela memória DRAM, ou seja, a largura de banda efetiva da memória. Como visto anteriormente, o acesso frequente para diferentes linhas de bancos de memória gera um *overhead* de acesso que reduz significativamente a largura de banda sustentada pela memória. Dessa forma, é preciso explorar a localidade espacial garantindo que clientes acessem a memória em sequências de rajadas, dessa forma provendo múltiplos ciclos de acesso de coluna para um único acesso de linha. Como parte do método de trabalho proposto, é feita a avaliação do uso de transações com sequências de rajadas maiores e o seu impacto no desempenho dos acessos. Essa análise mostra que a largura de banda sustentada da memória pode ser maximizada com o aumento da granularidade, mas em contrapartida existe o aumento da latência dos acessos. Existe um compromisso entre largura de banda e latência que influencia os limites de funcionamento do sistema.

Como parte do método é proposto um modelo baseado em atrasos para determinar os piores casos de desempenho para os acessos gerados por um conjunto de clientes. O funcionamento do modelo é analisado e seus limites são avaliados dentro dos limites previstos de funcionamento do sistema de memória. Por fim, a descrição do método apresenta a proposta do controlador de memória com árbitro adaptativo para agendamento de acessos dos clientes. A arquitetura do árbitro adaptativo e o funcionamento dos seus módulos principais são descritos concluindo o método.

Este Capítulo está organizado da seguinte forma. A seção 4.1 apresenta o projeto do subsistema de memória baseado em um controlador adaptativo para DRAM. A seção 4.2 apresenta o modelo matemático utilizado para prever o comportamento do subsistema de memória frente às diferentes características de acesso dos clientes conectados à esse subsistema. A seção 4.3 apresenta a descrição da implementação do controlador de memória e dos módulos que compõem o árbitro adaptativo que leva em conta os requisitos de acesso dos clientes.

4.1 Projeto do Subsistema de Memória

Em um SoC multimídia, a maioria das transferências de dados passam pela memória principal, de forma que um projeto "centralizado em memória" é essencial para alcançar Qualidade de Serviço desejada para cada elemento do sistema. Dessa forma é possível separar e organizar os clientes que precisam de acesso a dados em tempo real para a memória externa dos demais clientes, que possuem limitações de tempo menos rígidas. No projeto de um subsistema de memória, é importante observar as características que aumentem a eficiência das transações e que garantam os prazos de tempo para conclusão das mesmas. Além disso, é preciso permitir a escalabilidade das interfaces para clientes e canais de memória e a adaptatividade para ajustar o subsistema de memória frente às variações de acessos.

Memórias internas são usadas para armazenar temporariamente as informações entre intervalos de tempo das transferências para a DRAM. Dessa forma, o subsistema de memória é organizado como uma hierarquia de memórias, que consegue obter o melhor aproveitamento das rajadas e minimiza os efeitos de latência apresentados anteriormente. Elementos de processamento que possuem elevada localidade temporal da referência devem estar conectados a elementos de memória do tipo *cache*, para recuperar com maior rapidez uma informação que está "espelhada na memória principal", reduzindo acessos externos. Outros elementos de um SoC que possuem uma interface de dados com endereçamento sequencial, como ocorre nos controladores de vídeo e saída de módulos de decodificação e codificação, devem estar conectados à memórias temporárias como filas ou buffers. Tais estruturas são necessárias para conseguir armazenamento temporário que permita oscilações do fluxo de processamento da informação e transferências de dados em blocos maiores.

O projeto do subsistema de memória proposto nesse trabalho é orientado para a integração dos elementos do SoC, fazendo o gerenciamento dos acessos à memória DRAM a partir de um controlador de memória. Além disso, o subsistema de memória deve ser implementado seguindo o fluxo de projeto centralizado em memória, método que é proposto neste trabalho, que define uma interface de comunicação entre os elementos de processamento do SoC e o subsistema de memória que considere duas informações adicionais, além das informações de endereço, comando e dados: 1) prazo de tempo máximo para completar a transação e 2) tamanho mínimo da transação solicitada. Dessa forma, o subsistema de memória pode gerenciar da melhor forma o ordenamento dos acessos e a granularidade das transações. Neste trabalho usaremos os seguintes termos:

- **Transação:** representa uma requisição de acesso de escrita ou leitura originada pelo cliente e compreende as etapas de solicitação de transferência de dados, a espera pelo acesso permitido pelo árbitro do controlador de memória, o processamento do comando pelo controlador de memória externa e a escrita ou leitura dos dados na memória externa. A transação que está em andamento pode ser interrompida por outra transação gerada por um cliente com maior prioridade, dentro de um intervalo mínimo denominado de granu-

laridade.

- **Granularidade:** a granularidade das transações é a fração mínima de uma transferência de dados que não pode ser interrompida antes que o cliente que está com acesso liberado pelo controlador seja suspenso e um outro cliente assuma a transferência. Com a garantia da granularidade, uma transferência de dados ocorre em um número mínimo de rajadas de forma atômica. A granularidade é importante para garantir níveis mínimos de largura de banda da memória externa para um determinado cliente que pode sofrer constantes interrupções quando em um sistema com modo de arbitragem classificado por prioridades.

4.1.1 Arquitetura de Subsistema de Memória

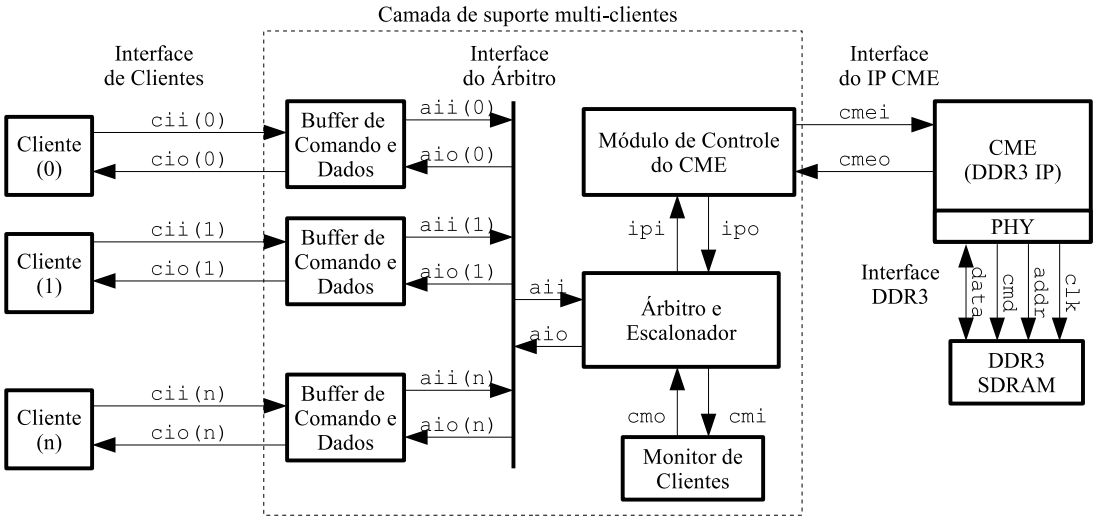
A arquitetura do subsistema de memória é formada por uma hierarquia de memórias, em diferentes níveis, tamanhos e funcionalidades, por elementos de controle e de multiplexação dos dados. O controlador de DRAM, ou controlador de memória externa (CME), implementa a interface de mais baixo nível com a memória externa. Ele é usado para gerar a sequência de comandos adequada para a DRAM a partir dos comandos recebidos da parte que faz interface com os clientes do sistema. Além disso, ele faz a tradução dos endereços gerados a partir de um espaço de endereçamento linear para o espaço de endereçamento formado por bancos, linhas e colunas. O CME contém um módulo duro (*Hard-IP*) de interface com a memória externa, denominado de PHY (*Physical Interface*), que implementa os registradores e *drivers* de conexão física entre o dispositivo de memória externa e a pastilha que contém o SoC.

O CME utilizado é um controlador estático de memória externa. Os controladores de memória estáticos são utilizados em sistemas de processamento crítico com características bem conhecidas e são configurados em tempo de projeto do sistema. Já os controladores de memória dinâmicos são utilizados em sistemas de processamento geral, para sistema de tempo real não crítico e com requisitos de acesso imprevisíveis. Controladores dinâmicos permitem que o subsistema de memória seja adaptado constantemente para atender alterações no comportamento dos clientes. Trabalham com escalonamento em tempo de execução das tarefas, sendo flexíveis às mudanças de comportamento dos acessos dos clientes. Podem ser usados para implementar melhorias no acessos aos dados como: atualizar o escalonamento dos clientes, reordenar as rajadas para reduzir o número de conflitos de bancos na memória, ordenar leituras e escritas para reduzir intervalos entre trocas de operações e permite diversificar o serviço para limites de acessos que não foram previstos inicialmente no projeto. No entanto, controladores dinâmicos podem ter comportamento imprevisível, pois o escalonamento não é conhecido anteriormente e depende das características de acesso dos clientes. Portanto, para calcular a eficiência da memória com maior precisão possível é necessário evitar o reordenamento de comandos, que possibilita prever com exatidão os piores casos de acesso em termos de tempo de resposta. Somente dessa forma é possível atender clientes com requisitos de acesso em tempo-real.

O suporte para múltiplos clientes é inserido na parte frontal do controlador de memória, entre o CME e o sistema. A Figura 4.1 mostra a arquitetura do controlador de memória multi-

cliente, composto a partir de interfaces para acesso individual por cliente, um árbitro para controlar os acessos, o monitor de acessos e o módulo de interface com o CME. Um árbitro implementa o algoritmo de escalonamento dos acessos dos clientes, baseado nas informações de prioridades e características dos acessos requeridos. Um monitor de acessos é implementado na parte de controle do subsistema de memória e gera estatísticas dos acessos dos clientes para o gerenciamento do subsistema de memória. Pedidos de transações gerados nas interfaces de clientes são enfileirados na entrada usando um conjunto de buffers para comandos, endereços, dados e máscara de escrita. A bufferização é usada para realizar transações em sequência entre um cliente e a memória, com tamanho controlado pelo subsistema de memória.

Figura 4.1: Arquitetura do subsistema de memória. O barramento de comandos de entrada e saída da interface com clientes (cii e cio) pode ser escalado para um número configurável de clientes. Da mesma forma, o barramento de interface com árbitro (aii e aio) pode ser escalado.



Fonte: elaborado pelo autor.

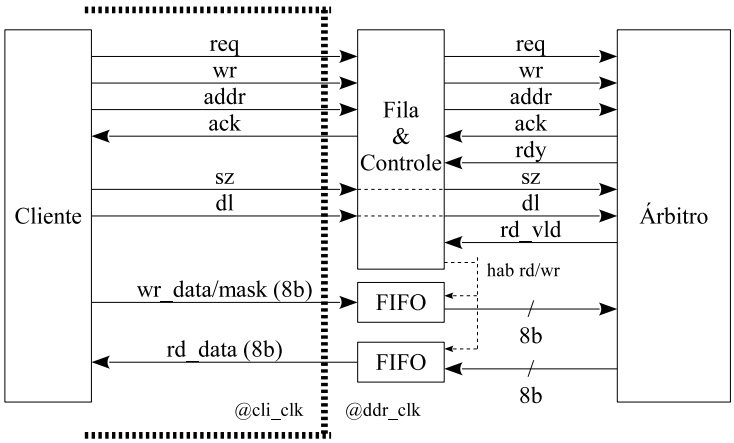
Na Figura 4.1, as interfaces entre módulos são mostradas através de barramentos de sinais de entrada e saída, respectivamente com finais i e o. A interface de entrada e saída de sinais do árbitro é denominada de aii e aio, podendo ser escalada conforme o número de clientes. As interfaces de entrada e saída do módulo de controle do CME é denominada de ipi e ipo. As interfaces de entrada e saída de sinais com o CME são denominadas de cmei e cmeo. Além disso, os sinais de interface da DDR3 estão simplificados, contendo somente relógio, comando, endereço e dados (bidirecional).

A Figura 4.2 mostra os principais sinais e componentes da interface entre um cliente e o árbitro, gerenciada por uma fila de comandos com controle de armazenamento. A fila de comandos é implementada seguindo um protocolo de comunicação assíncrono com *handshake* gerado por sinais de *request* e *acknowledgement*, fazendo uma transferência de comando a cada pedido. A fila recebe os comandos, endereços e dados de escrita gerados por cliente a taxa de um pedido por sinalização req-ack, armazena os comandos em uma fila e espera pela liberação do acesso sinalizada pelo árbitro do controlador. Os sinais ack, req, wr e addr são usados

na sinalização de pedido de escrita ou leitura (por parte do cliente) e entrega ou captura de dados. Os dados e máscara de bytes escritos são colocados no barramento wr_data e wr_mask, para uma escrita de dados na memória, e no barramento rd_data para capturar dados lidos da memória. O sinal rd_vld marca a chegada de dados lidos da memória externa.

O barramento de dados lidos e escritos tem largura em bits igual ao tamanho da rajada (*Burst-Length* – BL) vezes a largura em bits do barramento de dados da memória externa (b). Dessa forma, em cada ciclo de relógio ddr_clk é feita a transferência da quantidade de dados suficiente para completar uma rajada na memória externa. O sinai sz e dl correspondem respectivamente ao tamanho da transação que o cliente está solicitando e o prazo máximo para completar a transação. Tais parâmetros são usados no gerenciamento da fila de comandos e são passados diretamente para o árbitro, que os utiliza para gerenciar os acessos para o CME.

Figura 4.2: Interface entre cliente e árbitro.



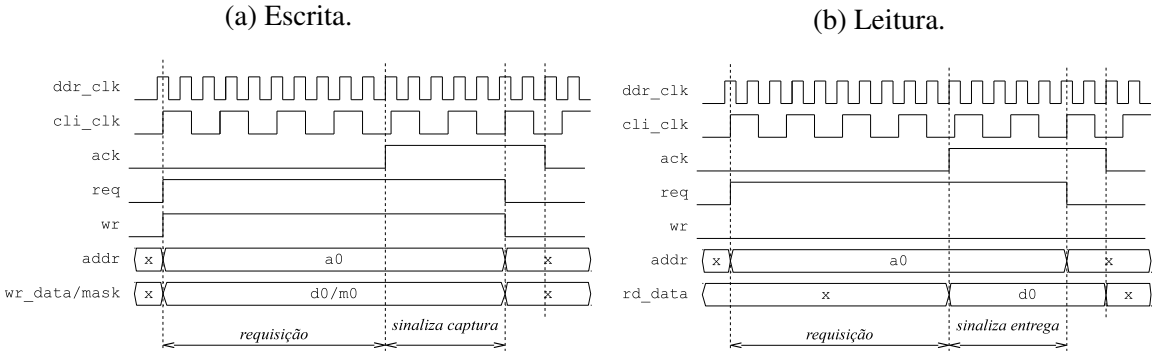
Fonte: elaborado pelo autor.

O parâmetro sz define número de requisições pendentes que um cliente pode esperar até que seja atendido pelo controlador de memória. No caso de arquiteturas de processamento de vídeo, módulos de hardware que fazem interface com a memória externa podem transferir grandes sequências de dados com acessos maiores que uma rajada, por exemplo, no processamento de uma região retangular de pixels com tamanho igual a 256 pixels, são necessários 4 sequências de rajadas de 64 bytes para transferir a informação entre memória e módulo de processamento. O suporte para múltiplas requisições pendentes também é implementado em processadores, como os IBM UltraSparc II que suportam até 3 leituras suspensas e ARM Corte-A57 que suporta até 12 leituras suspensas. Com essa funcionalidade implementada devido ao paralelismo de execução de tarefas nos processadores, uma falta de referência em sua *cachê* de último nível não gera um bloqueio da CPU, que continua com a execução de outra tarefa até que a referência seja buscada na memória externa.

Como um SoC pode conter diferentes regiões com domínios de relógio que operam em velocidades diferentes, o controlador de memória deve prever uma interface assíncrona de comunicação entre os clientes e a memória. A interface entre os clientes é implementada utilizando

um protocolo de sinalização do tipo req-ack (*request and acknowledgement*), a fim de suportar uma interface assíncrona sem riscos de falha causadas por metaestabilidade na transferência de comandos, dados ou endereços (PASRICHA; DUTT, 2008). Os diagramas de tempo de funcionamento para essa arquitetura são mostrados em 4.3a e 4.3b.

Figura 4.3: Diagramas de tempo para escrita e leitura de dados geradas por um cliente através do protocolo req-ack na interface de cliente.



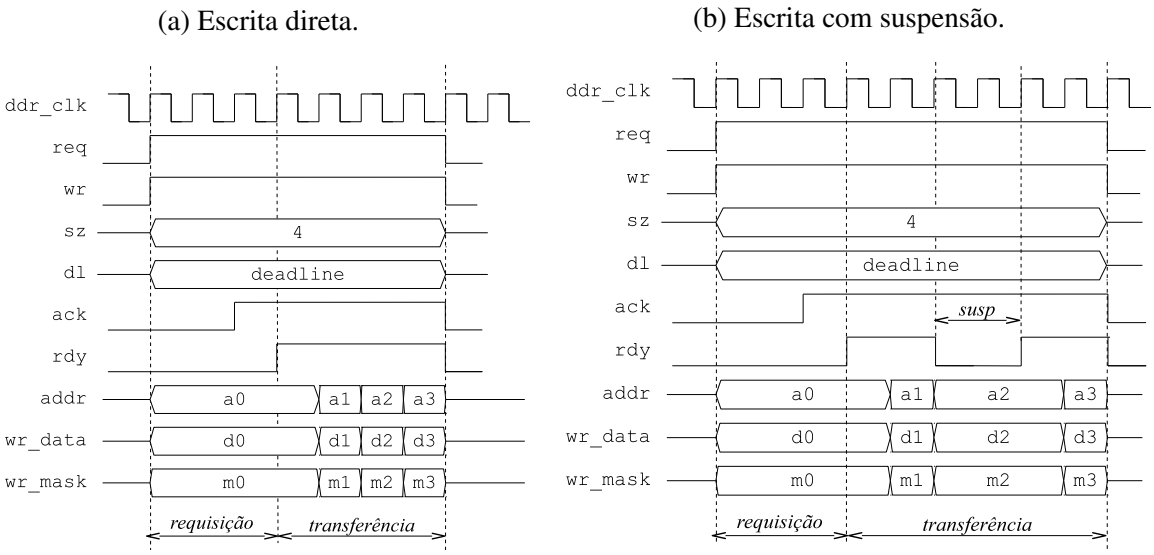
Fonte: elaborado pelo autor.

Após armazenar comandos, endereços e eventuais dados na interface de clientes, o árbitro inicia a transação para a memória externa liberando a comunicação entre a interface de clientes e o módulo de interface com IP. Essa interface opera de modo síncrono usando um protocolo de sinalização req-ack com *pipeline* de transferência de dados, a taxa de um comando por ciclo de relógio (Figura 4.4). Como o árbitro é projetado para suspender um cliente durante um determinado intervalo de tempo, para que outro cliente tenha acesso ou para que seja feita operações de manutenção da memória externa, essa interface é implementada com dois sinais de sinalização, ack e rdy. O sinal rdy é usado pelo árbitro para permitir que a interface de cliente opere na taxa de um comando por ciclo ou permaneça suspensa durante uma interrupção.

A arquitetura proposta é flexível e pode ser estendida para um número maior de clientes, limitados pela largura de banda do subsistema de memória e pela largura de banda solicitada pelos clientes. Um exemplo de implementação do subsistema de memória contendo quatro interfaces para clientes e três níveis de memória é mostrado na Figura 4.5. O nível 0 de memória é formado pelas memórias locais nos elementos de processamento (processos). O nível 1 de memória é formado pelas filas de comandos. O nível 2 de memória é a DRAM, que é acessada através de um CME que recebe os comandos da parte de controle do subsistema de memória.

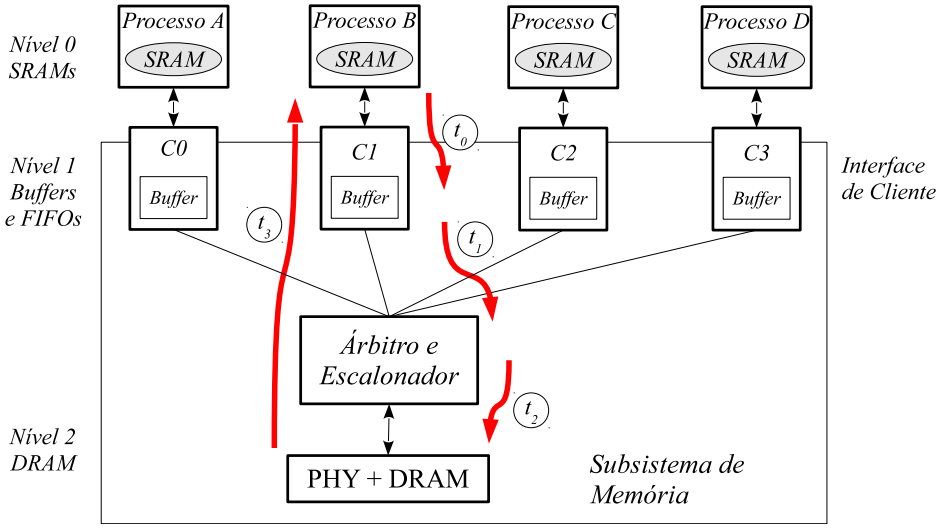
Na Figura 4.5, os intervalos de tempo apresentados estão relacionados às etapas necessárias para completar uma transação de escrita ou leitura, e serão analisados no modelo matemático do subsistema de memória. As requisições de escrita e leitura são enfileiradas durante o intervalo de tempo t0. A passagem do comando para o controlador de memória, após a arbitragem, é representada pelo intervalo t1. A tradução desse comando para a DRAM, realizando a abertura de banco/linha e a escrita dos dados é representada por t2. A transmissão dos dados lidos até o cliente é representada em t3. Portanto, nessa representação uma escrita envolve as etapas de t0

Figura 4.4: Diagramas de tempo para escrita com e sem suspensão da interface do cliente pelo árbitro. A transação suspensa permanece em espera até que o árbitro levante o sinal rdy, sinalizando para o cliente prosseguir com a transferência para a memória.



Fonte: elaborado pelo autor.

Figura 4.5: Exemplo de implementação de um subsistema contendo quatro clientes. A figura mostra os níveis de memória e os intervalos de tempo associados à transferência de dados em uma transação com a DRAM.

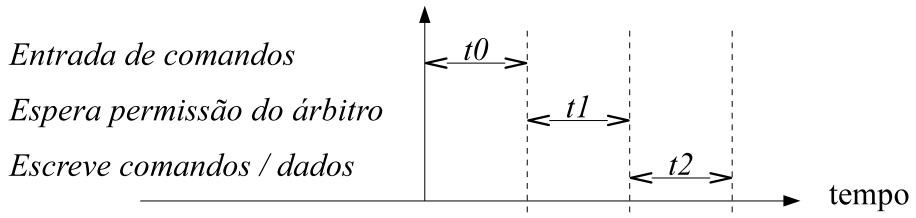


Fonte: elaborado pelo autor.

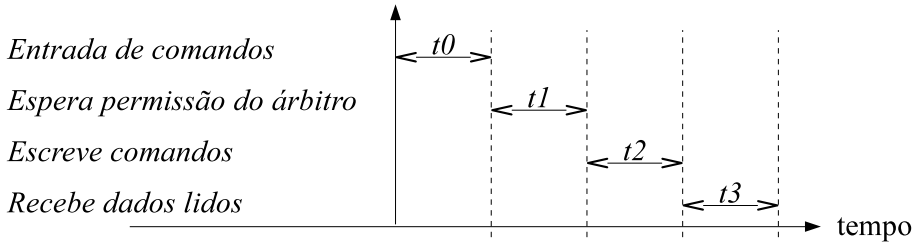
até t2 e uma leitura envolve as etapas de t0 até t3. A Figura 4.6a e 4.6b mostram diagramas de tempo para escrita e leitura de dados entre um cliente e a memória externa. Essa representação é feita de uma forma genérica e permite avaliar em que condições o controlador de memória pode ter o funcionamento predizível no tempo.

Figura 4.6: Intervalos de tempo associados a escrita e leitura de dados.

(a) Escrita de dados.



(b) Leitura de dados.



Fonte: elaborado pelo autor.

Os buffers na interface de clientes permitem que os comandos pendentes em cada entrada possam ser armazenados temporariamente até que haja a permissão de acesso pelo árbitro. A sequência de comandos gerados por um cliente é gerenciada a partir das informações de prazo limite para conclusão (dl) e de tamanho de transferência suportada por esse cliente (sz), seja para escrita ou para leitura de dados na memória. A partir da arquitetura descrita nessa seção será desenvolvido o modelo de funcionamento do subsistema de memória, que estabelece atrasos em ciclos para o funcionamento dos módulos internos do subsistema de memória.

4.2 Modelo do Subsistema de Memória

Esta Seção descreve o modelo de comportamento do subsistema de memória baseado em latência dos acessos à DRAM, denominado aqui por “modelo baseado em atrasos”. O modelo matemático é usado para avaliar o comportamento do sistema de memória. Ele é descrito a partir de equacionamento que considera o funcionamento ao longo do tempo dos elementos existentes dentro do sistema de memória. A partir dessa análise é possível prever, para uma determinada configuração do sistema de memória, os valores mínimos e máximos de latência e largura de banda, em função da granularidade dos acessos.

A granularidade dos acessos impacta diretamente no intervalo de tempo que o árbitro ge-

rencia as interrupções de clientes geradas por pedidos de maior prioridade. Além disso, a granularidade afeta a eficiência da transferência de dados para a DRAM, que é feita em sequências de rajadas de tamanho programado pelo árbitro do controlador. O aproveitamento das rajadas melhora a largura de banda máxima sustentada pela memória. No entanto, granularidades de tamanho elevado reduzem o tempo de troca entre clientes e um cliente com maior prioridade é penalizado com um tempo maior de espera pela permissão do árbitro.

Outro fator que impacta diretamente na latência dos acessos é o tamanho dos buffers internos no controlador de memória. Quanto maior o buffer, maior a oscilação dos tempos de escrita e leitura de dados em um sistema onde a terminação pode gerar interrupções esporádicas, nesse caso a memória externa. A forma de armazenamento e retirada dos dados no buffer interfere na latência, tanto se for FIFO, FILO, como buffer temporário de armazenamento, sempre uma quantidade de latência é inserida. Buffers de tamanho adequado podem proporcionar ao sistema de memória um melhor aproveitamento das rajadas para a memória DRAM.

O modelo de atrasos dos elementos do subsistema de memória é utilizado para determinar os piores e melhores casos de execução desse sistema, para realizar transações de dados com a memória. Esse modelo deve ter precisão suficiente para garantir que os acessos dos clientes sejam atendidos dentro dos prazos solicitados. A abordagem de integração centralizada no subsistema de memória é utilizada para conectar elementos de processamento à memória principal de forma adaptada para atender requisitos da aplicação como largura de banda dos clientes e prazo de conclusão para realizar as transações.

4.2.1 Análise do Pior Caso

Esta Seção apresenta o método proposto para a análise detalhada de pior caso de execução para acessos à memória no sistema compartilhado entre múltiplos clientes. O comportamento do subsistema de memória é avaliado a partir do cálculo do tempo de execução no pior caso WCET (*Worst-Case Execution Time*), que serve como um parâmetro de entrada para a função de arbitragem no subsistema de memória. Os resultados analíticos mostram que um limite superior pode ser estabelecido para WCET para o subsistema de memória, ajustando o tamanho das transações de dados e de armazenamento temporário das memórias internas. O modelo de atrasos proposto é utilizado para estimar o pior caso de tempo de resposta WCRT (*Worst-Case Response Time*), ou seja, o pior caso de realização de uma transação para um cliente conectado ao subsistema de memória. Este modelo leva em conta na sua análise detalhada os parâmetros de tempo que descrevem o comportamento das memórias DDR.

O tempo de resposta para um acesso à memória em um sistema multiprocessado depende da interferência dos acessos gerados pelos demais elementos do sistema. Os conflitos de acesso são resolvidos usando um árbitro de controle de acesso ao canal de memória. A análise do WCET é feita calculando o tempo de resposta para o acesso gerado por um cliente somado aos tempos de resposta dos acessos dos demais clientes, que deve considerar também as penalidades causadas pelo escalonamento dos acessos à memória. Estudos anteriores demonstram que a

análise detalhada do comportamento do subsistema de memória, mesmo com base no pior caso, é eficiente para gerar uma estimativa dos tempos de acesso que ocorrem em um SoC, tal como é apresentado em (SHAH; KNOLL; AKESSON, 2013), (SHAH; RAABE; KNOLL, 2012) e (PAOLIERI et al., 2009). Uma premissa de projeto para determinar corretamente o WCET do sistema é prevenir anomalias temporais de execução dos acessos, ou seja, prevenir de situações onde um pior caso local não contribui para um pior caso global (WILHELM et al., 2009). A análise do pior caso é usada para construção do modelo de comportamento do subsistema de memória e utiliza os parâmetros de temporização da memória. Assim, o modelo de memória pode ser utilizado em diferentes gerações de memória DDR e Wide-I/O. Esse método é implementado pelo árbitro de canal de memória, a fim de ajustar o comportamento do subsistema de memória em tempo de execução do sistema. No modelo proposto, a análise da granularidade dos acessos é usada para manter um valor mínimo de largura de banda para os clientes.

Dados na DRAM são organizados em bancos, linhas e colunas de conjuntos de bits. Quando um banco é ativado, ele carrega uma linha inteira para um buffer de linha interno à DRAM, também chamado de página de linha, que permite o acesso para escrita ou leitura. Controladores de memória podem implementar duas políticas de gerenciamento de páginas de linha: página aberta e página fechada. Na política de página aberta (*open-page policy*), após cada acesso à memória a página atual continua aberta permitindo que um acesso futuro, se coincidir com a mesma página, tenha uma penalidade menor para ser concluído. Essa política permite reduzir a ocorrência de comandos de ativação e pré-carga de linhas durante a execução da plataforma, atrasando especulativamente a realização da pré-carga da linha. No entanto, esse tipo de política quando aplicada aos sistemas multiprocessados não é eficiente devido aos numerosos acessos recebidos pelo controlador gerados por diferentes aplicações rodando em uma plataforma, que geralmente implicam em troca de página (KASERIDIS; STUECHELI; JOHN, 2011), (MOSCIBRODA; MUTLU, 2007). Árbitros podem implementar níveis diferentes de prioridade para acessos à páginas que já estejam abertas, mas isso pode implicar em problemas de trancamento de outros acessos (KASERIDIS; STUECHELI; JOHN, 2011). Já a política de página fechada evita a complexidade de gerenciamento dos buffers de linha realizando um único acesso para cada ativação de linha e fechando a linha após o acesso.

Análise do pior caso proposta nesse trabalho leva em conta o uso da política de página fechada (*closed-page policy*), que torna o sistema mais previsível do que usando página aberta e reduz a complexidade da análise, a interferência gerada pelo árbitro para preempção dos acessos por outro cliente e a interferência da atualização automática dos bancos de memória. O modelo implementado nesse trabalho para política de página fechada é o mesmo usado nos controladores (PAOLIERI et al., 2009), (SHAH; RAABE; KNOLL, 2012) e (AKESSON; GOOSSENS, 2011), com fechamento da página automaticamente após o acesso aos dados.

O equacionamento do modelo leva em conta três fatores principais: o algoritmo de arbitragem utilizado (que interfere no ordenamento dos acessos), a largura dos acessos (granularidade) e o tamanho dos buffers de memória. O algoritmo de arbitragem utilizado deve considerar a

classificação dos clientes para gerenciar da melhor forma o tráfego de dados. Além disso, a arbitragem gerencia a preempção dos clientes para controle da granularidade dos acessos e controle dos acessos com maior prioridade. A preempção é utilizada para interromper um acesso de menor prioridade e garantir que um acesso de maior prioridade utilize o canal durante uma transação. A transação é definida como uma sequência de acessos com tamanho mínimo definido pela granularidade dos acessos. O número mínimo de acessos por transação influencia o desempenho do sistema de três formas:

- Transações de longas sequências de rajadas provocam o aumento da largura de banda sustentada da memória, portanto, o uso de uma granularidade maior aumenta a largura de banda na memória;

- Clientes que fazem longas transações para a memória tendem a dominar o acesso ao canal por longos períodos, dificultando o acesso para clientes com prioridade menor. O árbitro gerencia os acessos e preempta clientes dentro de intervalos de tempo definidos pela granularidade mínima. Dessa forma, a redução da granularidade melhora o desempenho da arbitragem reduzindo o atraso no acesso aos dados;

- A quantidade de dados transferidos por transação que é utilizada pelo cliente deve tender ao tamanho de uma transação, para que os acessos sejam feitos da melhor forma para o sistema de memória.

Em resumo, a eficiência do sistema de memória depende do tamanho das transações e da utilização das mesmas (em termos de quantidade de dados transferidos por transação). A arquitetura do sistema de memória deve ser projetada com vistas a uma melhor utilização das transações para um determinado sistema.

4.2.2 Algoritmos de Arbitragem

O controle do fluxo de comandos que são inseridos no dispositivo DRAM é feito através do enfileiramento desses comandos, em uma fila única ou em múltiplas filas. Cabe ao controlador de memória gerenciar a prioridade dos comandos enfileirados, para organizar a sequência de acessos, que dependem de diferentes fatores como a ordem de prioridade dos comandos ou os recursos de memória disponíveis para efetivar um comando. Árbitros de ordenamento de comandos para controladores de memória são implementados de diferentes formas com foco para diferentes tipos de sistemas de computação. Sistemas com único processador e múltiplas tarefas ou com múltiplos processadores possuem uma única interface de entrada de comandos para o processamento do árbitro. Sistemas com múltiplas entradas, como controladores multi-clientes, possuem diversas entradas de comandos que são processadas pelo árbitro. As filas de entrada de comandos podem ser processadas seguindo uma sequência de classificação por ordenamento de faixa de endereçamento, como no ordenamento por bancos de memória, por classificação de prioridades ou por ordenamento de sequências de escritas e leituras. A inclusão

de uma política de prioridade de acesso leva em conta a origem que gerou o acesso para a memória ou os recursos disponíveis de memória. Outro método de classificar a sequência de processamento de filas de entrada é fazendo a divisão de largura de banda.

O modo mais simples de arbitragem é o "First Come, First Served" (FCFS), no qual os comandos são processados pelo controlador de memória a partir da sua ordem de chegada. Os comandos que estão esperando a mais tempo na fila são priorizados em relação aos comandos mais recentes recebidos pelo controlador, sendo processados na mesma ordem de chegada. É uma política de arbitragem que não implementa preempção prioridades e interrupção de acessos. A equidade de distribuição do uso do canal de memória é justa, pois garante que os clientes consigam realizar os acessos, no entanto o tempo de resposta pode ser elevado, pois clientes que fazem longas sequências de acessos tendem a dominar o uso do canal. O uso dessa política de acesso elimina a possibilidade de que um cliente fique bloqueado. Não existe priorização nesse sistema de escalonamento, de forma que assim que um dos clientes começa a utilizar o canal de memória, ele segue acessando até terminar completamente a transação.

O algoritmo de ordenamento do tipo Round-Robin (RR) faz um rodízio dos acessos gerados por diferentes elementos que compartilham o canal de memória, de forma sequencial, alocando uma parte fixa da largura de banda para cada elemento (SMITH, 1981). Esse esquema faz uma alocação fixa de largura de banda da memória por cliente, que pode ser implementado com duração de tempo determinado por um parâmetro denominado de "quantum" ou com duração determinada pelo cliente, denominado de "quantum infinito". O controlador de memória passa o comando para o novo cliente após um período de tempo igual ao quantum ou após o cliente encerrar o acesso ao canal de memória. O tamanho do quantum utilizado determina a utilização do canal de memória, sendo que um quantum pequeno tende a diminuir a largura de banda sustentada pela memória. Já um quantum elevado tende a aumentar a latência dos demais clientes. O tempo de virada depende do número de clientes. Round-Robin funciona de forma similar ao esquema de arbitragem por fila de ordem de chegada, exceto pela implementação do quantum de fatia de tempo. O uso dessa política de acesso elimina a possibilidade de que um cliente fique bloqueado, assim como ocorre no FCFS.

Outra forma de ordenamento de acessos é gerada usando o algoritmo de divisão por fatias de tempo, denominado de "Time Division Multiplex" (TDM). O TDM é implementado a partir da alocação de unidades de fatia de tempo fixas para cada cliente, liberando o acesso na forma de rodízio circular como no Round-Robin. Para garantir a redução da latência em clientes com acesso crítico pode-se utilizar um esquema de priorização dos acessos prioridades, implementado nos árbitros TDM. O acesso ao canal de memória durante a fatia de tempo determinada para um cliente pode ser interrompido para que um cliente crítico utilize o canal. No entanto, o uso de prioridades de acesso pode causar penalizações excessivas nos clientes não prioritários e também o bloqueio dos mesmos. A priorização pode ser implementada também em árbitros TDM, bem como nos árbitros Round-Robin. Os árbitros denominados de Prioridade Fixa com Preempção ("Fixed Priority with Preemption" – FPP) implementam a seleção de clientes no

modo RR ou TDM com preempção para um ou mais clientes com prioridade de acesso.

A partir desses modos arbitragem são desenvolvidos árbitros de controle de acesso à memória com diferentes aplicações alvo. Exemplos de árbitro TDM são apresentados por (GOMONY; AKESSON; GOOSSENS, 2013), (AKESSON; STEFFENS; GOOSSENS, 2009) denominado de árbitro por controle de créditos, baseado na teoria de servidores de latência para controle justo de multiplexação de tráfego de dados (STILIADIS; VARMA, 1998). O árbitro por controle de créditos é um árbitro TDM com fatia de tempo ajustável pelo tráfego, alocada de acordo com o nível de acessos gerados pelos clientes que são atendidos e dos clientes que estão em espera. Outro tipo de TDM que é utilizado em sistemas com múltiplos processadores é apresentado em (NESBIT et al., 2006), que aloca uma fatia de tempo proporcional ao número de processadores e conta o tempo de acordo com o parâmetro *tRL* (denominado pelos autores de "RAS latency"). Ou seja, para cada cliente é alocado um valor de fatia de tempo proporcional ao tempo de acesso aos dados na linha da DRAM. A partir disso, o controlador distribui igualmente a banda de memória entre as filas de bancos prevenindo problemas de "starvation". Na implementação apresentada em (GOMONY; AKESSON; GOOSSENS, 2013), escolheu-se uma unidade de serviço de 64 bytes (de tamanho igual a uma rajada) com um fatia de tempo igual a 13 ciclos de relógio, que é igual ao tempo necessário para realizar uma escrita ou leitura. O tamanho do frame é igual a 8 fatias de tempo. Esses intervalos de funcionamento são escolhidos em tempo de projeto do sistema e permanecem estáticos durante a sua execução. Além disso, os parâmetros utilizados para configurar o sistema são baseados em uma tabela de especificação de categorias de latência para os clientes.

Classificação por recursos disponíveis na memória faz o ordenamento dos comandos através dos bancos que estão disponíveis para acesso na memória DRAM. Os algoritmos de escalonamento de memória melhoram o desempenho do sistema reordenando os pedidos de comandos que chegam no controlador para priorizar os acessos que são direcionados para bancos abertos, resultando em um acerto de página de memória. Caso contrário, priorizam os acessos que estão esperando por mais tempo para serem processados, armazenados em uma fila de ordem de chegada. Um exemplo disso é o árbitro FR-FCFS (RIXNER et al., 2000). O algoritmo FR-FCFS melhora a vazão de dados na memória DRAM mas pode causar problemas de distribuição de banda entre processos que acessam o canal, pois reduzem a prioridade de processos com menor localidade no buffer de página de memória e com menor intensidade de acessos, como mostrado em (MOSCIBRODA; MUTLU, 2007). A priorização dos acessos direcionados à bancos abertos e reordenamento dos comandos reduz a previsibilidade de funcionamento do controlador de memória, que passa a ser determinado pelo conteúdo.

Outra forma de ordenar os comandos é implementar uma fila para cada um dos bancos de memória, assumindo que cada comando tem o mesmo nível de prioridade. Os comandos direcionados para o mesmo banco de memória são enfileirados e um árbitro do tipo Round-Robin escalona os acessos fazendo um rodízio de bancos. Essa técnica pode ser utilizada em controladores com uma entrada de comandos, classificando os pedidos através do endereçamento. O

uso de uma política de ordenamento de acessos do tipo Round-Robin garante que o atraso máximo de uma transação que sofre interferência da concorrência de outros clientes não dependa do conteúdo acessado mas somente da quantidade de clientes. Os chamados "application-aware memory controllers" são propostos para prover o melhor balanço entre desempenho de acesso à memória e equilíbrio da distribuição de banda. O algoritmo denominado de "Parallelism-aware Batch Scheduling" (PAR-BS) (MUTLU; MOSCIBRODA, 2008) organiza lotes de comandos na entrada do controlador baseados nos tempos de chegada e prioriza os comandos mais antigos entre os demais com equidade de acessos. No lote de comandos, aplicações são priorizadas para manter o paralelismo de bancos, ordenando acessos em lotes para bancos que estão disponíveis.

O algoritmo ATLAS ("Adaptive per-Thread Least-Attained-Service memory scheduling" (KIM et al., 2010) é usado para priorizar pedidos de comandos em sistemas multiprocessados gerados por aplicações que receberam o menor serviço de memória. Como resultado, aplicações com baixa intensidade de geração de acessos à memória são priorizadas melhorando o desempenho do sistema. No entanto, aplicações com elevada intensidade de acesso à memória são menos priorizadas e a quantidade de acessos é reduzida, diminuindo a equidade. O algoritmo "Thread Cluster Memory scheduling" (TCM) (KIM et al., 2010) melhora a equidade de distribuição da banda de memória ordenando dinamicamente os acessos gerados para a memória nos grupos de baixa e alta taxa de acessos para a memória, classificados acessos pela frequência de geração de acessos. Para melhorar o desempenho do sistema, o algoritmo TCM prioriza os acessos do grupo de menor frequência.

A classificação dos clientes por prioridades é utilizada para priorizar clientes que tem restrições limitadas e severas para tempos de acesso à memória. O árbitro gerencia acessos enfileirados conforme classificação de prioridade do cliente. Esse tipo de árbitro pode ser implementado a partir de múltiplas filas de entrada, uma para cada cliente. O escalonamento é feito priorizando filas de clientes com maior prioridade. A preempção também pode ser utilizada para que o cliente com maior prioridade coloque o de menor prioridade em estado de suspensão enquanto realiza o acesso. Ao completar o acesso, o cliente de menor prioridade retoma a transação. No sistema de arbitragem classificado por prioridades, pois o comportamento de acesso dos clientes que geram interrupções influencia significativamente no funcionamento do sistema de memória que passa a ser não-preditivo. A classificação de clientes baseada em prioridades estáticas necessita de um conhecimento prévio do comportamento da aplicação e garantir que o seu comportamento seja mantido ao longo do tempo de execução da plataforma. Trabalhos anteriores descrevem a classificação estática de clientes para plataformas multimídias, como proposto em (LEE; CHANG, 2006) e (WOLF; GEUZEBROEK, 2011). Os requisitos de QoS da aplicação são estabelecidos para caracterizar as diferentes características de acesso para cada cliente conectado ao subsistema de memória. Em geral, os clientes são classificados em três categorias diferentes de acordo com os requisitos de funcionamento e os prazos de execução: tempo-real, sensível à latência e sensível a largura de banda.

O modelo de arbitragem proposto nesse trabalho é baseado[1] em uma classificação dos clientes por rodízio, como ocorre no algoritmo RR, com quantum definido pela granularidade mínima definida para as transações de cada cliente. No árbitro adaptativo é implementado um esquema de preempção de transações baseado em prioridades dinâmicas, calculadas a partir dos requisitos de prazo de conclusão das transações informados por cada cliente. A prioridade dos clientes é levada em conta durante a etapa de seleção de clientes, no rodízio implementado na forma Round-Robin, com preempção ocorrendo somente durante a conclusão da granularidade mínima de uma transferência. Esse algoritmo de arbitragem garante uma implementação com funcionamento previsível durante o tempo e ao longo da execução das aplicações. Algoritmos de arbitragem ATLAS, TCM e PAR-BS, citados anteriormente, são implementados com base em algoritmos de arbitragem de filas ou divisão por fatia de tempo, mas com difícil modelagem, pois a organização dos acessos ordenada através do sequenciamento de endereços acessados insere problemas de previsibilidade do funcionamento do controlador.

Diferentemente dos árbitros por classificação de níveis de prioridade, como os citados anteriormente, o árbitro adaptativo proposto organiza os acessos gerados pelos clientes sempre com a mesma prioridade durante os prazos estabelecidos pelos clientes. No momento que um prazo está próximo de ser violado, o árbitro adaptativo prioriza esse cliente gerando uma interrupção. Esse método garante uma distribuição justa de uso do canal de memória entre os clientes, garantido os prazos de conclusão das transações dentro dos limites possíveis de funcionamento da memória externa.

4.2.3 Modelo Baseado em Atrasos

Esse trabalho propõe o uso de um modelo baseado em atrasos, de construção simples, mas que garanta prever os piores casos de comportamento para os clientes conectados ao controlador de memória. A partir desse modelo, o controlador pode calcular dinamicamente os piores casos e gerenciar os acessos à memória externa. O gerenciamento ocorre de forma adaptativa, calculando novamente os parâmetros do sistema a cada modificação das suas variáveis de entrada. O modelo é construído a partir da estimativa de atrasos em ciclos de relógio em cada parte do subsistema de memória por onde passam comandos e dados entre um cliente e a memória compartilhada. A Tabela 4.1 contém uma descrição dos parâmetros que serão utilizados ao longo desta subsecção.

A Figura 4.7 mostra o diagrama simplificado da arquitetura que faz interface entre os clientes e o IP controlador de memória e externa com os caminhos de comunicação de comandos marcados como A, B, C, D com comportamento mostrado na Figura 4.8. Nesse exemplo, dois clientes fazem acessos concorrentes à memória externa. O cliente C0 inicia o acesso ganhando a permissão do árbitro que repassa a sequência de comandos armazenados no buffer temporário para o IP de controle da memória externa. O IP traduz a sequência de comandos para a memória DDR ativando, se necessário, páginas para o acesso e gerando a sequência de rajadas de escrita

[1] O árbitro adaptativo será descrito na seção 4.3.1.

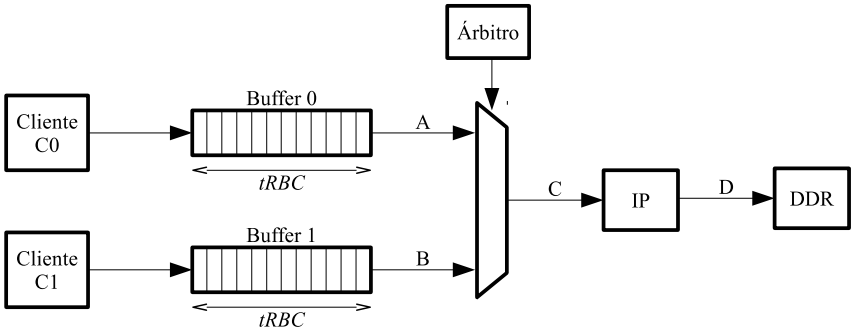
Tabela 4.1: Parâmetros do Modelo do Subsistema de Memória.

Parâmetro	***Descrição***	***Unidade***
g	Granularidade das Transações	ciclos
$l(n)$	Duração da Transação para um cliente n	ciclos
$dl(n)$	Prazo para completar a transação do cliente n	ciclos
$tR(l)$	Tempo de resposta para um cliente n	ciclos
$tRBC(l)$	Tempo de retenção no buffer de comandos para o cliente n	ciclos
$tLD(l)$	Tempo para efetivar os comandos de leitura de dados para uma transação de tamanho l mais a latência de leitura	ciclos
$tED(l)$	Tempo para efetivar os comandos de escrita de dados para uma transação de tamanho l	ciclos
tRL	Latência de dados lidos da memória	ciclos
$tCCD$	Tempo entre comandos de acesso à coluna	ciclos

Fonte: elaborado pelo autor.

ou leitura. Enquanto o cliente C0 acessa o canal de memória, o cliente C1 permanece em estado de espera e a sequência de comandos armazenada no buffer. O cliente C1 espera pela passagem dos comandos para o IP e pelo processamento dos comandos do cliente C0, até completar a transação. Após isso, o cliente C1 consegue iniciar a transação para a memória externa, pelos caminhos B-C-D.

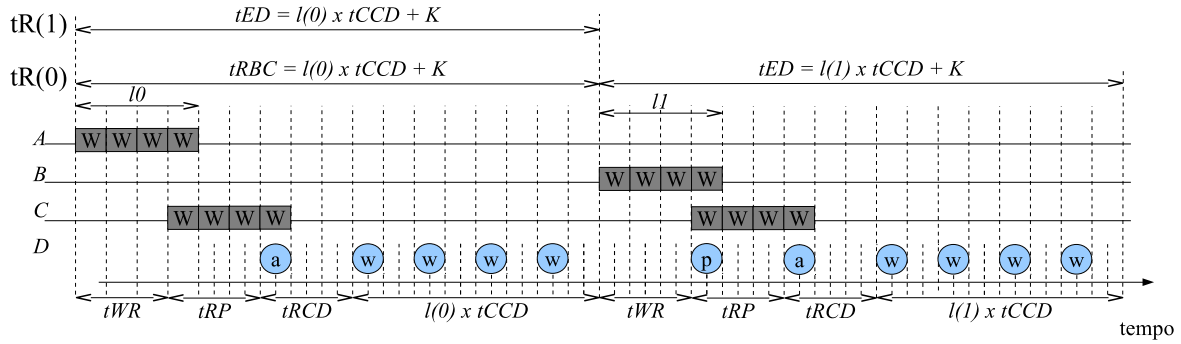
Figura 4.7: Diagrama de blocos simplificado com os caminhos de comunicação entre dois clientes e a memória externa. Cada cliente armazena os comandos de escrita ou leitura em uma fila de entrada até receber a permissão do árbitro.



Fonte: elaborado pelo autor.

Cada cliente contém uma fila de entrada de comandos, que são armazenados para esperar a liberação do árbitro para o acesso ao canal de memória. Quando o cliente recebe a permissão do árbitro, ele passa os comandos para o IP de memória na taxa de um comando por ciclo de relógio. Dessa forma, o pior caso de tempo de retenção no buffer de comandos (tRBC) é medido pelo número de clientes e pelo tamanho das transações de cada cliente. Para simplificar essa análise, uniformizaremos o tamanho das transações de cada cliente para 4 rajadas consecutivas, de forma que todos os clientes trabalhem com o mesmo tamanho de transação $l(n)$. O tempo de retenção no pior dos casos é dado por (4.1), onde a constante K representa a penalidade para a mudança de pagina na DRAM, assumindo que no pior dos casos a transação será feita

Figura 4.8: Diagrama de tempo da comunicação entre dois clientes gerenciada pelo árbitro com esquema do tipo FCFS. Cada cliente faz transferência de tamanho igual a 4 rajadas consecutivas para uma DDR3-SDRAM. O cliente C0 inicia o acesso enquanto que o cliente C1 espera pela permissão. Essa figura ilustra os tempos de acesso em um pior caso para o cliente C1.



Fonte: elaborado pelo autor.

em uma página fechada. Para uma sequência de escritas em uma página, a constante é igual a $K = tWR + tRP + tRCD$.

$$tRBC(l) = (l(0) \cdot tCCD + K) \tag{4.1}$$

Esse modelo presume que o atraso de processamento do árbitro seja menor do que a constante K. Quando o IP de interface com a memória externa começa a receber os comandos de um cliente, ele precisa verificar se a página solicitada está aberta para então iniciar a escrita ou se é preciso fechar a página atual e abrir uma nova. Essa penalidade já está considerada em (4.1). O tempo necessário para que o IP processe os comandos de escrita ou leitura depende do intervalo entre a colocação de comandos sucessivos na memória DDR, dado pelo parâmetro tCCD. Além disso, no caso de uma leitura de dados, os dados somente serão colocados no barramento após a latência de leitura, dada pelo parâmetro tRL. O tempo de escrita de dados é dado em (4.2) e o tempo de leitura de dados é dado em (4.3).

$$tED(l) = tCCD \cdot l(n) + K \tag{4.2}$$

$$tLD(l) = tCCD \cdot l(n) + tRL + K \tag{4.3}$$

O pior caso de tempo de resposta para escrita de um cliente n com transação de tamanho $l(n)$ é dado pela soma do tempo de espera no buffer mais o tempo gasto pelo IP de memória para efetivar a escrita dos comandos, resultando em $tR_{En}(l) = tRBC + tED$. Já o tempo de resposta no pior dos casos para leitura de um cliente n com transação de tamanho $l(n)$ é dado em $tR_{Ln}(l) = tRBC + tLD$. Além do tempo de resposta, é preciso considerar os efeitos gerados pela atualização automática da memória, que é realizada a cada tREFI nanosegundos e tem uma duração de tempo igual a tRFC. O WCRT para escritas e leituras deve considerar que uma auto-atualização ocorra ao mesmo tempo em que uma transação, de forma que o cliente permanecerá

em estado de suspensão. Como a auto-atualização necessita de que a página acessada seja fechada, os dados sejam gravados no núcleo de DRAM e a página seja aberta novamente para prosseguir com o acesso, o tempo de auto-atualização é igual a $tAA = tRFC + (tWR + tRP + tRCD)$. Dessa forma, o WCRT para escrita e leitura é dado respectivamente em (4.4) e (4.5).

$$wcrt_E(l) = tRBC(l) + tED(l) + tAA \tag{4.4}$$

$$wcrt_L(l) = tRBC(l) + tLD(l) + tAA \tag{4.5}$$

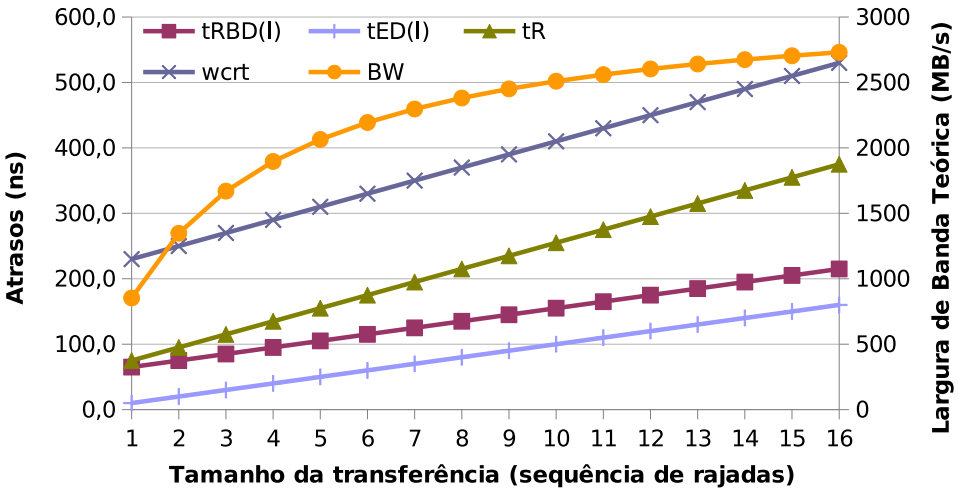
A Figura 4.9 mostra a análise de comportamento das curvas de atraso em função do tamanho das transferências geradas pelos clientes de uma plataforma com 2 clientes. O WCRT é uma função linear com derivada positiva em função do tamanho das transferências. A diferença entre os tempos tED e tLD é a latência de leitura dada por tRL que representa o tempo entre a escrita de comandos de leitura e a saída dos dados no barramento, portanto não interfere no uso do barramento de comandos. Dessa forma, o cálculo do WCRT pode ser feito para a análise do caso de escrita de dados, que necessita de um tempo maior do que a leitura devido ao *write recovery time* (tWR). A largura de banda também apresenta uma derivada positiva, mas não linear. Dessa forma, é possível perceber que o aumento da granularidade das transferências para os clientes traz benefícios em relação à largura de banda, mas o tempo de resposta é maior. Na Figura 4.9, o pior caso do tempo de resposta para os clientes é mostrado na curva *wcrt*, que para uma transferência de $l = 1$ rajada é de 230 ns e para uma sequência de $l = 16$ rajadas consecutivas é de 530 ns. Enquanto isso, a largura de banda mínima sustentada (BW) para cada cliente passa de 850 MB/s para 2700 MB/s com transações feitas em $l = 16$ sequências de rajadas. Essa análise é feita para uma memória DDR3-800 de 64 bits de barramento com largura de banda de pico igual a 6400 MB/s.

4.2.4 Análise em Função da Granularidade

A inserção de um valor mínimo de granularidade dos acessos para interrupção de clientes faz um equilíbrio entre a redução do tempo de resposta para o cliente que gera a interrupção e a melhora da largura de banda para o cliente preemptado. A granularidade, definida por g, representa a menor sequência de rajadas que um cliente pode fazer antes de ser interrompido pelo árbitro. A equação que mostra o tempo de retenção no buffer de comandos deve ser modificada inserindo um fator que representa o tempo adicional de espera quando o cliente é preemptado. Manter uma granularidade mínima de interrupção de um cliente gera um aumento significativo da largura de banda para o cliente que está sendo suspenso ao custo de um pequeno aumento de latência para o cliente que está solicitando acesso.

Nesse modelo não são consideradas prioridades diferentes entre os clientes, mas a preempção é considerada seguindo um critério de prioridades baseado em uma classificação adaptativa dos clientes. O árbitro gerencia dinamicamente qual cliente pode acessar o canal e gerar a preempção dos demais. No modelo será considerado o pior caso de tempo de acesso para o cliente

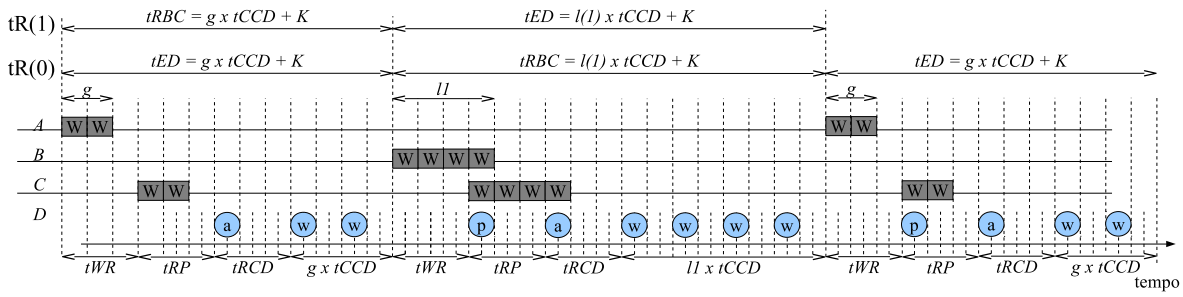
Figura 4.9: Comparação entre as curvas de atraso nos elementos do controlador de memória e o tamanho da transferência feita pelos clientes no cliente que espera pelo acesso ($n = 2$, $l(n) \in [1, 16]$).



Fonte: elaborado pelo autor.

que interrompe os demais, que tem acesso prioritário, e o pior caso para o cliente que está sendo interrompido. Um exemplo é mostrado na Figura 4.10

Figura 4.10: Diagrama de tempo da comunicação entre dois clientes e o árbitro. O cliente C0 inicia a transação com a memória externa quando é interrompido após passar g comandos para o IP.



Fonte: elaborado pelo autor.

Para o cliente C1, o pior caso do tempo de resposta para completar a transação é proporcional à granularidade mínima de interrupção do cliente que está acessando o canal de memória mais a penalidade para troca de página. Para o cliente C0, o tempo de resposta para completar a transação deve contar a preempção com as penalidades de acesso à página na memória mais a transação completa do outro cliente. A Figura 4.10 mostra um exemplo para dois clientes. A relação entre o tamanho das transações $l(n)$ e a granularidade com que ela é executada adiciona um fator igual a $\lceil l/g \rceil - 1$ para o tempo de retenção no buffer de comandos. Quando l/g resulta em 1 significa que a granularidade mínima é igual ao tamanho da transação e portanto não é

possível preemptar um cliente.

O tempo de retenção no buffer de dados (*tRBD*) é igual ao tempo gasto para processar um conjunto de dados transferidos dentro da menor granularidade possível, mostrado em (4.6). Na análise do pior caso é estimado que para cada grão de transferência é feita uma troca de linha na memória, pois considera uma troca de cliente acessando uma região diferente (outra página no mesmo banco ou uma página em um banco que está fechado). Portanto, essa função depende da relação entre o tamanho da transferência de um cliente e a granularidade da transferência definida pelo sistema de memória. Essa função considera o tempo para uma troca de linha da DRAM utilizada, definido pela constante de penalidade de troca de linha (K).

$$tRBD(l) = \left(\lceil l/g \rceil - 1\right) \cdot (l(1) \cdot tCCD + K) \tag{4.6}$$

O tempo de processamento do controlador de memória externa e na DRAM está representado em (4.7). Esse intervalo de tempo depende do número de repetições da transação para a DRAM e da latência de leitura. Esse cálculo é direcionado para a leitura de dados, que representa o pior caso de uma transação para a memória. A granularidade dos acessos definida para o sistema de memória influencia diretamente no intervalo de tempo $tED(l)$.

$$tED(l) = tCCD \cdot l(0) + K \tag{4.7}$$

O tempo de resposta ($tR(n)$) para um determinado cliente n é calculado pela soma das equações (4.6) e (4.7), resultando em $tR(l) = tRBC(l) + tLD(l)$, que expandida em cada um dos seus termos resulta em (4.8). Para o cliente que está gerando a preempção, o tempo de resposta é dado por $tR(l) = tED(l) + tRBD(l(1))$ em (4.9).

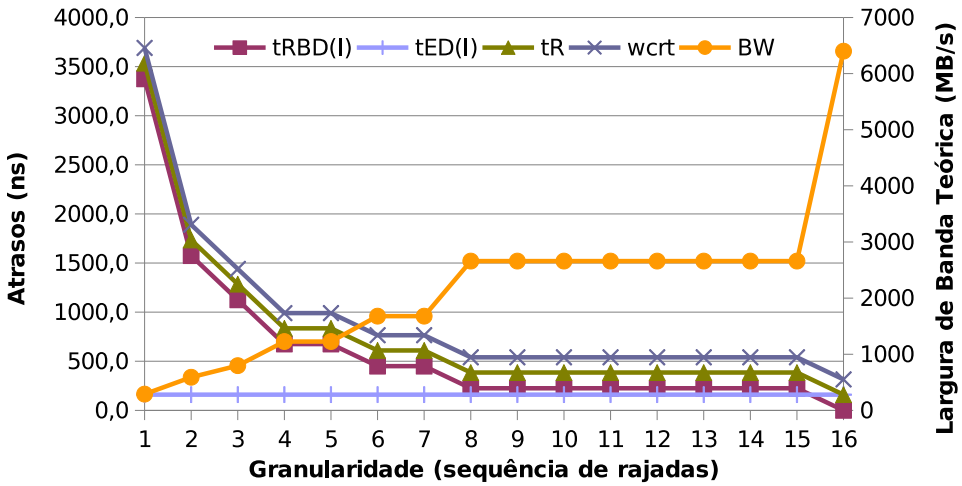
$$tR(0) = (g \cdot tCCD + K) + (l(1) \cdot tCCD + K) + (g \cdot tCCD + K) \tag{4.8}$$

$$tR(1) = (g \cdot tCCD + K) + (l(1) \cdot tCCD + K) \tag{4.9}$$

A equação utilizada para calcular o tempo de resposta para cada acesso de um cliente n depende de parâmetros referentes ao funcionamento da DRAM (como *tRC*, *tCCD* e *tRL*), usados aqui em ciclos de relógio, e de parâmetros de ajuste do subsistema de memória, como a granularidade das transferências (g) e o tamanho da transferência alocada para cada cliente ($l(n)$). A Figura 4.11 mostra o comportamento das curvas de atraso em função do tamanho das transferências geradas pelos clientes de uma plataforma com 2 clientes para o cliente que é preemptado, descrito em (4.8). A derivada do tempo de retenção no buffer em relação à granularidade é negativa, então o aumento da granularidade diminui o tempo de retenção. Essa relação tem influencia significativa no tempo de resposta para o cliente e na largura de banda estimada.

Para o intervalo de granularidades de interrupção de um cliente compreendido entre [8,15],

Figura 4.11: Árbitro com controle de interrupção por granularidade. Comparação entre as curvas de atraso nos elementos do controlador de memória e o tamanho da transferência feita pelos clientes ($n = 2$, $l(0) = l(1) = 16$, $g \in [1, 16]$).



Fonte: elaborado pelo autor.

o tempo de resposta (tR) não varia pois o número de interrupções que o cliente sofre para completar a transação de 16 sequências de rajadas é o mesmo e igual a 1 interrupção. Isso ocorre de forma semelhante para o intervalo [6,7], que pode acarretar em duas interrupções, e para o intervalo [4,5], que pode acarretar em três interrupções do cliente. Segundo a análise mostrada na Figura 4.11, a largura de banda mínima sustentada pela memória para cada cliente (BW), dependendo do número de interrupções, pode variar entre 290 MB/s e 2660 MB/s para uma granularidade igual entre 8 e 15 rajadas consecutivas[2]. Um algoritmo de arbitragem com preempção que não faça controle de granularidade mínima pode operar com $g = 1$, pois interrompe imediatamente o cliente que está acessando o canal. Esse efeito pode causar a redução significativa da largura de banda do sistema devido às trocas frequentes.

4.2.5 Granularidade das Transações

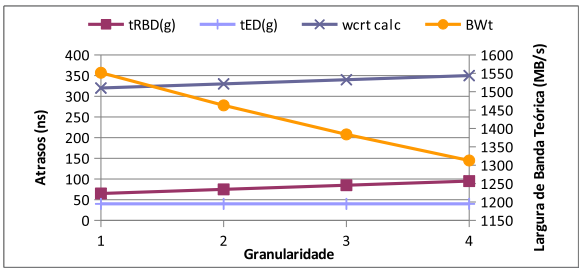
A comparação da análise do tempo de resposta para dois clientes fazendo transferências de tamanho igual a quatro rajadas consecutivas para a memória é mostrada na Figura 4.12. Nessa análise percebe-se que a granularidade mínima de transação interfere significativamente no tempo de resposta para o cliente que está sendo preemptado. As Figuras 4.12a e 4.12b mostram o tempo de resposta calculado a partir das equações (4.8) e (4.9) para o cliente que está preemptando e para o cliente que está sendo preemptado.

O último ponto de granularidade para o cliente que está sendo preemptado não deve ser considerado, pois quando $g = l$ significa que não há preempção. Fazendo essa mesma análise

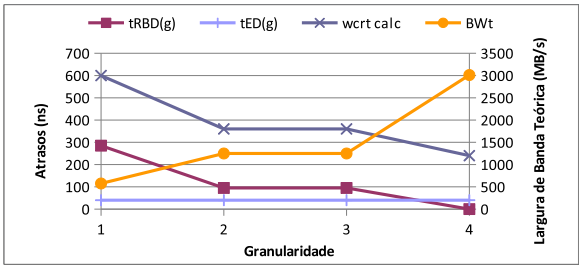
[2]Essa análise gera uma exceção para $g = 16$ que significa que o cliente não é interrompido nunca e ocupa completamente o canal de memória, tendo uma largura de banda alocada de 6400 MB/s. Esse resultado é válido somente na teoria, pois na prática é impossível atingir a largura de banda de pico da memória.

Figura 4.12: Análise dos tempos de resposta estimados para $l(0) = l(1) = 4$ e $g \in [1,4]$.

(a) Cliente que preempta.



(b) Cliente preemptado.

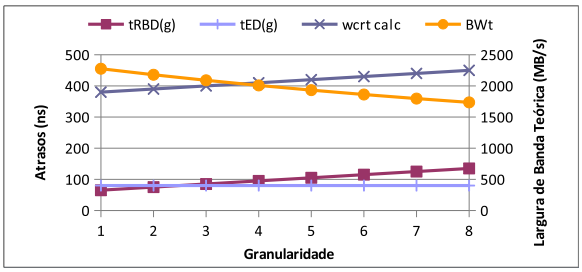


Fonte: elaborado pelo autor.

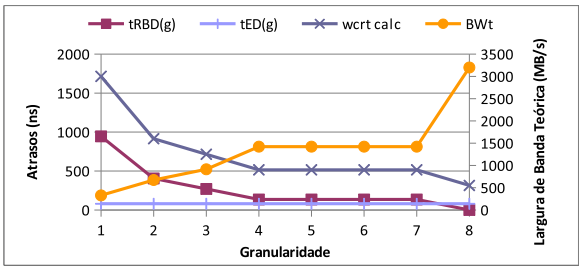
para clientes com transações de tamanho igual a oito, resultam as Figuras 4.13a e 4.13b. Para o cliente que está sendo preemptado, a granularidade entre valores 4 e 7 não resulta em aumento do tempo de resposta da transação, pois significa somente uma preempção por transação. A granularidade igual a 8 significa que nenhuma preempção ocorre.

Figura 4.13: Análise dos tempos de resposta estimados para $l(0) = l(1) = 8$ e $g \in [1,8]$.

(a) Cliente que preempta.



(b) Cliente preemptado



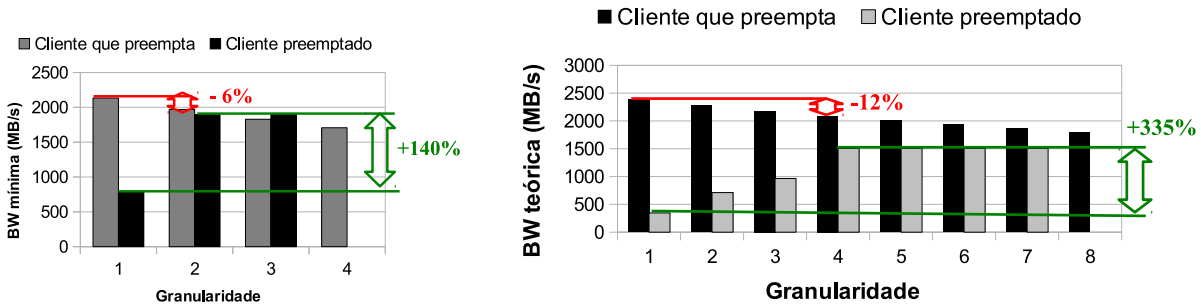
Fonte: elaborado pelo autor.

Na análise apresentada nessa subseção, a não linearidade que existe nas equações de cálculo do tempo de resposta para o cliente que está sendo preemptado ocorre pelo número de interrupções possíveis no intervalo de tempo em que transação solicitada ocorre. O cliente de maior prioridade sofre a penalidade de ter que esperar alguns ciclos de relógio para interromper o cliente que está acessando o canal e pelos tempos de ativação de linha na memória. O cliente que é interrompido sofre a penalidade de perder a prioridade de acesso ao canal de memória e ter que esperar por uma ou mais transações completas de clientes mais prioritários. Na situação limite, o cliente pode ser interrompido a cada comando de uma transação gerada para a memória, que degrada significativamente a largura de banda do cliente. A Figura 4.14 mostra a modificação da largura de banda prevista em função dos piores casos de tempo de resposta para um sistema com dois clientes acessando a memória em transações de tamanho diferentes com granularidades variando entre 1 e o máximo. Essa comparação mostra que a variação da granularidade influencia significativamente na largura de banda do cliente que está sendo preemptado.

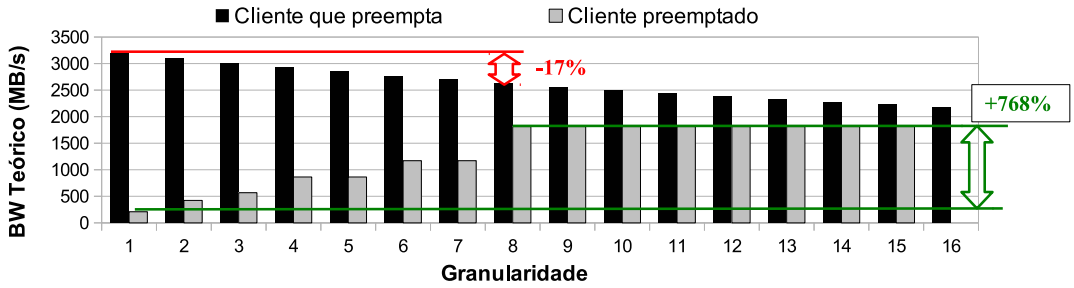
Figura 4.14: Análise da variação da largura de banda prevista para um sistema com dois clientes acessando o canal de memória em diferentes tamanhos de rajada e granularidades diferentes.

(a) Transações de tamanho igual a 4 rajadas.

(b) Transações de tamanho igual a 8 rajadas.



(c) Transações de tamanho igual a 16 rajadas.



Fonte: elaborado pelo autor.

Para implementar um controlador de memória com requisitos de garantia de largura de banda mínima para todos os clientes, pode-se determinar um valor mínimo de granularidade da transação antes de realizar uma interrupção. Como mostra a Figura 4.14a, a preempção de um cliente com granularidade g=2 pode aumentar a largura de banda do cliente mínima garantida do cliente preemptado em 140%, passando de 790 MB/s para 1890 MB/s. Já o cliente que está preemptando, por esperar um ciclo a mais de transferência do cliente preemptado, tem uma redução da largura de banda igual a 7,7%, passando de 2130 MB/s para 1970 MB/s. Esse aumento no tempo de resposta do cliente que está esperando é igual a 10 ns

Caso a granularidade de interrupção seja aumentada em uma sequência de rajadas, passando para $g = 3$, ocorre uma redução da largura de banda do cliente que está preemptando, pois o tempo de espera aumenta em 10 ns. Já para o cliente preemptado não ocorre nenhuma alteração na largura de banda mínima, pois somente ocorre uma interrupção da transação no pior caso como para $g = 2$. Essa análise quando estendida para transações de tamanho igual a 8 e 16 rajadas resulta nas Figuras 4.14b e 4.14c, que são obtidas a partir das equações (4.8) e (4.9), que apresentam uma diferença maior da variação da granularidade mínima.

Esta análise mostra que a escolha da granularidade de interrupção de uma transação que está em andamento é determinante para garantir uma largura de banda mínima necessária para clientes. A granularidade é um parâmetro de projeto que deve ser determinado previamente para o controlador de memória. O uso de granularidades maiores que 1 rajada, ou seja, garantir que

uma transação somente seja interrompida após a segunda rajada ou além, traz benefícios para o funcionamento do sistema pois eleva a largura de banda mínima dos clientes. Em um sistema de acesso à memória com escalonamento de acessos ordenados em prioridades dinâmicas, a garantia de largura de banda mínima através da granularidade é fundamental para manter o desempenho do sistema. A inclusão de um valor de granularidade de interrupção não inviabiliza a previsibilidade de funcionamento do sistema a partir de um modelo de funcionamento baseado em atrasos, como é proposto nessa tese. As equações (4.8) e (4.9), que formam o modelo de pior caso para os clientes, utilizam o parâmetro g como parte do modelo e o tamanho das transações ($l(n)$).

4.3 Implementação

A arquitetura do controlador de memória proposto nessa tese foi implementada em linguagem VHDL com foco para síntese para FPGA e integração de elementos de processamento em plataforma de desenvolvimento. Esta Seção descreve os aspectos funcionais e construtivos da arquitetura e também o funcionamento do árbitro adaptativo. O árbitro utiliza informações de prazo para conclusão dos acessos e tamanho dos acessos para gerenciar as transações para a memória externa. O modelo de comportamento do subsistema de memória leva em conta duas informações sobre os acessos dos clientes: a granularidade das transações e o tamanho da transação de cada cliente. O prazo de conclusão da transação é usado para o controle de acesso e escalonamento, para uma classificação adaptativa dos clientes em tempo de execução.

A arquitetura proposta para implementação do controlador multi-cliente de memória externa é baseada em uma parte de interface com a memória externa, denominada de “firm-IP”, e uma parte flexível denominada de “ soft-IP”. O “firm-IP” compõe a parte do circuito que deve ser mapeada de forma rígida para fazer a interface de sinais com o dispositivo de memória externa. Este módulo é descrito em linguagem de descrição de hardware mas é mapeado pela ferramenta de implementação do circuito para o FPGA, de forma que este módulo é estritamente dependente da plataforma alvo. O restante do controlador de memória multi-cliente está implementado em linguagem VHDL, com vistas à implementação em placa e simulação em precisão de ciclo. Além disso, o controlador foi descrito utilizando recursos de configuração de hardware em tempo de implementação, permitindo a configuração do número de clientes e dos parâmetros da memória alvo sem necessidade de alteração do código em si mas através de constantes definidas em um pacote adicional.

Esta implementação do controlador de memória multi-cliente batizada de MDDR (*Multi-client DDR*) é composta pelos módulos principais: árbitro, monitor de clientes, buffer de comandos e interface com IP. O IP de controle da DDR é um módulo externo ao MDDR, assim planejado para simplificar o porte do controlador para outras plataformas ou utilizando diferentes tecnologias de memória DDR SDRAM. O árbitro de controle dos clientes é a parte principal desenvolvida nesse trabalho, sendo composto pelos módulos: seletor de clientes, calculador da

granularidade, calculador do WCRT e gerador de interrupções. O seletor de clientes atua como um escalonador de transações baseado nas informações de acesso geradas pelo árbitro como prioridade dos clientes, granularidade e interrupções pendentes. O buffer de comandos é utilizado para armazenar temporariamente os comandos gerados pelos clientes para acessar o canal de memória e também recebe as informações de prazo de conclusão e tamanho da transação. O monitor de clientes é usado para monitorar quantos clientes estão efetivamente acessando o canal de memória, passando esta informação para o árbitro do controlador.

4.3.1 Árbitro Adaptativo

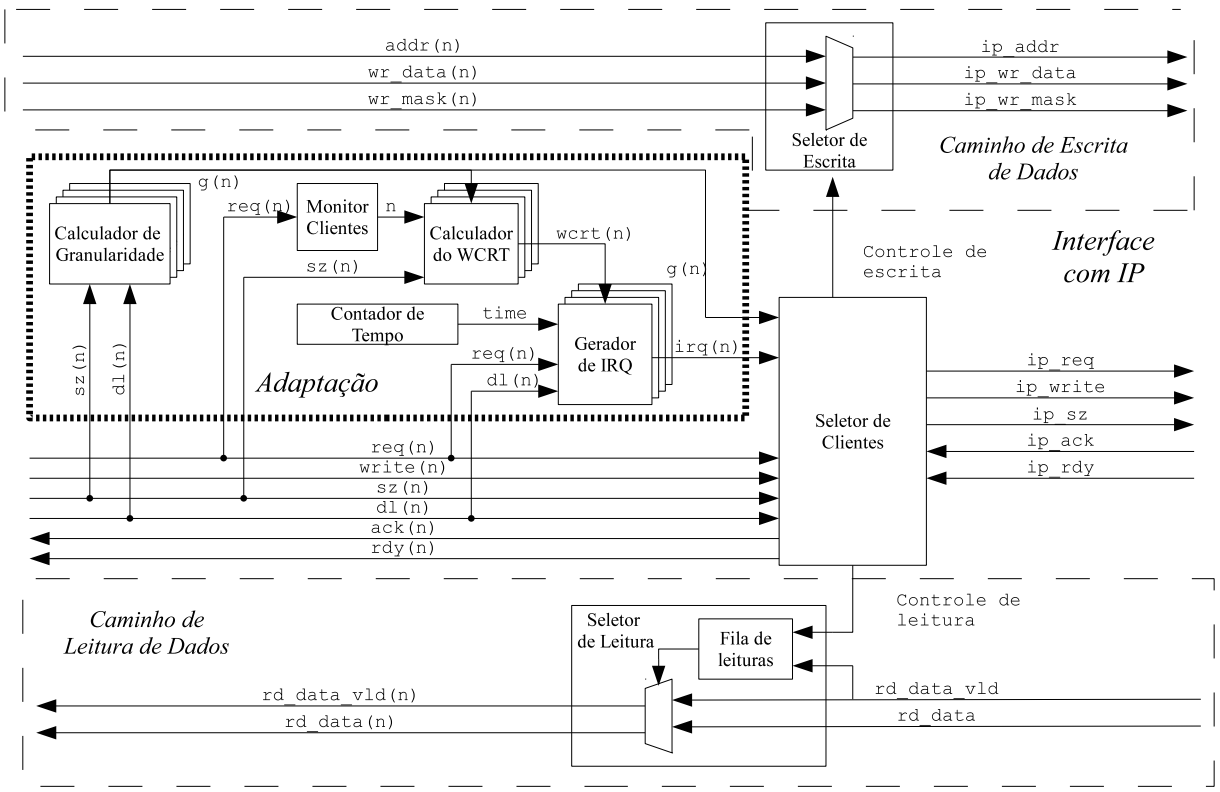
O árbitro adaptativo avalia em tempo de execução os parâmetros de acesso que tendem ao melhor funcionamento do sistema, com vistas ao conjunto de clientes que está gerando acessos ao canal de memória. A Figura 4.15 mostra o diagrama de blocos com os módulos principais que formam o árbitro adaptativo. A conexão com os n clientes conectados ao controlador de memória é representada nessa figura pelos sinais de interface de comandos e endereço (req, write, sz, dl, ack e rdy). O sinal sz é gerado pelo cliente e usado para sinalizar o tamanho da transação. O sinal dl é gerado pelo cliente e sinaliza ao árbitro qual é o prazo para concluir a transação. Os demais são utilizados para implementar o protocolo de transmissão de comandos. Também faz parte a interface de dados escritos e lidos, representados pelos sinais de escrita de dados, máscara e endereço (addr, wr_data e wr_mask) e os dados lidos que são enviados pelo IP para os clientes (rd_data e rd_data_vld).

Tanto na interface entre as entradas de clientes quanto na interface com o IP, a largura do barramento de dados é parametrizável e ajustada para comportar a quantidade de bytes de uma rajada na memória externa por ciclo de relógio. Dessa forma, para uma memória DDR com 64 bits de largura de barramento de dados e rajadas de tamanho igual a oito (BL8), o barramento de dados no controlador de memória é de 512 bits, ou seja, 64 bytes. Durante uma transação de escrita, a sequência de comandos é gerenciada pelo seletor de clientes que passa os comandos para a interface com o IP, responsável por traduzir para a memória externa. Durante uma escrita de dados, a sequência de dados escritos é passada juntamente com respectivos endereços de escrita através de um "seletor de escrita", implementado através de um multiplexador de dados controlado pelo "Seletor de clientes". Durante uma transação de leitura, somente é passada para o IP a sequência de comandos e endereços e os dados lidos são geridos pelo IP juntamente com uma sinalização de dado válido (rd_data_vld).

Como a leitura de dados tem latência elevada, as sequências de comandos de leitura são passadas pelo IP, que não faz reordenamento de comandos, e esta sequencia é armazenada em uma fila de comandos de leitura, necessária para demultiplexar os dados recebidos do IP para a interface de cliente respectiva. Cada comando de leitura de dados enviado para o IP retorna um conjunto de 64 bytes que são entregues em um único ciclo de relógio para o cliente.

A adaptação ao número de cliente e ao seu comportamento é implementada pelos módulos de cálculo de granularidade e de WCRT. O cálculo é realizado cada vez que exista uma alteração

Figura 4.15: Diagrama de blocos do árbitro adaptativo.



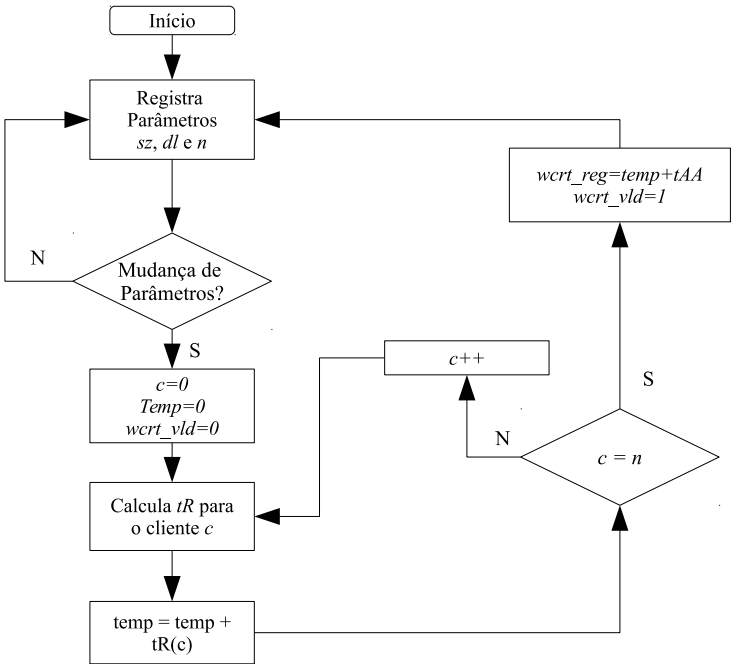
Fonte: Elaborado pelo autor.

dos parâmetros de entrada de clientes: *sz*, *dl* e *n*. O número de clientes ativos é estimado pelo "monitor de clientes", que verifica a atividade dos clientes conectados ao controlador e detecta se estão acessando o canal de memória dentro de um intervalo médio de ciclos de relógio. O fator *n* é utilizado para calcular o WCRT para os clientes, juntamente com a granularidade *g* calculada para o cliente é preemptado. A granularidade utilizada para preemptar o cliente é gerada a partir da análise apresentada na seção 4.2.5 a partir da estimativa de melhoria da largura de banda para o cliente com menor prioridade. Nessa análise foram apresentadas margens teóricas de piores casos possíveis para redução da largura de banda para o cliente que preempta e melhor caso possível de melhoria de largura de banda para o cliente que está sendo preemptado.

Para simplificar a implementação utilizaremos a divisão do tamanho da rajada por dois, que pode ser implementado com baixo custo de hardware usando um deslocamento à direita. O objetivo do cálculo do WCRT para cada cliente é alimentar o gerador de interrupções que compara o tempo absoluto com o prazo estabelecido pelo cliente e com o tempo de transação para dessa forma gerar um aviso ao escalonador quando o prazo desse cliente estourar. Dessa forma, o cliente é priorizado e consegue ser atendido em tempo. A Figura 4.16 mostra o fluxograma utilizado para cálculo do WCRT para o conjunto de *n* clientes que está conectado ao controlador de memória.

O calculador do WCRT implementa para cada interface de cliente um modulo que calcula o tempo de resposta para o cliente, em função o tamanho da transação solicitada pelo cliente, e

Figura 4.16: Fluxograma de cálculo do WCRT.



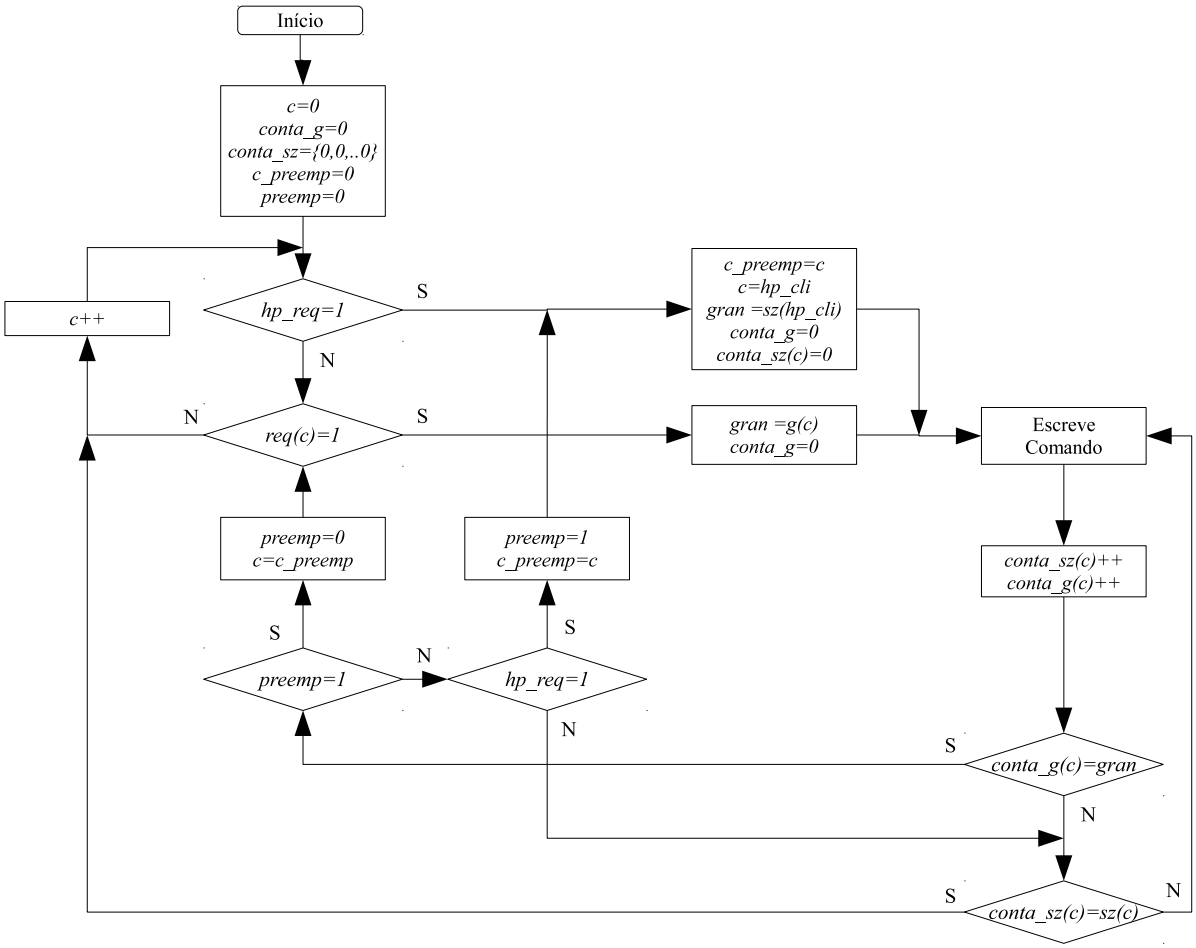
Fonte: Elaborado pelo autor.

quando ocorre uma mudança nos parâmetros de entrada ele executa o laço de soma dos tempos de resposta. Este módulo constantemente monitora a entrada de clientes e calcula os tempos de resposta usando a função (4.8), que gera uma estimativa do pior caso de tempo de resposta para o cliente considerando que ele seja preemptado por outros clientes. Como não existe classificação por prioridades fixas, qualquer cliente pode ser preemptado. O resultado de (4.8) é somado ao atraso gerado por uma auto-atualização (tAA), gerando o WCRT para qualquer cliente. A característica de previsibilidade do subsistema de memória é mantida através da utilização de uma granularidade de transferência mínima para todos os clientes, que permite conhecer a priori os piores casos de latência e de largura de banda para os clientes.

As Figuras 4.17 e 4.18 apresentam respectivamente o fluxograma do escalonador de clientes a a representação dos estados da máquina de estados finitos que implementa o escalonador de clientes. Na Figuras 4.17, após a inicialização do sistema o escalonador realiza uma verificação de requisição de cliente com maior prioridade a partir da variável hp_req. Caso não haja uma requisição de maior prioridade, então o escalonador verifica se existe uma requisição de cliente pela variável req(c) para um cliente *c*. O controle da granularidade dos acessos é feito a partir da variável conta_g(c), que é monitorada para cada um dos clientes.

A MEF que implementa o escanolador é descrita na Figura 4.18. O estado preempta é utilizado para verificar se um cliente de maior prioridade está solicitando acesso ao canal de memória. Esta verificação também é feita no estado ver_cli. Os estados ver_cli e prox_cli são usados para verificar todas entradas de clientes se existe algum pedido de comando válido armazenado no buffer de comandos. No estado ver_cli também é feita a verificação de pedido de

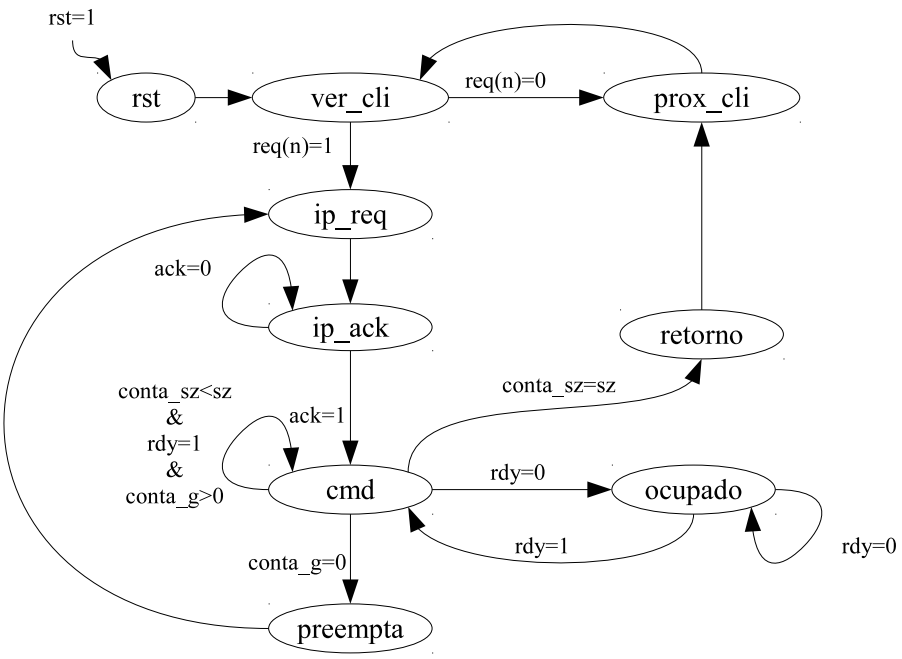
Figura 4.17: Fluxograma do escalonador de clientes.



Fonte: Elaborado pelo autor.

IRQ gerado por um cliente. Assim que um pedido é recebido a máquina passa para os estados ip_req e ip_ack, onde começa a negociação de início da transação com o IP controlador de memória. No estado ip_req são carregados os contadores conta_sz e conta_g, usados para verificar os limites aplicados para a transação. O contador conta_sz é usado para controlar quando a transação solicitada pelo cliente é completada. O contador conta_g é usado para controlar quando é atingida a granularidade mínima para que o cliente atual possa ser interrompido por outro cliente com maior prioridade. Ao receber a permissão de entrada de comandos do IP, a máquina passa para o estado cmd, o qual é utilizado para escrever a transação para o IP. Este estado também verifica se o IP está ocupado, através do sinal rdy, se a transação chegou ao final, através do contador conta_sz ou estourou o contador de granularidade. Quando é atingida a granularidade da transação, há uma mudança para o estado preempta, que faz a troca de cliente caso haja outro cliente solicitando acesso. Caso contrário, o mesmo cliente continua acessando o canal de memória.

Figura 4.18: Representação da máquina de estados finitos implementada para fazer a seleção de clientes.



Fonte: Elaborado pelo autor.

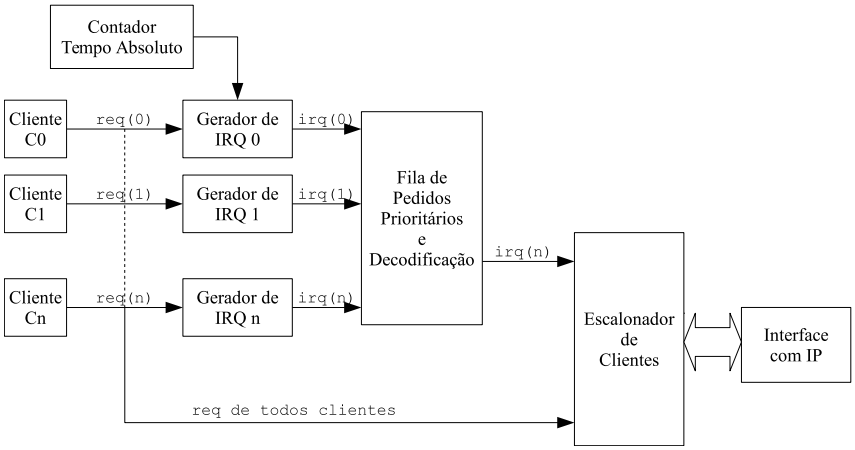
4.3.2 Agendamento de Início da Transação

O agendamento é feito com base no cálculo do WCRT, feito previamente para cada cliente. Se acontecer a mudança de um parâmetro na hora do pedido, o novo cálculo é feito em alguns ciclos de relógio, dependendo da forma como é implementado o cálculo de granularidade e de cálculo do WCRT. A granularidade dos acessos é calculada fazendo a divisão por dois (ou seja, um deslocamento para direita) do valor de tamanho da transação de cada cliente. O agendamento de pedidos é feito com um contador de tempo absoluto que compara o instante que cada pedido é realizado, o prazo solicitado pelo cliente e o WCRT para o cliente. O circuito gera um sinal de pedido de início imediato para o cliente que está com o prazo em seu limite. Esta informação é passa para um módulo que ordena os pedidos mais prioritários, alimentando o escalonador que ao receber um pedido prioritário faz a preempção do cliente que está acessando o canal de memória e passa o controle para esse cliente.

A Figura 4.19 mostra a parte do árbitro adaptativo que contém o escalonador de clientes conectado à entrada do IP de controle da memória externa, a fila de clientes prioritários que geram IRQs e os módulos de geração de IRQ.

Esta arquitetura utiliza um contador de tempo absoluto de 16 bits sem controle de virada, e implementa o calculo do instante em que deve ser gerado o pedido de preempção do cliente que está acessando o canal de memória. A função que calcula o instante é dada por $\delta irq(n) = \delta req(n) + dl(n) - wcrt(n)$, onde $\delta irq(n)$ é o instante absoluto em que deve ser gerado o sinal de interrupção para o escalonador de clientes, $\delta req(n)$ é o instante absoluto em que foi gerado

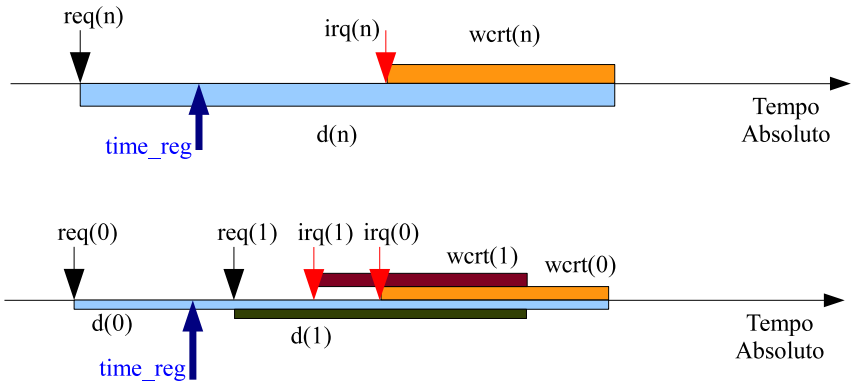
Figura 4.19: Diagrama do circuito que implementa os geradores de IRQs para cada um dos clientes a partir da comparação com o contador de tempo absoluto e fila de IRQs pendentes para o seletor de clientes.



Fonte: Elaborado pelo autor.

o pedido de transação pelo cliente, $dl(n)$ é o tempo de prazo limite para completar a transação e $wcrt(n)$ é o tempo necessário no pior caso para completar a transação. Diagrama de tempo do calculo do pedido de interrupção a partir do gerador de interrupção e do contador de tempo absoluto e mostrado na Figura 4.20.

Figura 4.20: Diagrama de tempo que representa dois casos de interrupção gerada por clientes que estão em espera para acessar o canal de memória. No segundo caso, a interrupção gerada pelo cliente 1 (irq1) é processada antes da interrupção gerada pelo cliente 0.

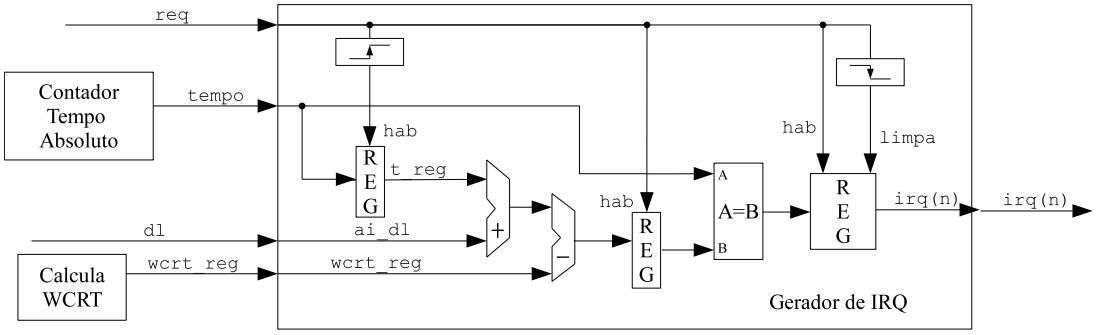


Fonte: Elaborado pelo autor.

A arquitetura do gerador de interrupção é descrita pela Figura 4.21. Esta arquitetura é replicada para cada cliente conectado ao controlador de memória e deve estar conectada ao mesmo contador de tempo absoluto que os demais geradores de interrupção. A arquitetura contém um somador, um subtrator e um comparador, todos de 16 bits. A entrada wcrt_reg contém o valor calculado previamente com o WCRT para o cliente. Este módulo contém um registrador t_reg que armazena o instante em que ocorre a borda de subida do sinal req gerado pelo cliente, que é detectado usando um detector de borda ascendente. O instante em que

ocorre a requisição fica registrado a partir do valor do contador de tempo absoluto, sendo usado no conjunto somador-subtrator que gera o instante futuro em que deverá ocorrer a interrupção. Quando o contador de tempo atinge o instante futuro, o comparador gera o sinal de interrupção. O registrador de saída é limpo quando ocorre a borda de descida do sinal req, usando um detector de borda descendente, sinalizando que o cliente finalizou a transação para o canal de memória.

Figura 4.21: Diagrama do circuito de cálculo do instante de tempo para gerar o IRQ.



Fonte: Elaborado pelo autor.

4.4 Resumo do Capítulo

Este Capítulo apresentou uma descrição do método proposto para um fluxo de projeto centralizado em memória, que fundamenta a implementação de controladores de memória com garantias de prazo de conclusão e de maximização da largura de banda de memória. A arquitetura do controlador de memória com árbitro adaptativo foi descrita, bem como a proposta de integração com módulos que compõem um SoC. No projeto centralizado em memória, uma arquitetura dedicada de controle de acessos é implementada para gerenciar transações entre módulos do SoC e memória com garantias de prazo máximo e de largura de banda mínima. Para que o método proposto seja aplicada ao sistema, os módulos que compõem o SoC devem gerar informações adicionais para o acesso à memória: prazo de conclusão e tamanho mínimo da transferência. O controlador de memória prioriza a manutenção das transferências em sequências de rajadas, aumentando a granularidade dos acessos de forma que a largura de banda do sistema é maximizada. A análise dos piores casos de acesso para o conjunto dos clientes conectados ao subsistema de memória é descrita em função dos parâmetros temporais de funcionamento da memória DRAM, utilizada como memória principal do sistema. Este Capítulo discutiu o uso de diferentes árbitros para ordenar comandos para o controlador de memória e a sua influência na previsibilidade do sistema a na preempção de clientes. A granularidade de interrupção de uma transação foi discutida e apresentada como um fator de melhoria da largura de banda para o sistema. O próximo Capítulo apresenta os resultados da implementação e avaliação do método proposto.

5 ANÁLISE DOS RESULTADOS

A análise dos resultados apresentada neste Capítulo descreve os benefícios de uso do fluxo de projeto centralizado em memória. Os resultados foram obtidos através de simulação comportamental com precisão de ciclo de relógio. O sistema consiste de um controlador de memória multi-cliente com árbitro de gerenciamento das transações geradas por clientes. Para a obtenção dos resultados utilizou-se um modelo comportamental de memória DDR com precisão de ciclo de relógio, que é capaz de reproduzir o comportamento ao longo do tempo de um dispositivo de memória DDR3 SDRAM conectado ao controlador de memória.

Os resultados apresentados nessa seção mostram que o controlador de memória proposto, com árbitro adaptativo baseado na análise do pior caso do tempo de resposta dos clientes, pode ser usado para gerenciar acessos de clientes classificados como de tempo real estrito (*hard real-time*) quando a demanda de acesso solicitada pelos clientes é menor do que a largura de banda máxima sustentada pela memória. Quando a demanda de acesso dos clientes supera a largura de banda máxima sustentada pela memória, o controlador é capaz de satisfazer requisitos de tempo real firme (*firm real-time*)[1].

A avaliação dos resultados é uma tarefa com espectro muito amplo de possibilidades, visto que existem ao menos quatro variáveis diferentes que devem ser ajustadas para simulação: o número de clientes conectados ao controlador de memória, o tamanho das transações realizadas pelos clientes (homogêneas ou heterogêneas), a largura de banda solicitada por cada cliente e os prazos de conclusão determinados pelos clientes. Essas variáveis são modificadas a fim de avaliar os resultados de tempo de resposta e de largura de banda em cada uma das configurações. Além disso, as simulações são feitas utilizando o árbitro adaptativo proposto neste trabalho e outros árbitros implementados em conjunto com o controlador de memória, tipicamente utilizados em controladores de memória, para permitir comparação.

O objetivo do controlador proposto é o controle dos prazos de execução das transações para que seja mantida a largura de banda solicitada pelo cliente. Quando um prazo é perdido, o cliente pode perder um acesso à memória, atrasando o seu funcionamento e por consequência

[1] Processos do tipo *firm real-time* são classificados como tolerantes à algumas falhas de prazos de conclusão, que podem degradar o funcionamento do sistema reduzindo a qualidade mas não provocam uma falha de funcionamento. Já os processos do tipo tempo-real estrito (*hard real-time*) não são tolerantes à falhas de conclusão dos prazos de acesso.

reduzindo a largura de banda alocada para este cliente. Para tanto, primeiramente serão avaliados os limites de funcionamento possíveis para a arquitetura alvo, composta pelo controlador de memória e pela memória externa. Tais limites são evidenciados através da análise dos piores casos de execução.

Este Capítulo primeiramente descreve as características do ambiente de simulação utilizado e na sequência apresenta os resultados de síntese do controlador de memória. A análise e comparação do modelo proposto para estimar o pior caso de tempo de resposta é apresentada na sequência, comparando os resultados do modelo com os resultados obtidos em simulação. Esta análise é feita para diferentes configurações de clientes e de acessos conectados ao controlador de memória. Ao final é feita a análise dos tempos de resposta e de largura de banda para diferentes configurações de clientes e tamanhos de transações, comparando resultados obtidos em simulação para o árbitro adaptativo proposto e outros algoritmos de arbitragem e controle de acessos.

5.1 Descrição do Ambiente de Simulação

O controlador de memória com suporte para multi-clientes foi descrito em linguagem VHDL sintetizável para hardware, com suporte para configuração a partir de parâmetros de escalabilidade de interfaces de entrada como: número de clientes conectados ao controlador, largura de barramento de dados e endereços e parâmetros de funcionamento da memória DDR. O controlador de memória foi implementado com suporte para três árbitros diferentes, tipicamente utilizados no gerenciamento de acesso à recursos compartilhados, além do árbitro proposto neste trabalho. O árbitro pode ser configurado para operar como esquema adaptativo (AA), fila de ordem de chegada de comandos (FCFS), Round-Robin com quantum infinito (RR) e prioridade fixa com preempção (FPP). Cada um dos árbitros possui características próprias que serão avaliadas nessa seção a partir dos resultados de simulação apresentados. Cada um dos árbitros implementados está inserido no núcleo do controlador de memória, sendo que possuem a mesma interface de entrada e saída para outros módulos do controlador de memória e são configurados em tempo de síntese. Somente um árbitro pode ser utilizado após a configuração do sistema. O propósito dessa implementação é validar as vantagens de cada um dos árbitros, de forma a comparar o desempenho final do sistema.

O controlador de memória multicliente utiliza um módulo de interface física para a memória externa disponibilizado pelo fabricante de FPGAs Xilinx, que pode ser gerado através do programa *Memory Interface Generator* (MIG) (XILINX, 2014a). Este IP está disponível sob licença de uso das ferramentas de síntese que acompanham a placa de desenvolvimento para FPGA utilizada nesse trabalho, Xilinx ML605 (XILINX, 2014b). Esta placa de desenvolvimento contém um FPGA XC6VLX240T-1FFG1156 e um pente de memória externa DDR3 SDRAM 1GB com quatro dispositivos de memória com largura de barramento total igual a 64 bits e velocidade de relógio do barramento de dados igual a 400 MHz. A memória da placa de

prototipagem está configurada para latência de leitura (CL) igual a 6 ciclos de relógio (15 ns) e funciona a uma taxa máxima de 800 mega-transferências de 64 bits por segundo, ou seja, uma lagura de banda de pico igual a 6,4GB/s. Demais parâmetros da memória estão apresentados na Tabela (5.1).

Tabela 5.1: Parâmetros da memória DDR3-800 CL6.

Parâmetro	***ns***	***ciclos***	***Parâmetro***	***ns***	***ciclos***
tCK	2,5	1	tRC	52,5	21
tRAS	37,5	15	tRP	12,5	5
tRTP	10,0	4	tCCD	10,0	4
tRL	15	6	tRFC	110	44
tRCD	12,5	5	tCWD	15	6
tWR	15	6	tBURST	10,0	4

Fonte: elaborado pelo autor.

As avaliações mostradas nessa seção foram obtidas com um modelo de memória descrito em linguagem Verilog, para a memória DDR3-800 que é utilizada na placa de prototipagem ML605. O modelo de memória é fornecido pelo fabricante Micron (MICRON, 2009).

Foram implementados geradores de acesso para simular o funcionamento dos clientes conectados ao controlador de memória. Cada gerador de acesso tem um período de repetição do pedido de transação e gera informações de tamanho, prazo de conclusão e endereçamento do acesso. Os testes apresentados foram simulados para uma demanda de comandos em sequências de 50% escritas e 50% leituras intercaladas. Para os simuladores de acesso, a geração de endereços força a troca de páginas no controlador de memória entre rajadas consecutivas, sejam de escrita ou e leitura, de forma que o IP de controlador de memória é obrigado a operar no modo de página fechada. Este representa o pior caso de operação do sistema, onde cada transação é feita com a ativação e fechamento da página de memória. Os intervalos de acesso gerados pelos simuladores de clientes também são configurados com uma margem aleatória de instante de ocorrência.

5.2 Síntese para Hardware

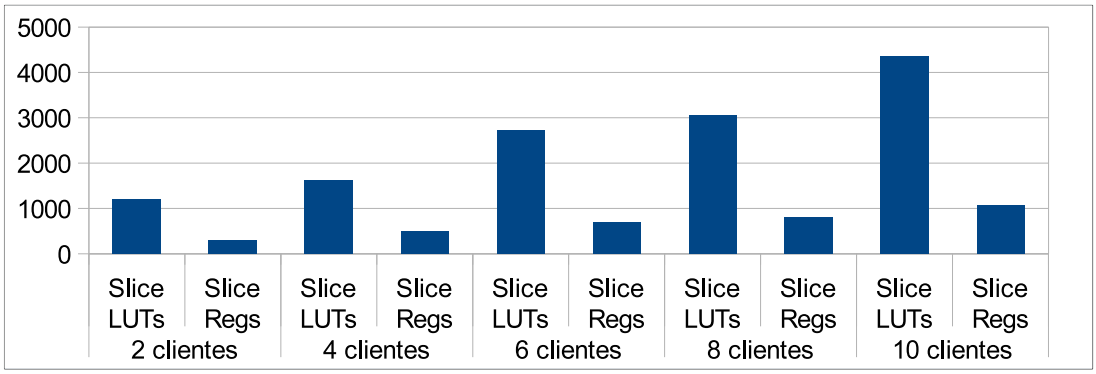
Os resultados de síntese para os módulos que compõem o controlador de memória multi-cliente e os resultados para a sua implementação em diferentes quantidades de interface com clientes são apresentados na Tabela 5.2 e na Figura 5.1. A área ocupada pelo controlador de memória é uma função do número de clientes conectados, pois os módulos de cálculo do WCRT e gerador de IRQ são gerados um para cada cliente. A interface com o IP, o árbitro, o módulo de seleção de clientes (escalonador) e o monitor de clientes são gerados com interfaces escaláveis para o número de clientes. Esta implementação é capaz de rodar a a uma frequência de 200 MHz suportando até 10 clientes, devido à uma limitação de velocidade do seletor de clientes.

Tabela 5.2: Resultados de Síntese

	Slice LUTs		***Slice Regs***		***Fmax***
	Uso	% Disp	Uso	% Disp	MHz
Interface IP	1961	1,30%	1204	0,40%	363.3
Seletor de Clientes	134	0,09%	51	0,02%	287.3
Calculador do WCRT	108	0,07%	96	0,03%	251
Gerador de IRQ	97	0,06%	18	0,01%	231
Monitor de clientes	96	0,06%	64	0,02%	307
Árbitro (2 cli)	1204	0,80%	229	0,08%	228
TOTAL	3504	2,32%	1598	0,53%	

Fonte: elaborado pelo autor.

Figura 5.1: Resultados de síntese para FPGA Virtex-6 XC6VLX240T.



Fonte: elaborado pelo autor.

5.3 Análise do Pior Caso

Esta Seção apresenta resultados de simulação do controlador de memória implementado para validar os melhores e os piores casos de execução que são previstos no modelo matemático simplificado utilizado para calcular o WCRT. O melhor caso (“best-case”) de execução ocorre quando o cliente que solicita o acesso à memória gera o pedido de transação no momento em que o canal está livre e o árbitro não está processando alguma outra requisição de outro cliente. Para configurar um melhor caso, a memória externa também deve estar livre sem a ocorrência de uma pré-carga. A avaliação dos resultados de simulação de piores casos para o tempo de resposta do circuito descrito em VHDL em relação aos valores calculados no modelo proposto na Seção 4.2. A comparação dos resultados simulados com os calculados é feita a partir de (5.1) e (5.2), para tempo de resposta (tR) e largura de banda (BW), respectivamente.

$$tR\ normalizado = \frac{tR\ simulado}{tR\ calculado} \tag{5.1}$$

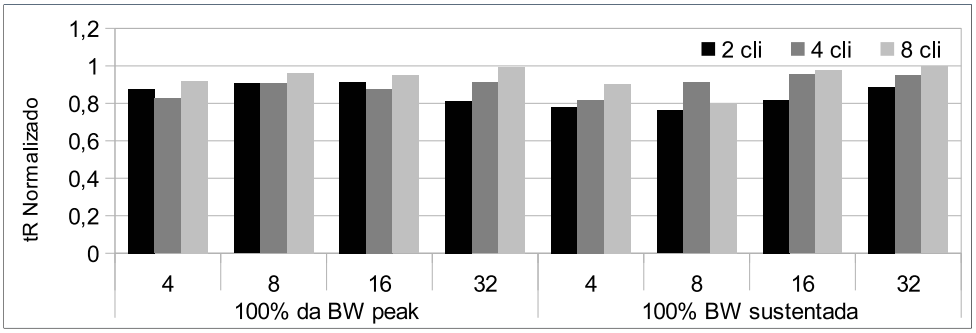
$$BW\ normalizado = \frac{BW\ simulado}{BW\ calculado} \tag{5.2}$$

A Figura 5.2 mostra a avaliação dos resultados simulados para o WCRT em duas situações

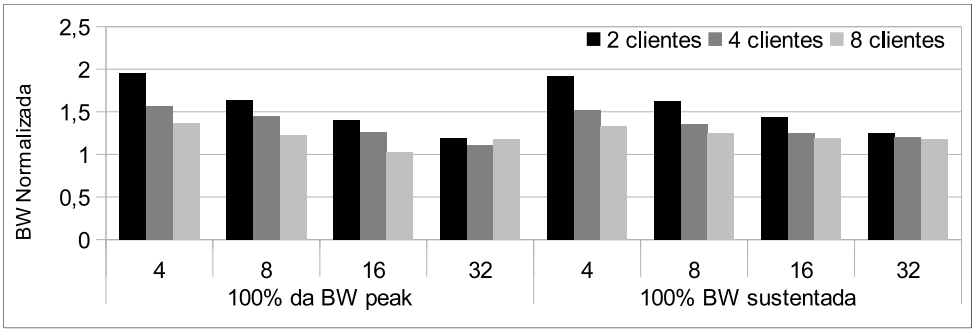
diferentes de demanda de acesso geradas pelos clientes. A primeira situação mostra o controlador sobrecarregado com acessos gerados para ocupar 100% da largura de banda de pico da memória. A segunda situação ocorre com os clientes solicitando 100% da largura de banda máxima sustentada da memória. O tempo de resposta normalizado que está apresentado na Figura 5.2a mostra que os valores calculados para diferentes situações de implementação do controlador de memória são suficientes para prever os piores casos de execução do sistema. Na análise apresentada, geradores de acesso são utilizados para inserir comandos de leitura e escrita (50% leitura e 50% escrita intercalados) em intervalos de tempo reduzidos o suficiente para ocupar a largura de banda de pico e a sustentada pela memória. Nas duas situações de ocupação de largura de banda apresentada, os piores casos de execução do controlador de memória ficam abaixo do que previsto pelo modelo.

Figura 5.2: Avaliação do WCRT para o controlador de memória em situações com 2, 4 e 8 clientes gerando solicitações de transação de tamanhos 4 até 32 rajadas consecutivas.

(a) Tempo de resposta normalizado em função do previsto pelo modelo.



(b) Largura de banda normalizada em função da prevista pelo modelo.



Fonte: elaborado pelo autor.

Para esta análise não foram usados limites de prazo de conclusão das transações, pois os prazos foram configurados com valor muito acima do que é o pior caso, sendo dessa forma, o controlador não gera interrupções e todos os clientes são atendidos em ordem de chegada, como em um árbitro do tipo FCFS. Esta análise mostra que o modelo simplificado para os piores casos é confiável quando o sistema está recebendo acessos até o limite previsto de largura de banda

de memória. Quando este limite é excedido, clientes que estão em espera pelo acesso geram sinal de interrupção e o controlador deve lidar com múltiplos clientes gerando interrupções simultaneamente, provocando um comportamento de alternância entre clientes preemptados e clientes que estão preemptando. As medições foram feitas para uma memória DDR3-800 com 50% RD/WR intercaladas.

A partir do WCRT é possível ter um valor aproximado da largura de banda mínima sustentada pelo conjunto controlador de memória e memória externa. A Figura 5.2b mostra um comparativo obtido através de simulação entre os valores de largura de banda máxima para clientes com requisitos iguais de acesso. A largura de banda obtida nas simulações é superior à estimada pela análise do pior caso, que é uma análise pessimista e nem sempre os acessos são coincidentes com o pior caso possível. Nessa avaliação, os prazos de conclusão da transação para os clientes são superiores ao WCRT, portanto não são geradas interrupções e a distribuição da banda é a mais equitativa possível. Essa avaliação é útil para caracterizar o funcionamento do controlador de memória para projetar as aplicações que irão rodar sobre a plataforma alvo.

Os valores apresentados para piores casos do tempo de resposta normalizados no gráfico da Figura 5.2 mostra que existe uma variação entre o tempo de resposta que é medido no sistema em execução em relação ao tempo de resposta que é calculado. Esta variação é de 0,8 e 1,0 para o caso onde o controlador de memória é exigido ao máximo e entre 0,6 e 1,0 quando o controlador é exigido até o limite da capacidade teórica sustentada pela memória. Esta variação, embora represente que a análise está sendo pessimista em relação ao que ocorre no sistema em execução, apresenta uma variação menor do que os resultados apresentados em (SHAH; KNOLL; AKESSON, 2013), que superestima os tempos de execução em 2 até 7 vezes o que ocorre na realidade. Embora a análise aqui apresentada seja simples de implementar ela mostra resultados mais precisos sobre os piores casos de execução.

A principal vantagem da análise simplificada do pior caso é a simplicidade de implementação em hardware. Como apresentado em (GOMONY; AKESSON; GOOSSENS, 2013), uma análise detalhada dos piores casos de execução e de alocação de largura de banda em um SoC composto por 25 clientes pode levar até 3 horas e com 100 clientes até 2 dias, inviabilizando o cálculo em tempo-real de uma aplicação. Portanto, a análise simplificada que é apresentada nessa tese além de ser eficiente em termos de precisão da estimativa dos piores casos para que o controlador de memória tenha uma estimativa da qualidade de serviço fornecido aos clientes e da alocação de largura de banda.

5.4 Estudo de Caso - Controle dos tempos de Resposta

As garantias impostas de completar as transações dentro do tempo de resposta foram avaliadas em duas situações diferentes, com 4 clientes compartilhando o acesso ao canal de memória. A análise é feita para diferentes situações de frequência dos acessos gerados pelos clientes e de solicitação de prazos de conclusão das transações. Esta análise faz uma comparação entre

o funcionamento do árbitro adaptativo em relação à árbitros FCFS, RR e prioridade fixa. A largura de banda máxima sustentada pela memória é calculada a partir dos atrasos na memória para gerar a sequência de acessos aos dados, no entanto, este valor é o máximo teorico possível para acessos em rajadas, como explicado anteriormente na seção 3.2. Para acessos no formato de 50% RW/WR em sequências de 16 rajadas por transação, o limite teórico da DDR3-800 é igual a 4951,2 MB/s. A Tabela 5.3 mostra as diferentes configurações de prazo e largura de banda solicitada para cada cliente nos Casos 1 até 6 simulados a partir da descrição VHDL.

Tabela 5.3: Configuração dos casos simulados.

Clientes	**Caso 1**		**Caso 2**		**Caso 3**	
	$BW(MB/s)$	$dl(n)$	$BW(MB/s)$	$dl(n)$	$BW(MB/s)$	$dl(n)$
C0	618	5000 ns	1236	700 ns	1236	700 ns
C1	618	5000 ns	618	5000 ns	1236	700 ns
C2	618	5000 ns	618	5000 ns	618	5000 ns
C3	618	5000 ns	618	5000 ns	618	5000 ns
Clientes	**Caso 4**		**Caso 5**		**Caso 6**	
	$BW(MB/s)$	$dl(n)$	$BW(MB/s)$	$dl(n)$	$BW(MB/s)$	$dl(n)$
C0	1236	5000 ns	1236	800 ns	1236	800 ns
C1	1236	5000 ns	1236	5000 ns	1236	800 ns
C2	1236	5000 ns	1236	5000 ns	1236	5000 ns
C3	1236	5000 ns	1236	5000 ns	1236	5000 ns

Fonte: elaborado pelo autor.

O primeiro caso avaliado (Caso 1) é com cada cliente ocupando 12,5% da largura de banda sustentada pelo canal de memória para transações feitas em sequências de 16 rajadas, onde cada cliente solicita uma transação a cada 1.655 ns com jitter de 10%. Isso resulta em uma largura de banda solicitada por cliente igual a 618 MB/s. Nesse primeiro caso os prazos de conclusão das transações para todos os clientes é igual a 5000 ns, de forma que o controlador com árbitro adaptativo fica livre para gerenciar de qualquer forma os acessos.

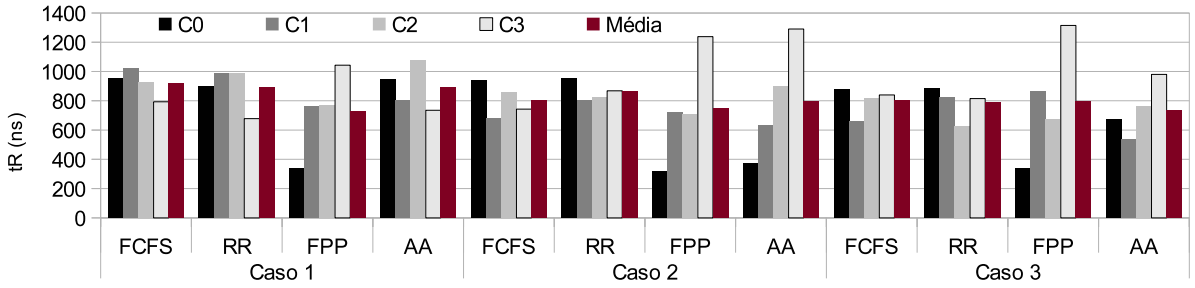
O segundo caso é configurado de forma que o cliente c0 passa a ocupar 25% da largura de banda sustentada, perfazendo um acesso a cada 827 ns com jitter de 10%, que gera uma largura de banda solicitada igual a 1236 MB/s e os três outros clientes ocupam 12,5% da largura de banda sustentada da memória e não tem prazos de conclusão. Nessa situação, o cliente c0 informa ao controlador um prazo de conclusão igual a 700 ns, que está abaixo do WCRT para esta configuração de 1.215 ns.

O terceiro caso avaliado ocorre quando o cliente c1 aumenta a largura de banda solicitada para 25% da largura de banda sustentada e impõe um prazo de conclusão de 700 ns. Tais prazos são realizáveis dentro da configuração atual desse sistema, visto que o tempo para realizar uma transação de tamanho de 16 rajadas para um cliente é igual a 265 ns e o tempo de uma operação de auto-refresh é de 155 ns, totalizando para 2 clientes 685 ns no pior dos casos. A Figura 5.3 mostra os resultados dessa análise.

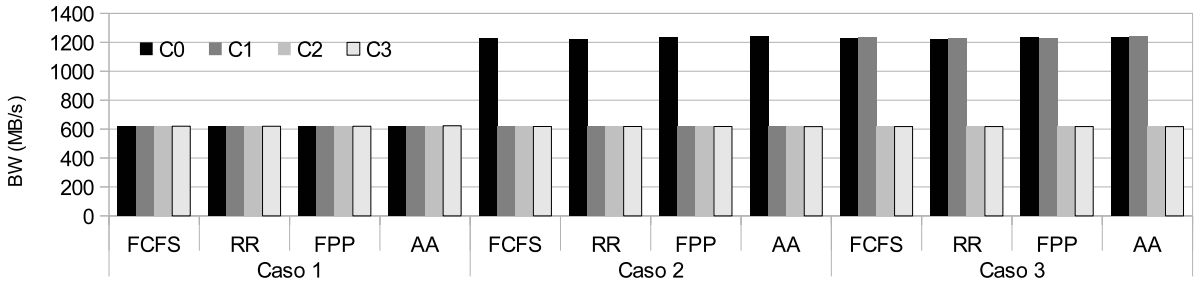
Na Figura 5.3a são mostrados os piores casos do tempo de resposta para cada um dos clien-

Figura 5.3: Análise dos piores casos de execução para um sistema com quatro clientes fazendo transações de tamanho igual a 16 rajadas para 50%, 62% e 74% da largura de banda máxima sustentada pela memória.

(a) Piores casos de tempo de resposta.



(b) Largura de banda por cliente.



Fonte: elaborado pelo autor.

tes em cada um dos três casos simulados. Cada um dos casos apresentados foi simulado durante 1 milhão de ciclos de relógio com o conjunto de quatro clientes, suficiente para completar mais de 120 etapas de auto-atualização da memória. No caso 1, todos os quatro modos de arbitragem conseguem satisfazer a largura de banda mínima solicitada pelos clientes, que é igual a 618 MB/s, e também conseguem satisfazer o prazo de 5000 ns para as transferências. Os árbitros RR, FCFS e AA tem comportamento semelhante mas dentre os três o árbitro adaptativo apresenta a menor média de tempo de resposta (3% abaixo que FCFS e 1% abaixo que RR), e conseguem fornecer mais de 50% da largura de banda sustentada pela memória, que teoricamente é de 4951,2 MB/s para 50% RD/WR. Nesse primeiro caso, o árbitro FPP prioriza o cliente C0 reduzindo significativamente o tempo de resposta para este cliente e consegue manter a largura de banda mínima solicitada para todos os clientes.

A primeira alteração de comportamento (caso 2) ocorre quando o cliente C0 duplica a largura de banda solicitada e reduz o prazo para conclusão, sendo que os demais clientes permanecem com o mesmo comportamento de acessos. O árbitro adaptativo e o de prioridade fixa conseguem reduzir o pior caso de tempo de resposta para cumprir com o prazo solicitado, já os demais FCFS e RR não conseguem cumprir o prazo, já que não fazem controle de temporização dos acesos. Além disso, no esquema de arbitragem AA e FPP, o controlador de memória conse-

gue manter a largura de banda solicitada pelo cliente C0, fazendo 1237,6 MB/s no árbitro FPP e 1238,9 MB/s no árbitro AA. Já os modos de arbitragem FCFS e RR não conseguem suprir a largura de banda e ficam 1% abaixo do necessário. Nessa situação, o controlador de memória consegue fornecer 62,5% da largura de banda máxima sustentada da memória para o conjunto de clientes, em qualquer modo de arbitragem.

Na segunda alteração de comportamento (caso 3), o cliente C1 dobra a largura de banda solicitada da memória e reduz o prazo de conclusão solicitado para 700 ns, sendo que os demais clientes permanecem na mesma condição de acesso. Nessa situação, somente o esquema de arbitragem adaptativa consegue fornecer a largura de banda mínima solicitada para o conjunto de clientes e cumpre com os prazos dos clientes C0 e C1 que são de 700 ns. O árbitro FPP coloca maior prioridade para os clientes C0 e C1, mas não consegue manter a largura de banda e o prazo para o cliente C1, que ficam em 1% abaixo do mínimo solicitado e 19% acima do prazo máximo para completar a transação. Os árbitros FCFS e RR não conseguem atender a largura de banda solicitada e os prazos de conclusão são maiores do que os solicitados pelos clientes. A média dos piores tempos de resposta para o caso 3 é menor no AA do que nos demais clientes, ficando 8% abaixo do árbitro FPP. Nessa configuração de clientes, o controlador de memória com árbitro adaptativo consegue fornecer 75% da largura de banda máxima sustentada pela memória e cumpre com os prazos solicitados de 700 ns para dois dos quatro clientes.

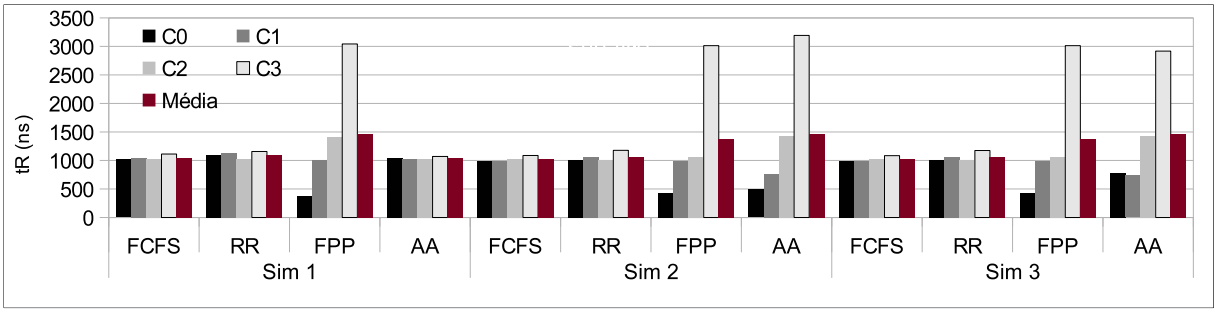
A avaliação apresentada anteriormente para quatro clientes gerando transações para a memória de tamanho igual a 16 rajadas consecutivas e repetida aqui, mas com largura de banda solicitada pelos clientes igual ao valor máximo de largura de banda sustentada teoricamente pela memória. A Figura 5.4 mostra os resultados para a análise dos tempos de resposta máximos e de largura de banda para os três casos de diferentes demandas geradas pelos clientes. Inicialmente, todos clientes geram acessos com intervalos periódicos de 825 ns com jitter de 10% e um prazo de conclusão igual a 5000 ns. Na configuração inicial (Sim 1), cada cliente solicita banda igual a 25% da banda máxima sustentada pela memória, que corresponde à 1238 MB/s, mas o conjunto controlador de memória e memória externa somente conseguem fornecer 85% desse valor, para todos clientes, representando uma banda de aproximadamente 1050 MB/s para cada cliente. Nessa situação, o árbitro AA consegue atingir 86% da largura de banda máxima e com a distribuição mais equilibrada de banda entre todos clientes.

A primeira alteração nessa configuração (Sim 2) ocorre quando o cliente C0 reduz o prazo solicitado para 800 ns, sendo que os demais clientes continuam com prazos elevados. Nessa situação, o árbitro adaptativo regula os prazos de acesso de forma a priorizar o cliente C0, que passa a ser atendido antes dos demais, em um tempo menor que o intervalo de repetição dos acessos (que é de 825 ns). Com esta alteração o cliente C0 passa a ganhar do controlador toda a largura de banda solicitada, atingindo 1238,2 MB/s que representa um aumento de 16,7% de largura de banda em relação à situação inicial. Na segunda alteração (Sim 3), o cliente C1 reduz o prazo solicitado para 800 ns e passa a ser um cliente com maior prioridade do que os clientes C2 e C3. Dessa forma, o árbitro prioriza os prazos desse cliente e aloca uma largura de

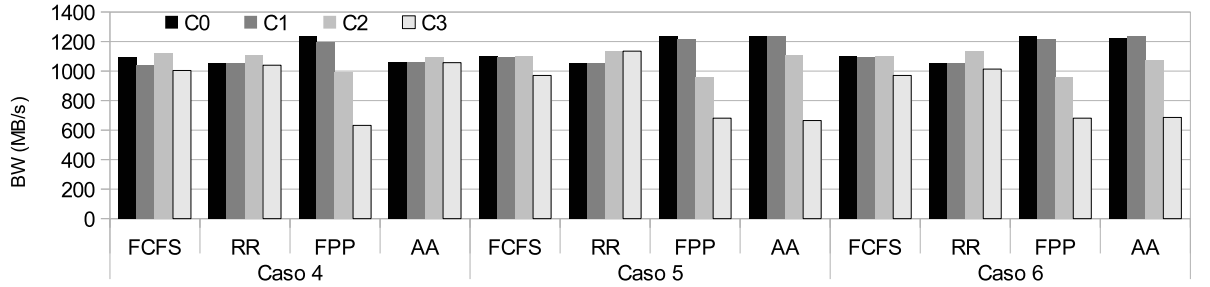
banda maior que passa a ter 1237,6 MB/s, 16,7% a mais que a situação inicial. Com a segunda alteração, a largura de banda do cliente C0 é penalizada em 1,1%. Os demais clientes que não tem restrições tão limitadas de prazo são penalizados em 3,4% para C2 e 35,1% para C3. Isso ocorre pois o sistema atinge o limite de serviço de largura de banda para o conjunto de clientes. No caso do árbitro FPP, a penalização é menor e inclui o próprio cliente C1, que não consegue ter a largura de banda solicitada atendida pelo controlador. Penalizações são de 12,4% e 35,6% para os clientes C2 e C3.

Figura 5.4: Análise dos piores casos de execução para um sistema com quatro clientes fazendo transações de tamanho igual a 16 rajadas para ocupar a largura de banda máxima sustentada pela memória.

(a) Piores casos de tempo de resposta.



(b) Largura de banda por cliente.



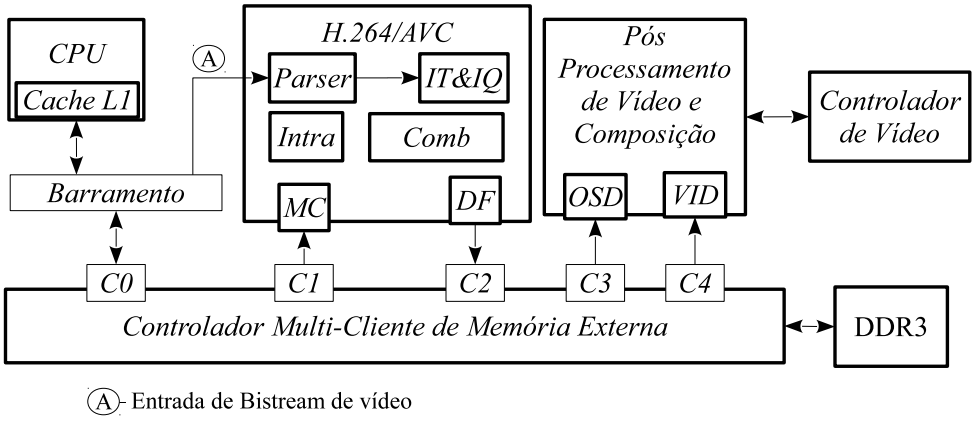
Fonte: elaborado pelo autor.

5.5 Estudo de Caso: Terminal de Acesso de Televisão Digital

Em aplicações multimídia, com o aumento da resolução das imagens e do número de quadros exibidos por segundo, a quantidade de dados em memória aumenta significativamente. Dessa forma, o desempenho das interfaces de memória deve ser melhorado para suportar o processamento de vídeos em formatos de alta definição como 4K ou UHD, com taxas de quadros para permitir filmes em 3 dimensões, como 120 e 240 Hz. Um Terminal de Acesso de Televisão Digital (*Set-Top Box*) é usado como estudo de caso para validação em placa dos pressupostos dessa tese. A Figura 5.5 mostra o diagrama de blocos da implementação do subsistema de

memória para integrar os módulos de processamento multimídia de um Terminal de Acesso de Televisão Digital. Nesse SoC, o subsistema de memória foi implementado a partir de um controlador multi-cliente de memória DRAM, contendo interfaces dedicadas para cinco clientes.

Figura 5.5: Arquitetura de estudo de caso: terminal de acesso de televisão digital.



Fonte: elaborado pelo autor.

Esta arquitetura apresentada na Figura 5.5 utiliza como base de integração o controlador de memória proposto nessa tese, que conecta os módulos de processamento de vídeo, exibição de imagens e gerenciamento de dados para uma CPU. Uma interface de cliente foi utilizada para integrar uma CPU Leon-3 (arquitetura SparcV8). Duas interfaces de clientes foram utilizadas para o módulo de decodificação de vídeo no padrão H.264/AVC. Outras duas interfaces foram utilizadas para o módulo de pós-processamento de vídeo e composição, utilizado para combinar vídeo e gráfico gerado pela CPU em uma única imagem para a saída.

O desenvolvimento dessa arquitetura foi realizado de forma incremental. Os primeiros resultados da implementação da arquitetura foram publicados na primeira edição do LASCAS (*IEEE Latin American Symposium on Circuits and Systems* (BONATTO; SOARES; SUSIN, 2010). O controlador multi-cliente proposto foi utilizado na integração dos módulos de processamento de vídeo de uma plataforma de televisão digital, com resultados apresentados nas edições do SBCCI em 2011 e 2010 (SOARES; BONATTO; SUSIN, 2011), (BONATTO et al., 2010) e no SPL (*Southern Programmable Logic Conference*) (BONATTO; SOARES; SUSIN, 2011). Melhorias foram incluídas na arquitetura do controlador multi-cliente de memória e foram publicados em no Workshop de Sistemas Embarcados SBESC (BONATTO et al., 2012), e no *International Journal of Reconfigurable Systems* (SOARES; BONATTO; SUSIN, 2013). Além disso, este controlador foi utilizado como base de integração de um sistema de processamento de vídeo publicado no SPL em 2012 (NEGREIROS et al., 2012).

A Tabela 5.4 mostra os parâmetros utilizados para configuração dos clientes durante a simulação do comportamento dos acessos à memória do STB e os parâmetros de tempo da memória DDR3-800. A largura de banda solicitada por cada cliente (BW) está descrita na Tabela 5.4 e

os limites de prazo de conclusão para as transações ($dl(n)$) de tamanho igual a $l(n)$ sequências de rajadas para a memória externa. O parâmetro $l(n)$ nos clientes H264, MC, Vídeo, e Gráfico variam entre 1 rajada simples e o máximo de rajadas, para a transação solicitada: 6 no caso do Vídeo, Gráfico e H264 e 18 no caso do MC. Para a memória externa com barramento de dados igual a 64 bits, 6 rajadas BL8 consecutivas para a memória conseguem transportar um macrobloco de 256 pixels no formato YCbCr 4:2:2, que contém 384 bytes.

Tabela 5.4: Parâmetros de configuração da simulação do STB de televisão digital para processar imagens em definição Full-HD.

Parâmetros de configuração dos clientes					
Clientes	***BW (MB/s)***	$l(n)$	$dl(n)$		
CPU	220	1	290		
H264	93,3	1 – 6	680 – 4100		
MC	769,8	1 – 18	500		
Vídeo	93,3	1 – 6	680 – 4100		
Gráfico	186,6	1 – 6	340 – 2050		
Parâmetros da DDR3-800					
Parâmetro	**Ciclos**	**ns**	**Parâmetro**	**Ciclos**	**ns**
tCK	1	2,5	tCCD	4	10
tRC	21	52,5	tRL	6	12,5
tWR	6	15	tRP	6	15
tRCD	5	12,5	tRFC	44	110

Fonte: elaborado pelo autor.

O prazo de conclusão das transações é definido a partir da frequência mínima de acessos para a memória externa, que no caso do processamento de vídeo é determinado pela resolução da imagem que está sendo processada. Para o processamento de vídeo em resolução Full-HD 30fps, a frequência de acessos para a memória gerada pelos clientes H264 e Vídeo varia entre 680 e 4100 ns, dependendo do tamanho das transações, se de 1 rajada ou 6 rajadas. Para o cliente de saída de gráficos o intervalo varia entre 340 e 2050 ns. Para o cliente CPU estabeleceu-se um prazo para conclusão das transações de 290 ns, para conseguir atingir os 220 MB/s em acessos de rajada simples.

Para o cliente MC, que possui a maior demanda de banda da memória, determinou-se um prazo de conclusão fixo de 500 ns. Para este cliente, os 770 MB/s são facilmente transferidos quando fazendo transações de tamanho igual a 18 rajadas consecutivas, que são repetidas a cada 1400 ns. No entanto, para rajadas simples esta banda solicitada é cerca de 90% da largura de banda máxima sustentada da memória, que é limitada em 850 MB/s, e as repetições de transações são feitas a cada 85 ns. Este cliente, quando configurado para transações de rajada simples, ocupa praticamente toda a largura de banda disponível da memória, gerando uma sobrecarga de acessos com elevada concorrência com os demais clientes. O prazo fixo determinado de 500 ns foi estipulado para ser maior do que o da CPU (290 ns) e suficiente para processar uma transação de tamanho igual a 18 com margem para a interrupção de outro cliente mais o tempo de uma auto-atualização da memória.

A fim de avaliar o funcionamento do árbitro adaptativo e fazer uma comparação com outros algoritmos de arbitragem, determinou-se uma simulação do comportamento dos acessos para a memória a partir de 4 casos que abrangem os extremos de funcionamento dessa plataforma. O somatório da largura de banda solicitada pelo conjunto de clientes é cerca de 1360 MB/s que representa 20% da largura de banda de pico da memória externa. Embora seja um valor muito inferior ao limite máximo da memória, a largura de banda máxima sustentada para rajadas simples é de aproximadamente 850 MB/s. A configuração escolhida para comparação é apresentada na Tabela 5.5. Nessa tabela também são apresentados os piores casos previstos de tempo de resposta para o conjunto de clientes. Para todos os quatro casos apresentados, o prazo de conclusão dos clientes CPU e MC é menor do que o WCRT, fazendo com que para esses dois clientes sejam gerados IRQs de preempção imediatamente quando ocorre o pedido de transação para memória. Nos casos 1 e 3, o cliente Gráfico gera IRQ de preempção imediato.

Tabela 5.5: Configuração dos casos simulados para o STB de televisão digital para processar imagens em definição Full-HD.

Clientes	**Caso 1**		**Caso 2**		**Caso 3**		**Caso 4**	
	$l(n)$	$dl(n)$	$l(n)$	$dl(n)$	$l(n)$	$dl(n)$	$l(n)$	$dl(n)$
CPU	1	290 ns	1	290 ns	1	290 ns	1	290 ns
MC	1	500 ns	1	500 ns	18	500 ns	18	500 ns
H264	1	680 ns	6	4100 ns	1	680 ns	6	4100 ns
Video	1	680 ns	6	4100 ns	1	680 ns	6	4100 ns
Gráfico	1	340 ns	6	2050 ns	1	340 ns	6	2050 ns
WCRT	530 ns		680 ns		700 ns		850 ns	

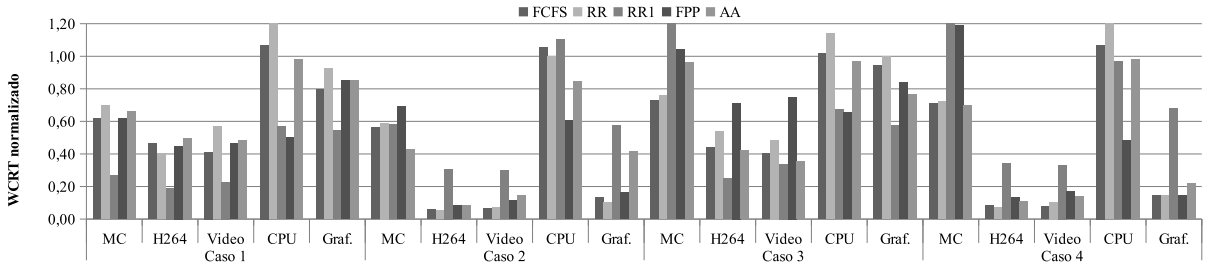
Fonte: elaborado pelo autor.

O funcionamento do árbitro adaptativo (AA) proposto nesse trabalho e implementado no controlador de memória multicliente foi comparado com o algoritmo de arbitragem com prioridade fixa e preempção (FPP), com fila de ordem de chegada (FCFS) e com round-robin (RR). O árbitro round-robin foi implementado de duas formas, com "quantum" de tamanho igual a 1, que força o uso de transações de rajada simples, e com quantum infinito, permitindo que o cliente complete a transação solicitada em seu tamanho integral. O árbitro de prioridade fixa foi implementado com prioridade maior para a CPU. Os resultados de piores casos de tempo de resposta e largura de banda para todos os clientes, comparando os diferentes árbitros implementados, são apresentados nas Figura 5.6, que mostra o comparativo de WCRT normalizado em relação ao prazo *dl* na Figura 5.6a e a largura de banda normalizada em relação ao mínimo solicitado, na Figura 5.6b.

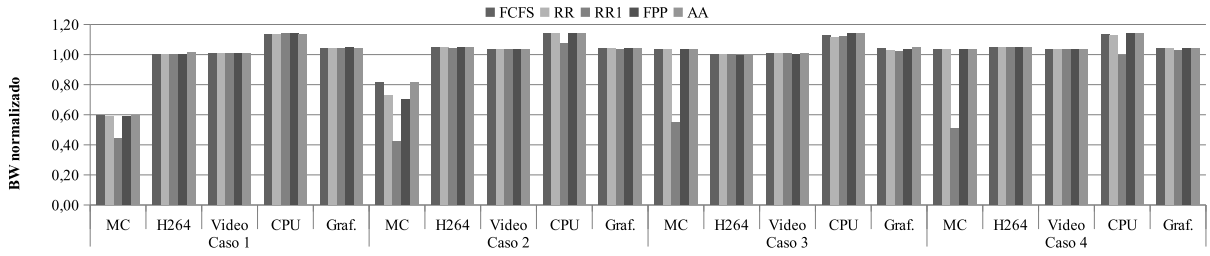
Os tempos de resposta médios medidos a partir da simulação com o árbitro adaptativo para os clientes MC e CPU são, respectivamente: 90 e 85 ns (caso 1), 55 e 55 ns (caso 2), 245 e 60 ns (caso 3) e 195 e 55 ns (caso 4). Tais tempos de resposta médios são inferiores aos prazos solicitados pelos dois clientes. Com exceção do cliente MC no caso 1, a média do tempo de resposta é superior ao intervalo médio dos acessos que é de 80 ns, limitando severamente a largura de banda para este cliente. Isso reflete na redução da largura de banda do cliente MC

Figura 5.6: Piores casos de tempo de resposta e largura de banda para os clientes do STB em 4 casos estudados, comparando implementações com diferentes árbitros.

(a) Comparativo dos piores casos de tempo de resposta.



(b) Comparativo da largura de banda.



Fonte: elaborado pelo autor.

que pode ser vista na Figura 5.6b, que atinge somente 60% da BW solicitada. No caso 2, a largura de banda do cliente MC é um pouco maior do que 80% da solicitada, pois a média do tempo de resposta para este cliente é reduzida para 55 ns, mas o WCRT ainda é maior do que o intervalo médio de acessos solicitado (80 ns). Dessa forma, o cliente MC ainda é penalizado no caso 2, por fazer transações de rajada simples, embora que com os modos de arbitragem AA e FCFS a largura de banda seja melhorada em relação aos árbitros FPP, RR e RR1, atingindo 634 MB/s que correspondem a 74% da largura de banda máxima sustentada.

O cliente CPU não consegue ter o prazo garantido com modos de arbitragem FCFS e RR nos casos 1 e 2, que pode ser garantida usando os árbitros AA e FPP. O árbitro FPP coloca maior prioridade para o cliente CPU em detrimento do desempenho dos demais clientes. Isso pode ser verificado nos casos 3 e 4, onde o cliente MC não consegue atingir o prazo de 500 ns para o modo de arbitragem FPP.

Transações de sequências de rajadas maiores foram avaliadas nos casos 3 e 4, com o cliente MC acessando o canal de memória em sequências de 18 rajadas consecutivas. Em ambos os casos 3 e 4, o prazo estabelecido pelo cliente MC de 500 ns somente é satisfeito com modos de arbitragem AA, FCFS e AA. No entanto, o cliente CPU é prejudicado com modo de arbitragem RR, que não consegue manter o prazo de 290 ns. Somente o árbitro adaptativo consegue manter tal prazo solicitado pela CPU sem prejudicar os demais clientes, mantendo a largura de banda.

A avaliação apresentada nessa subseção mostra a capacidade de adaptação do árbitro proposto nesse trabalho para diferentes configurações do SoC em relação aos acessos gerados pela memória. A adaptação é feita de forma automática, com duração de oito ciclos de relógio (40 ns) devido aos pipelines internos dos estágios de adaptação. Esta adaptação ocorre somente uma vez a cada alteração de algum dos parâmetros de acesso dos clientes. O árbitro adaptativo consegue estabelecer níveis de largura de banda mínimos para os clientes do sistema e manter os prazos de conclusão das transações, dentro dos limites estabelecidos pela forma de acesso aos dados na memória e penalidades impostas pela troca de linhas.

5.6 Resumo do Capítulo

Este Capítulo apresentou a análise dos resultados de implementação e simulação comportamental com precisão de ciclo de relógio para o árbitro adaptativo com o controlador de memória multi-clientes proposto nesse trabalho. Nessa seção verificou-se a validade do modelo de atrasos utilizado para estimar os piores casos de tempo de resposta para o sistema, para um conjunto de 2, 4 e 8 clientes acessando o canal de memória com largura de banda elevadas. Foi apresentado um estudo de caso para avaliação do funcionamento do controle dos tempos de resposta para um sistema composto por 4 clientes acessando o canal de memória em sequências de 16 rajadas consecutivas, com diferentes níveis de largura de banda exigida da memória. Ao final, foi apresentado o estudo de caso de um set-top box de televisão digital para processamento de vídeo em resolução Full-HD. A análise dos resultados apresentados mostra que a principal vantagem de utilização do algoritmo de arbitragem adaptativa baseado no controle dos tempos de resposta traz benefícios para um sistema composto por clientes heterogêneos com variações nas formas de acesso ao canal de memória. O árbitro mostra capacidade de adaptação frente à diferentes formas de acesso e tráfego de dados, utilizando como base as informações de prazo de conclusão e tamanho das transações solicitadas para a memória.

6 CONCLUSÕES E COMENTÁRIOS FINAIS

Esta tese apresenta um método para projeto de sistemas-em-chip orientado a partir de um fluxo de projeto centralizado em memória, que permite a implementação de um sistema de gerenciamento adaptativo dos acessos à memória compartilhada. Na proposta apresentada, o projeto é orientado a partir dos aspectos funcionais de DRAM, através da implementação de um subsistema de memória com gerenciamento adaptativo de prioridades dos acessos. A adaptação é feita a partir da avaliação das características de acesso à memória dos clientes conectados ao sistema, em relação aos prazos de conclusão e tamanho das transações, em tempo de execução do sistema. O subsistema de memória centraliza a comunicação de dados e gerencia os acessos dos diversos clientes do sistema, de forma que a comunicação seja balanceada de acordo com as necessidades de cada aplicação. São apresentadas três contribuições principais: um método de fluxo de projeto de sistemas centralizado em memória, que orienta o projeto para os aspectos funcionais da memória compartilhada; um modelo baseado em atrasos para estimar o pior caso de execução do sistema, quanto aos tempos de resposta e largura de banda mínima alocada por cliente; um árbitro adaptativo para gerenciamento dos acessos à memória externa com garantias de prazos de execução das transações.

Os estudos realizados até o presente momento apontam resultados satisfatórios da comunicação de dados entre os clientes e a memória principal. A estrutura de memória proposta consegue aproveitar da melhor forma as características de acesso em rajadas da memória externa e apresenta um comportamento preditivo em relação às características dos acessos e ao número de clientes. É apresentado um estudo de caso aplicado à Terminal de Acesso de Televisão Digital, compatível com o padrão brasileiro (SBTVD), utilizado para implementar e validar o subsistema de memória proposto. O modelo do subsistema de memória para sistemas multimídia é apresentado nessa tese, baseado na latência usando estimativas de tempo de execução no pior caso. Uma arquitetura de subsistema de memória multi-cliente é proposta em conjunto com o seu modelo baseado em latência. Utilizando a abordagem proposta, um subsistema de memória, pode ser modelado a partir dos parâmetros de temporização da memória DRAM alvo e dos parâmetros dos elementos que formam o subsistema.

Os resultados obtidos até o momento mostram que a classificação automática das transações geradas pelos clientes, implementada através de um algoritmo de arbitragem adaptativa baseado

em agendamento dos acessos, traz benefícios para o sistema de memória. A partir do fluxo de projeto centralizado em memória é possível garantir que os tempos de acesso dos clientes sejam respeitados e, por consequência, a largura de banda seja mantida para cada cliente. Esta tese propõe um algoritmo de referência que implementa o modelo matemático baseado em atrasos, para classificar e ordenar os acessos dos clientes. O modelo proposto é gerado a partir das informações temporais que descrevem o comportamento dos elementos que formam o subsistema de memória. Os acessos são monitorados e classificados de acordo com as características do padrão de transferência: escrita ou de leitura, prazo de conclusão e tamanho da transferência. Os resultados obtidos através de simulação comportamental do circuito implementado mostraram que o desempenho pode ser melhorado utilizando o método proposto para classificação das transações de dados para a DRAM.

A implementação do controlador de memória DDR3 é sintetizável e pode ser implementada em placa de desenvolvimento para fazer a integração de módulos de um SoC com a memória externa. O controlador de memória multi-clientes está implementado com interfaces escaláveis para um número de até 32 clientes, composto por árbitro de acessos, interface com IP de memória externa e o controle de adaptação. Esta arquitetura está sendo utilizada em um SoC real que implementa um set-top box de televisão digital em plataforma FPGA.

6.1 Trabalhos Futuros

A partir dos resultados obtidos até o presente momento, pretende-se desenvolver como trabalhos futuros uma ferramenta de análise comportamental dos limites teóricos do canal de memória em um determinado SoC. O método proposto pode ser usada para guiar projetistas durante a concepção dos sistemas integrados, provendo antecipadamente informações sobre tempos de resposta e largura de banda mínima para os clientes a partir das informações de acesso.

Outra contribuição futura é implementar um monitor de clientes que faça uma avaliação contínua de quantos clientes estão “acordados” acessando o canal de memória e quantos estão apagados. Em SoCs é comum utilizar a técnica de “power-down mode” para desligar módulos que não estão sendo utilizados durante algum período de tempo. A proposta é de que o monitor de clientes calcule uma estimativa do intervalo médio entre os últimos acessos gerados por um cliente e a partir desse valor faça o acompanhamento de uma nova requisição do cliente. Caso o cliente não faça mais pedidos em um intervalo de tempo proporcional à média calculada, ele é considerado como apagado. Dessa forma, o cálculo do WCRT não inclui o cliente que está apagado e torna-se mais realista para a situação presente.

A alocação dinâmica da largura de banda por cliente é uma etapa de projeto do controle do subsistema de memória que ainda não está implementada. O modelo proposto para o subsistema de memória não garante alocação de largura de banda para os clientes. Está previsto para esse modelo a implementação de uma função que calcule a alocação de largura de banda.

Esse cálculo será feito a partir das características de acesso dos clientes, como frequência de acessos e tamanho das transferências. Dessa forma, o algoritmo de escalonamento poderá alocar a quantidade de banda necessária por cliente, de forma dinâmica. O modelo desenvolvido atualmente utiliza como única métrica a latência das transações geradas por cada cliente do subsistema de memória. É importante que o subsistema consiga garantir, além dos requisitos de latência, a largura de banda necessária para cada elemento de processamento. A métrica de largura de banda deve ser implementada no modelo de memória e utilizada no algoritmo de escalonamento das transações.

REFERÊNCIAS

ACCELLERA. **Accellera Systems Initiative.** 2013.

AGRAWAL, S.; DAS, M. Internet of Things – A paradigm shift of future Internet applications. In: ENGINEERING (NUICONE), 2011 NIRMA UNIVERSITY INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.1–7.

AKESSON, B.; GOOSSENS, K. Architectures and modeling of predictable memory controllers for improved system integration. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2011. **Anais...** [S.l.: s.n.], 2011. p.1 –6.

AKESSON, B.; GOOSSENS, K.; RINGHOFER, M. Predator: a predictable sdram memory controller. In: HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS (CODES+ISSS), 2007 5TH IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2007. p.251–256.

AKESSON, B.; STEFFENS, L.; GOOSSENS, K. Efficient Service Allocation in Hardware Using Credit-Controlled Static-Priority Arbitration. In: EMBEDDED AND REAL-TIME COMPUTING SYSTEMS AND APPLICATIONS, 2009. RTCSA '09. 15TH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.59–68.

ANDREWS, J.; BAKER, N. Xbox 360 System Architecture. **IEEE Micro**, Los Alamitos, CA, USA, v.26, n.2, p.25–37, Mar. 2006.

ARM. **CoreLink Memory Controllers.** 2013.

BONATTO, A. C. et al. A 720p H.264/AVC decoder ASIC implementation for digital television set-top boxes. In: INTEGRATED CIRCUITS AND SYSTEM DESIGN, 23., New York, NY, USA. **Proceedings...** ACM, 2010. p.168–173. (SBCCI '10).

BONATTO, A. et al. Towards an Efficient Memory Architecture for Video Decoding Systems. In: COMPUTING SYSTEM ENGINEERING (SBESC), 2012 BRAZILIAN SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2012. p.198–203.

BONATTO, A.; SOARES, A.; SUSIN, A. High Efficiency Reference Frames Storage for H.264/AVC Decoder Hardware Implementation. In: LASCAS 2010, IEEE LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Anais...** [S.l.: s.n.], 2010. p.304–307.

BONATTO, A.; SOARES, A.; SUSIN, A. Multichannel SDRAM controller design for H.264/AVC video decoder. In: PROGRAMMABLE LOGIC (SPL), 2011 VII SOUTHERN CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.137–142.

BORKAR, S.; CHIEN, A. A. The future of microprocessors. **Commun. ACM**, New York, NY, USA, v.54, n.5, p.67–77, May 2011.

BOSSEN, F. et al. HEVC Complexity and Implementation Analysis. **Circuits and Systems for Video Technology, IEEE Transactions on**, [S.l.], v.22, n.12, p.1685–1696, 2012.

CATTHOOR, F.; GREEF, E. d.; SUYTACK, S. **Custom Memory Management Methodology**: exploration of memory organisation for embedded multimedia system design. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

CUMMING, P. **Winning the SoC Revolution**: experiences in real design. [S.l.]: Kluwer Academic Publishers, 2003. p.97 – 119.

CUPPU, V.; JACOB, B. Concurrency, latency, or system overhead: which has the largest impact on uniprocessor dram-system performance? In: COMPUTER ARCHITECTURE, 2001. PROCEEDINGS. 28TH ANNUAL INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2001. p.62–71.

DENNING, P. J. The locality principle. **Commun. ACM**, New York, NY, USA, v.48, n.7, p.19–24, July 2005.

DENNING, P. J.; SCHWARTZ, S. C. Properties of the working-set model. **Commun. ACM**, New York, NY, USA, v.15, n.3, p.191–198, Mar. 1972.

DUTTA, S.; JENSEN, R.; RIECKMANN, A. Viper: a multiprocessor soc for advanced set-top box and digital tv systems. **IEEE Des. Test**, Los Alamitos, CA, USA, v.18, n.5, p.21–31, Sept. 2001.

ESMAEILZADEH, H. et al. Power challenges may end the multicore era. **Commun. ACM**, New York, NY, USA, v.56, n.2, p.93–102, Feb. 2013.

FULLER, S.; MILLETT, L. Computing Performance: game over or next level? **Computer**, [S.l.], v.44, n.1, p.31–38, 2011.

GAJSKI, D. D. et al. **Embedded System Design**. [S.l.]: Springer, 2009.

GAJSKI, D. D.; KUHN, R. H. Guest Editors' Introduction: new vlsi tools. **Computer**, [S.l.], v.16, n.12, p.11–14, 1983.

GAJSKI, D. et al. **SPECC**: specification language and methodology. New York, USA: Springer, 2000.

GOMONY, M. D.; AKESSON, B.; GOOSSENS, K. Architecture and optimal configuration of a real-time multi-channel memory controller. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, San Jose, CA, USA. **Proceedings...** EDA Consortium, 2013. p.1307–1312. (DATE '13).

GOMONY, M. D.; AKESSON, B.; GOOSSENS, K. Coupling TDM NoC and DRAM controller for cost and performance optimization of real-time systems. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION (DATE), 2014. **Anais...** [S.l.: s.n.], 2014. p.1–6.

HAMZAOGLU, F. et al. 13.1 A 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology. In: SOLID-STATE CIRCUITS CONFERENCE DIGEST OF TECHNICAL PAPERS (ISSCC), 2014 IEEE INTERNATIONAL. **Anais...** [S.l.: s.n.], 2014. p.230–231.

HENKEL, J. Closing the SoC design gap. **Computer**, [S.l.], v.36, n.9, p.119–121, 2003.

HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture**: a quantitative approach. 4.ed. San Fransisco, CA, USA: Morgan Kaufmann, 2006.

HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture**: a quantitative approach. 5.ed. San Fransisco, CA, USA: Morgan Kaufmann, 2011.

HENRIKSSON, T. et al. Network Calculus Applied to Verification of Memory Access Performance in SoCs. In: EMBEDDED SYSTEMS FOR REAL-TIME MULTIMEDIA, 2007. ESTIMEDIA 2007. IEEE/ACM/IFIP WORKSHOP ON. **Anais...** [S.l.: s.n.], 2007. p.21 –26.

HOROWITZ, M. et al. H.264/AVC baseline profile decoder complexity analysis. **Circuits and Systems for Video Technology, IEEE Transactions on**, [S.l.], v.13, n.7, p.704–716, 2003.

HUR, I.; LIN, C. Adaptive History-Based Memory Schedulers. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 37., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2004. p.343–354. (MICRO 37).

HYNIX. **4Gb DDR4 SDRAM**. [S.l.]: Hynix, 2013.

INTEL. **Mobile 4th Generation Intel Core Family - Datasheet volume 1 of 2**. [S.l.]: Intel, 2014. Disponível em: < http://ark.intel.com/products/76642 >. Acesso em: abr. 2014.

ITRS. **International technology roadmap for semiconductors 2011 edition**: design. [S.l.]: Semiconductor Industry Association, 2011. Disponível em: <http://www.itrs.net/Links/2011ITRS/Home2011.htm>. Acesso em: 10 ago. 2013.

ITRS. **International technology roadmap for semiconductors 2011 edition**: system drivers. [S.l.]: Semiconductor Industry Association, 2011. Disponível em: <http://www.itrs.net/Links/2011ITRS/Home2011.htm>. Acesso em: 10 ago. 2013.

ITU-T. **Recommendation H.264 – Advanced video coding for generic audiovisual services**. [S.l.]: ITU-T – International Telecommunication Union, 2003.

ITU-T. **SERIES H**: audiovisual and multimedia systems: infrastructure of audiovisual services - coding of moving video - h.265. [S.l.]: ITU-T – International Telecommunication Union, 2013.

JACOB, B.; NG, S.; WANG, D. **Memory Systems**: cache, dram, disk. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

JEDEC. **JEDEC-JESD229**: wide I/O single data rate. Virginia, USA: JEDEC Solid State Technology Association, 2011.

JEONG, M. K. et al. A QoS-aware memory controller for dynamically balancing GPU and CPU bandwidth use in an MPSoC. In: DESIGN AUTOMATION CONFERENCE (DAC), 2012 49TH ACM/EDAC/IEEE. **Anais...** [S.l.: s.n.], 2012. p.850–855.

KAHLE, J. A. et al. Introduction to the Cell multiprocessor. **IBM J. Res. Dev.**, Riverton, NJ, USA, v.49, n.4/5, p.589–604, July 2005.

KASERIDIS, D.; STUECHELI, J.; JOHN, L. K. Minimalist Open-page: a dram page-mode scheduling policy for the many-core era. In: ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 44., New York, NY, USA. **Proceedings...** ACM, 2011. p.24–35. (MICRO-44).

KEATING, M. **The Simple Art of SoC Design**. [S.l.]: Springer New York, 2011.

KEATING, M.; BRICAUD, P. **Reuse methodology manual**: for system-on-a-chip designs. 2.ed. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

KIM, J.-S. et al. A 1.2V 12.8GB/s 2Gb mobile Wide-I/O DRAM with 4x128 I/Os using TSV-based stacking. In: SOLID-STATE CIRCUITS CONFERENCE DIGEST OF TECHNICAL PAPERS (ISSCC), 2011 IEEE INTERNATIONAL. **Anais...** [S.l.: s.n.], 2011. p.496–498.

KIM, Y. et al. ATLAS: a scalable and high-performance scheduling algorithm for multiple memory controllers. In: HIGH PERFORMANCE COMPUTER ARCHITECTURE (HPCA), 2010 IEEE 16TH INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.1–12.

KIM, Y. et al. Thread Cluster Memory Scheduling: exploiting differences in memory access behavior. In: MICROARCHITECTURE (MICRO), 2010 43RD ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.65–76.

KOLLIG, P.; OSBORNE, C.; HENRIKSSON, T. Heterogeneous multi-core platform for consumer multimedia applications. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION, 2009. DATE '09. **Anais...** [S.l.: s.n.], 2009. p.1254–1259.

KURD, N. et al. 5.9 Haswell: a family of ia 22nm processors. In: SOLID-STATE CIRCUITS CONFERENCE DIGEST OF TECHNICAL PAPERS (ISSCC), 2014 IEEE INTERNATIONAL. **Anais...** [S.l.: s.n.], 2014. p.112–113.

LEE, D. et al. Tiered-latency DRAM: a low latency and low cost dram architecture. In: IEEE 19TH INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTER ARCHITECTURE (HPCA), 2013., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2013. p.615–626. (HPCA '13).

LEE, K.-B.; CHANG, T.-S. **Essential Issues in SOC Design**. [S.l.]: Springer Netherlands, 2006. p.73–118.

LIU, J. et al. RAIDR: retention-aware intelligent dram refresh. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 39., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p.1–12. (ISCA '12).

MENG, J.; KAWAKAMI, K.; COSKUN, A. Optimizing energy efficiency of 3-D multicore systems with stacked DRAM under power and thermal constraints. In: DESIGN AUTOMATION CONFERENCE (DAC), 2012 49TH ACM/EDAC/IEEE. **Anais...** [S.l.: s.n.], 2012. p.648–655.

MICRON. **DDR2 SDRAM**: 2gb: x4, x8, x16 ddr2 sdram features. [S.l.]: Micron Technology, Inc, 2007.

MICRON. **DDR3 SDRAM**: 4gb: x4, x8, x16 ddr3 sdram features. [S.l.]: Micron Technology, Inc, 2009.

MICRON. **TwinDie 1.2V DDR4 SDRAM**: 8gb: x4, x8 twindie ddr4 sdram. [S.l.]: Micron Technology, Inc, 2013.

MOORE, G. Cramming More Components Onto Integrated Circuits. **Proceedings of the IEEE**, [S.l.], v.86, n.1, p.82–85, 1998.

MOORE, G. Cramming More Components Onto Integrated Circuits. **Proceedings of the IEEE**, [S.l.], v.86, n.1, p.82–85, 1998.

MOSCIBRODA, T.; MUTLU, O. Memory Performance Attacks: denial of memory service in multi-core systems. In: USENIX SECURITY SYMPOSIUM ON USENIX SECURITY SYMPOSIUM, 16., Berkeley, CA, USA. **Proceedings...** USENIX Association, 2007. p.18:1–18:18. (SS'07).

MUTLU, O. Memory scaling: a systems architecture perspective. In: MEMORY WORKSHOP (IMW), 2013 5TH IEEE INTERNATIONAL. **Anais...** [S.l.: s.n.], 2013. p.21–25.

MUTLU, O.; MOSCIBRODA, T. Parallelism-Aware Batch Scheduling: enhancing both performance and fairness of shared dram systems. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 35., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008. p.63–74. (ISCA '08).

MUTLU, O.; MOSCIBRODA, T. Parallelism-Aware Batch Scheduling: enabling high-performance and fair shared memory controllers. **Micro, IEEE**, [S.l.], v.29, n.1, p.22–32, Jan 2009.

NEGREIROS, M. et al. Towards a video processing architecture for SBTVD. In: PROGRAMMABLE LOGIC (SPL), 2012 VIII SOUTHERN CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.1–6.

NESBIT, K. J. et al. Fair Queuing Memory Systems. In: ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 39., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.208–222. (MICRO 39).

NEWMAN, L. H. Piz Daint Supercomputer Shows the Way Ahead on Efficiency. **IEEE Spectrum**, [S.l.], Jan. 2014.

PAGANINI, M. Nomadik: a mobile multimedia application processor platform. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 2007., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.749–750. (ASP-DAC '07).

PAOLIERI, M. et al. An Analyzable Memory Controller for Hard Real-Time CMPs. **Embedded Systems Letters, IEEE**, [S.l.], v.1, n.4, p.86–90, Dec 2009.

PASRICHA, S.; DUTT, N. **On-Chip Communication Architectures**: system on chip interconnect. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

RIXNER, S. et al. Memory Access Scheduling. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 27., New York, NY, USA. **Proceedings...** ACM, 2000. p.128–138. (ISCA '00).

ROGERS, B. M. et al. Scaling the bandwidth wall: challenges in and avenues for cmp scaling. In: COMPUTER ARCHITECTURE, 36., New York, NY, USA. **Proceedings...** ACM, 2009. p.371–382. (ISCA '09).

SALEH, R. et al. System-on-Chip: reuse and integration. **Proceedings of the IEEE**, [S.l.], v.94, n.6, p.1050–1069, 2006.

SEICULESCU, C. et al. A DRAM Centric NoC Architecture and Topology Design Approach. In: VLSI (ISVLSI), 2011 IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2011. p.54–59.

SESHADRI, V. et al. RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In: MICRO 2013. **Anais...** [S.l.: s.n.], 2013. p.1–13.

SHAH, H.; KNOLL, A.; AKESSON, B. Bounding SDRAM interference: detailed analysis vs. latency-rate analysis. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, San Jose, CA, USA. **Proceedings...** EDA Consortium, 2013. p.308–313. (DATE '13).

SHAH, H.; RAABE, A.; KNOLL, A. Bounding WCET of applications using SDRAM with Priority Based Budget Scheduling in MPSoCs. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2012. **Anais...** [S.l.: s.n.], 2012. p.665–670.

SHAO, J.; DAVIS, B. A Burst Scheduling Access Reordering Mechanism. In: HIGH PERFORMANCE COMPUTER ARCHITECTURE, 2007. HPCA 2007. IEEE 13TH INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2007. p.285–294.

SMITH, A. Internal Scheduling and Memory Contention. **Software Engineering, IEEE Transactions on**, [S.l.], v.SE-7, n.1, p.135–146, Jan 1981.

SOARES, A. B.; BONATTO, A. C. a.; SUSIN, A. A. Integration issues on the development of an h.264/AVC video decoder SoC for SBTVD set top box. In: INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 24., New York, NY, USA. **Proceedings...** ACM, 2011. p.125–130. (SBCCI '11).

SOARES, A. B.; BONATTO, A. C.; SUSIN, A. A. Development of a SoC for Digital Television Set-Top Box: architecture and system integration issues. **International Journal of Reconfigurable Computing**, [S.l.], v.2013, n.1, Jan. 2013.

SON, Y. H. et al. Reducing memory access latency with asymmetric DRAM bank organizations. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.41, n.3, p.380–391, June 2013.

STILIADIS, D.; VARMA, A. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. **IEEE/ACM Trans. Netw.**, Piscataway, NJ, USA, v.6, n.5, p.611–624, Oct. 1998.

SULLIVAN, G. et al. Overview of the High Efficiency Video Coding (HEVC) Standard. **Circuits and Systems for Video Technology, IEEE Transactions on**, [S.l.], v.22, n.12, p.1649–1668, 2012.

SUZIM, A. A. **Sistemas Digitais**: uma visão integrada do processo de síntese. Porto Alegre, RS, Brasil: Universidade Federal do Rio Grande do Sul, 1989.

SUZIM, A. Data Processing Section for Microprocessor-Like Integrated Circuits. **Solid-State Circuits, IEEE Journal of**, [S.l.], v.16, n.3, p.233–235, 1981.

SZE, V.; BUDAGAVI, M. **Design and Implementation of Next Generation Video Coding Systems (H.265/HEVC Tutorial)**. [S.l.]: Tutorial apresentado no ISCAS 2014, 2014.

THIELE, L.; WILHELM, R. Design for Timing Predictability. **Real-Time Syst.**, Norwell, MA, USA, v.28, n.2-3, p.157–177, Nov. 2004.

TSAI, S.-F. et al. A 1062Mpixels/s 8192x4320p High Efficiency Video Coding (H.265) encoder chip. In: VLSI CIRCUITS (VLSIC), 2013 SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2013. p.C188–C189.

UNIQUIFY. **Uniquify – DDR Memory Subsystem IP**. [S.l.]: Uniquify, 2014. Disponível em: < http://www.uniquify.com/wp-content/uploads/2014/05/UNQ_Datasheet_DDRRevolutionRev_Web.pdf >. Acesso em: 20 abr. 2014.

VINK, J.; BERKEL, K. van; WOLF, P. van der. Performance Analysis of SoC Architectures Based on Latency-Rate Servers. In: DESIGN, AUTOMATION AND TEST IN EUROPE, 2008. DATE '08. **Anais...** [S.l.: s.n.], 2008. p.200 –205.

WEISER, M. The computer for the 21st century. **SIGMOBILE Mob. Comput. Commun. Rev.**, New York, NY, USA, v.3, n.3, p.3–11, July 1999.

WILHELM, R. et al. Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, [S.l.], v.28, n.7, p.966–978, July 2009.

WILKES, M. V. The memory gap and the future of high performance memories. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.29, n.1, p.2–7, Mar. 2001.

WOLF, P. van der; GEUZEBROEK, J. SoC infrastructures for predictable system integration. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2011. **Anais...** [S.l.: s.n.], 2011. p.1 –6.

WOLF, P. van der; HENRIKSSON, T. Video Processing Requirements on SoC Infrastructures. In: DESIGN, AUTOMATION AND TEST IN EUROPE, 2008. DATE '08. **Anais...** [S.l.: s.n.], 2008. p.1124 –1125.

WOLF, W. The future of multiprocessor systems-on-chips. In: DESIGN AUTOMATION CONFERENCE, 41., New York, NY, USA. **Proceedings...** ACM, 2004. p.681–685. (DAC '04).

WOO, D. H. et al. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In: HIGH PERFORMANCE COMPUTER ARCHITECTURE (HPCA), 2010 IEEE 16TH INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.1–12.

WULF, W. A.; MCKEE, S. A. Hitting the memory wall: implications of the obvious. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.23, n.1, p.20–24, Mar. 1995.

XIE, Y. Future memory and interconnect technologies. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2013. **Anais...** [S.l.: s.n.], 2013. p.964–969.

XILINX. **LogiCORE IP Multi-Port Memory Controller (v6.06.a)**. [S.l.]: Xilinx Inc., 2012.

XILINX. **Memory Interface Generator.** 2014.

XILINX. **Virtex-6 FPGA ML605 Evaluation Kit**. [S.l.]: Xilinx Inc., 2014.

ZHANG, G. et al. A Laxity-Aware Memory Access Scheduler for High Performance Multimedia SoC. In: COMPUTER AND INFORMATION TECHNOLOGY (CIT), 2011 IEEE 11TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.603–608.