UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ALEXSANDRO CRISTOVÃO BONATTO

NÚCLEOS DE INTERFACE DE MEMÓRIA DDR SDRAM PARA SISTEMAS-EM-CHIP

Porto Alegre
2009

ALEXSANDRO CRISTOVÃO BONATTO

NÚCLEOS DE INTERFACE DE MEMÓRIA DDR SDRAM PARA SISTEMAS-EM-CHIP

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Automação e Instrumentação Eletro-Eletrônica

ORIENTADOR: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre
2009

ALEXSANDRO CRISTOVÃO BONATTO

NÚCLEOS DE INTERFACE DE MEMÓRIA DDR SDRAM PARA SISTEMAS-EM-CHIP

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Altamiro Amadeu Susin, UFRGS-DELET
Doutor pela INPG – Grenoble, França

Banca Examinadora:

Prof. Dr. Sergio Bampi, UFRGS
Doutor pela Stanford University – Stanford, Estados Unidos

Prof. Dr. Alexandre Balbinot, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Cesar Albenes Zeferino, UNIVALI
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Coordenador do PPGEE: _____
Prof. Dr. Arturo Suman Bretas

Porto Alegre, março de 2009.

AGRADECIMENTOS

Primeiramente, meus agradecimentos aos que me apoiaram diretamente na realização desse trabalho: ao Prof. Altamiro Susin pelo apoio constante e por sempre acreditar nas minhas idéias e no meu potencial de trabalho e; ao Prof. André Borin pelo seu apoio com o desenvolvimento do projeto, a escrita dos artigos e com a constante troca de idéias e de informações.

Em segundo lugar, é prioritário agradecer àqueles que participaram indiretamente na realização desse trabalho, os colegas de laboratório: Negreiros, Letícia, Márlon, Fábio e Vinícius. Também gostaria de agradecer aos professores da pós-graduação, que compartilharam seus conhecimentos entre salas de aulas e conversas nos corredores: Luigi, Luba, Gilson e Schuck.

Em especial, gostaria de agradecer à Gabriela pelo seu carinho e companheirismo durante esses dois anos de mestrado.

RESUMO

Dispositivos integrados de sistemas-em-chip (SoC), especialmente aqueles dedicados às aplicações multimídia, processam grandes quantidades de dados armazenados em memórias. O desempenho das portas de memória afeta diretamente no desempenho do sistema. A melhor utilização do espaço de armazenamento de dados e a redução do custo e do consumo de potência dos sistemas eletrônicos encorajam o desenvolvimento de arquiteturas eficientes para controladores de memória. Essa melhoria deve ser alcançada tanto para interfaces com memórias internas quanto externas ao chip. Em sistemas de processamento de vídeo, por exemplo, memórias de grande capacidade são necessárias para armazenar vários quadros de imagem enquanto que os algoritmos de compressão fazem a busca por redundâncias. No caso de sistemas implementados em tecnologia FPGA é possível utilizar os blocos de memória disponíveis internamente ao FPGA, os quais são limitados a poucos mega-bytes de dados. Para aumentar a capacidade de armazenamento de dados é necessário usar elementos de memória externa e um núcleo de propriedade intelectual (IP) de controlador de memória é necessário. Contudo, seu desenvolvimento é uma tarefa muito complexa e nem sempre é possível utilizar uma solução "sob demanda". O uso de FPGAs para prototipar sistemas permite ao desenvolvedor integrar módulos rapidamente. Nesse caso, a verificação do projeto é uma questão importante a ser considerada no desenvolvimento de um sistema complexo. Controladores de memória de alta velocidade são extremamente sensíveis aos atrasos de propagação da lógica e do roteamento. A síntese a partir de uma descrição em linguagem de hardware (HDL) necessita da verificação de sua compatibilidade com as especificações de temporização pré-determinadas. Como solução para esse problema, é apresentado nesse trabalho um IP do controlador de memória DDR SDRAM com função de BIST (*Built-In Self-Test*) integrada, onde o teste de memória é utilizado para verificar o funcionamento correto do controlador.

Palavras-chave: Controlador de Memória, *Double Data Rate* SDRAM, *firm*-IPs, Sistemas-em-Chip, FPGA, Teste de Memória.

ABSTRACT

Many integrated Systems-on-Chip (SoC) devices, specially those dedicated to multimedia applications, process large amounts of data stored on memories. The performance of the memories ports directly affects the performance of the system. Optimization of the usage of data storage and reduction of cost and power consumption of the electronic systems encourage the development of efficient architectures for memory controllers. This improvement must be reached either for embedded or external memories. In systems for video processing, for example, large memory arrays are needed to store several video frames while compression algorithms search for redundancies. In the case of FPGA system implementation, it is possible to use memory blocks available inside FPGA, but for only a few megabytes of data. To increase data storage capacity it is necessary to use external memory devices and a memory controller intellectual property (IP) core is required. Nevertheless, its development is a very complex task and it is not always possible to have a custom solution. Using FPGA for system prototyping allows the developer to perform rapid integration of modules to exercise a hardware version. In this case, test is an important issue to be considered in a complex system design. High speed memory controllers are very sensitive to gate and routing delays and the synthesis from a hardware description language (HDL) needs to be verified to comply with predefined timing specifications. To overcome these problems, a DDR SDRAM controller IP was developed which integrate the BIST (*Built-In Self-Test*) function, where the memory test is used to check the correct functioning of the DDR controller.

Keywords: Memory controller, double data rate SDRAM, firm-IPs, system-on-a-chip, FPGA, memory test.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ASIC	*Application Specific Integrated Circuit*
ASIP	*Application Specific Instruction Set Processor*
BIST	*Built-In Self-Test*
BISV	*Built-In Self-Verification*
BL	*Burst Length*
CAS	*Column-Address Strobe*
CI	Circuito Integrado
CL	*CAS Latency*
CLB	*Configurable Logic Block*
CPLD	*Complex Programmable Logic Device*
CUV	*Circuit-Under-Verification*
DCM	*Digital Clock Manager*
DDR	*Double Data Rate*
DIMM	*Dual In-line Memory Module*
DLL	*Delay-Locked Loop*
DRAM	*Dynamic Random Access Memory*
DSP	*Digital Signal Processing*
EDA	*Electronic Design Automation*
ES	Entrada e Saída
FIFO	*First-In first-Out*
FPGA	*Field Programmable Gate Array*
GDSII	*Graphic Design Station II*
HDL	*Hardware Description Language*
HW	*Hardware*
IP	*Intellectual Property*
IOB	*Input-Output Block*

ITRS	*International Technology Roadmap for Semiconductors*
JEDEC	*Joint Electron Device Engineering Council*
JTAG	*Joint Test Action Group*
LFSR	*Linear Feedback Shift Register*
LUT	*Look-Up Table*
MEF	Máquina de Estados Finitos
MIG	*Memory Interface Generator*
MOS	*Metal-Oxide Semiconductor*
NoC	*Network on Chip*
PLD	*Programmable Logic Device*
RAM	*Random Access Memory*
RAS	*Row-Address Strobe*
ROM	*Read Only Memory*
RTL	*Register-Transfer Level*
SBTVD	Sistema Brasileiro de Televisão Digital
SDF	*Standard Delay Format*
SDR	*Single Data Rate*
SDRAM	*Synchronous Dynamic RAM*
SIA	*Semiconductor Industry Association*
SRAM	*Static Random Access Memory*
SoC	*System-on-a-Chip*
STA	*Static Timing Analysis*
SW	*Software*
TVD	Televisão Digital
UCF	*User Constraints File*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VIP	*Verification IP*
VLSI	*Very Large Scale Integration*
VSIA	*Virtual Socket Interface Alliance*

LISTA DE SÍMBOLOS

F_{BUS}	Frequência do barramento de dados
t_{AC}	Tempo de acesso aos dados
t_{DQSS}	Tempo entre dado e amostrador
t_{RPRE}	Primeiro ciclo de transição do amostrador
t_{WTR}	Número mínimo de ciclos entre comandos de escrita e leitura
t_{DQS}	Tempo de atraso do sinal de amostragem
$t_p data$	Tempo de propagação entre a saída e chegada ao controlador
$t_p L$	Tempo de propagação das trilhas da placa
$t_p R$	Tempo de propagação do roteamento entre controlador e memória
t_{DQSQ}	Variação entre dados e amostrador
D_n	Bit de dado no instante atual
D_{n+1}	Bit de dado no próximo ciclo de relógio
t_{CK}	Intervalo do relógio
t_{CH}	Intervalo de ciclo alto do relógio
t_{CL}	Intervalo de ciclo baixo do relógio
t_{SD}	Intervalo de propagação da linha de atraso ideal
t_{DV}	Intervalo de janela de dado válido
t_{SU}	Intervalo de incerteza do amostrador
t_S	Tempo de *set* do registrador
t_H	Tempo de *hold* do registrador
t_{PHL}	Tempo de transição de alto para baixo
t_{PLH}	Tempo de transição de baixo para alto
t_{LUT4}	Tempo de propagação do sinal pela LUT4
t_{NET}	Tempo de propagação do sinal pela conexão
t_{DQ}	Atraso de propagação de uma trilha de dado
t_{DQS}	Atraso de propagação de uma trilha de amostragem

t_{DQSd}	Atraso de propagação de uma trilha de amostragem após a linha de atraso
t_{DQSdS}	Atraso de propagação de uma trilha de amostragem de subida atrasada
t_{DQSdD}	Atraso de propagação de uma trilha de amostragem de descida atrasada
t_{RCD}	Intervalo entre ativar uma linha e realizar uma escrita ou leitura
t_{WTR}	Intervalo mínimo entre uma escrita e leitura
t_{WR}	Intervalo mínimo entre uma leitura e desativar uma linha
$t_{SD'}$	Intervalo t_{SD} considerando as variações da placa, memória e FPGA
t_{PCI}	Variação da placa sobre os sinais de dado e amostragem
t_{QH}	Tempo de estabilidade do bit de dado após a transição do amostrador
t_{DQm}	Variação média do tempo de propagação dos sinais de dado
t_{DQSm}	Variação média do tempo de propagação dos sinais de amostragem
$\sum_{i=0}^{7}$	Somatório dos termos de índice i
t_{DQimax}	Maior tempo de propagação do sinal de dado de índice i
$t_{DQSimax}$	Maior tempo de propagação do sinal de amostragem de índice i
t_{DQimin}	Menor tempo de propagação do sinal de dado de índice i
$t_{DQSimin}$	Menor tempo de propagação do sinal de amostragem de índice i
$t_{SD''}$	Intervalo $t_{SD'}$ considerando as variações internas no FPGA
$\Uparrow$	Ordem crescente de endereçamento
$\Downarrow$	Ordem decrescente de endereçamento
$\Updownarrow$	Ordem crescente ou decrescente de endereçamento
r	Operação de leitura na memória
w	Operação de escrita na memória
y	Padrão de dados do teste de memória
$\bar{y}$	Complemento do padrão de dados do teste de memória
t_w	Número de ciclos para uma operação de escrita na memória
t_r	Número de ciclos para uma operação de leitura na memória

SUMÁRIO

1 INTRODUÇÃO

O segmento industrial ligado à cadeia da Eletrônica atinge marcas de importância social e econômica que só os setores de infra-estrutura e energia conseguiam. A Eletrônica abrange desde os componentes semicondutores até os equipamentos eletrônicos e as aplicações em sistemas de Informática, Comunicação e Tecnologia da Informação. Com o nível atual de interação que existe entre os sistemas eletrônicos e a vida cotidiana, é difícil imaginar o mundo atual sem a eletrônica. A base da eletrônica atual é a microeletrônica e o mercado de semicondutores é um dos segmentos mais sensíveis da economia mundial. Também é um setor estratégico para qualquer nação que deseja estar tecnologicamente integrada no concerto das nações. O projeto de sistemas que possuem eletrônica embarcada é um desafio crescente. Atualmente, a complexidade dos circuitos semicondutores duplica a cada 24 meses (INTEL, 2005), considerando a complexidade e a quantidade de transistores num único chip.

Os sistemas de processamento de alto desempenho permitem o desenvolvimento de aplicações que tratam grandes quantidades de dados em tempo real, como multimídia interativa. Tais sistemas podem realizar, por exemplo, a compressão e/ou a descompressão de vídeos de alta qualidade que exige altas taxas de processamento. Uma memória externa de dados e de programa é, em geral, necessária. A velocidade de acesso às memórias de programa e de dados passa a ser o principal gargalo de desempenho do sistema. O desenvolvimento de técnicas de otimização do acesso à memória é extremamente relevante. São utilizadas técnicas de acesso em rajada, transferência nas duas bordas do relógio, uso de um ou mais níveis de memória *cache*, etc.

Esse estudo versa sobre o projeto de um controlador de memória tipo *Double Data Rate* (DDR) Synchronous DRAM (SDRAM), amplamente utilizada em sistemas digitais de computação e de processamento dedicado. As memórias externas ao chip têm um custo inferior, pois são fabricadas em grandes quantidades e com processo de fabricação sintonizado para maior rendimento. No caso em foco neste trabalho (memórias DDR) o ônus para o projetista do sistema digital é a utilização de um controlador para implementar o protocolo específico destas memórias. As restrições temporais entre os sinais de um controlador de memória DDR são bastante restritas: os atrasos de roteamento devem

ser controlados para assegurar a correta operação do circuito. Há todo o interesse em sintetizar o controlador DDR junto com o sistema digital: é o que será apresentado nesta dissertação de mestrado.

1.1 PROJETO DE SISTEMAS-EM-CHIP

O termo *sistema-em-chip* (do inglês *System-on-a-Chip* ou SoC) refere-se à integração de diversos componentes de um sistema eletrônico num único circuito integrado. Tal sistema pode 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 processadores, 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 (barramento de alta velocidade e baixa velocidade) ou uma rede-em-chip (NoC).

A evolução das técnicas de fabricação de circuitos em silício foi o que possibilitou atingir um nível de integração tão elevado, como para fabricar um SoC. A tecnologia utilizada para construir circuitos digitais sofre uma evolução constante desde as últimas quatro décadas. Até os anos 60, os circuitos lógicos eram construídos a partir de elementos discretos, como transistores e resistores, conectados por fios. Com o advento do circuito integrado, um número maior de transistores pode ser integrado sobre uma única pastilha de silício. Em 1972 a Intel lança o processador 4040, usado em calculadoras e outros dispositivos eletrônicos. Foi possível integrar sobre uma única pastilha todos os elementos necessários para produzir um microprocessador. A partir disso, os circuitos integrados vêm tendo uma evolução constante, como prevista por Gordon E. Moore[1] em seu artigo de 1965 (MOORE, 1965). Essa tendência dos semicondutores é conhecida como *Lei de Moore* que continua prevalecendo até os dias atuais: onde o número de transistores por chip aumenta de forma exponencial. Atualmente, os chips podem conter mais de dois bilhões de transistores, como em alguns microprocessadores de alto desempenho.

O avanço crescente nos processos de fabricação e o aumento da quantidade de transistores por circuito integrado permitem o avanço tecnológico em direção à maior integração de diferentes partes no mesmo substrato. Por consequência, a maior complexidade da tecnologia, que disponibiliza mais recursos de hardware, faz com que o projeto de circuitos integrados seja uma tarefa mais trabalhosa. O preço do aumento de escalas de integração, de desempenho, de funcionalidade e a redução de consumo de energia é a maior comple-

[1] Gordon E. Moore é um dos co-fundadores da Intel

xidade do projeto. A maior complexidade demanda, além de um tempo maior de projeto, o aumento de custeio em recursos humanos (homens/hora) e equipamentos (ferramentas de hardware e software). A Figura 1.1 reporta a crescente diferença entre a complexidade dos chips e a produtividade de projeto de hardware, reportada na última previsão do *International Technology Roadmap for Semiconductors* (ITRS) no relatório (ITRS, 2007). A lacuna de produtividade de hardware representa a diferença que existe entre o número de transistores disponível num chip e a produtividade de projeto. A produtividade de hardware é indicada pelo número de portas geradas pelo projetista durante o dia. Com o uso de módulos de hardware pré-projetados e 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. Também é ilustrada a evolução da quantidade de software necessário para operar o hardware gerado.

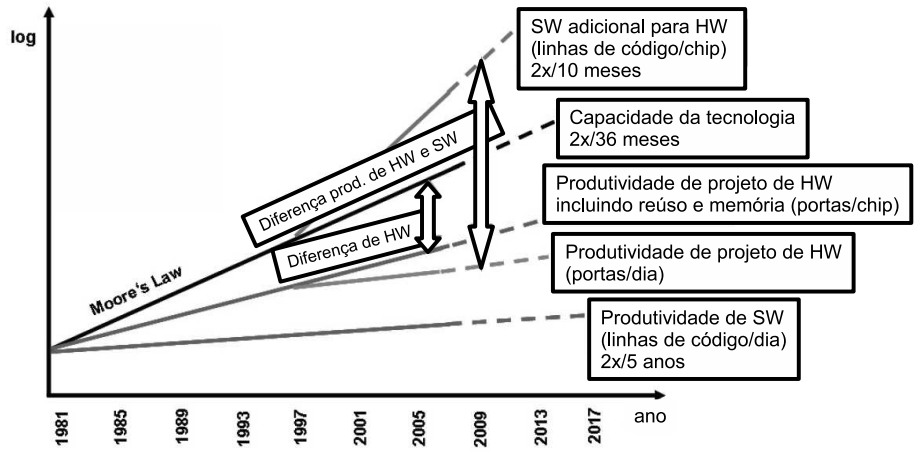

Figura 1.1: Diferença de produtividade para hardware e software. Fonte: (ITRS, 2007).

O aumento da quantidade de engenheiros de projeto por equipe pode reduzir a diferença de produtividade, tanto de hardware como de software. Entretanto, não deve ser esquecido que grandes equipes necessitam de um grande esforço de coordenação de pessoas, atividades e tarefas. Além disso, todo novo projeto necessita de um estudo da especificação do sistema a ser projetado e de suas subpartes, especificação das ferramentas utilizadas, da estratégia de verificação, etc. O reúso de módulos de hardware pré-projetados e a padronização de interfaces entre os módulos são formas de aliviar esse déficit de produtividade. Mesmo com o preenchimento de espaço disponível através do reúso e de memórias em chip, a diferença de produtividade tende a crescer segundo a previsão do ITRS.

Como o projeto de sistemas integrados é orientado para o hardware, tais estudos apontam que a diferença de produtividade somente será superada se for modificada a metodologia de projeto. Atualmente, circuitos complexos são projetados a partir de um alto nível de abstração como transferência de registradores e máquinas de estado. Níveis mais altos

de descrição estão em desenvolvimento e o salto de produtividade somente será vencido com projetos em nível de sistema (*system-level design*) realizados inteiramente com suporte de ferramentas de *Electronic Design Automation* (EDA). O projeto em nível de sistema deve ser feito a partir de uma descrição comportamental eficiente com uso de núcleos de HW pré-projetados e SW de alto nível parametrizáveis. Com a capacidade atual de integração de transistores no silício podem ser fabricados sistemas extremamente complexos. O desafio para explorar totalmente esse potencial é enorme, pois a produtividade de projeto deve ser aumentada em cerca de 50 vezes (ITRS, 2007).

Um sistema integrado numa pastilha de silício é desenvolvido de forma hierárquica, com subpartes ou módulos projetados separadamente. O termo núcleo (do inglês *core*) é utilizado para denominar as subpartes constituintes do sistema. O projeto de SoCs é baseado na integração de núcleos, pré-projetados ou projetados para o sistema em desenvolvimento. Esses núcleos podem ser projetados para executar uma tarefa específica para esse sistema ou para tarefas geralmente necessárias em sistemas integrados. Quando o núcleo é projetado como um produto para ser reutilizado (chamado de entregável, do inglês *deliverable*) ele é denominado de propriedade intelectual ou IP e é um bem que pode ser comercializado. Um IP é um módulo de hardware pronto para ser integrado ao sistema e que atende aos padrões de interface de comunicação, ambiente de verificação e contém uma documentação completa. O projeto de IPs é conveniente para reduzir o tempo gasto com projeto de módulos de HW frequentemente utilizados nos sistemas. Os IPs são projetados e verificados previamente, antes da sua integração no sistema. Sistemas digitais complexos são gerados a partir da integração dos núcleos interligados por sistema de comunicação interno ao chip.

O projeto de um SoC a partir de IPs segue um fluxo de projeto específico. Se o sistema contém processadores, em paralelo com projeto do hardware deve-se projetar o software e as funções utilizadas pelo processador do sistema para comunicar e controlar os módulos (*drivers*). Como a plataforma de hardware pode ser gerada a partir de IPs desenvolvidos previamente e de módulos novos de hardware que são projetados especificamente para o sistema, o desenvolvimento de um SoC pode ser visto como um fluxo triplo de tarefas concorrentes, como um *3-codesign*, formado pelo desenvolvimento de hardware, de IPs e de software. Os IPs selecionados para integrar o sistema podem ser descritos usando diferentes níveis de abstração, desde uma descrição comportamental até o desenho dos transistores e das interconexões. O desenvolvimento é realizado em etapas, onde as ferramentas utilizadas interpretam uma descrição de entrada gerando um produto para a próxima etapa de projeto, até chegar ao produto final. Para permitir a verificação do sistema, é importante que estejam disponíveis descrições dos IPs em diversos níveis. As ferramentas utilizadas para a verificação das etapas de projeto, a simulação do hardware e do software, a compilação do código de entrada e a síntese física do hardware constituem o ambiente de *co-design* (ou projeto conjunto de hardware e software).

A atividade de produção de circuitos integrados é um processo muito complexo que envolve atividades desde a especificação e o projeto detalhado das máscaras (também chamada "fase de projeto"), até a fabricação, encapsulamento e teste. Vários métodos foram desenvolvidos para diminuir o tempo e o custo do processo. Alguns métodos procuram reduzir o tempo de projeto enquanto que outros procuram acelerar a fabricação pela redução do número de etapas. Para acelerar o projeto são utilizados sistemas de síntese automática que geram o leiaute a partir de descrições de nível mais alto. Essas ferramentas podem receber como entrada desde uma descrição de células a serem justapostas até uma descrição em alto nível. A qualidade do circuito gerado depende da ferramenta. Ferramentas de síntese automática em geral não conseguem gerar circuitos que aproveitem todo o potencial dos processos de fabricação. Costuma-se classificar os circuitos integrados como *full custom* quando todas as etapas de projeto e processo são executadas. Quando o projeto é feito com células de uma biblioteca (*standard cells*), é classificado como *semi-custom*. Diz-se que é feita síntese automática quando a especificação de entrada é feita em alto nível, como as linguagens de descrição de hardware. Para acelerar a fase de fabricação são utilizados os circuitos pré-fabricados, onde as etapas mais complexas e caras são executadas gerando uma matriz de transistores chamados de *gate arrays*. Nesse circuito é processada somente a camada de metal sobre a matriz de transistores determinando a interconexão entre eles. Já os dispositivos de lógica programável (*programmable logic*) são formados por uma matriz de células lógicas programáveis e interconexões também programáveis, chamada *field programable gate arrays* ou FPGA ou também *programmable logic*. Em todos esses casos há sempre um balanço a fazer entre custo do hardware, tempo de projeto, número de circuitos a serem produzidos e desempenho necessário ou desejado. Uma discussão mais aprofundada sobre outras formas de lógica programável, como microprogramação ou utilização de microprocessadores fogem ao escopo do presente trabalho.

Ainda são utilizados circuitos eletrônicos de diferentes tipos e tecnologias de fabricação, implementados com poucos transistores e que realizam funções lógicas amplamente utilizadas, chamados *circuitos de catálogo*. Um sistema eletrônico formado por circuitos de catálogo utiliza uma placa de circuito impresso para as interconexões. O projetista escolhe os circuitos adequados para realizar uma certa tarefa e projeta as interconexões a serem feitas pelo circuito impresso. Esses sistemas são cada vez menos utilizados, pois seu custo pode torná-los inviáveis economicamente.

Dispositivos lógicos programáveis são amplamente utilizados para o desenvolvimento e prototipação de módulos para sistemas digitais. O projeto de um sistema em chip pode ser realizado utilizando dispositivos de lógica programável de grande capacidade. Esses são formados por uma matriz de elementos lógicos programáveis interconectados através de chaves programáveis. O resultado é um circuito lógico que realiza uma função programada pelo desenvolvedor. Tais circuitos são chamados de *Programmable Logic Devices*

(PLDs) e podem ser programados diversas vezes. A principal vantagem de usar tais circuitos é a possibilidade e modificar o projeto, reprogramando o dispositivo. Circuitos de lógica programável existem de diferentes tamanhos e, atualmente, são classificados em duas grandes categorias: os CPLDs e os FPGAs.

Uma premissa comum no desenvolvimento de sistemas digitais baseados em núcleos é que novos projetos iniciam em uma das linguagens de descrição de hardware utilizadas popularmente (GUPTA; ZORIAN, 1997): VHDL ou Verilog. As ferramentas de síntese do circuito digital utilizam como entrada a descrição, em nível de transferência entre registradores, gerando o arquivo de programação de dispositivos FPGA e CPLD. Esses dispositivos implementam funções lógicas combinacionais, circuitos sequenciais, tabelas (lógica combinacional) e memórias a partir de um arranjo de elementos base conectados a uma matriz programável.

Mesmo sendo amplamente utilizados no projeto de sistemas digitais, os dispositivos programáveis possuem algumas desvantagens se comparado com ASICs, como menor desempenho e maior consumo de potência. O uso de chaves para interconectar os elementos lógicos reduz o desempenho do sistema. Para grandes quantidades produzidas a partir de um mesmo projeto, os ASICS têm um custo menor.

1.2 MEMÓRIA EM SISTEMAS DIGITAIS

Sistemas de processamento de dados de alto desempenho necessitam de uma larga banda de acesso à memória. A evolução dos sistemas digitais tende a aplicações cada vez mais complexas computacionalmente e que utilizam grandes quantidades de memória. Em sistemas embarcados a capacidade de memória tende a ser um limitante e o acesso aos dados armazenados na memória do sistema é o gargalo para o 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 do contexto atual. 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 no contexto atual, 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. Segundo (HENNESSY; PATTERSON, 2006), o custo de uma memória produzida com tecnologia SRAM é cerca de oito a dezesseis vezes maior do que memórias de tecnologia DRAM. No entanto, as SRAM são cerca de oito a dezesseis vezes mais rápidas que DRAMs. Já em termos de capacidade de armazenamento, DRAMs podem ser construídas com quatro a oito vezes a capacidade de uma SRAM. O tema pode atingir grande complexidade quando aplicado a sistemas com múltiplos processadores, vários níveis de *cache*, memórias de massa como discos magnéticos, etc. No presente trabalho o foco é um sistema embarcado com dois ou três níveis de memória onde os processos têm acesso a espaços exclusivos de memória e não há necessidade de controle de coerência de *cache*.

A criação de hierarquia de memórias para sistemas de processamento ganhou importância com a evolução dos processadores. A Figura 1.2 ilustra uma projeção histórica do desempenho de processadores comparado ao tempo gasto no acesso à memória principal. Mesmo com a redução do tempo de acesso à memória principal, a evolução da capacidade de processamento dos processadores gera uma diferença de desempenho. A velocidade de acesso ao dado armazenado em memória (considerando uma hierarquia de memória) 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 (PATTERSON et al., 1997; HENNESSY; PATTERSON, 2006).

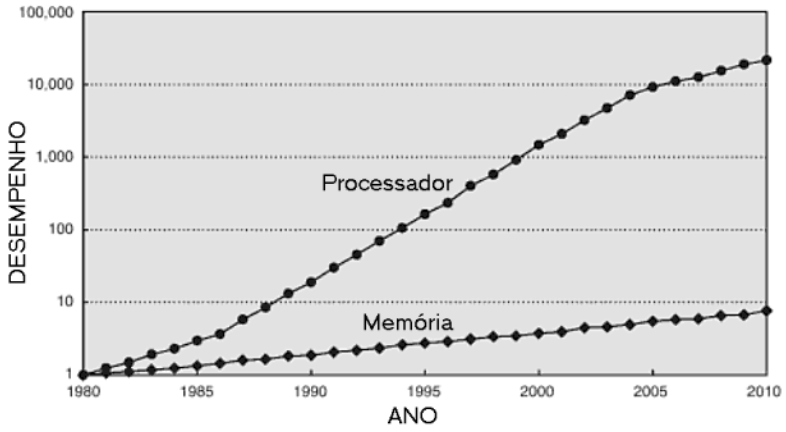

Figura 1.2: Diferença de velocidade entre processador e memória. Fonte: (HENNESSY; PATTERSON, 2006).

Essa diferença é conhecida como *Processor-Memory Performance Gap*. O resultado disso é o aumento do tempo de acesso aos dados armazenado em memória, do ponto de vista do processador, que deve esperar mais ciclos para seguir o processamento. É necessário o uso de uma hierarquia de memória para explorar a localidade de processamento

da informação. Assim, uma parte dos dados usados pelo processador está armazenada na memória mais próxima, com velocidade maior de acesso. 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*; etc.

Para aumentar a eficiência de processamento de dados em memória, sistemas de decodificação de vídeo necessitam de uma hierarquia de memória e de otimização dos algoritmos. O objetivo da hierarquia de memória é reduzir o número de acessos à memória externa. Liu apresenta (LIU; LEE, 2008) uma hierarquia de memória de três níveis (registradores, memória local SRAM e memória externa DRAM), otimizada para um sistema de processamento de vídeo com processador central. Esse trabalho mostra que a manutenção dos pixeis vizinhos aos pixeis processados reduz o número de acessos à memória externa, melhorando o desempenho do sistema.

Kim apresenta (KIM; SUNWOO, 2008) a arquitetura de um ASIP utilizado no processamento em tempo real de vídeo no padrão H.264/AVC. Esse processador possui um conjunto de instruções específicas para computação intensiva das partes de vídeo, reduzindo o número de acessos à memória externa se comparado a processadores DSP.

Processadores de alto desempenho fabricados atualmente vêm com o controlador de memória embutido no mesmo encapsulamento (*on-die memory controller*). Um exemplo disso é o processador Intel Core i7 que possui um controlador de memória integrado com três canais DDR3 de 1066 MHz e 64-bits, com capacidade máxima de 24 GBytes de endereçamento, atingindo uma taxa de 26,5 Gbit/s de acesso aos dados.

O uso de memórias DDR SDRAM se beneficia do aumento da largura de banda e da velocidade de acesso aos dados. Essas memórias têm grande capacidade de armazenamento com baixo custo, se comparadas às SRAM. Em contrapartida, a implementação do controlador de memória é uma tarefa de grande complexidade e seu uso em FPGAs requer diversos cuidados na implementação da interface de acesso aos dados. Os sinais de interface com a memória DDR SDRAM operam em taxas elevadas de relógio. As gerações de memórias DDR, como DDR2 e DDR3, têm explorado o aumento da frequência de acesso aos dados, mantendo a largura de barramento de dados.

Quando o SoC é projetado a partir de um FPGA, a integração de memórias de grande capacidade pode ter um custo elevado. A Figura 1.3 relaciona o custo de capacidade de armazenamento em relação à tecnologia utilizada, se FPGAs (nesse caso do fabricante

Xilinx) ou elementos de memória (do fabricante Micron)[2].

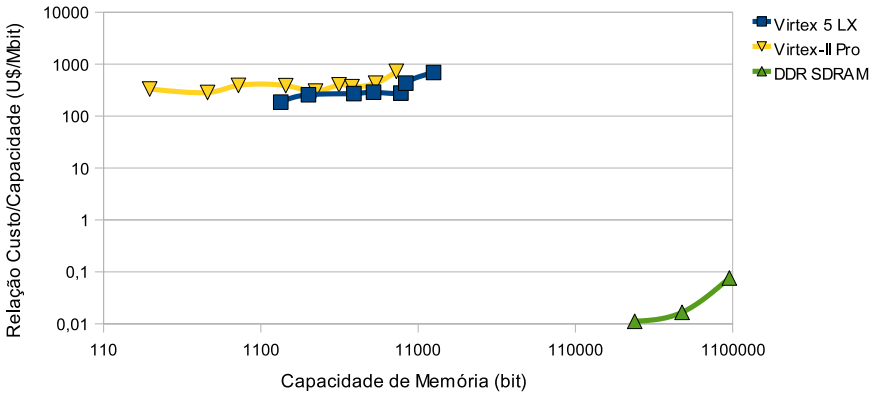

Figura 1.3: Comparativo entre a capacidade de memória e seu custo por mega-byte.

Memórias DDR SDRAM têm custo baixo e grande capacidade de armazenamento de dados se comparadas às tecnologias FPGA. O custo de um elemento de memória é menor que o de um circuito programável, que não foi projetado e otimizado para atender um único fim. Além disso, a implementação de memórias usando tecnologia FPGA utiliza diversos recursos adicionais, como as interconexões comutáveis e elementos de lógica combinacional, que não são otimizados para esse fim. Alguns FPGAs contêm blocos de RAM internos ao chip. Entretanto, circuitos de memória produzidos em larga escala têm custo mais reduzido.

1.3 DESENVOLVIMENTO DE NÚCLEOS DE IP

Os circuitos integrados complexos são decompostos em partes para serem mais facilmente projetados ou distribuídos para as diversas equipes de desenvolvimento. Algumas destas partes podem ser, por sua vez, muito complexas. Se a função é genérica e essas partes ou módulos forem úteis para outros circuitos, há interesse em projetá-las de forma que possam ser reutilizadas. Evidentemente incide um esforço adicional para implementar um circuito para diversas aplicações: um parâmetro adicional, documentação mais detalhada, etc. O projeto para reúso consolidou-se em atividade específica na área de microeletrônica produzindo bens de valor comercial chamados Propriedade Intelectual (do termo em inglês *Intellectual Property* ou IP): é uma parte de hardware (analógico ou digital) ou de software projetado com a finalidade de ser reutilizado. Um IP é formado por uma descrição sintetizável de funcionamento encapsulado num módulo (ou objeto no caso de SW), realizando uma tarefa específica de forma autônoma. Os módulos de hard-

[2]Os valores de preço para componentes de memória e dispositivos FPGA foram obtidos da página do fornecedor Avnet (http://www.avnet.com) em agosto de 2008. Os totais apresentados de capacidade de armazenamento em memória para os FPGAs é calculado considerando tanto as BlockRAMs internas quanto as RAMs distribuídas.

ware na literatura podem ser denominados pelos termos *módulo*, *núcleo* ou *bloco*. Nesse trabalho será utilizado o termo *núcleo*.

Existem duas metodologias de projeto de módulos de hardware: o projeto para uso e o projeto para reúso. Durante o projeto para uso, o módulo é bem documentado, deve possuir um código bem comentado, deve ser gerado e verificado usando um ambiente de EDA conhecido e deve possuir *scritps* para síntese e implementação. Quando o projeto do módulo é orientado para reúso, gerando um IP, são necessárias algumas etapas adicionais de desenvolvimento (KEATING; BRICAUD, 1999):

- Deve ser projetado para resolver diferentes problemas: durante a especificação do IP, o projetista deve prever diferentes funcionalidades para que o IP atenda a diferentes sistemas. Em resumo, deve ser parametrizável;

- Deve ser projetado para diferentes tecnologias alvo: o projeto de um IP que possa ser utilizado tanto em ASICs quanto em FPGAs deve prever o uso de diferentes ferramentas de EDA. A fim de evitar problemas durante o porte para diferentes tecnologias, devem ser evitadas construções de linguagem específicas de uma ferramenta. Da mesma forma, o uso de elementos instanciados em bibliotecas deve ser limitado, pois pode dificultar a sua implementação em outra tecnologia ou com outra ferramenta;

- Diversificação de ferramentas de simulação: os *scripts* de simulação do IP devem ser gerados para diferentes simuladores e permitir diferentes situações de verificação;

- Verificação em hardware independente da plataforma: o IP projetado deve conter um conjunto completo de núcleos de teste do tipo *stand-alone*. Permitindo a verificação do núcleo de forma independente da plataforma utilizada;

- Verificação em alto nível de fiabilidade: o ambiente de teste do IP deve prever situações reais de funcionamento para verificação do sistema. Dessa forma são evitadas situações de teste viciado;

- Documentação completa: a documentação deve conter em detalhe as restrições de uso do núcleo, a relação de todas aplicações e configurações possíveis e as etapas de teste do IP.

As etapas adicionais mencionadas acima aumentam o tempo do projeto do núcleo, principalmente pelo maior detalhamento de verificação do seu funcionamento. O tempo de um projeto para uso é menor do que para reúso. Quando o núcleo é reutilizado em outro projeto ou por outra equipe de projeto, se ganha tempo de desenvolvimento e de teste.

Durante o andamento de um projeto, sob pressão de cumprir prazos e com eventuais problemas de projeto, engenheiros nem sempre cumprem todas etapas de documentação e verificação. Para guiar o andamento de projeto de um IP, os itens mencionados anteriormente são adicionados ao fluxo de projeto.

Segundo afirma Keating no Manual de Metodologia de Reúso (KEATING; BRICAUD, 1999), para vencer o desafio de projetar SoCs complexos, equipes de projeto mudaram o fluxo de projeto de duas formas importantes: (1) misturar metodologias de projeto *top-down* e *botton-up* e; (2) utilizar modelo *espiral* de projeto ao contrário do modelo *waterfall*. No modelo de projeto water-fall, as etapas são consecutivas, bem definidas e não repetidas. Após cada etapa finalizada, não há possibilidade de retorno para modificações em etapas anteriores. Já no modelo *espiral* é feito o particionamento de hardware e de software e após a especificação do sistema, as tarefas de projeto são executadas em paralelo. Tal modelo prevê a execução concorrente de tarefas, como projeto e verificação, por mais de uma equipe de projeto. É utilizado para projetos grandes e prevê a inclusão de núcleos pré-projetados.

O modelo clássico de projeto *top-down* pode ser visto como uma rotina que inicia com a especificação do núcleo e termina com a verificação do funcionamento. As etapas importantes são: (1) escrita e especificação completas para o sistema e sub-sistemas projetados; (2) refinamento da arquitetura e dos algoritmos incluindo o projeto de software e co-simulação; (3) decomposição da arquitetura em sub-módulos bem definidos; (4) projeto dos sub-módulos ou seleção de núcleos de IP pré-projetados; (5) integração dos sub-módulos em nível de topo, verificando seu funcionamento e; (6) verificação dos resultados do núcleo para atender às especificações requeridas de temporização, área e potência.

No entanto, essa metodologia assume que todos os sub-módulos que compõem o sistema são realizáveis. Isso nem sempre acontece e pode ser necessário retornar o processo de desenvolvimento. Para vencer a exigência de tempo de projeto, utiliza-se o modelo *botton-up* associado ao *top-down*, onde os núcleos complexos de nível mais baixo são projetados inicialmente, em paralelo à execução do projeto.

Quando o núcleo projetado para ser disponibilizado para outros projetos ou equipes como um produto final, ele é denominado de entregável. Os núcleos podem ser classificados como *hard*, *soft* ou *firm*, dependendo do nível de flexibilidade e otimização para uma determinada tecnologia ou processo de fabricação:

- Núcleo tipo *Soft* – são núcleos gerados a partir de uma descrição RTL sintetizável, com maior flexibilidade de implementação. Não possuem nenhum tipo de restrição de roteamento (*routing*) e/ou posicionamento (*placement*) de portas e tampouco qualquer dependência de tecnologia. Em contrapartida, é difícil prever algumas características importantes como potência, área e desempenho já que dependem da tecnologia utilizada e da ferramenta de síntese;

- Núcleo tipo *Hard* – etapas adicionais de posicionamento dos elementos, roteamento das conexões e leiaute geram um núcleo tipo *hard*. São núcleos de hardware projetados para implementação utilizando uma tecnologia específica e otimizadas para desempenho, área e potência. Os núcleos entregáveis são fornecidos no formato de leiaute, tipo GDSII. Núcleos hard apresentam desempenho melhor que núcleos soft, geralmente ocupando um número menor de recursos;

- Núcleo tipo *Firm* – são otimizados para ter melhor funcionamento em uma tecnologia. São entregues no formato de código RTL contendo informações adicionais do projeto físico para uma plataforma alvo. São mais flexíveis e portáveis que as hard e possuem melhor desempenho, menor área e menor consumo se comparados com os soft. Os núcleos do tipo *firm* foram definidos pela VSIA como uma forma intermediária de circuito entregável, entre os *hard* e os *soft*.

A portabilidade do núcleo é uma questão importante quando se planeja fazer uma conversão de projeto entre diferentes tecnologias de FPGA ou entre FPGA e ASIC. Dois fatores importantes interferem na portabilidade e reusabilidade: a independência de detalhes de processo durante a sua implementação e; o formato de dados em que o núcleo é entregue para utilização. O núcleo pode conter elementos internos utilizados somente para uma tecnologia específica, dificultando a sua adaptação ou alteração. Isso torna necessário uma nova etapa de projeto, aumentando o tempo de integração do sistema. A melhor forma de evitar a recorrência de engenharia é incluir no projeto do SoC o maior número possível de núcleos do tipo soft. Mas nem sempre é possível implementar em hardware um núcleo soft, principalmente quando os quesitos de temporização são estritos ou os recursos são limitados. Núcleos projetados para FPGAs, que possuam restrições de posicionamento ou de roteamento, são projetados como *firm*.

1.4 CONTRIBUIÇÃO DA DISSERTAÇÃO DE MESTRADO

O objetivo dessa dissertação de mestrado é o projeto e implementação de um controlador de memória DDR SDRAM para sistemas-em-chip prototipados em tecnologia FPGA, com características de reúso e de portabilidade. O projeto do controlador DDR objetiva a criação de um núcleo de hardware entregável, um IP do tipo *firm*. Esse trabalho está inserido no projeto de pesquisa e desenvolvimento de um sistema de decodificação de vídeo digital no padrão H.264/AVC (conhecido como MPEG-4 Parte 10) (ITU-T, 2005), utilizado no Sistema Brasileiro de Televisão Digital (ABNT, 2007). Tal sistema de processamento de vídeo necessita de uma interface de memória externa para armazenar grandes quantidades de dados, como imagens de referência em alta definição. Memórias DDR e DDR2 são amplamente utilizadas em placas de prototipação com circuitos programáveis. Por possuírem características de baixo custo e elevadas taxas de transferência de dados,

são preferidas para armazenamento de informações durante etapas de processamento.

O decodificador de vídeo pode ser visto como um SoC, formado por núcleos de processamento com funcionamento bem definido, com implementação alvo para tecnologias FPGA. Cada núcleo é projetado de forma concorrente como descrito na Dissertação de Mestrado (STAEHLER, 2006). Para a implementação e validação do decodificador de vídeo em FPGA são utilizados dispositivos programáveis. O uso de memória externa ao chip faz-se necessário, visto que os quesitos de memória do sistema são superiores à capacidade de armazenamento dos dispositivos programáveis. O controlador de memória foi projetado para implementação com o decodificador de vídeo, como um IP para ser reutilizado. Esse tipo de núcleo é classificado como *firm*, por possuir restrições de temporização e de posicionamento de elementos lógicos no FPGA. Interfaces de memória do tipo DDR SDRAM possuem uma interface de ES com a memória e uma parte de controle. A interface de ES possui temporização crítica sendo portanto implementadas como núcleos do tipo *hard* ou *firm*. Já a parte de controle é geralmente implementada como um núcleo *soft*. É importante verificar a compatibilidade de funcionamento entre as duas partes. Isolar os elementos dependentes da tecnologia num único sub-módulo é uma forma eficiente de minimizar o esforço de engenharia recorrente. Uma descrição comportamental desse tipo de núcleo não é suficiente para garantir o seu funcionamento correto. É necessário haver restrições de implementação para que os dados sejam transferidos corretamente entre a interface física do controlador e a memória DDR SDRAM. Entende-se por nível físico o sub-módulo que contém os elementos de interface de ES que conectam o controlador à memória.

A implementação de um controlador de memória DDR SDRAM em dispositivos FPGA requer um projeto da interface de ES que utilize recursos próprios do dispositivo. Devido às restrições críticas de temporização e outras restrições de funcionamento, a implementação desse núcleo em FPGAs de diferentes famílias e/ou fabricantes é restrita. O projeto de um controlador de memória externa deve objetivar o desenvolvimento de um núcleo genérico, que possa ser usado em diferentes sistemas. Como as interfaces de memória são padronizadas, seu projeto deve ser orientado para facilitar a integração em diferentes plataformas FPGA, de diversos fabricantes ou de diversas famílias de dispositivos. Nesse trabalho é explorado o projeto para reúso do IP de controlador de memória com funcionalidade de BIST integrado. A verificação do funcionamento do IP utilizando uma placa ou dispositivo FPGA real é parte importante do projeto do SoC com memória externa.

Durante a realização desse trabalho, foram produzidos três artigos apresentados em *workshops* internacionais: um sobre a validação da implementação do controlador de memória em FPGA (BONATTO; SOARES; SUSIN, 2008a); o segundo sobre uma metodologia de teste de memória DDR externa utilizando o controlador em-chip (BONATTO; SOARES; SUSIN, 2008b) e; o terceiro sobre o projeto de um IP do controlador de me-

mória para reúso (BONATTO; SOARES; SUSIN, 2008c). Também foi apresentado um artigo fórum de estudantes regional sobre o projeto do IP de controlador de memória (BONATTO; SOARES; SUSIN, 2008d). Uma técnica de teste de memória DDR SDRAM utilizando algoritmos March com rajadas de dados foi apresentada num simpósio internacional (SOARES; BONATTO; SUSIN, 2008).

O projeto da interface de memória inicia na Seção 2 com o estudo das tecnologias de memória DDR SDRAM e seu funcionamento. Em seguida, na Seção 3, é feito o estudo de funcionamento do controlador DDR e sua implementação em HDL. O projeto do controlador DDR orientado para reúso é detalhado da Seção 4. Na seção 5 é descrita a validação do controlador de memória num circuito real. As conclusões são apresentadas na Seção 6.

2 TECNOLOGIA DE MEMÓRIAS DDR SDRAM

Nesse capítulo é introduzida a tecnologia de memórias síncronas DRAM de dupla amostragem por ciclo de relógio. Ao longo do texto é apresentada a descrição da arquitetura interna e algumas características importantes de seu funcionamento. Primeiramente será feita uma revisão da tecnologia atual das memórias encontradas à venda no mercado e na sequência será feita uma descrição da arquitetura e do funcionamento das memórias do tipo DDR SDRAM, alvo da implementação desse trabalho. As informações apresentadas aqui, abrangendo aspectos evolutivos e construtivos das memórias, foram retiradas dos manuais e artigos técnicos como: (VOLLRATH, 2003), (MICRON, 2003), (MICRON, 2006) e (MOSAID, 2006).

Por convenção, será usada a seguinte terminologia no decorrer do texto: *célula de memória* é o menor elemento de armazenamento da memória DRAM, formado por um transistor e um capacitor; *banco de memória* é o arranjo de células de memória organizado na forma de conjuntos que podem ser acessadas dentro de uma determinada faixa de endereços; *elemento de memória* é o circuito integrado encapsulado que contém a matriz de células de memória como núcleo de armazenamento, um circuito de controle e um circuito de interface de dado, endereço e comando.

As memórias DRAM síncronas de taxa simples de transferência de dados introduziram a arquitetura de agendamento de dados para sincronização com o barramento externo. Sua arquitetura baseaia-se na separação entre o núcleo de memória DRAM e o circuito de sincronização, que faz a interface com o barramento externo. Essa arquitetura permite que o núcleo de memória funcione à uma taxa de relógio diferente da interface de dados. O circuito de sincronização faz a conexão entre o núcleo e a interface externa. A partir disso foi criada a arquitetura de dupla amostragem de dados por ciclo de relógio, tecnologia denominada de DDR SDRAM. A evolução das memórias DDR SDRAM pode ser vista na Figura 2.1, retirada do relatório de análise de mercado (SAMSUNG, 2007). Esse gráfico mostra a evolução de produção de memórias para computadores pessoais. Atualmente, o carro-chefe de vendas de memórias para PCs são as DDR2, mas segundo essa previsão a partir de 2010 as DDR3 dominarão o mercado de vendas.

Atualmente, memórias DDR SDRAM representam cerca de 2% do total de vendas

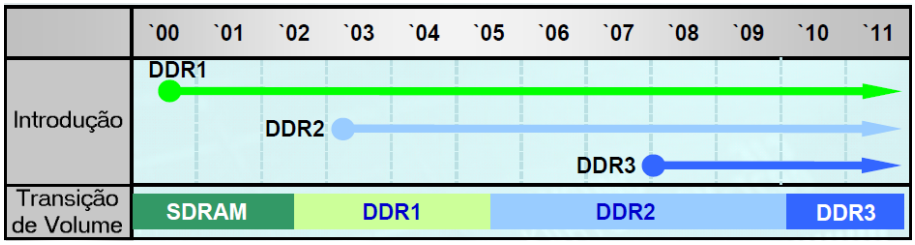

Figura 2.1: Evolução das memórias DDR SDRAM.

para o mercado de computadores pessoais sendo que a tecnologia sucessora (DDR2) é mais vendida atualmente. Fabricantes de controladores de memória DDR estão projetando controladores mistos DDR/DDR2. Poucas alterações de projeto são necessárias, como é discutido por (SHUANG-YAN et al., 2005). Nesse trabalho são apontadas as diferenças construtivas dos controladores para as duas memórias e alterações de projeto necessárias para implementar o controlador DDR/DDR2.

O fato de duplicar a taxa de transferência de dados enquanto o núcleo de memória continua operando na mesma frequência tem importantes implicações sobre o custo da memória. Aumentar a frequência de operação do núcleo de memória implica em novos processos de fabricação, o que eleva o custo da memória. Além disso, o projeto das interfaces de comando, de endereço e de dado é mais complexo. A melhor solução encontrada, do ponto de vista econômico, para aumentar a velocidade de acesso aos dados é de aumentar a largura da memória em relação ao barramento de dado. Dessa forma, o núcleo de memória pode operar à uma taxa de frequência menor que a taxa de acesso ao dado pelo circuito externo à memória. A geração de memórias DDR SDRAM possui o núcleo interno de memória com largura da palavra de dado igual ao dobro da largura do barramento externo. No outro extremo das possibilidades de evolução está a memória DDR3 (terceira geração DDR) que possui um núcleo com 8 vezes o tamanho da interface de dados. Por razões de custo, essa evolução tem seguido pequenos passos em direção ao aumento de velocidade e largura de núcleo de memória. Quando uma meta de desempenho é alcançada, esbarrando num limite da tecnologia, a indústria cria uma nova geração de memórias DDR.

2.1 GERAÇÕES DE MEMÓRIAS DRAM

As memórias de tecnologia SDRAM foram as precursoras dos elementos de memória com altas taxas de transferência de dados, atingindo 133 MHz na interface de comunicação com o controlador. Memórias desse tipo possuem o núcleo de memória, chamado de *Memory Cell Array*, rodando à mesma frequência que o barramento de interface de dados. A velocidade de transferência de dados é atrelada à velocidade do núcleo de memória DRAM. Percebeu-se então a dificuldade em aumentar a frequência de operação do núcleo de memória. Portanto, para aumentar a velocidade de transferência de dados sem mexer

no núcleo de memória, projetou-se uma arquitetura de memória que isolasse o núcleo de memória do barramento de dados.

A evolução das SDRAM veio com as memórias *Double Data Rate* (DDR) SDRAM, que introduziu a arquitetura *2n-prefetch*[1] de dupla transferência de dados por ciclo de relógio. Além do aumento da taxa de relógio da interface dessas memórias, a dupla taxa de transferência de dados possibilitou elevar ainda mais a velocidade de acesso aos dados armazenados. Outra melhoria tecnológica advinda do uso de memórias DDR foi a alteração da tensão de alimentação, reduzida de 3,3V nas SDRAM para 2,5V.

A nova geração de memórias veio com a criação das DDR2 e DDR3 SDRAM, que trouxeram algumas modificações construtivas como aumento da taxa de relógio da interface e aumento do número de palavras transferidas por ciclo de relógio. A última geração dessas memórias veio com a DDR3 SDRAM, com arquitetura *8n-prefetch*, que permitiu o aumento da taxa de transferência de dados para 1600 Mbps. Para isso, algumas modificações tecnológicas de interface elétrica foram realizadas como a redução da tensão de 1,8V para 1,5V, reduzindo em aproximadamente 30% o consumo de potência se comparada com as DDR2 (MICRON, 2006). Outra característica das DDR3 é o aumento da capacidade de armazenamento de cada elemento, chegando à 8 Gbit por circuito integrado. A seguir são apresentadas as principais características das três gerações de memória DDR.

2.1.1 Double Data Rate (DDR) SDRAM

Algumas características importantes das memórias DDR são (JEDEC, 2003):

- Taxa de transferência de dados variando entre 200 Mbps e 400 Mbps;
- Interface elétrica de IO do tipo SSTL-2.5V;
- Rajada (*burst*) de dados de tamanhos 2, 4 ou 8;
- Arquitetura *2n-prefetch* com latência de CAS igual a 2, 2,5 ou 3 ciclos de relógio;
- Capacidade máxima de 1 Gbit por circuito integrado;
- Disponível com encapsulamentos TSOP de 66 pinos e BGPA de 60 esferas;
- Sinal de amostragem de terminação simples, não contínuo e bidirecional.

2.1.2 Double Data Rate (DDR2) SDRAM

Memórias DDR2 operam com dupla taxa de transferência de dados e com o dobro da frequência de barramento em relação às DDR. O núcleo de memória DRAM opera à

[1] O termo *prefetch* refere-se à busca de palavras da memória e seu arranjo em fila para entregar ao barramento de dados. $2n$ representa que são buscadas do núcleo de memória duas palavras de n-bits por ciclo de relógio do núcleo.

mesma frequência que nas memórias DDR. Memórias DDR2 são definidas nas normas JESD79-2E e JESD208 (JEDEC, 2008, 2007a).

Algumas características importantes das memórias DDR2 são:

- Taxa de transferência de dados variando entre 400 Mbps e 1066 Mbps;
- Interface elétrica de IO do tipo SSTL-1.8V;
- Rajada de dados de tamanhos 4 ou 8;
- Arquitetura *4n-prefetch* com latência de CAS de 3, 4 ou 5 ciclos de relógio;
- Capacidade máxima de 4 Gbit por circuito integrado;
- Disponível com encapsulamentos TSOP de 66 pinos e BGPA de 60 esferas;
- Sinal de amostragem de terminação simples ou diferencial, não contínuo e bidirecional.

2.1.3 Double Data Rate (DDR3) SDRAM

Algumas características importantes das memórias DDR3 são (JEDEC, 2007b):

- Taxa de transferência de dados variando 800 Mbps e 1600 Mbps;
- Interface elétrica de IO do tipo SSTL-1.5V;
- Rajadas de dados de tamanho 8;
- Arquitetura *8n-prefetch* com latência mínima de 5 ciclos de relógio;
- Capacidade máxima de 8 Gbit por circuito integrado;
- Sinal de amostragem diferencial, não contínuo e bidirecional;
- Topologia DIMM tipo *fly-by* (Ilustrada na Figura 2.2);
- Contém reset global assíncrono da memória.

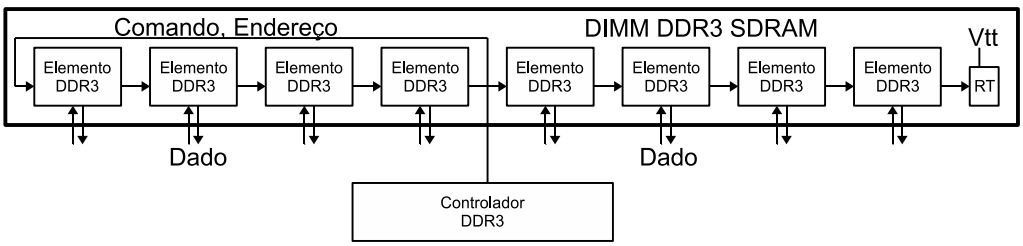

Figura 2.2: Diagrama de blocos simplificado de um pente DIMM DDR3 SDRAM.

O princípio de funcionamento de taxa de dupla amostragem por ciclo de relógio continuou sendo usado nas memórias DDR2 e DDR3, juntamente com o aumento da velocidade do barramento de dados. Na última geração, a topologia de circuitos DIMM do tipo *fly-by* permitiu que os sinais de controle sejam conectados serialmente à cada um dos elementos de memória do módulo. O aumento da capacidade dos elementos de memória DDR3 e a redução da tensão de alimentação estão diretamente relacionados à evolução da tecnologia.

2.2 DESCRIÇÃO DAS MEMÓRIAS DDR SDRAM

As memórias *Double Data Rate* SDRAM representam um salto tecnológico em relação às antecessoras *Single Data Rate* SDRAM. A transferência dos dados ocorre nas bordas de subida e de descida do relógio de sincronização da memória. A taxa de transferência de dados é duplicada sem modificar a taxa de relógio do núcleo de memória. Uma interface de memória operando à taxa de relógio de 100 MHz atinge uma taxa de transferência de dados de 200 Mbps por pino de dado. Larguras típicas da interface de dado de elementos de memória DDR SDRAM são 4, 8 ou 16-bits. O padrão de interface e de funcionamento é definido na norma JESD79C (JEDEC, 2003). O grupo JEDEC é responsável pela padronização de engenharia de semicondutores da EIA (*Electronic Industries Alliance*), responsável por estabelecer a norma de padronização das SDRAM. A norma é dividida em duas partes: (1) a primeira é referente ao circuito de memória e (2) a segunda aos módulos de memória. Os módulos são compostos geralmente de 8 elementos (ou 16 no caso de memórias *dual-rank*) integrados de memória em cada face do módulo DIMM, como ilustra a Figura 2.3).

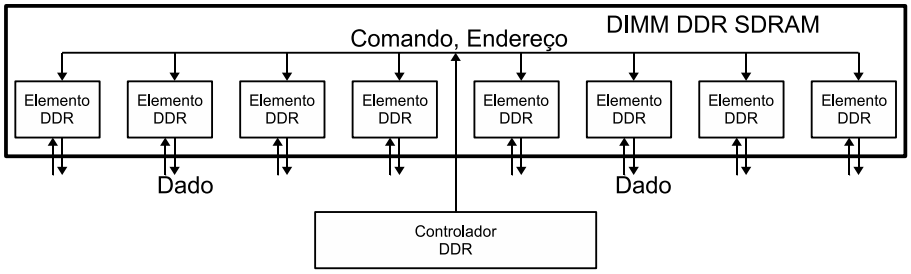

Figura 2.3: Diagrama de blocos simplificado de um módulo DIMM DDR SDRAM.

Os 8 elementos de memória em paralelo formam um barramento de dados de largura total de 64-bits. A taxa máxima de transferência de dados para módulos de memória é dada por (2.1) (em Mbps), onde F_{BUS} é a frequência do barramento de dados (em MHz).

$$F_{BUS} \cdot 2 \cdot 64/8 \tag{2.1}$$

Para F_{BUS}=100 Mhz, a taxa de transferência de dados é de 1600 MBps. A Tabela 2.1

mostra um comparativo de desempenho e de aspectos funcionais dos diferentes módulos DDR SDRAM encontrados no mercado.

Tabela 2.1: Comparativo entre módulos DIMM de memória DDR SDRAM.

Tipo de Memória	Relógio do Barramento	Taxa de transferência por segundo	Nome do Módulo	Taxa de transferência Máxima
DDR-200	100 MHz	200 Milhões	PC-1600	1.600 GBps
DDR-266	133 MHz	266 Milhões	PC-2100	2.133 GBps
DDR-333	166 MHz	333 Milhões	PC-2700	2.667 GBps
DDR-400	200 MHz	400 Milhões	PC-3200	3.200 GBps

No entanto, as taxas de transferência mostradas na Tabela 2.1 não são efetivas, pois existe latência no acesso aos dados. Isso ocorre devido à arquitetura de busca de palavras no núcleo DRAM, à multiplexação dos dados na entrada e saída e à organização dos dados na memória. O controlador deve aguardar alguns ciclos de relógio antes de receber uma sequência de dados lidos. Isso caracteriza a latência de leitura da memória introduzida pelo circuito de sincronização que conecta o núcleo ao barramento externo. Além disso, a organização de dados na memória obedece ao esquema de banco, linha e coluna. Antes de acessar uma coluna, o controlador deve ativar um banco e linha do núcleo, para após ler ou escrever o dado. Para trocar de linha ou banco, o controlador também deve desativar a linha ou banco acessados anteriormente. Essas etapas são ilustradas na Figura 2.4, onde a etapa de leitura de dados com troca de linha é a que leva mais ciclos para que o dado esteja disponível no barramento.

Figura 2.4: Exemplos de latência no acesso em diferentes tipos de acesso.

Os ciclos adicionais gastos no controle da memória reduzem a taxa de transferência de dados. A vantagem em termos de eficiência no uso de memórias síncronas é a possibilidade de repetir as sequências de rajadas de acesso de dados. Um sistema eficiente de memória deve poder agendar comandos e organizá-los da melhor forma a acessar dados constantemente, tirando o melhor proveito da interface de memória.

2.3 ARQUITETURA E FUNCIONAMENTO

A necessidade de aumentar a capacidade de armazenamento e a velocidade no acesso aos dados leva engenheiros à busca de novas soluções em tecnologia de memórias. As gerações de memória DDR deixaram de ser simples matrizes de acumulação de dados para tornar-se elementos complexos contendo máquinas de estados, controle de transferência de dados e funcionamento em taxas elevadas de relógio. Nessa seção será analisado o funcionamento e alguns aspectos construtivos de uma memória DDR SDRAM.

2.3.1 Arquitetura de Memória DDR SDRAM

Um diagrama das interfaces de um elemento de memória DDR SDRAM é mostrado na Figura 2.5. A memória contém três barramentos: de dados, de endereços e de controle. O barramento de controle é formado pelos sinais CASn (*column-address strobe*), RASn (*row-address strobe*), WEn (*write enable*), CKE (*clock enable*) e CSn (*chip select*). A nomeclatura que termina com *n* representa que o sinal é ativo em nível lógico baixo ou '0'. O barramento de dados contém o sinal bidirecional de dados DQ, de máscara de dados DM e bidirecional de amostragem de dados DQS. O barramento de endereços é formado pelos sinais de endereço de linha (*row address* - RA), endereço de coluna (*column address* - CA) e banco de memória (*bank address* - BA). Além disso existem duas entradas de relógio (CK e CKn) defasados de 180°. A interface da memória é do tipo *source synchronous*, ou seja, juntamente com o dado é entregue um sinal de sincronização enviado pelo circuito de origem para o de circuito destino.

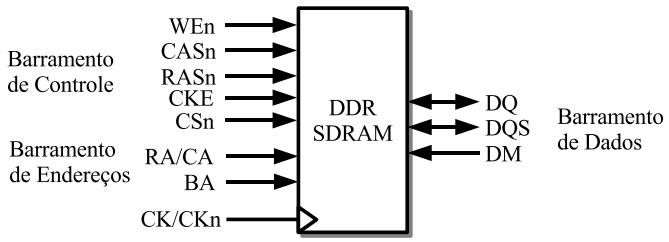

Figura 2.5: Diagrama de sinais da interface de memória DDR SDRAM.

O conjunto de células lógicas de armazenamento de dados é organizado em quatro bancos independentes e todos são acessados pelo mesmo barramento de endereços e de dados. Cada elemento de memória contém uma parte de controle que é responsável pela manutenção dos dados armazenados na memória, fazendo o *auto-refresh* periódico, que deve ser solicitado pelo controle externo à memória. Um diagrama de blocos que ilustra as interfaces do núcleo de memória e da parte de controle de um elemento DDR SDRAM é apresentado na Figura 2.6.

A memória contém uma lógica de controle que interpreta o comando recebido na entrada de decodificação de comandos, gerando sinais de controle para acessar dados armazenados no núcleo DRAM. Os quatro bancos de memória são controlados por re-

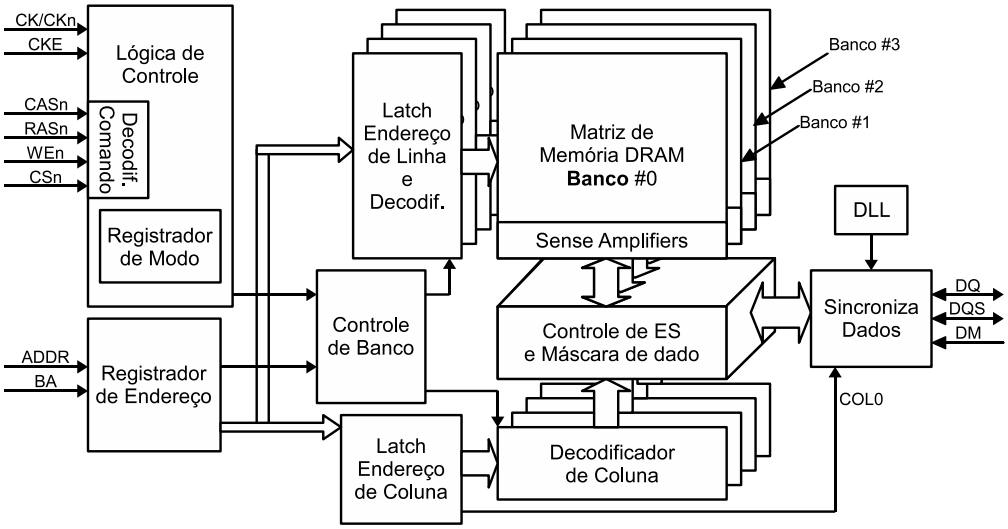

Figura 2.6: Diagrama de blocos simplificado da arquitetura interna da memória DDR SDRAM.

gistradores de endereço de linha e coluna. A parte de sincronização de dados controla a entrada e saída (escrita e leitura) do dado no banco de memória selecionado. O bit COL0 de coluna faz a multiplexação do dado durante a leitura da memória. A Figura 2.7 ilustra o circuito de interface de dados com o núcleo de memória DRAM.

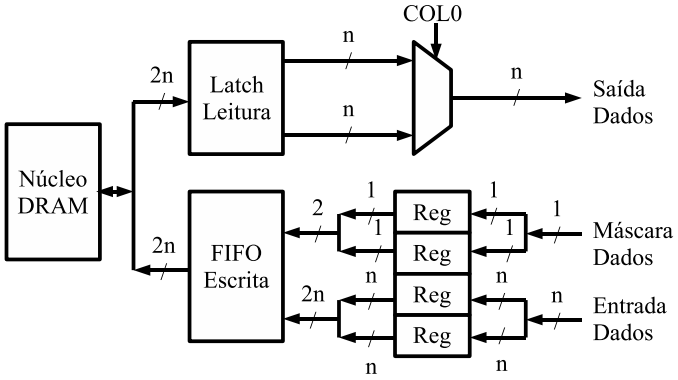

Figura 2.7: Arquitetura *2n-prefetch*: circuito de sincronização de dados da memória.

O sinal de máscara de dado DM é utilizado para inibir a escrita na memória, protegendo os bits de dados pela configuração da máscara. Os sinais de relógio entregues à DDR são gerados pelo controlador da memória. Os sinais de controle e de dados da interface entre o controlador e a memória devem ser sincronizados a partir desses relógios. Um *Delay Locked-Loop* (DLL) faz o ajuste de fase dos sinais de dado DQ e DQS lidos da memória, alinhando a borda de transição desses sinais com a borda de CK e CKn.

Os dados DQ são escritos e lidos nas duas bordas do relógio e os sinais de amostragem DQS são usados para sincronizar os dados com destino. Os sinais bidirecionais DQS são utilizados na sincronização dos dados nas etapas de escrita e leitura, sendo transmitido em paralelo com o barramento de dados e utilizado pelo circuito de captura. Durante a fase

de leitura da memória, o sinal de amostragem é transmitido pela DDR alinhado com a borda de transição do dado. Durante a fase de escrita na memória, o sinal de amostragem é transmitido pelo controlador da memória alinhado no centro dos dados.

A escrita e leitura da memória são orientadas por rajadas de dados, obedecendo uma sequência pré-estabelecida de comandos. A sequência de escritas ou de leituras começa em um endereço inicial e continua automaticamente por um número pré-programado de endereços. A organização interna dos dados na memória é feita a partir de uma matriz, com M linhas e N colunas. A rajada de acessos à memória é realizada somente dentro da mesma linha da matriz, no mesmo banco. O tamanho da rajada é fixo para as memórias DDR SDRAM e pode ser programado para 2, 4 ou 8 palavras.

A parte de controle da DDR implementa funcionalidades de economia de energia como *Power-Saving* (para economia de energia durante períodos de inatividade) e *Power-Down Mode* e *Auto Pre-charge* que elimina a necessidade do controle de *Pre-charge* feito pelo controlador.

2.3.2 Funcionamento de Memória DDR SDRAM

A memória contém uma máquina de controle interna que deve ser inicializada corretamente antes de entrar em modo de operação normal. Depois que a memória é energizada e que os níveis corretos de tensão sejam atingidos, uma sequência de comandos deve ser enviada à memória para garantir seu correto funcionamento.

Após a energização da memória e que o relógio do circuito esteja estável, é necessário aguardar o intervalo de 200µs antes de executar qualquer comando na memória. Após esse intervalo de tempo, um comando DESELECT ou NOP deve ser requisitado pelo controlador e CKE deve estar em nível alto. Em seguida, um comando de PRECHARGE_ALL deve ser aplicado. A próxima etapa é a inicialização do DLL interno e programação dos parâmetros de operação, através da configuração do *Extended Mode-Register* utilizando-se o comando MODE_REGISTER_SET, seguido pelo comando de reset do DLL interno. Após isso deve-se aguardar 200 ciclos para estabilização do DLL interno da memória. Em seguida, o comando PRECHARGE_ALL deve ser aplicado para que o circuito esteja no estado *all banks idle*.

Quando no estado de espera, dois comandos de AUTO_REFRESH devem ser aplicados seguidos pelo comando MODE_REGISTER_SET com os mesmos parâmetros de programação da memória mas com o bit de reset do DLL desligado (a fim de tirar o DLL interno do estado de reset). Ao final dessa sequência de passos, a memória está pronta para operar normalmente.

O registrador de modo (*Mode Register*), ou de controle, é utilizado para programar o modo de operação da memória. A configuração do modo de operação da memória DDR é feita usando os barramentos de endereço e de comando. O valor da configuração do registrador é programado pelo barramento de endereço, para isso é necessário solicitar o

comando de programação do registrador através do barramento de comando. Essa operação deve ser feita pelo controlador da memória. A Figura 2.8 ilustra a sequência de bits que formam o registrador de modo da memória e o bit correspondente no barramento de endereço composto pelos sinais de endereço A[0:n] e de banco BA[0:1].

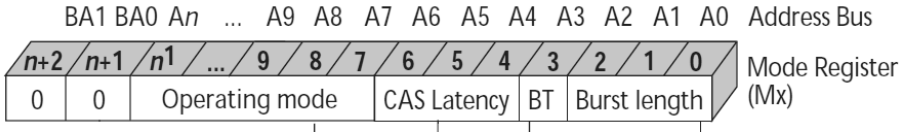

Figura 2.8: Registrador de Modo da DDR SDRAM. Fonte (JEDEC, 2003).

Esse registrador é utilizado para configurar o funcionamento da memória, através dos parâmetros *burst length*, *burst type*, *CAS latency*, e modo de operação. Os parâmetros são descritos abaixo:

- Burst length (A2...A0) – define o tamanho da rajada de acesso aos dados. Esse parâmetro é pré-programado durante a inicialização do circuito de memória determinando o número máximo de colunas que podem ser acessados sequencialmente;

- Burst type (A3) – o tipo de rajada é definido como entrelaçado ou sequencial. No tipo sequencial, os endereços de coluna acessados pela operação de leitura ou escrita são incrementados de um valor unitário, na forma 0-1-2-3-4-5-6-7. Já no modo entrelaçado, o acesso é feito invertendo-se o bit menos significativo do offset, da forma 1-0-3-2-5-4-7-6;

- CAS latency (A6...A4) – usado para programar a latência na leitura dos dados da memória. É o tempo, em ciclos do relógio, entre o registro de um comando de leitura e o dado estar pronto no barramento;

- Operating mode (A13...A7) – determina o modo de operação da memória. Geralmente são disponíveis os modos normal de operação e o modo de reset do DLL.

O registrador de modo deve ser programado somente enquanto os bancos de memória estejam em estado de espera e nenhuma operação de leitura ou de escrita esteja sendo executada. Também deve ser programado durante a inicialização do sistema, antes de utilizar a memória. Ela não armazena a última configuração do registrador de modo.

O acesso ao conteúdo armazenado no núcleo de memória DRAM é feito em duas etapas: primeiro é ativada uma linha de memória, comando conhecido por RAS (*row-active strobe*); em seguida é ativada a coluna onde os dados são escritos ou lidos na memória, é conhecido como CAS (*column-active strobe*). A linha da memória permanece ativa até que ela seja desativada (desabilitada) através do comando *Pre-charge*. O comando *Pre-charge* deve ser executado ao final dos acessos aos dados em uma linha da matriz de

memória, quando deseja-se acessar dados em outra linha. Durante o comando de escrita, o bit de endereço A10 em nível alto determina que a escrita é sucedida por um comando de *auto-precharge*. Caso contrário o controlador deve realizar o comando de *auto-precharge* para liberar a linha antes da nova escrita.

A Tabela 2.2 relaciona os comandos para controle de operações da memória e a combinação dos sinais de interface de comando e endereço.

Tabela 2.2: Comandos de controle da memória DDR SDRAM.

Comando	CSn	RASn	CASn	WEn	ADDR	BA
Não seleciona (NOP)	1	x	x	x	x	x
Sem operação (NOP)	0	1	1	1	x	x
Seleciona banco e ativa linha (ACT)	0	0	1	1	Linha	Banco
Escrita (ESCR)	0	1	0	0	Coluna	Banco
Leitura (LEIT)	0	1	0	1	Coluna	Banco
Termina Rajada	0	1	1	0	x	x
Desativa linhas (PRE)	0	0	1	0	A10	Banco
Auto-refresh ou *Self-Refresh*	0	0	0	1	x	x
Programa registrador de modo	0	0	0	0	MRD	x

O comando PRE pode ser utilizado para desativar todos os bancos se A10 = 0 ou somente o banco selecionado por BA se A10 = 1. O símbolo *x* representa “não importa”.

A lógica de controle da memória contém uma máquina de estados com 15 estados como ilustra a Figura 2.9. As operações de leitura e de escrita de dados passam pela ativação de linha da matriz de memória, representado pelo estado “row active”. Tanto a escrita como a leitura podem ser feitas por repetições do ciclo de leitura, sem necessidade de fazer o *pre-charge*. A troca de linha é realizada a partir do comando *pre-charge* que pode ser automático, passando pelos estados “read A” ou “write A”, ou pode ser realizado manualmente. Também existe a opção de executar o “write-to-read” com ou sem *pre-charge* automático.

Para compreender melhor o funcionamento do controlador de memória, o funcionamento simplificado da memória DDR SDRAM foi apresentado nesse capítulo. O próximo capítulo é apresentado o projeto do controlador de memória em-chip e os desafios de implementação de uma interface de dados DDR para FPGA.

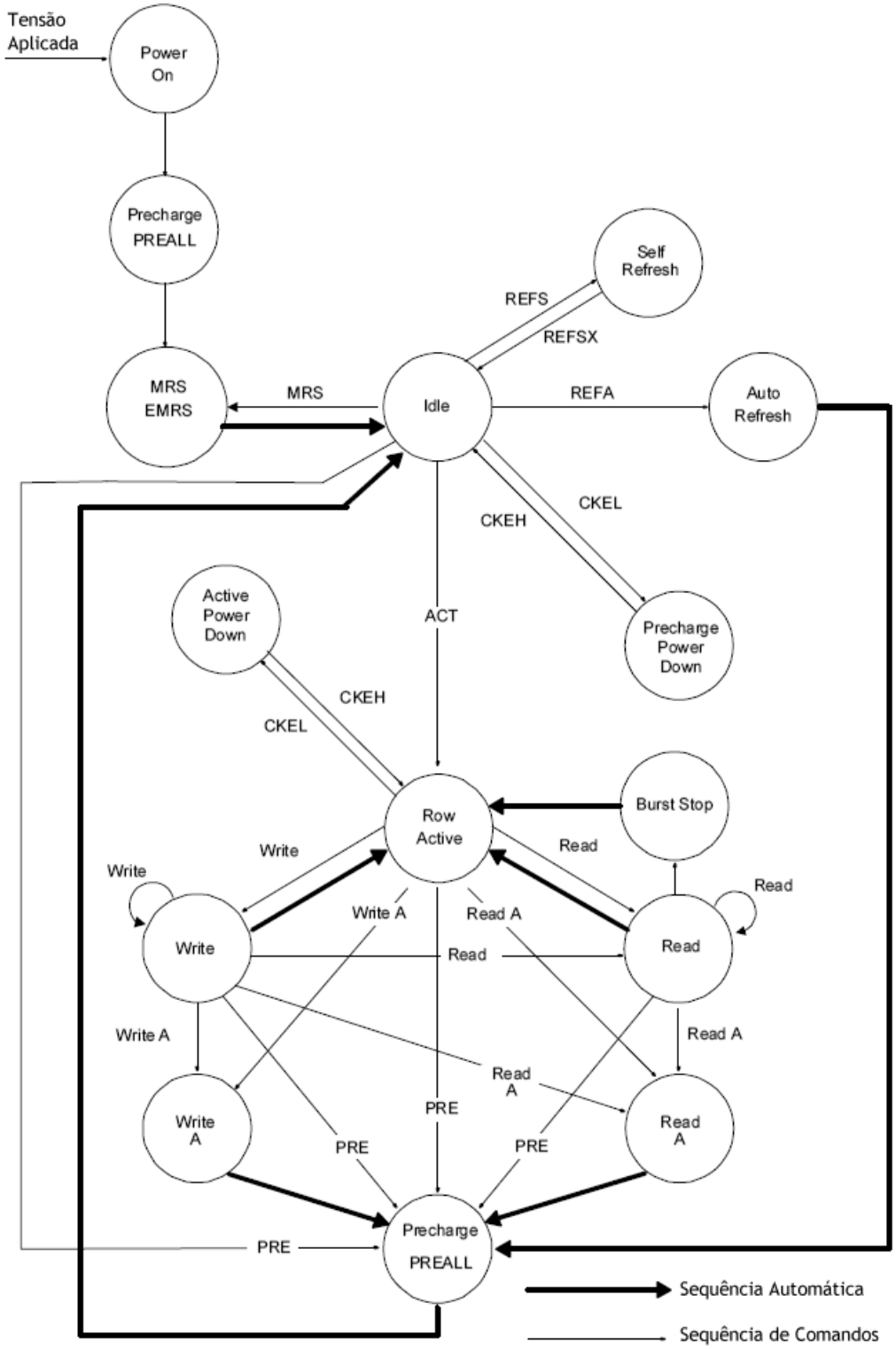

Figura 2.9: Diagrama de estados da máquina de controle de uma memória DDR SDRAM. Fonte: (JEDEC, 2003).

3 CONTROLADOR DE MEMÓRIA EM CHIP

Memórias DDR têm funcionamento diferenciado devido ao circuito de arquitetura com um buffer duplo de carga de dados, denominado de arquitetura *2n-prefetch*. O núcleo de memória DRAM possui largura de 2n-bits enquanto que a interface de dados possui largura de n-bits. O dado é transferido à dupla taxa de amostragem por ciclo de relógio. As operações de acesso ao núcleo de memória, tanto a escrita quanto a leitura, necessitam da execução de etapas de ativação de linha, ativação de coluna e uma finalização. Mesmo com a latência maior de acesso, se comparada à memória SRAM, esse tipo de memória vêm sofrendo uma evolução com o surgimento de DDR2 (4n-*prefetch*) e DDR3 (8n-*prefetch*). A cada nova geração, a capacidade total de armazenamento dos elementos de memória e a taxa de transferência de dados têm aumentado. Enquanto isso, o custo do mega-byte de armazenamento é menor a cada nova geração de memórias DDR SDRAM. Por esses motivos, se comprova que a arquitetura de *prefetch* é eficiente.

Quando se criou o modelo de memórias DDR SDRAM a partir do conceito de dupla amostragem por ciclo de relógio, algumas decisões importantes do seu funcionamento foram especificadas: (1) o sistema de transferência de dados entre memória e controlador é sincronizado na origem (*source synchronous*). Portanto, o controlador de memória entrega para a DDR dois sinais de relógio CK e CKn, sincronizados e defasados de 180°; (2) os sinais de dado e de amostragem gerados no chip de memória contém circuitos DLL usados para alinhar as transições de DQS e DQ com CK. Isso elimina qualquer defasagem entre o dado e o relógio de entrada (recebido do controlador) nos pinos do chip de memória. Esse atraso é denominado tempo de acesso aos dados (do inglês *Data Access Time* ou tAC). Além disso, os amostradores são sinais bi-direcionais, o mesmo sinal é utilizado para alinhar o dado lido ou escrito. Por fim, (3) os chips de memória são desprovidos de qualquer circuito de correção de fase, necessário para alinhar o sinal de amostragem DQS no centro do sinal DQ. Isso simplifica o circuito de leitura de dados interno à memória, mas transfere a complexidade da captura de dados para o controlador. Na escrita o sinal de amostragem DQS é entregue defasado de 90° em relação ao dado DQ, como ilustra a Figura 3.1. A memória registra o dado escrito nas bordas de subida e de descida do sinal de DQS recebido. Os bits de dado DQ lidos da memória são entregues alinhados ao sinal

de amostragem DQS e ambos em fase com o relógio CK, como ilustra a Figura 3.2.

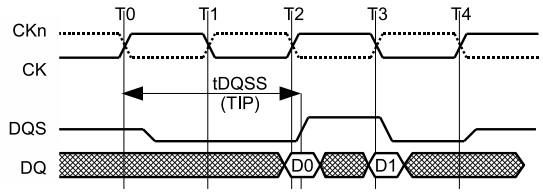

Figura 3.1: Diagrama de tempo de escrita, dado DQ e amostrador DQS são entregues à memória defasados de 90°.

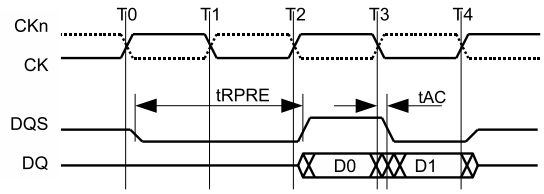

Figura 3.2: Diagrama de tempo de leitura, dado DQ e amostrador DQS são entregues ao controlador alinhados.

O intervalo tDQSS típico representa o intervalo de tempo entre a recepção do comando de escrita e a chegada da primeira transição do amostrador DQS na memória. Já durante a leitura, o intervalo tRPRE representa o tempo em que o amostrador DQS passa a nível baixo e sua primeira transição. Tipicamente, esses dois intervalos são iguais a um ciclo.

Essas decisões foram tomadas para simplificar e reduzir o custo dos chips de memória, transferindo a complexidade para o controlador. Em módulos de memória DIMM, geralmente tem-se oito ou dezesseis chips de memória conectados em paralelo, controlados por um único controlador. O controlador é um circuito extremamente sensível a atrasos e sua implementação é uma tarefa complicada do ponto de vista de desempenho. Para transferir dados com a memória externa, técnicas de projeto são utilizadas na construção do controlador de memória DDR. Nas seções seguintes são explicadas técnicas utilizadas para gerar o relógio de sincronismo e para fazer a leitura da memória externa.

3.1 PROJETO DO CONTROLADOR DDR

3.1.1 Esquema de distribuição do relógio

A memória DDR SDRAM recebe do controlador dois relógios de sincronismo complementares, denominados CK e CKn. Esses são utilizados para a sincronização dos circuitos de leitura e de escrita de dados no núcleo interno de memória DRAM. A interface de dados da memória com o controlador faz o ajuste de sincronismo do dado entregue durante uma leitura, alinhando a transição do dado e amostrador às bordas de transição de CK e CKn. Esse sincronismo é feito utilizando um DLL que controla a fase dos sinais a partir dos relógios complementares externos. Durante uma escrita na memória, o amostrador deve ser gerado pelo controlador alinhado com o centro do dado.

Essas considerações acima, sobre geração de relógios de sincronismo complementares e alinhamento de dados e amostrador, sugerem que o controlador de memória contém deve conter mais que um único domínio de relógio. O controlador entrega para a memória um relógio de sincronismo CK, que é o relógio principal do controlador, e também sua versão defasada de 180° CKn. Durante a escrita de dados na memória DRAM, a sequência de dados (DW0 e DW1) é colocada no barramento DQ com fases de 270° (para a borda

de subida do amostrador) e de 90° (para a borda de descida do amostrador). O sinal de amostragem é sincronizado aos relógios CK e CKn. Durante a leitura, a memória coloca o sequência de dados (DR0 e DR1) no barramento DQ alinhados com as bordas de subida e de descida de CK. Já os sinais de controle e endereço são gerados pelo controlador e sincronizados com a borda de descida de CK, à taxa simples de transferência. A Figura 3.3 sintetiza as informações acima, relacionando no diagrama de tempos as transições e com que relógio são sincronizadas.

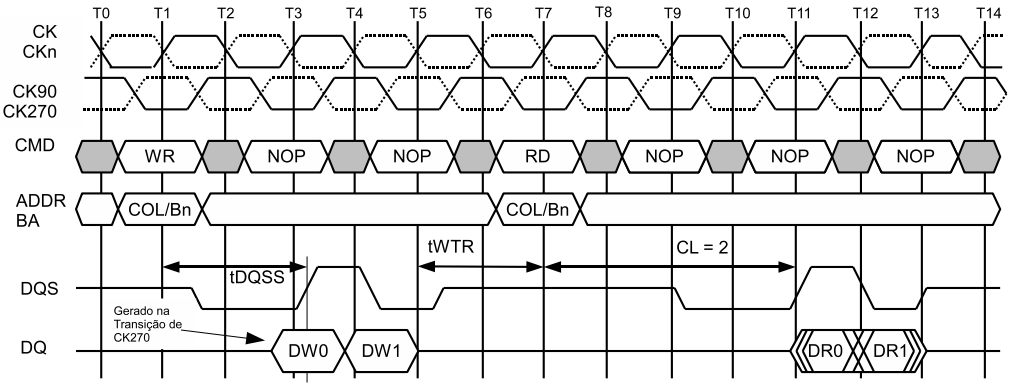

Figura 3.3: Diagrama de tempo de escrita e leitura na memória DDR.

O intervalo de tempo denominado tWTR representa o número mínimo de ciclos entre um comando de escrita e leitura. A latência de acesso aos dados após a realização de um comando de leitura na memória e simbolizada por CL (*CAS Latency*).

Internamente ao controlador de memória, existem três versões do relógio do sistema defasados de 90°, 180° e 270°, sendo que dois desses relógios (CK e CKn) são transmitidos à memória externa. Dispositivos FPGA contém um circuito interno de ajuste de fase do relógio do sistema. Nos FPGAs Xilinx Virtex-2 Pro existem elementos internos de ajuste do relógio construídos em meio à matriz de elementos programáveis, denominados de *Digital Clock Manager* (DCM). São circuitos utilizados para regenerar o relógio externo e distribuí-lo pelo dispositivo através de linhas de relógio dedicadas. Também são utilizados para gerar um sinal de relógio em fase com o relógio do sistema, técnica denominada de síntese de frequência.

O sistema que utiliza o controlador de memória deve conter um circuito de geração de relógio em fase com o relógio do sistema. A partir de um DCM, pode ser gerado o relógio CK90, que é distribuído pelo controlador de memória. O fabricante Xilinx recomenda que seja utilizada a técnica de inversão local dos relógios CK e CK90, gerando os relógios em fases de 180° e 270° respectivamente. Essa técnica é recomendada para reduzir o efeito de variação do relógio pelo circuito FPGA. Cada elemento lógico programável, denominado pelo fabricante Xilinx de *Configurable Logic Block* (CLB), possui uma entrada de relógio direta e uma negada. Dessa forma, o sinal de relógio é invertido localmente em cada registrador, com mínima defasagem. A Figura 3.4 mostra o diagrama

de blocos simplificado da geração de relógio utilizando um DCM e o método de inversão local nos registradores.

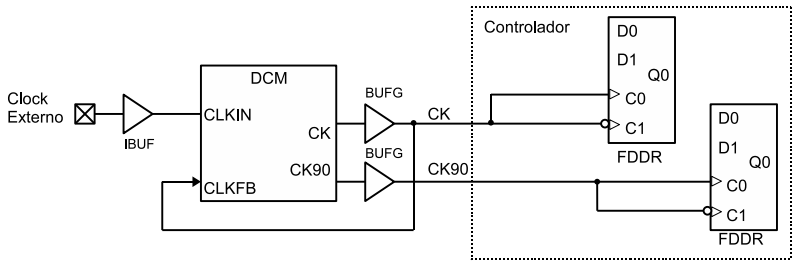

Figura 3.4: Diagrama de blocos da distribuição de relógios com inversão local.

As entradas CLKIN e CLKFB são respectivamente a entrada de relógio gerado externamente ao FPGA e a realimentação da saída CK. Os buffers de entrada IBUF e de conexão com trilhas de distribuição global de relógio BUFG são utilizados para distribuir o relógio pelo FPGA. O relógio invertido CKn é gerado internamente ao próprio CLB que implementa o registrador DDR, a partir de um inversor localizado na entrada de relógio de sincronismo do bloco configurável.

3.1.2 Leitura de dados da memória

O projeto de um controlador DDR SDRAM rompe o conceito tradicional de um sistema síncrono (com um único relógio) usado para acessar os dados armazenados na memória. A interface de dados de memória DDR é do tipo *source synchronous*, ou seja, na leitura ou escrita dos dados é entregue um sinal de sincronização. Tanto para a escrita como para a leitura na memória, a borda de transição do sinal DQS é utilizada para capturar o dado. A leitura da memória DDR SDRAM é a tarefa que oferece maior desafio ao projeto do controlador de memória.

Os dados lidos da memória DDR são recebidos pelo controlador juntamente com o sinal de amostragem DQS alinhado com os sinais DQ. Existe um sinal DQS para cada oito sinais de dado DQ. Tanto DQ como DQS gerados na DDR estão alinhados ao relógio CK gerado pelo controlador. O controlador utiliza o sinal de amostragem para amostrar os dados lidos da memória, tanto na borda de subida quanto na de descida.

O sinal de amostragem chega no FPGA atrasado em relação ao relógio CK transmitido à memória externa, devido ao tempo de propagação entre a memória e o controlador. Existe um *skew* negativo entre o amostrador e CK pois o relógio é enviado à memória pelo controlador e os dados são enviados pela memória ao controlador. Dessa forma, a captura de dados está sujeita a problemas de violação de tempo de *setup*. Esse atraso é dependente do tempo de propagação das trilhas que conectam a memória DDR ao FPGA, do atraso de propagação dos pinos do FPGA e do atraso das conexões internas. Para amostrar corretamente os dados lidos da memória, deve-se atrasar o sinal DQS de um intervalo igual a t_{DQS}, como ilustrado em Figura 3.5. A Figura 3.6 ilustra o caminho de

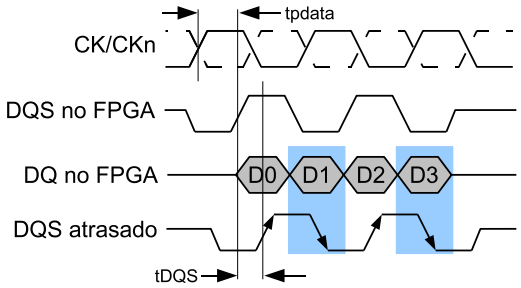

Figura 3.5: Diagrama de tempo da leitura dos dados com DQS atrasado.

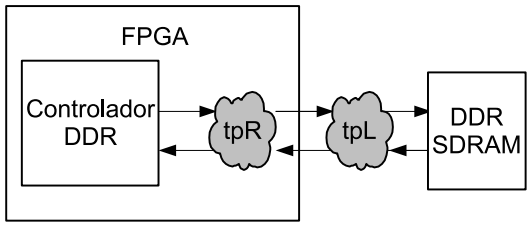

Figura 3.6: Tempos de propagação nas conexões entre memória e controlador.

propagação dos dados entre controlador e memória. O intervalo de tempo denominado t_pdata é o tempo decorrido entre a transição de subida do relógio gerado pelo controlador DDR e o instante em que os dados são recebidos pelo controlador. Esse intervalo de tempo é dado pela soma dos tempos de propagação nas trilhas da placa (t_pL) mais o tempo de propagação dentro do FPGA (do roteamento ou (t_pR)) mostrados na Figura 3.6.

O tempo de propagação t_pL depende do projeto da placa contendo o FPGA e a memória. Essa variável envolve o tempo de propagação das conexões entre memória e FPGA e dos pinos dos componentes. Esse atraso pode ser considerado constante para uma implementação do controlador DDR em uma determinada placa, a menos pela tolerância dos componentes. Já os atrasos internos ao FPGA gerados pelos elementos de trilhas e conexões pode variar conforme a implementação e roteamento realizados pela ferramenta de síntese.

Outro desafio da leitura de dados é a demultiplexação dos dados lidos, para troca de n-bits em dupla taxa por ciclo de relógio para 2n-bits à taxa simples. Após isso, existe uma transição de domínios de relógio em que o dado lido deve ser alinhado novamente com o relógio do sistema CK. Essa transferência deve levar em conta o atraso relativo entre as transições do amostrador DQS e as transições do relógio do sistema, a fim de respeitar os intervalos de *hold* e *setup* dos registradores de captura.

O controlador de memória deve ser implementado com especificações críticas de temporização, principalmente quando da leitura de dados da memória externa. Antes de fazer uma análise de cálculo de temporização do dado, devem ser esclarecidos os conceitos de dado válido e variação (*skew*). Internamente à DRAM, existe um relógio de sincronização que faz o controle dos multiplexadores de dado e amostradores para a saída. Mas devido à variações de roteamento internas ao chip de memória e efeitos dinâmicos de chaveamento simultâneo dos pinos de saída, existe uma variação no instante da transição dos sinais de saída. Isso gera um pequeno espalhamento no barramento de dados. Essa variação, entre todos os pinos de dado e seus respectivos amostradores é denominada de tDQSQ. A Figura 3.7 mostra que a janela válida do dado acontece entre a última variação de transição do dado Dn e a primeira variação de $Dn+1$.

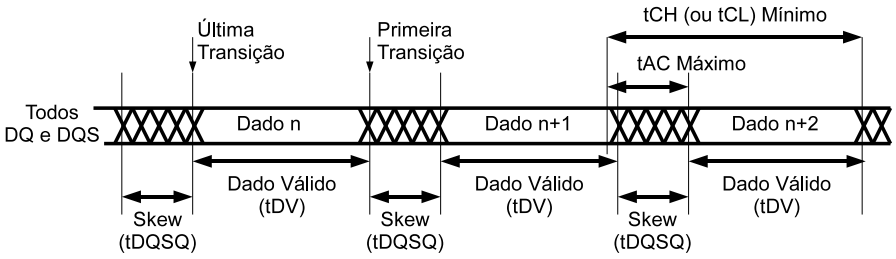

Figura 3.7: Janela de dado válido e variação entre os sinais DQ e DQS gerados pela memória DDR.

Outro conceito importante na análise de dados gerados pela memória é o tempo de acesso ao dado (*access time*), denominado tAC. O chip de memória contém um DLL interno que ajusta a transição dos sinais de dado e amostrador de saída com a borda do relógio recebido do controlador. O tempo de acesso pode variar devido à transição simultânea dos sinais de saída, jitter interno do relógio, variação da temperatura afetando o DLL, entre outros fatores (RYAN, 1999). A especificação desse parâmetro faz referência à variação de longo tempo e não é afetada pela variação do duty-cycle do relógio. A diferença entre tAC e tDQSQ é que esse último representa a diferença entre a primeira e a última variação de um sinal num grupo sincronizados pelo mesmo relógio. Já tAC representa a diferença de tempo entre o evento de transição do relógio e a última variação de um sinal do grupo controlado por esse relógio. Como tAC é dependente do intervalo de tempo de relógio (tCH ou tCL), ele pode sofrer alguma variação ao longo do tempo, mas sempre estará limitado entre duas transições do relógio.

Os dados enviados pela memória e recebidos pelo controlador são capturados a partir das transições do sinal de amostragem DQS. Internamente ao controlador, esse sinal deve ser alinhado com o centro da janela de dado válido. A quantidade de tempo que esse sinal deve ser atrasado é denominada tSD (*strobe delay*). Entre os sinais serem enviados pela memória e chegarem ao controlador, existe a variação causada pelas trilhas de roteamento da placa. Na Figura 3.8 é ilustrada a redução da janela de dado válido, causado pelas variações geradas pelas trilhas da placa.

O valor de tSD deve ser calculado para centrar a transição de DQS no centro da janela de dado válido. O valor ideal é tSD = (tDQSS + tDV)/2. Mas projetar o atraso do amostragem para um valor ideal não é suficiente. Se o efeito da placa reduzir muito o tempo de dado válido, o controlador não poderá capturar dados corretamente. Existem os casos máximo e mínimo para o cálculo do atraso inserido em DQS, ilustrados na Figura 3.9.

Considerando-se o pior caso em que o amostrador chega muito cedo ao controlador (mais para a esquerda), é acrescentado ao valor ideal metade da variação do grupo de dados, ou seja, tSD = tDV/2 + tDQSQ. Já para o melhor caso com o amostrador chegando muito tarde (mais para a direita), o atraso é igual à metade da janela de dado válido, ou seja, tSD = tDV/2. A influência total dos efeitos das conexões da placa que conectam o

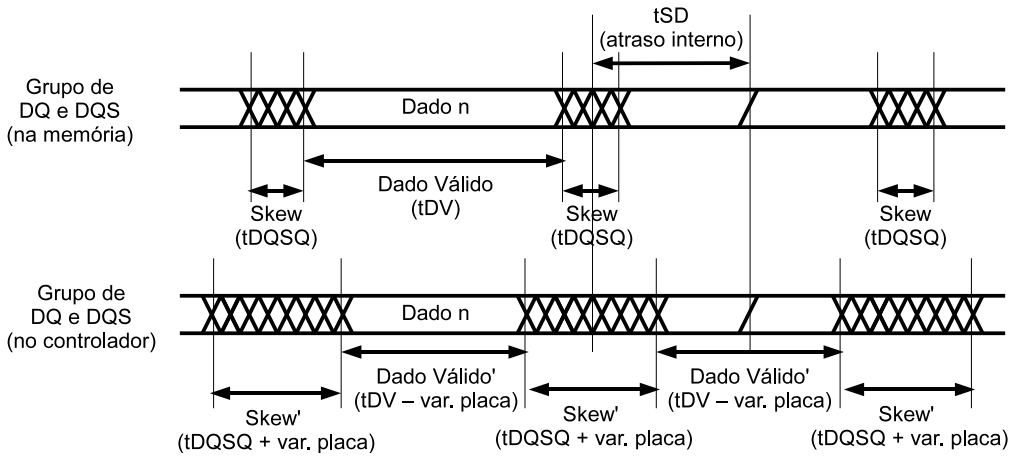

Figura 3.8: Redução da janela de dado válido pelas variações da placa.

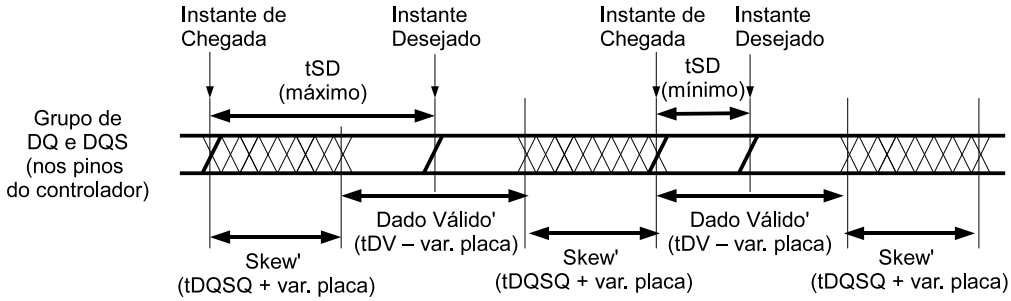

Figura 3.9: Intervalos mínimo e máximo de tSD para atraso do amostrador DQS.

dispositivo FPGA e a memória é uma grandeza difícil de ser mensurada. O fabricante da placa de prototipação não informa valores de diferença entre o tempo de propagação das trilhas de conexão da memória. Um valor típico assumido no projeto do controlador de memória é de 50 ps de variação entre as trilhas de dados e de amostragem. A análise de janela de dado válido pode ser feita assumindo que todos efeitos das trilhas sobre os sinais são simétricos para dado e amostrador. Dessa forma, a análise utiliza as transições do sinal de amostrador como referência. A Figura 3.10 ilustra a análise de incerteza do amostrador DQS na captura de dados DQ. Nesse caso, assume-se que o intervalo de atraso do amostrador tSD é o ideal.

Essa análise também leva em conta o intervalo de tempo de incerteza total do amostrador tSU (*strobe uncertainty*) no controlador de memória. A incerteza ocorre no circuito gerador do atraso tSD. Nesse caso, a variação dos dados (skew) é igual a 2 x (tDQSQ + variação da placa) e o tempo de dado válido é igual a tDV - tDQSQ - 2 x variação da placa. Também devem ser analisados os valores de setup e hold dos registradores do controlador, respecivamente tS e tH. A partir da largura da janela de dado válido, são subtraídos: duas vezes o valor dos efeitos totais da placa sobre o dado; o valor de erro de geração do amostrador atrasado no controlador; a incerteza de chegada do amostrador gerado na memória e; os valores de setup e hold do controlador. O resultante é a margem

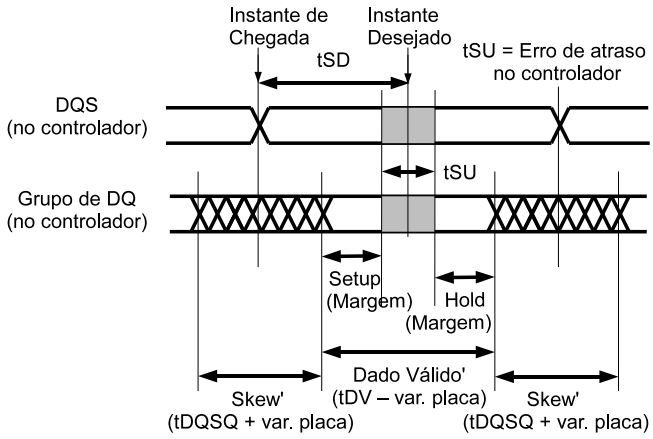

Figura 3.10: Incerteza total do amostrador e janelas de setup e hold.

de tempo que, sendo positiva, significa que os dados serão capturados corretamente pelo controlador.

3.1.3 Implementação do circuito de atraso

Existem diferentes formas de implementar o circuito de atraso do amostrador para a captura do dado no controlador de memória. Circuitos de atraso sofisticados podem garantir o melhor funcionamento do controlador de memória. Já circuitos mais simples, construídos a partir de poucos elementos, podem ser eficientes suficientemente para o funcionamento do controlador e com baixo custo.

Uma abordagem simplificada de geração de atraso é aumentar o tempo de propagação dos sinais de amostragem diretamente sobre as trilhas na placa, entre memória e FPGA. Durante a leitura de dados, o controlador recebe o amostrador alinhado com o centro da janela de dado válido. Em contrapartida, durante a escrita o controlador deve aplicar ao amostrador DQS o atraso correto, pois o efeito de atraso acontecerá tanto na escrita quando na leitura. Essa solução limita a frequência de operação do sistema controlador-memória não sendo interessante para o projeto de um controlador genérico.

Em controladores de memória de alto desempenho, o controle do atraso do sinal de amostragem é implementado internamente ao controlador. Um circuito de controle de atraso pode ser implementado utilizando elementos lógicos. O sinal de amostragem é então utilizado como relógio de sincronismo dos registradores que capturam os dados lidos da memória. Três tipos de circuitos de atraso podem ser implementados: com valor de atraso fixo, com atraso programável e com atraso proporcional ao período do relógio. A Figura 3.11 mostra o circuito de captura de dados no controlador de memória. A linha de atraso controla a borda de transição do sinal de amostragem, alinhando com o centro da janela de dado válido. O amostrador atrasado é utilizado para sincronizar os registradores de captura do conjunto de bits de dado DQ.

Uma linha de atraso fixa pode ser calculada para implementar tSD no caso ideal.

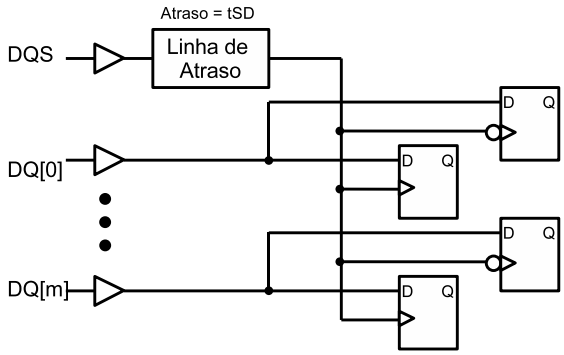

Figura 3.11: Circuito de captura de dados do controlador com amostrador atrasado.

Nesse caso, considera-se que os efeitos de variação da placa sobre os sinais de amostragem e dado gerados pela memória são simétricos. Pode-se estimar o cálculo de tSD sem considerar os efeitos de variação da placa, fazendo tSD = (tDV + tDQSQ)/2. O projeto de uma linha de atraso inicia com a escolha dos elementos lógicos adequados para prover um valor razoável de atraso e que não sofram excessiva variação de temperatura e de tensão elétrica. Outra preferência é de que os elementos lógicos utilizados não sejam inversores, para manter a polarização correta. A precisão da linha de atraso é função do tempo de propagação de cada elemento lógico, que deve ser idêntico tanto nas transições de alto-para-baixo (tPHL) quanto nas de baixo-para-alto (tPLH). A dificuldade de implementar esse tipo de circuito é que a resolução necessária para alcançar o valor de atraso necessário pode não ser atingida. Ainda, variações de roteamento do circuito podem alterar o tempo de atraso.

Existem diferentes maneiras de gerar o circuito de atraso programável como mostrado por (RYAN, 1999). Segundo o autor, a melhor forma de implementação do circuito de atraso, para reduzir problemas de roteamento e possibilitar uma possível expansão futura, é ilustrada na Figura 3.12. Nesse circuito a saída é gerada a partir de um número selecionado de estágios de atraso.

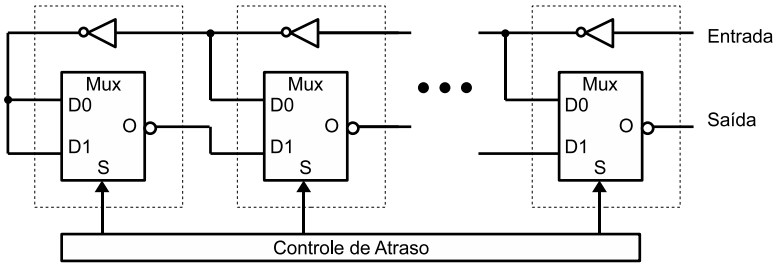

Figura 3.12: Circuito de geração de atraso com elementos programáveis.

Em controladores implementados em FPGA existe uma dificuldade associada para manter o tempo de propagação de um circuito lógico. Dependendo do roteamento do circuito, o atraso gerado pela linha de atraso pode variar significativamente, fazendo com que o circuito de captura de dados deixe de funcionar corretamente. Portanto, para controla-

dores de memória implementados em FPGAs é preferido utilizar o circuito com topologia ilustrado na Figura 3.12. Uma linha de atraso pode ser implementada a partir de uma série de células lógicas programáveis, que implementam a função lógica de multiplexadores de duas entradas e uma saída (Mux 2x1). A Figura 3.13 ilustra o uso de *Look-Up Tables* de quatro entradas (LUT4s) para atrasar o sinal DQS.

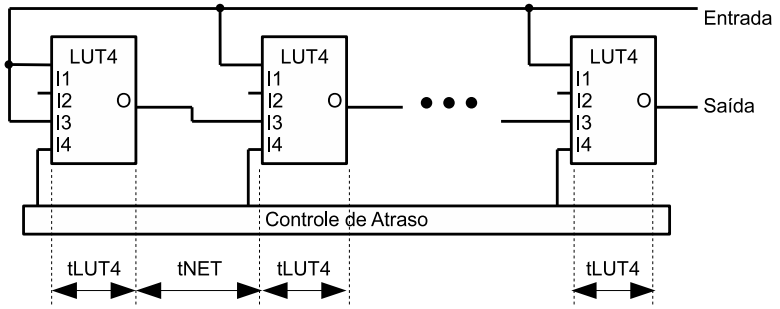

Figura 3.13: Circuito de geração de atraso programável para implementação em FPGA.

O tempo de propagação máximo do elemento combinacional LUT4 (tLUT4), para um dispositivo da família Virtex-2 Pro, é de 280 ps (XILINX, 2007). O tempo de propagação do sinal pelas conexões entre LUTs é denominado tNET e pode variar entre aproximadamente 20 ps e 300 ps dependendo se as LUTs estão no mesmo CLB ou em CLBs adjacentes[1]. Conforme seja feito o roteamento do controlador de memória no dispositivo FPGA, a linha de atraso poderá gerar valores maiores ou menores tSD. Controladores de memória DDR implementados em tecnologia Virtex-2 Pro devem possuir todos os elementos lógicos do circuito de captura de dados posicionados com colocação específica, para garantir o tempo correto de propagação dos sinais. Além disso, um circuito de ajuste de fase pode ser utilizado para compensar a variação (devido às variações de temperatura e tensão) do tempo de propagação do sinal através dos elementos lógicos que compõe a linha de atraso, como descrito em (GUPTA, 2005).

Os fabricantes de FPGAs também acompanharam a evolução das memórias DRAM e criaram novos recursos embutidos nos dispositivos programáveis a fim de prover uma interface eficiente de dados. Os dispositivos das famílias Xilinx Virtex-4 e Virtex-5 contém elementos de geração de atraso programável chamados IDELAY (GEORGE, 2007). São geradores de atraso programável com compensação à variação por temperatura e tensão, com resolução de tempo de propagação igual a 75 ps.

Usar um conjunto de elementos lógicos que podem ser inseridos ou removidos para ajustar o valor de atraso é uma solução interessante do ponto de vista de portabilidade do controlador. Dessa forma, é possível utilizar tal circuito em controladores com memórias de diferentes frequências de operação.

[1]Dados obtidos através de experimentação, resultado da implementação da linha de atraso num dispositivo Virtex-2 Pro usando mapeamento forçado dos elementos lógicos LUT4.

3.1.4 Restrições na Implementação

Ferramentas de síntese e implementação de circuitos lógicos em dispositivos configuráveis permitem que o projetista defina o melhor posicionamento dos elementos na matriz de CLBs. Essa abordagem é necessária para circuitos com especificações críticas de temporização, como é o caso do controlador de memória. A leitura de dados é a parte crítica de implementação do controlador. Dependendo do instante em que ocorra a borda de transição do sinal de amostragem, os elementos de captura de dados podem violar os tempos de setup e hold para amostrar o dado lido.

O posicionamento de elementos utilizados na leitura dos dados enviados pela memória DDR requer um cuidado maior na hora de implementar o controlador em FPGA. A diferença que existe no tempo de propagação de um sinal entre CLBs no FPGA pode resultar na amostragem incorreta dos dados lidos da memória. O correto posicionamento dos recursos utilizados pelo controlador nos elementos lógicos do FPGA deve ser otimizado para reduzir a variação de tempo de propagação entre os sinais. O posicionamento forçado dos elementos lógicos que implementam a linha de atraso e os registradores de captura de dado é necessário para que a ferramenta de síntese não aloque esses recursos em regiões do FPGA com excessivo atraso de roteamento entre os sinais. Internamente ao FPGA, os elementos que constituem a matriz de conexões podem gerar um atraso considerável, se comparado ao tempo de propagação dos sinais. A Figura 3.14 ilustra o diagrama de blocos simplificado do circuito de captura de dados e os tempos de atraso gerados pelo roteamento (linhas e conexões da matriz de FPGA).

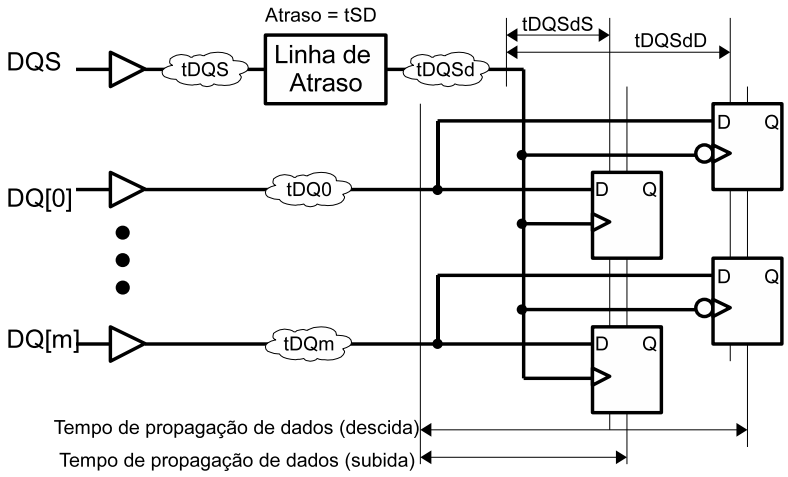

Figura 3.14: Diagrama de blocos simplificado do circuito de captura de dados e a representação dos tempos de propagação dos sinais pela matriz do FPGA.

O atraso gerado pelas trilhas do dado é representado por tDQ[0:m]. O atraso de propagação pela trilha do amostrador é representado por tDQS, enquanto que o amostrador atrasado é tDQSd. Idealmente, após o roteamento do controlador no FPGA, espera-se que a quantidade de atraso de roteamento de qualquer trilha de dado seja igual à soma dos

atrasos do amostrador, ou seja, tDQ[0:m] = tDQS + tDQSd. Da mesma forma, o roteamento deve garantir que o atraso de propagação para qualquer trilha de dado seja igual, ou seja, tDQ0 = tDQ1 = tDQm. Existe uma diferença do tempo de propagação do sinal de sincronismo tDQSd para os registradores de captura de borda de subida e descida. O inversor conectado à entrada de relógio do registrador aumenta o tempo de propagação de forma que tDQSdS $>$ tDQSdD. Para compensar essa variação, os registradores com inversão local do relógio são posicionados na coluna seguinte de blocos lógicos do FPGA. Assim, o tempo de propagação do dado amostrado na borda de descida é maior do que o tempo do dado amostrado na borda de subida. A colocação dos elementos lógicos da linha de atraso também deve ser supervisionada, de forma que o atraso entre as células não altere significativamente o valor de tSD.

Na implementação do controlador, utiliza-se um sinal de amostragem DQS para sincronizar os registradores de captura do dado lido da memória. Pela especificação das memórias DDR SDRAM, tipicamente é utilizado um sinal de amostragem para sincronizar a leitura de 8 sinais de dado. Esse sinal de sincronização deve ser distribuído pelo dispositivo reconfigurável de forma que não haja variação de tempo de propagação entre o bit de dado mais próximo e o mais distante. Dispositivos FPGA da família Virtex-2 Pro contém linhas locais de distribuição de relógio, com baixo skew. Os blocos de interface de ES estão conectados à matriz do FPGA através de uma matriz de conexões. A colocação forçada objetiva colocar os elementos na fronteira mais próxima do FPGA com a memória externa (Figura 3.15), reduzindo o tempo de propagação das linhas de dado e amostragem. Os elementos lógicos de endereço e controle não sofrem tais restrições.

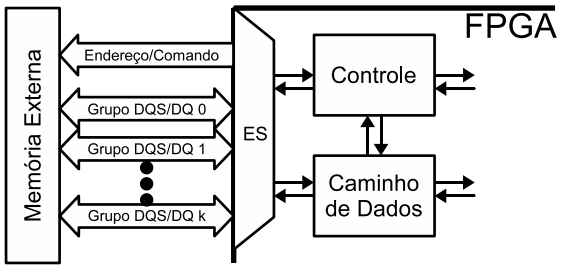

Figura 3.15: Interface de ES do controlador mapeada para a fronteira do FPGA mais próxima com a memória externa.

As técnicas de projeto mostradas aqui foram implementadas no controlador DDR para interface com a memória externa e FPGA. Na próxima seção será mostrado o projeto da interface de utilização do controlador (chamada de Interface do Usuário) e a arquitetura e funcionamento do controlador de memória DDR.

3.2 INTERFACE DE MEMÓRIA

Num sistema complexo contendo uma hierarquia de memória, o controlador DDR pode ser visto como o nível mais baixo de interface com a memória. Um sistema de processamento de dados, seja ele dedicado ou de propósito geral, pode conter um ou mais níveis de memória internas ou externas. A conexão do controlador ao sistema necessita de uma interface que "traduza" as solicitações de acesso aos dados armazenados para comandos conhecidos pelo controlador. Tal interface também é necessária para isolar diferentes domínios de relógio, já que o controlador deve rodar à mesma frequência que a memória externa e o sistema pode rodar em qualquer outra frequência diferente. Um sistema em chip pode possuir uma interface dedicada com seus periféricos, ou uma interface padronizada na forma de um barramento ou de uma NoC. A interface entre o controlador e o sistema é utilizada para realizar essa conexão, nesse trabalho será denominada de *interface do usuário*.

Nessa seção é feita a descrição do funcionamento e da arquitetura interna do controlador de memória utilizado e da interface do usuário implementada. A Figura 3.16 mostra o diagrama de blocos simplificado do sistema e da interface de memória com a memória externa DDR SDRAM.

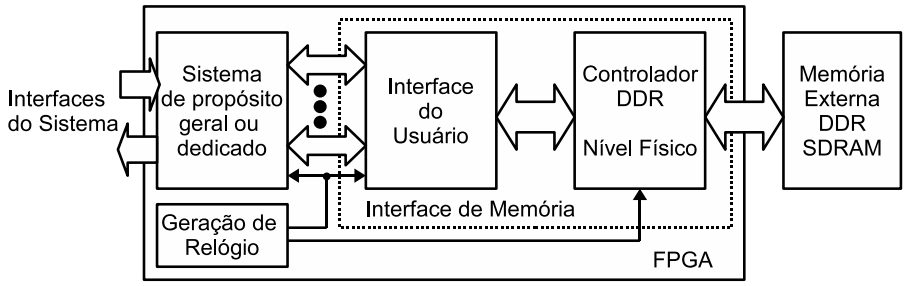

Figura 3.16: Diagrama de blocos simplificado da interface de memória.

A interface de memória é o módulo que controla o acesso aos dados armazenados na memória externa. É formada pelo controlador DDR e pela interface do usuário. O controlador de memória representa o nível físico de interface de dado, controle e endereço com a memória externa. O controlador de memória é um circuito de alta complexidade de implementação física, que deve ser projetado levando em conta técnicas mencionadas na seção 3.1.

A interface do usuário não é um circuito complexo do nível de implementação física, mas possui grande complexidade de funcionamento. É responsável pela gerência dos dados lidos e escritos na memória, pelo endereçamento e pela organização de uma sequência de comandos para reduzir a latência de acesso. Essa interface também pode ser implementada a partir de um modelo simples, que somente traduz os comandos recebidos do sistema para o controlador, ou como uma máquina complexa que faz a gerência de comandos com listas de prioridades e organiza uma fila de execução. Alguns sistemas re-

querem que a interface de memória contenha uma ou mais interfaces de acesso. Tudo isso depende da necessidade do sistema e da urgência de acesso às informações armazenadas na memória externa.

A interface de memória DDR, formada pelo controlador e pela interface de usuário, é explicada nessa seção. Primeiramente é feita a descrição dos aspectos funcionais do controlador de memória DDR SDRAM, como descrição das interfaces e comandos de controle. Em seguida, é explicado o funcionamento da interface de usuário projetada para realizar os testes de implementação do controlador de memória em placa.

3.2.1 Funcionamento do Controlador DDR

O controlador de memória DDR SDRAM gerado pela ferramenta MIG é programado para executar cinco operações de interface com a memória externa: (1) realizar a rotina de *auto-refresh* periódico da memória; (2) controlar as etapas de acesso para a leitura da memória; (3) realizar as etapas de acesso para a escrita na memória; (4) executar a sequência de comandos para a inicialização da memória e; (5) programar o funcionamento da memória através do registrador de modo. A Tabela 3.1 sintetiza os comandos de entrada para o controlador, que devem ser aplicados no sinal user_command_register[2:0].

Tabela 3.1: Comandos do controlador de memória DDR.

Comando	Descrição
000	NOP
010	Inicialização da memória
100	Escrita
101	Lê Registrador de Modo
110	Leitura
Outros	Reservado

Após a inicialização do sistema, a memória DDR SDRAM deve ser inicializada e ter seu registrador de modo configurado antes da execução de qualquer comando. O controlador de memória é responsável por essa tarefa. O controlador implementado pelo MIG contém a rotina de inicialização da memória mas isso deve ser solicitado ao controlador através de um comando. A interface do usuário deve prover o comando de inicialização da memória ao controlador DDR.

Os comandos disponíveis na interface do usuário são simplificados e atendem às necessidades de escrita e leitura da memória. Comandos como “Lê Registrador de Modo” e “Inicialização da memória” não são úteis ao sistema e podem ser implementados somente pela interface do usuário.

3.2.1.1 Inicialização da memória

O primeiro comando que o usuário deve executar antes de utilizar os recursos de memória do sistema é a inicialização da memória. Primeiramente, o valor correto de configuração do Registrador de Modo da memória é colocado no sinal user_mode_register. Depois, o comando de inicialização da memória deve ser aplicado no controlador de memória no sinal user_command_register[2:0] na borda de subida do relógio CLK180, imediatamente antes de completar dois ciclos de CLK0 da configuração de user_mode_ register. Após isso, deve-se aguardar até que o sinal user_init_val esteja em nível alto, indicando que a memória está pronta para ser utilizada. Esse processo pode ser visto na Figura 3.17.

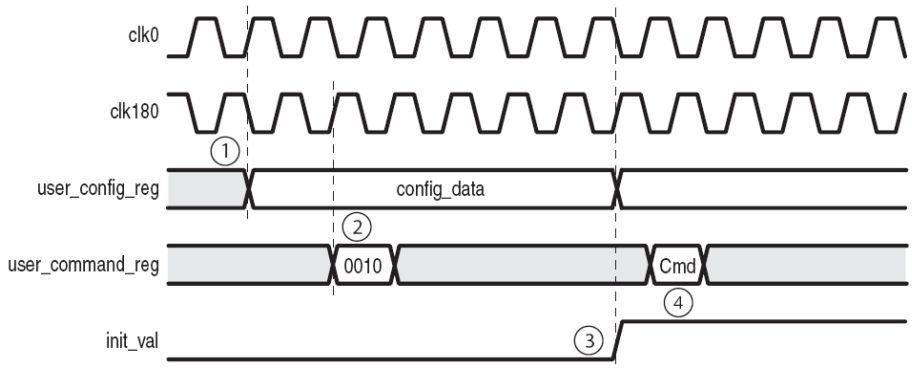

Figura 3.17: Diagrama temporal de inicialização da memória na entrada do controlador DDR. Fonte: (XILINX, 2005).

3.2.1.2 Escrita na memória

A escrita dos dados na memória é realizada em rajadas. Uma sequência de palavras deve ser colocada na entrada do controlador para serem escritas na memória em semi-ciclos consecutivos. A sequência de operações de escrita na memória realizada pelo controlador DDR é feita continuamente até que o usuário force o seu final. A Figura 3.18 mostra como exemplo duas sequências de escrita na memória com tamanho de rajada igual a 4. O tamanho da rajada depende da configuração da memória. Nesse exemplo, são escritos os dados D0-D7 em sequência, ou seja, 64 bytes em palavras de 64 bits. Esses dados devem estar disponíveis na entrada do controlador, sendo chaveados a cada ciclo de relógio após o início da escrita na memória.

O usuário inicia o processo de escrita na memória aplicando o comando na subida de CLK180, como resposta o controlador envia o sinal de confirmação (user_cmd_ack) do comando na subida de CLK180. O endereço deve estar programado no barramento e deve permanecer por 4,5 ciclos de relógio (4,5 é o período CAS-to-RAS, escrita da linha da matriz e após da coluna). Os endereços posteriores devem ser chaveados na transição de subida de CLK0, alternando os ciclos necessários para completar o tamanho de rajada

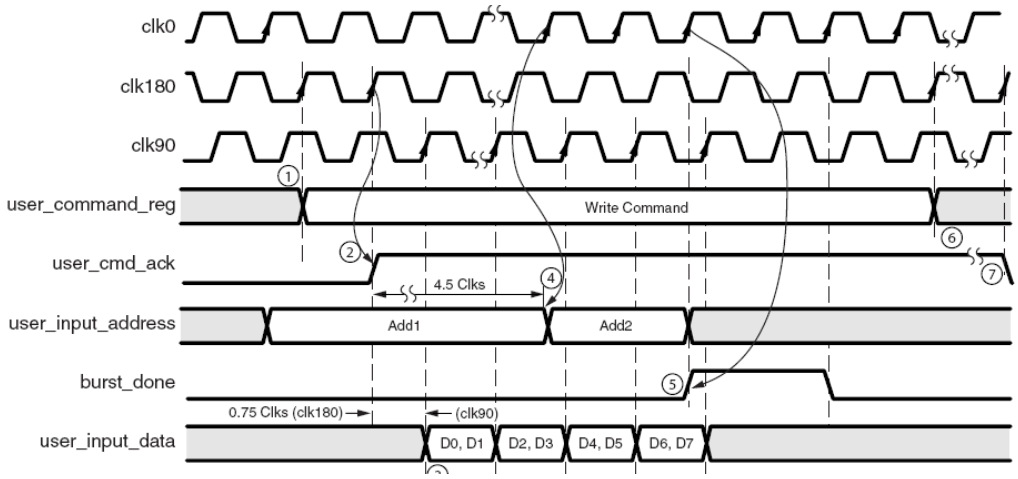

Figura 3.18: Diagrama temporal da interface com o controlador durante o processo de escrita. Fonte: (XILINX, 2005).

programado (2, 4 ou 8)[2].

Os dados devem ser colocados no barramento na subida de CLK90 após receber o sinal de confirmação do controlador e devem ser alternados a cada ciclo de CLK90. Os dados entrados no controlador tem largura igual a $2n$, são do tipo SDR, na saída do controlador eles são convertidos para DDR de largura igual a n.

O fim do ciclo de escrita é realizado com o sinal burst_done, levantado numa borda de subida de CLK0 e assim mantido por dois ciclos. Ao final o sinal de user_cmd_ack retorna ao nível baixo indicando que o controlador está livre para uma nova operação. O ciclo de escrita na memória pode ser continuado por períodos maiores, desde que o endereçamento dos dados na memória seja pertencente à mesma linha da matriz. Como a primeira escrita do ciclo é a que registra a linha, as subsequentes somente escrevem o endereço de coluna na memória.

3.2.1.3 Leitura da memória

A operação de leitura da memória DDR SDRAM é procedida de forma semelhante ao processo de escrita. O usuário coloca primeiramente o endereço inicial a ser lido, em seguida aplica o comando de leitura no controlador e troca os endereços sucessivos após receber o sinal de resposta do controlador. A Figura 3.19 mostra o diagrama temporal da operação de leitura de dados.

Os dados lidos estão disponíveis no barramento quando o sinal user_data_valid estiver em nível alto. Os dados lidos da memória são entregues na borda de subida de CLK90.

O fim do ciclo de leitura é realizado com o erguido numa borda de subida de CLK0 e assim mantido por dois ciclos. Ao final o sinal de user_cmd_ack retorna ao nível baixo indicando que o controlador está livre para uma nova operação.

[2]Rajadas de dados de tamanho 2 e 8 ainda não são suportados pelo controlador DDR gerado pela ferramenta MIG

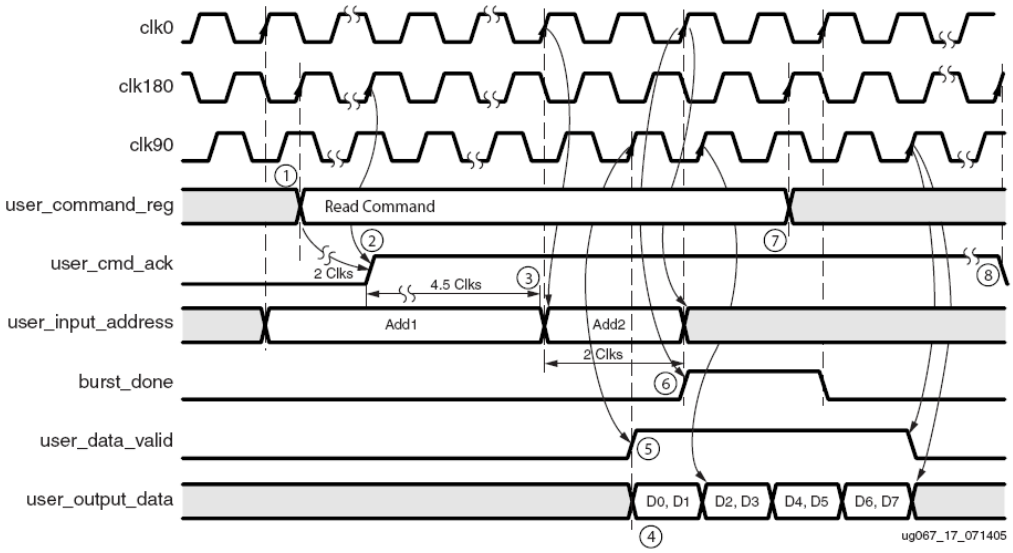

Figura 3.19: Diagrama temporal da interface com o controlador durante o processo de leitura. Fonte: (XILINX, 2005).

3.2.1.4 Auto-Refresh

O controlador de memória realiza periodicamente, a cada 7,7µs, o comando de auto-refresh necessário para manter os dados ativos na memória DDR SDRAM. Quando o usuário recebe um pedido de auto-refresh do controlador DDR, ele deve terminar a operação que está sendo realizada até que o sinal de ar_done seja erguido.

O tempo de duração do *auto-refresh* e o seu período de realização são controlados pelos parâmetros REF_FREQ_CNT e RFC_COUNTER que estão mais detalhados em (XILINX, 2005).

3.2.2 Interface com o usuário

A interface com o usuário é responsável pela troca dos dados lidos ou escritos na memória de uma forma mais "amigável" do que o meio físico é capaz de fazer. Foi projetada para operar no domínio de relógio do sistema, sendo responsável pela troca de domínio de relógio dos sinais recebidos do controlador. Dessa forma, pode ser conectada diretamente aos módulos do usuário que irão comunicar com a memória. Foi projetada de forma simples para implementar funções de acesso aos dados a fim de verificar o funcionamento do controlador. As modificações feitas foram em função da arquitetura do controlador, não foram feitas alterações no controle corrigindo limitações de funcionamento, somente alterações para melhorias no nível físico e de implementação, para testar em outras placas. Foram projetadas uma máquina de controle para a operação de escrita na memória e outra máquina para a leitura. As duas máquinas implementam as operações de acesso ao dado em memória externa a partir da sequência de comandos ACT + (RD/WR) + PRE. Quando da inicialização do sistema, após o reset, a máquina de escrita realiza a inicialização da memória DDR gerando para o controlador o comando INIT. Isso é feito somente após

decorrido o intervalo de espera de 200µs, necessários para que os circuitos DLL da memória estejam funcionando normalmente. A interface do usuário também é responsável pela configuração do registrador de modo de operação da memória DDR. Isso deve ser feito de acordo com o funcionamento do sistema, configurando parâmetros como latência de CAS (CL) e o tamanho e ordem das rajadas de dados. O controlador é responsável pela geração periódica dos comandos de refresh da memória.

A Figura 3.20 ilustra o diagrama de blocos simplificado das partes principais da interface de usuário projetada. Do lado do usuário são fornecidos comandos de escrita (ESCR) e de leitura (LEIT), juntamente com o endereço inicial da operação. A interface é responsável por gerar a sequência de comandos para o controlador e também a sequência de endereços a partir dos endereços de banco, linha e coluna.

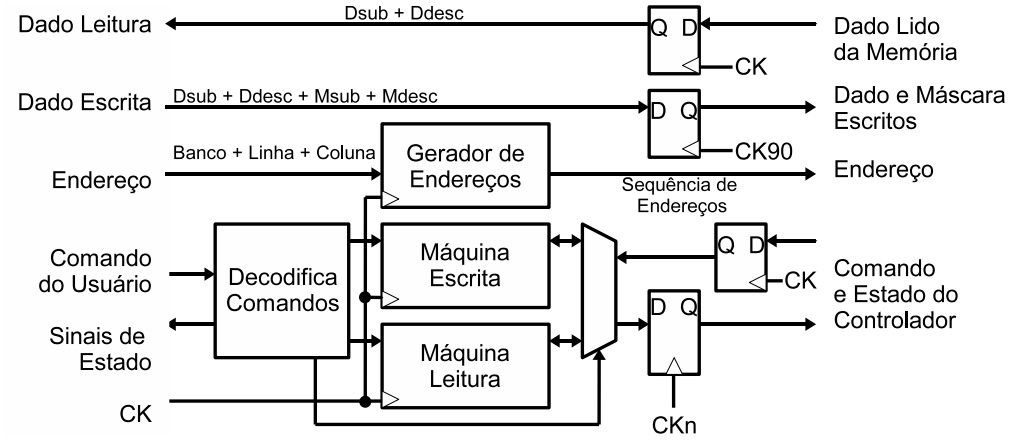

Figura 3.20: Diagrama de blocos simplificado da interface do usuário com o controlador DDR.

Do lado do usuário são colocados os dados para serem transmitidos na borda de subida (Dsub) e de descida (Ddesc) e as máscaras (Msub e Mdesc). O dado e a máscara são amostrados usando a borda de subida de CK90. Os sinais de interface com o controlador DDR que pertencem a outro domínio de relógio que não o do sistema (nesse caso denominado CK), são sincronizados. Os comandos transmitidos ao controlador são amostrados na borda de subida de CKn. As partes principais que compõe esse módulo são (Figura 3.20): máquina de leitura de dados, máquina de escrita de dados, decodificador de comandos, sincronização dos dados e geração de endereços.

Nesse projeto, assume-se que o relógio do sistema é CK, assim todos os sinais da interface com o usuário que comunicam com o resto do sistema devem ser condicionados corretamente para o domínio de relógio CK. No texto que segue será descrito o funcionamento e a arquitetura das partes integrantes da interface com o usuário.

Os sinais presentes na interface são mostrados na Figura 3.21. A interface com o controlador DDR é definida pela implementação do controlador feita pela ferramenta MIG. Essa implementação assume que a interface de memória DDR possui 64-bits de dados, 8-

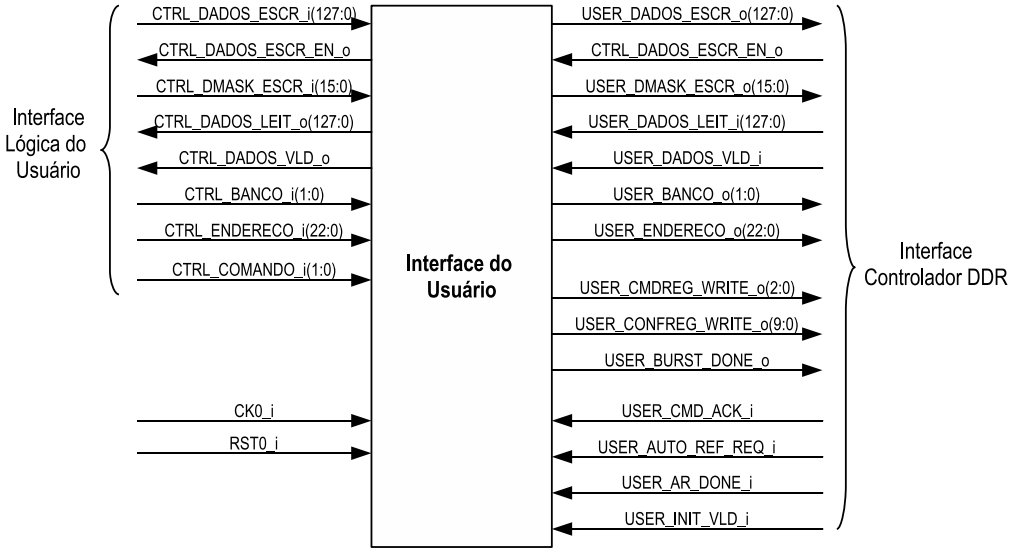

Figura 3.21: Sinais da interface com o usuário.

bits de máscara de dados, 13-bits de endereços (13 linhas), 3-bits de comandos e 2-bits de banco de memória. A entrada da interface do usuário usa 23-bits de endereço (13 linhas e 10 colunas concatenados), 128-bits de dados (2 x 64-bits por ciclo de relógio), 16-bits de máscara de dados (2x 8-bits por ciclo), 2-bits de comandos e 2-bits de banco de memória. Essa nomeclatura de sinais será usada nas descrições feitas nas seções seguintes.

3.2.2.1 Máquina de leitura

A máquina de leitura é formada por uma parte de controle que realiza o acesso à memória externa usando o controlador DDR. O diagrama de estados da máquina de leitura é mostrado na Figura 3.22. O estado inicial ESPERA_CMD mantém a máquina em es-

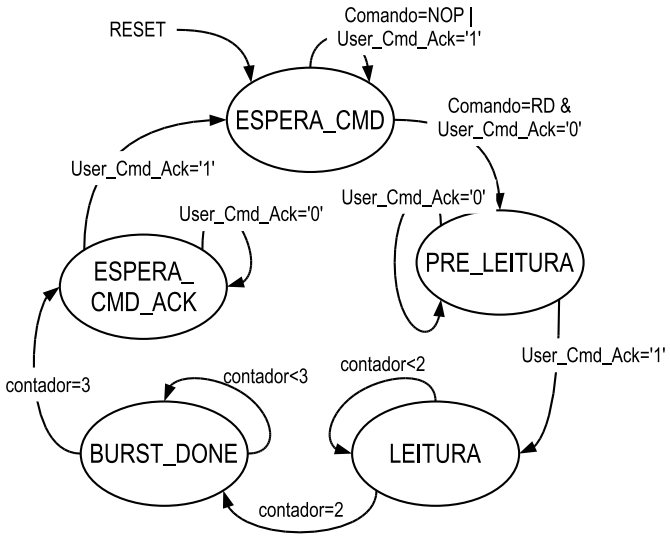

Figura 3.22: Diagrama de estados da máquina de leitura.

pera até que um comando de leitura seja recebido e que o controlador DDR esteja livre

(User_Cmd_Ack='0') para realizar outra operação. O comando de leitura é enviado ao controlador no estado PRE_LEITURA, em LEITURA é feita a espera pela latência de leitura programada da memória (CL) e no estado BURST_DONE o comando de leitura é terminado. Essa máquina realiza somente um único ciclo de burst de leitura da memória por comando recebido.

A máquina de leitura opera em um único domínio de relógio, todos os sinais recebidos do ou enviados ao controlador DDR são sincronizados no topo da interface do usuário.

A Figura 3.23 mostra o diagrama de tempo do processo de leitura de dados na interface do usuário. Os endereço (linha + coluna), o banco e o comando de leitura são colocados

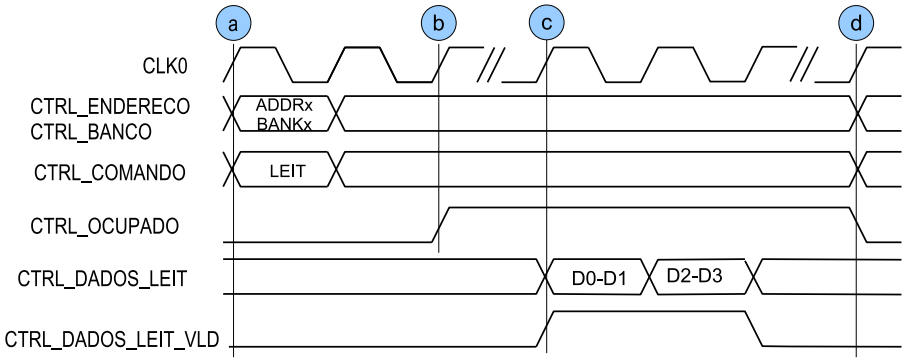

Figura 3.23: Diagrama de tempo da leitura de dados da interface do usuário.

na subida de CLK0 (marcador *a*); a interface sinaliza o início da operação pelo aviso de ocupado (marcador *b*); a interface sinaliza que os dados dados D0-D3 (2 x 128-bits) estão disponíveis para o usuário levantando o sinal CRTL_DADOS_LEIT_VLD (marcador *c*); ao final da operação de leitura, o sinal de ocupado é abaixado (marcador *d*). Na implementação autal da interface, com o controlador ativando a linha, fazendo a leitura e desativando a linha, o ciclo completo de leitura dura 20 ciclos.

3.2.2.2 Máquina de escrita

A máquina de escrita é formada por uma máquina de estado que realiza o acesso à memória externa durante uma escrita de dados. Funciona de forma semelhante à máquina de leitura, fazendo um ciclo de burst de escrita na memória a cada requisição do sistema. Outra diferença é que essa máquina também faz a inicialização da memória através do controlador DDR. O diagrama de blocos está ilustrado na Figura 3.24. Esse módulo faz a inicialização da memória com a configuração de BL=4 e CL=2.

A Figura 3.25 mostra o diagrama de tempo do processo de escrita de dados na interface do usuário. Os dados D0-D1 (128-bits) são colocados juntamente com o endereço (linha + coluna), o banco e o comando de escrita na subida de CLK0 (marcador *a*); a interface sinaliza o início da operação pelo aviso de ocupado (marcador *b*); a interface solicita os dados D2-D3 (128-bits) levantando o sinal CRTL_DADOS_ESCR_EN (marcador *c*); ao final da operação de escrita, o sinal de ocupado é abaixado (marcador *d*). Na

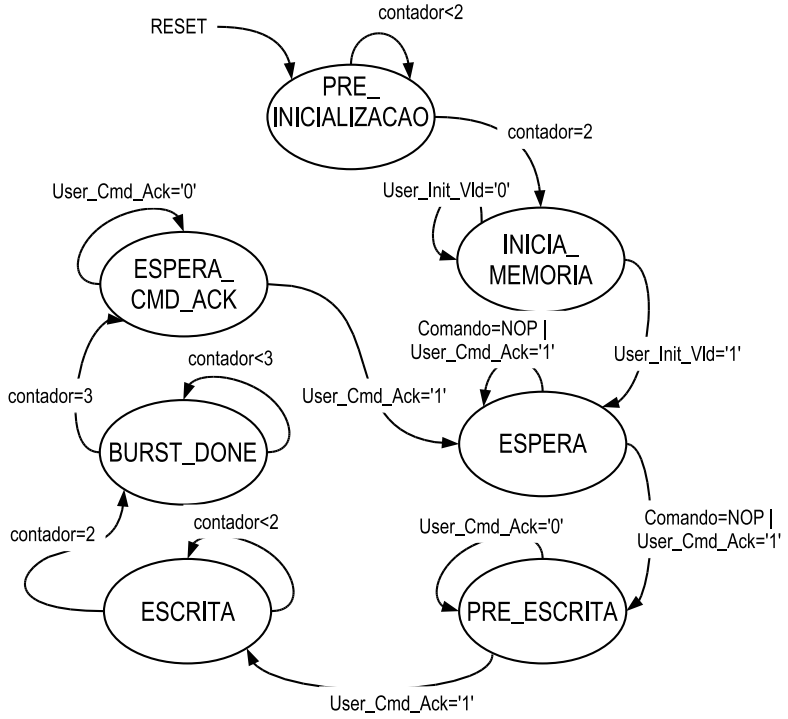

Figura 3.24: Diagrama de estados da máquina de escrita.

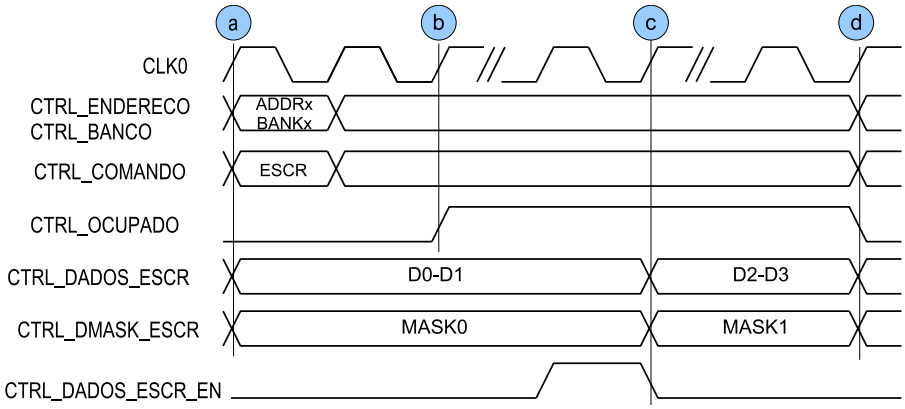

Figura 3.25: Diagrama de tempo da escrita de dados da interface do usuário.

implementação autal da interface, com o controlador ativando a linha, fazendo a escrita e desativando a linha, esse ciclo de escrita leva 19 ciclos.

3.2.2.3 Decodifica Comandos

O módulo de decodificação de comandos interpreta os comandos enviados pelo usuário e faz o controle da máquina de escrita e de leitura. Inicialmente, quando o sistema é inicializado, esse módulo aciona a máquina de escrita para que seja feita a inicialização da memória DDR.

A entrada de comandos do decodificador aceita os comandos de escrita e leitura de um ciclo. Esse módulo pode ser modificado para incluir comandos de repetição dos ciclos de leitura e de escrita na memória.

3.3 IMPLEMENTAÇÃO DO CONTROLADOR DDR

O controlador de memória utilizado nesse projeto é o controlador implementado pela ferramenta Xilinx *Memory Interface Generator* (MIG) (apêndice A). É um elemento complexo de acesso de dados possuindo diferentes domínios de relógio e restrições críticas de implementação em elementos FPGA. Para fazer a abstração entre o sistema e a memória, a interface do usuário é usada no isolamento dos diferentes domínios de relógio e na adaptação dos sinais de interface do controlador DDR ao sistema.

A ferramenta MIG permite a configuração de parâmetros de implementação de um controlador para FPGAs da Xilinx, em formato VHDL com arquivos fontes acessíveis ao projetista. Também gera o arquivo de colocação forçada dos elementos da interface de captura dos dados. Existem algumas limitações em seu projeto, tais como: o controlador gerado não suporta latências de leitura de 2,5 ciclos de relógio; o controlador não executa transações de dado com repetições de 2 e de 8, somente o faz para rajadas de tamanho 4 e; o controlador gerado utiliza elementos lógicos instanciados da biblioteca do fabricante diretamente no código fonte.

O controlador foi gerado para a placa Digilent XUPV2P (*Xilinx University Program*) que possui um FGPA Xilinx XC2VP30 contendo 13.696 slices, relógio de 100 MHz e um pente de memória DIMM DDR SDRAM com capacidade de até 1 GB. A latência da memória utiliza é de CL=2 para o relógio de 100 MHz, o que a caracteriza como uma PC2100 (mais conhecida por DDR200). A interface de memória dessa placa é formada por 64-bits bidirecionais de dado DQ, 8-bits bidirecionais de amostrador DQS e 8-bits de máscara de dados DM. A memória utilizada é do tipo *dual-rank*, contendo dois conjuntos de oito chips de memória. Portanto existem dois conjuntos de habilita (CKE) e seleção de chip (CSn).

Juntamente com os arquivos fonte VHDL do controlador, a ferramenta MIG gera um arquivo de configuração da implementação do controlador para a placa atual. Esse arquivo

contém as informações de ES do FPGA e as informações de restrições de posicionamento do controlador. Também são geradas informações sobre o relógio do sistema, que nesse caso é o oscilador externo de 100 MHz. A sintaxe utilizada para forçar o posicionamento dos elementos lógicos de captura de dados e para determinar os pinos de interface é mostrada no exemplo abaixo:

```
NET "DDR_DQ<0>" LOC = C27;
INST ".../fifo0_bit0" LOC = SLICE_X0Y153;
INST ".../fifo0_bit0" BEL = G;
```

A instrução LOC determina que o sinal DDR_DQ[0] deve ser conectado ao pino de ES C27 do FPGA. Essa informação é passada pelo projetista, que verifica com a documentação da plataforma utilizada qual pino de ES do FPGA está conectado à memória. Para forçar a colocação dos registradores de captura dos dados, a instrução LOC determina que fifo0_bit0 deve ser colocado no registrador G do CLB com coordenadas X0 e Y153 da matriz do FPGA. A ferramenta de síntese que realiza as etapas de mapeamento e de roteamento utiliza essas instruções para colocar o circuito no FPGA. A Figura 3.26 ilustra uma captura de tela do programa *FPGA Editor* contendo: o IOB com pino de ES DDR_DQ[0], os registradores de captura na borda de subida (denominados RAMs Subida) e os registradores de captura de dados na borda de descida de DQS.

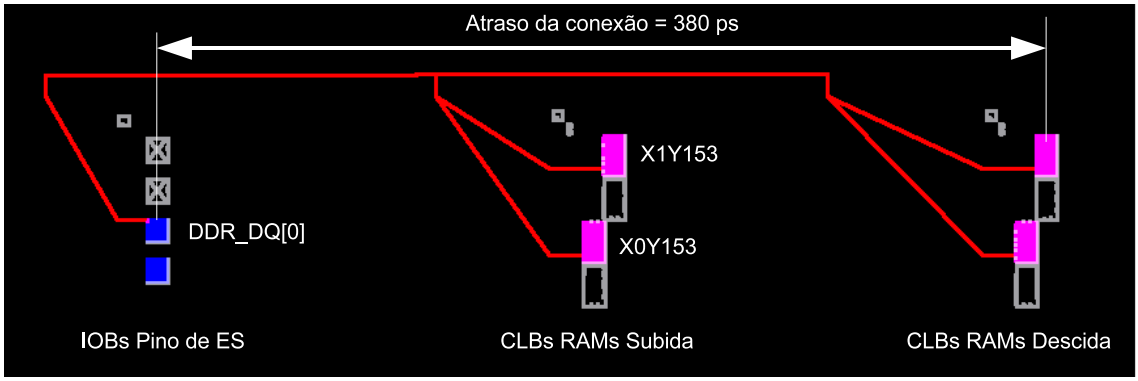

Figura 3.26: Registradores de captura com mapeamento forçado próximos ao pino de ES.

No caso ilustrado, o maior tempo de propagação do sinal de dado até o registrador, incluindo o tempo de propagação do buffer de entrada (1,4 ns) mais o tempo de setup do registrador (373 ps), é igual a 2,153 ps. O caminho de roteamento é ilustrado pelo traço vermelho. Com a colocação forçada dos registradores, o maior atraso da conexão entre o buffer de entrada e o registrador é igual a 380 ps. Os CLBs X0Y153 e X1Y153 formam em conjunto a FIFO de captura do bit de dado DQ[0], usando a borda de subida de amostragem. Esses resultados foram obtidos a partir da análise estática de temporização do circuito posicionado e com as conexões roteadas para o dispositivo FPGA alvo. Essa análise é feita utilizando as próprias ferramentas de verificação do fabricante do FPGA. As informações de temporização do circuito são especificadas para a ferramenta de síntese e de implementação a partir das linhas de código abaixo:

```
NET "MASTER_CLK" LOC = AJ15;
TIMESPEC "TS_MASTER_CLK" = PERIOD "MASTER_CLK" 10 ns HIGH 50%;
```

O sinal de relógio principal do sistema, gerado externamente ao FPGA, é denominado de MASTER_CLK e está conectado ao pino de ES AJ15. A instrução TIMESPEC especifica para a ferramenta de síntese que o período do relógio externo (MASTER_CLK) é de 10 ns com duty-cycle igual a 0,5. Essa instrução é importante para que a ferramenta possa efetuar a análise de temporização do circuito. Verificando se os caminhos críticos presentes no sistema poderão gerar violações de *setup* e *hold* ou se existem corridas no circuito.

Inicialmente foram encontradas dificuldades na implementação do controlador a partir da ferramenta MIG como: problemas de funcionamento da ferramenta; dificuldade na configuração dos pinos de interface com a memória externa e; falta de repetibilidade do controlador gerado a cada nova implementação. Esses problemas resultaram num maior tempo de implementação correta do controlador num circuito real. Então foi proposto projetar uma interface de teste para o controlador implementado em placa FPGA, para assim proceder alterações no projeto do controlador. A implementação do controlador de memória como um IP projetado para reuso é dificultada pelo uso das restrições de colocação explicadas anteriormente. Como o projeto do controlador objetiva sua portabilidade para outros dispositivos e placas, a principal tarefa desse trabalho é projetar um controlador parametrizável, otimizado para a sua implementação em diferentes plataformas. Inicialmente, projetou-se um ambiente para verificação da implementação do controlador em placa. Para simplificar a comunicação com o controlador, foi projetada uma interface de usuário com comandos de escrita e leitura de um único acesso.

No próximo capítulo desse trabalho são reportadas as alterações no projeto do controlador de memória DDR para reúso, bem como as etapas de implementação em FPGA e verificação dos parâmetros funcionais de projeto.

4 PROJETO DO CONTROLADOR PARA REÚSO

Esse capítulo discute questões relacionadas ao reúso no desenvolvimento do controlador de memória DDR SDRAM para sistemas implementados em tecnologia FPGA. O desenvolvimento de SoCs é baseado no reúso de núcleos de IP. Além disso, o tempo de desenvolvimento e de validação de sistemas complexos pode ser reduzido com o uso de descrições de hardware em alto nível. Entende-se por alto nível uma descrição de hardware em nível de transferência de registradores. A integração de módulos descritos em código RTL é uma forma rápida de construção de um protótipo para implementação e verificação utilizando um dispositivo FPGA.

A interface de dados entre o controlador DDR e a memória externa é extremamente sensível à atrasos e possui restrições críticas de temporização. Tal circuito não pode ser completamente implementado a partir de uma descrição genérica e reutilizável. Existe uma dependência em relação à tecnologia utilizada para gerar o protótipo, ou o circuito final. Essa dependência está ligada aos sinais da interface de dados, que operam à dupla taxa de amostragem em alta frequência (de 100 MHz a 200 MHz), e requerem de restrições críticas de temporização para sua implementação num circuito real.

4.1 FLUXO DE PROJETO

Memórias DDR SDRAM possuem sinais de interface e funcionamento padronizados, seja módulos de memória tipo DIMM ou dispositivos simples soldados em placa. O projeto do controlador de memória objetiva construir um núcleo reutilizável e reconfigurável, adaptado às necessidades do sistema. O uso de linguagens de descrição de hardware como VHDL ou Verilog traz benefícios, tais como:

- Utilizar uma descrição do sistema em alto-nível, propiciando uma descrição de código legível;
- Ser uma descrição de sistema independente da tecnologia utilizada, que pode ser reutilizada com pouco retrabalho caso elementos lógicos não sejam instanciados de uma biblioteca específica;

- Permitir a verificação do projeto por simulação de alto nível antes da implementação em nível de portas lógicas.

Apesar de existirem tais vantagens, o código HDL não foi criado originalmente para ser uma entrada para geração de hardware. Muitas construções da linguagem podem não ser suportadas pelas ferramentas de síntese, bem como muitos circuitos podem não ser descritos corretamente. As ferramentas de síntese utilizam sub-conjuntos de linguagem para traduzir a entrada de código num circuito lógico. Em se tratando de circuitos lógicos complexos, o projetista de hardware deve utilizar certas estratégias para criar um código fonte que possa ser sintetizado corretamente. O conhecimento completo dos processos da ferramenta de síntese e o conhecimento das características do FPGA alvo é importante para descrever corretamente o hardware. Da mesma forma, os parâmetros de temporização e posicionamento para o circuito lógico devem ser conhecidos, a fim de alcançar as especificações de funcionamento do sistema.

O fluxo de projeto de IPs orientado para concepção de um ASIC é proposto por Keating e Bricaud no Manual de Metodologia de Reúso (KEATING; BRICAUD, 1999). É orientado para o desenvolvimento e verificação de soft e hard IPs. Para o desenvolvimento de núcleos *firm* em FGPAs propõe-se o fluxo de projeto ilustrado na Figura 4.1. Esse fluxo é composto por cinco etapas de projeto e cada etapa acompanha um nível de verificação diferente, desde funcional até a implementação em placa. As etapas de projeto propostas são: (1) definição dos requisitos do sistema e especificação técnica dos sub-módulos; (2) concepção do projeto, test bench e especificações de temporização; (3) descrição HDL da entrada de projeto; (4) síntese lógica do circuito e; (5) implementação física do circuito e validação na placa de prototipação.

A entrada de projeto é a descrição do sistema em linguagem de hardware. Ela é estruturada usando sub-conjuntos de linguagem para a ferramenta de síntese, após a definição dos requisitos do sistema e especificações técnicas dos módulos. Essa entrada de código pode ser de duas formas: estrutural ou definida topologicamente, instanciando blocos pré-definidos a partir da biblioteca de modelos fornecida pelo fabricante e; comportamental, descrevendo o funcionamento do módulo num nível maior de abstração de hardware.

Na etapa de síntese lógica, o código de entrada é traduzido em elementos de hardware por inferência ou utilizando componentes da biblioteca do fabricante do dispositivo utilizado. A última etapa antes da realização do protótipo é a implementação física. Nessa etapa, os componentes instanciados são mapeados no dispositivo e são aplicadas as restrições de posicionamento e de temporização. Essas restrições influenciam no posicionamento dos componentes e no roteamento interno das conexões entre componentes.

Em paralelo ao fluxo de projeto do núcleo, cada etapa de desenvolvimento é acompanhada por um diferente estágio de verificação. A verificação em diferentes níveis de projeto objetiva assegurar que o produto de cada etapa atingiu as especificações necessárias para iniciar a etapa seguinte de projeto. Segundo (BUSHNELL; AGRAWAL, 2002),

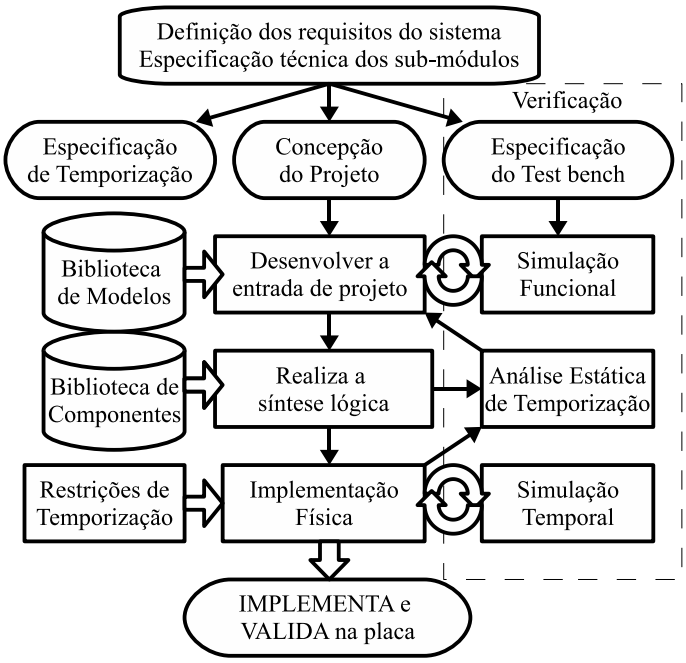

Figura 4.1: Fluxo de projeto de um núcleo *firm* para FPGA.

o comportamento e funcionamento de um circuito é analisado por duas duas categorias de teste, classificados em: teste funcional e teste estrutural. O teste funcional é necessário para verificar o funcionamento do projeto, em nível de comportamento lógico e de transferência de dados. Já o teste estrutural (ou teste de hardware) objetiva assegurar que a estrutura do circuito funciona corretamente, verificando se todos os elementos lógicos e as interconexões foram implementados corretamente. O teste envolve o desenvolvimento de algoritmos de teste e assume modelo de faltas, que não serão abordados nesse trabalho. Portanto, o teste funcional será denominado de verificação enquanto que o teste estrutural de validação.

O primeiro nível de verificação do projeto é do código de entrada, assegurando o funcionamento correto do núcleo. É feito a partir de *test benchs* de simulação do comportamento do sistema, gerando as entradas e saídas para o núcleo. Essa etapa de verificação é precedida pela especificação do ambiente de validação do projeto, que deve prever todos casos possíveis de utilização e configuração (ou parametrização) do núcleo. As próximas etapas de verificação são a análise estática de temporização e simulação temporal. Através da análise estática de temporização (do inglês *Static Timing Analysis* ou STA) é avaliado se o resultado de síntese atende às especificações de temporização estipuladas no início do projeto do núcleo. A última etapa de verificação do núcleo é a simulação temporal, utilizando as informações de atraso de roteamento do sistema mapeado no dispositivo. Essas informações são geradas pela ferramenta de síntese e armazenadas num arquivo de formato padrão de atraso (*Standard Delay Format* ou SDF).

Cada etapa de verificação durante o desenvolvimento do núcleo pode ser bem sucedida ou não. Deve-se prever durante o projeto do núcleo o retorno à uma etapa anterior,

para correção de erros ou acerto do código HDL, correção do arquivo de restrições ou modificação dos parâmetros da ferramenta de síntese e implementação.

O controlador de memória DDR SDRAM utiliza componentes de uma arquitetura específica à qual está sendo implementado. Tais componentes são instanciados no código de entrada como blocos, usando a biblioteca do fabricante, ou como código HDL para serem inferidos pela ferramenta de síntese. Os componentes instanciados de bibliotecas não podem ser inferidos pela ferramenta sem ter acesso à biblioteca específica ou usando uma biblioteca de outro fabricante. Dessa forma, para implementar o controlador num dispositivo diferente, se deve projetar novamente o núcleo do controlador de memória. Isso requer que o código de entrada seja modificado, substituindo os componentes dependentes da tecnologia utilizada.

Outra questão importante sobre o reúso do núcleo *firm* é sobre as restrições de posicionamento e de temporização. Na seção 3.1 foi mostrado que a implementação do controlador utilizando um dispositivo da família Virtex-2 Pro requer o posicionamento específico dos elementos de ES. Essas definições estão contidas no arquivo de restrições UCF. Qualquer alteração nesse arquivo requer um conhecimento detalhado da arquitetura interna do núcleo, além do conhecimento da arquitetura interna do dispositivo FPGA alvo e da placa usada para implementar o protótipo. Deve-se conhecer além da pinagem de interface com a memória DDR, fazer um mapeamento manual dos elementos lógicos de captura de dados lidos da memória externa. Isso faz com que a reutilização do IP seja impraticável, a menos que o projetista faça uma análise detalhada de funcionamento e da arquitetura interna do núcleo.

O objetivo principal do reúso de núcleos no projeto de sistemas complexos é reduzir o tempo de desenvolvimento. Inserir um núcleo pré-projetado no sistema é um atalho que elimina etapas de especificação e de desenvolvimento do núcleo e reduz significativamente a etapa de verificação. Além disso, qualquer alteração no projeto do controlador necessita de uma nova etapa de teste e verificação, para assegurar o funcionamento correto. Além do esforço de modificação do projeto, deve-se prever o esforço (e tempo gasto) com as etapas de verificação.

Essas características de posicionamento e de temporização necessárias ao funcionamento do núcleo caracterizam o firm-IP. Nesse trabalho é discutida uma modificação do projeto do controlador de memória DDR com a finalidade de isolar os elementos dependentes de tecnologia, facilitando o processo de substituição e alteração. Também é discutida aqui a modificação do uso do arquivo de restrições, substituindo as restrições de posicionamento por restrições de temporização. Isso facilita a alteração do projeto, dispensando o entendimento detalhado do funcionamento e da arquitetura interna do núcleo.

4.2 ARQUITETURA DO CONTROLADOR DDR

O projeto de um controlador de memória DDR SDRAM para sistemas implementados em FPGA objetiva a criação de um núcleo de alto desempenho para transferir dados com uma memória externa. A interface física com a memória possui elementos lógicos específicos dependentes da tecnologia. O projeto do controlador deve levar em consideração a análise detalhada de funcionamento e as características físicas e construtivas de tais elementos lógicos. Essa consideração já é discutida anteriormente por (OLLOQUI et al., 2004), que faz uma análise dos elementos dependentes de tecnologia para FPGAs dos fabricantes Altera e Xilinx.

A implementação do controlador de memória em outra plataforma requer modificações de projeto e de implementação do circuito. A arquitetura proposta para implementar o controlador de memória baseia-se na separação dos elementos dependentes de tecnologia em um único módulo do tipo firm. Dessa forma, essa arquitetura reduz o esforço de entendimento do código para proceder as alterações e propõe interfaces padrão para o módulo firm.

A arquitetura do controlador de memória é dividida em três módulos importantes, ilustrados na Figura 4.2: o controle, a interface de dados e o módulo de ES. Seu projeto objetiva a criação de um controlador genérico parametrizável para ser adaptado à diferentes sistemas. Do lado da interface com a memória DDR, é necessário configurar: a largura do barramento de dados (DQ) e o número de amostradores (DQS); a largura do barramento de endereços (ADDR); o número de pares de relógio (CK e CKb), seleção de chip (CSb) e habilita relógio (CKE) e o tamanho da rajada de leitura e escrita de dados.

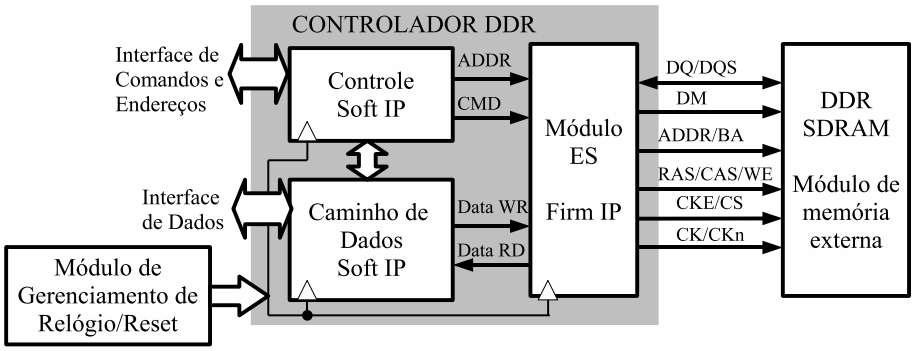

Figura 4.2: Diagrama de blocos da arquitetura do IP do controlador de memória DDR.

Os três módulos principais do controlador são:

- Controle – responsável pelo controle de acesso aos dados armazenados na memória e funcionamento do controlador. Faz a tradução dos comandos gerados pelo sistema (interface de usuário) para comandos para a memória externa. Também traduz a entrada de endereços para o esquema linha e coluna de acesso aos dados na memória. Faz o controle dos ciclos de escrita e leitura, o tempo de latência dos

dados, a inicialização da memória SDRAM e realiza funções de manutenção dos dados como o *auto-refresh* periódico. Esse módulo é escrito em código HDL do tipo *soft*;

- Caminho de dados (*datapath*)– os dados escritos e lidos na memória externa passam por esse módulo, que contém os registradores de escrita de leitura. Esse módulo é descrito em HDL do tipo *soft*;

- Entrada e Saída – módulo que contém os elementos de interface física com a memória externa, como buffers de entrada, saída e de terceiro estado. Tanto os buffers quanto os registradores DDR estão presentes nos IOBs do FPGA. Esse módulo é descrito em HDL do tipo *firm*.

Além dos módulos internos do controlador de memória, é necessário um módulo de controle de geração de relógio de sincronismo e resets. Cada módulo do controlador de memória têm uma função e interfaces bem definidas. Os sinais de interface com a memória DDR SDRAM são padronizados pela norma JEDEC (JEDEC, 2003). As modificações que são apresentadas nessa interface referem-se ao número de elementos de memória, que podem ser agrupados em paralelo para aumentar a largura dos barramentos de dados ou de endereçamento.

Os sinais da interface do módulo de ES com os módulos de controle e de caminho de dados (Figura 4.2) podem ser padronizados. Isso facilitaria a troca desse módulo por outro com mesma função mas desenhado para uma tecnologia diferente. A padronização dessa interface vem sendo discutida por alguns grupos de trabalho e empresas, a fim de facilitar a troca do módulo de ES entre fabricantes. A interface com o sistema também pode ser padronizada pois existem sinais elementares que devem existir como: barramento de dados de leitura e de escrita com largura igual a 2 x DQ cada; barramento de máscara de dados com largura igual a 2 x DM; barramento de endereçamento e sinais de relógio de sincronismo. Os sinais que podem ser alterados são o barramento de comandos para o controlador e os sinais de estado do controlador. Os sinais de amostragem DQS somente são utilizados na interface com a memória DDR, gerados internamente no módulo de ES. Internamente ao código VHDL de descrição dos módulos do controlador foram colocados trechos de código para verificação durante a síntese do controlador. São denominados de *asserts* e têm a função de verificar se os parâmetros utilizados pelo projetista na configuração do controlador são corretos.

O texto que segue explica o funcionamento de cada módulo do controlador, reportando as melhorias realizadas ao projeto original de interface com a memória DDR externa ao chip.

4.2.1 Módulo de Controle

O controle do controlador de memória foi aproveitado inteiramente do projeto original gerado pela ferramenta MIG. Algumas pequenas alterações foram necessárias para remover alguns registradores instanciados no código fonte e substituí-los por código genérico. O controle é responsável por interpretar o comando recebido da interface do usuário e gerar os sinais de endereço e comando para a memória externa. A memória amostra sinais de controle na borda de subida de CK. Portanto, esse módulo opera com o relógio negado CKn, para enviar comandos e endereços à memória externa na borda de descida de CK. Outra alteração feita no controle é a geração do sinal de configuração da linha de atraso para controlar o intervalo de tempo tSD. Com esse sinal partindo do controle, é possível implementar a uma rotina de calibração da linha de atraso a partir da inicialização do controlador, após a inicialização da memória.

A largura de bits dos sinais de endereço, banco de memória e sinais de seleção de memória externa foram parametrizados através do uso de *generics* na entidade do controle. O controle faz interface com o módulo de ES, através dos sinais de endereço (ADDR e BA), de comando (CASn, RASn, WEn, CKE e CSn) e de controle da linha de atraso. Também gera o sinal de habilitação da escrita para o caminho de dados, que ativa os buffers de terceiro estado para escrever nos barramentos de dado DQ, amostragem DQS e máscara DM. A Figura 4.3 ilustra o diagrama de tempos que sintetiza os níveis dos sinais de comando para realizar uma escrita (ESCR), leitura (LEIT), ativação (ACT) e desativação (PRE) de linha e banco de memória.

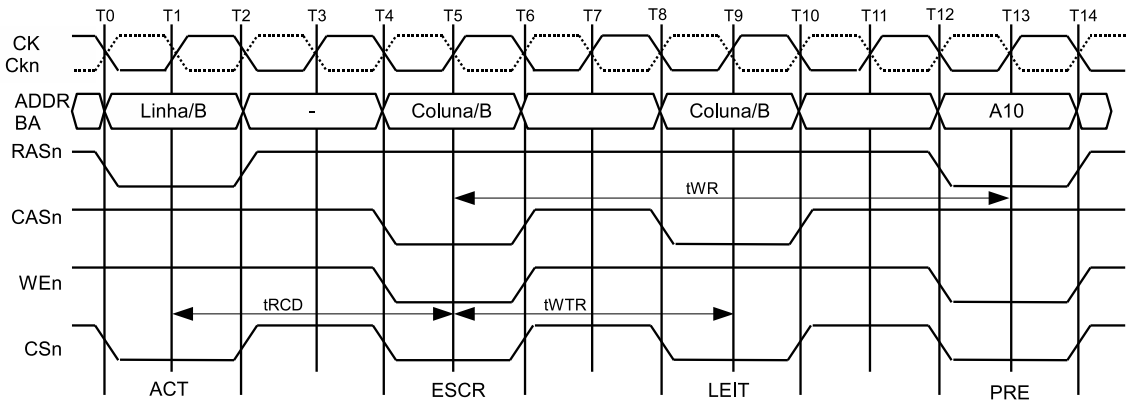

Figura 4.3: Diagrama de tempo dos sinais de controle e endereço.

O sinal de habilita do relógio CKE deve estar sempre ativo para que a memória interprete os comandos. Uma escrita ou leitura na memória somente pode ser realizada após o comando ACT quando decorrido o intervalo de tempo tRCD. O intervalo mínimo de tempo entre um comando de escrita e o de leitura é determinado por tWTR. Antes disso, o comando de leitura é interrompido. Para acessar outra linha da matriz de memória, a linha ativa e banco devem ser desativados através do comando PRE. Após receber o co-

mando de leitura, a memória executa uma instrução PRE sem interromper a leitura depois de *x* ciclos, onde *x* é o número de pares de dados lidos. No caso de uma escrita, antes de enviar o comando PRE é necessário esperar tWR após a última transição de DQS. O bit de endereço A10 em nível alto determina que todos os bancos devem ser desativados, já em nível baixo desativa somente o banco selecionado.

A máquina de estados finitos (MEF) que implementa o controle contém 13 estados e 36 transições. Existem algumas limitações na implementação dessa máquina de controle, que não foram resolvidas nesse trabalho. Uma limitação é que essa máquina não implementa acesso em rajadas de 2 ou 8 palavras e também não pode operar com latências de CAS de 2,5 ciclos. Outra limitação é que o controle não implementa nenhum caminho direto de transição entre escrita e leitura ou vice-versa sem passar pelos comandos de PRE, fazendo o *pre-charge* automático como está definido na norma JEDEC. Os parâmetros de intervalo de tempo entre operações da memória, como os intervalos tRCD, tWTR e tWR, são calculados por contadores. A saída de cada contador passa por um comparador que verifica se o valor atual da contagem é igual ao valor programado (por constantes no código VHDL), gerando sinais de estado para a MEF de controle. Por exemplo, o estado ACTIVE_WAIT espera que o intervalo tRCD seja satisfeito antes de iniciar uma leitura ou escrita. Tipicamente, esse intervalo é igual a dois ciclos de relógio da memória.

4.2.2 Módulo de Caminho de Dados

Tanto o dado escrito na memória como o lido passam pelo caminho de dados. Esse módulo contém os registradores tipo DDR usados na escrita da memória. É organizado em dois sub-módulos principais: o de leitura da memória e o de escrita na memória. O caminho de escrita é responsável pelo controle dos buffers de terceiro estado que conectam o controlador no barramento de dados com a memória externa. Também é esse sub-módulo que controla os sinais de máscara de dado para a memória. O caminho de escrita é implementado com registradores tipo FDCE (flip-flops tipo D com clock-enable e reset) com inversão local do relógio de sincronismo. O dado é enviado à memória nas transições de subida e de descida do relógio com fase de 90° (CK90). A Figura 4.4 ilustra o diagrama de blocos do caminho de escrita de dado, juntamente com o sinal de controle do barramento (habilita escrita), e a conexão de saída com o barramento de dados da memória (área pontilhada).

O pipeline de 4 estágios na entrada de dados é utilizado para sincronizar os dados da entrada do controlador com o tempo de ativar a linha da memória mais o intervalo de tRCD. Todos registradores utilizados foram implementados em código VHDL para inferência. A entrada de dado do lado do sistema possui largura de 2n-bits, que são transmitidos à memória externa com largura de n-bits multiplexados pelo registrador DDR. O registrador DDR de saída de dado é instanciado no módulo de ES, diretamente da biblioteca do fabricante Xilinx, denominado FDDRRSE. Os registradores de escrita de dados

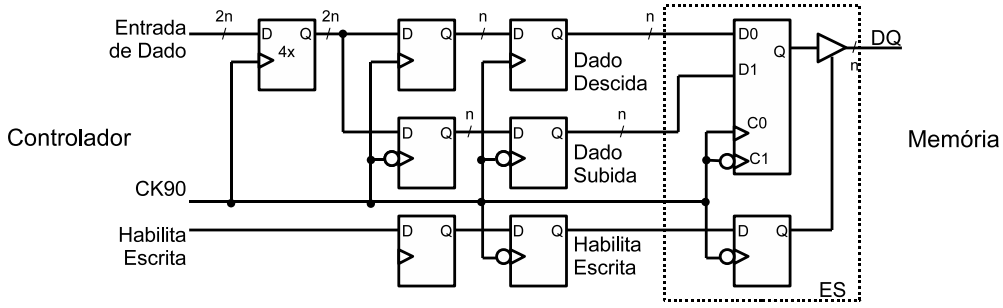

Figura 4.4: Caminho de escrita de dados na memória.

foram descritos de forma parametrizável, formando conjuntos de de n-bits de dados para cada sinal de amostragem DQS.

O circuito de captura de dados também foi modificado a fim de possibilitar a parametrização do controlador de memória. Originalmente, o caminho de leitura do controlador gerado pela ferramenta MIG contém elementos lógicos instanciados diretamente da biblioteca do fabricante Xilinx. Esses elementos são instanciados no código para permitir o posicionamento forçado do controlador no FPGA, como foi detalhado na seção 3.1.4. Esse é o maior desafio do projeto de um controlador de memória genérico: substituir os elementos lógicos instanciados por código genérico reutilizável sem afetar seu funcionamento. Nesse circuito, os registradores e memórias dedicadas (chamadas RAM16X1D) são substituídos por código VHDL genérico.

Novas diretrizes devem ser passadas para a ferramenta de síntese e implementação para orientar o posicionamento correto dos elementos lógicos de interface de dados. Existe aqui uma alteração de forma de projeto, que anteriormente estava orientado para forçar o posicionamento exato dos elementos lógicos no FPGA. O novo projeto requer, com a re-estruturação do controlador, que sejam utilizadas diretrizes de temporização. Dessa forma, são delimitados intervalos máximos de propagação dos sinais internamente. A ferramenta de síntese deve ser capaz de fazer o posicionamento dos elementos lógicos e o roteamento das conexões de forma a respeitar os limites de temporização impostos pelo projetista.

O circuito de leitura da memória externa é formado por duas FIFOs, uma para captura na borda de subida de DQSd e outra para a borda de descida. O sinal de amostragem é atrasado no módulo de ES, que contém a linha de atraso, gerando o sinal DQSd. A inversão local é feita para chavear os registradores usando tanto a borda de subida quanto a de descida de DQSd. Existe um sinal de realimentação externa ao FPGA, denominado Loop, que utiliza uma trilha com comprimento igual ao caminho de ida e volta dos sinais à memória externa. Essa trilha é utilizada para gerar o sinal de habilitação da geração de endereços das FIFOs e também para habilitar a escrita. A Figura 4.5 ilustra o diagrama de blocos simplificado do circuito de leitura de dados.

As FIFOs são utilizadas para isolar os dois domínios de relógio diferentes: do lado

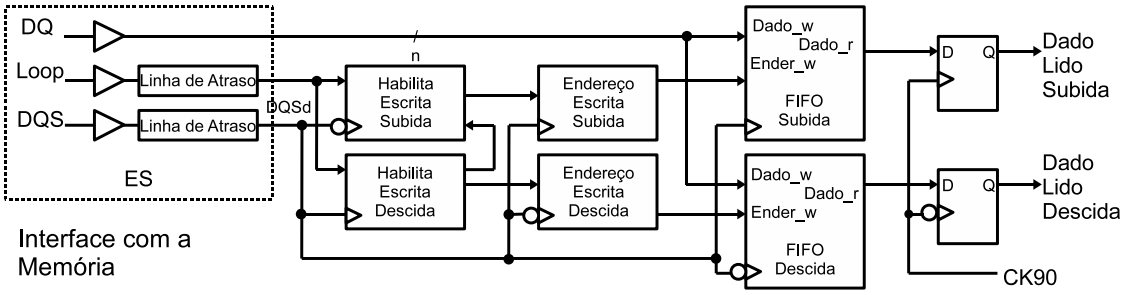

Figura 4.5: Caminho de leitura de dados na memória.

da escrita na FIFO, os registradores são sincronizados por DQSd; já do lado da leitura os registradores são sincronizados por CK90. A saída de dado (dado_r) das FIFOs é assíncrona, mas a geração do endereço de leitura é sincronizada por CK90. Um circuito de comparação dos ponteiros de leitura e de escrita nas duas FIFOs gera o sinal de incremento dos ponteiros de leitura. Esse circuito não está representado na Figura 4.5. Os ponteiros de escrita e leitura são gerados utilizando contadores tipo *gray*, para reduzir a probabilidade de meta-estabilidade na troca do endereço entre domínios de relógio.

O uso de uma FIFO para capturar dados externos é preferível ao uso de registradores por não se saber exatamente qual é a relação de fase entre o relógio local e o externo. No caso do circuito apresentado na Figura 4.5, a relação de fase desconhecida acontece entre os relógios DQSd e CK90. Para essa análise, vamos segmentar o período do relógio CK90 em 4 intervalos de tempo iguais. Supondo que o período do relógio seja de 10ns, a primeira região está entre 0 e 2,5ns, a segunda entre 2,5ns e 5,0ns e assim sucessivamente. Supondo que os sinais de dado DQ e DQS cheguem ao FPGA com muito atraso, na quarta região, a amostragem dos dados usando CK90 pode sofrer violações de hold. Nesse caso o dado deveria ser registrado na borda de descida de CK90, ou seja, utilizando CK270. As FIFOs de leitura implementadas para captura na subida e na descida contém quatro registradores tipo-D para cada bit de dado DQ.

4.2.3 Módulo de Entrada-Saída

O módulo de ES é a parte do controlador que contém os elementos lógicos instanciados no código a partir da biblioteca do fabricante. Seu projeto objetiva o arranjo dos elementos lógicos da interface com a memória de forma que possam ser modificados ou trocados. Dessa forma, a implementação do controlador para outras arquiteturas ou dispositivos é simplificada. A possibilidade de parametrização desse módulo também é importante. A Figura 4.6 ilustra o diagrama de blocos simplificado do módulo de ES com os sinais de interface com a memória externa e com os módulos de caminho de dados e de controle.

A interface física com a memória externa contém os buffers de terceiro estado tipo SSTL_2 para conectar o controlador ao barramento de dados. Os sinais de dado e de

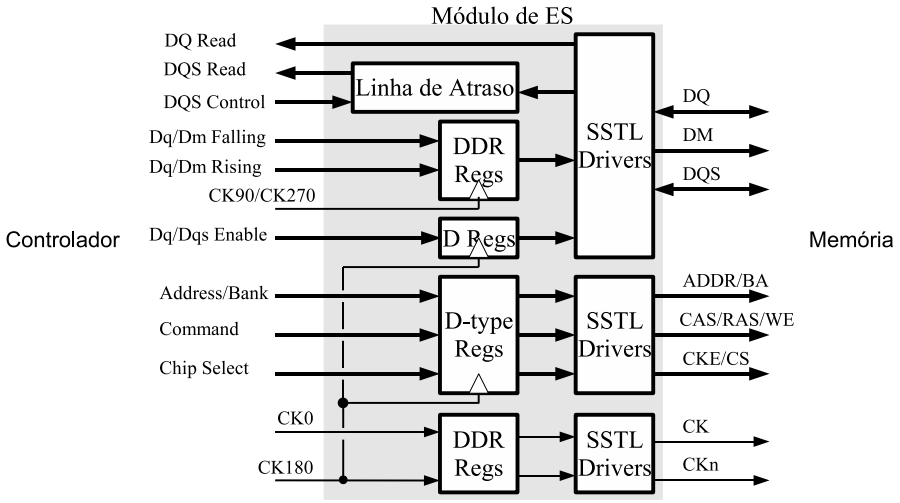

Figura 4.6: Diagrama de blocos do módulo de ES.

máscara são alinhados às transições de subida e de descida de CK90. Já o sinal de amostragem é alinhado com as transições do relógio CK enviado à memória externa. Os sinais de endereço e de controle são sincronizados com registradores tipo D usando a borda de descida do relógio, ou seja, usando o relógio complementar CKn. A geração do relógio para a memória é feita por esse módulo, que utiliza registradores DDR para gerar os relógios com as bordas de transição alinhadas e variação mínima.

O projeto da linha de atraso para o controlador de memória é baseado no circuito apresentado na Seção 3.1.3. Uma série de LUT4s é conectada e o módulo de controle faz a configuração do atraso. O cálculo da quantidade de atraso que deve ser inserido ao sinal de amostragem DQS é detalhado na Seção 3.1.2, resultando na relação tSD = (tDQSQ + tDV)/2 considerando o caso ideal. O único efeito da placa sobre os sinais é a variação entre dado e amostragem tPCI. Os parâmetros utilizados nesse cálculo são referentes à memória DRAM utilizada e ao dispositivo FPGA. A Tabela 4.1 resume as especificações dos fabricantes.

Tabela 4.1: Relação de parâmetros para cálculo de tSD'.

Origem	Parâmetro	Valor (ps)
Sistema	Período do relógio (tCK)	10.000
	Variação da placa (tPCI)	50
DDR SDRAM	Janela de dado válido (tDV)	3.250
	Variação dado-amostragem (tDQSQ)(Max)	500
Controlador	Tempo de set (tS)	373
	Tempo de hold (tH)	-46
Cálculo de tSD' = (tDQSQ + tDV)/2 -tS +tH -tPCI		1406

O valor de atraso do amostrador, considerando as variações do controlador, da memória e da placa, é denominado tSD'. A janela de dado válido é calculada a partir da

relação tDV = tQH - tDQSQ, onde tQH = tHP - tQHS representa o tempo máximo em que qualquer bit de dado permanece estável após uma transição de DQS. Das informações do fabricante de memória Micron (MICRON, 2003) para chip de memória DDR SDRAM tipo DDR200, 100 MHz em CL2, tem-se: tHP = min(tCL,tCH) = 0,45 tCK e; tQHS = 0,75 ns (Max). Dessa análise tem-se que o valor de tDV = 3,25 ns e o tempo de propagação total para para a linha de atraso é de tSD' = 1406 ps.

A linha de atraso é implementada em dispositivos Virtex-2 Pro usando elementos LUT4. O tempo de propagação máximo de uma LUT4 é de 275 ps e o atraso de roteamento entre elementos combinacionais pode variar de 20 a 300 ps. Portanto, o atraso de cada elemento combinacional mais o roteamento varia entre 295 ps e 575 ps, ou seja, 435 ps $\pm$ 32,2%. O intervalo de tSD' pode ser implementado com 4 LUT4s, totalizando tSD' $\pm$ 32,2%. Na seção de implementação do controlador e análise temporal estática será discutida o efeito de variação do tempo de propagação da linha de atraso sobre a captura de dados lidos da memória.

4.3 ELEMENTOS LÓGICOS DEPENDENTES DE TECNOLOGIA

Os elementos lógicos do controlador que são dependentes da tecnologia da plataforma utilizada são descritos nessa seção. São eles: registradores DDR, drivers de ES tipo SSTL-2, elementos lógicos de linha de atraso e controlador de fase e sincronização de relógio. O circuito que gera o relógio do sistema é responsável por gerar também os relógios em fases de 90°, 180° e 270° para o controlador DDR. Esse circuito deve ser implementado externamente gerando o relógio para todo o sistema. As boas práticas de projeto de SoC requerem que o relógio de sincronismo e os resets do sistema sejam gerados a partir de um módulo centralizado, localizado no nível de topo do sistema (KEATING; BRICAUD, 1999).

O controlador de memória DDR foi implementado utilizando dispositivos FPGA do fabricante Xilinx. Como não foram feitos testes utilizando tecnologia Altera, não será feita a análise detalhada dos elementos de tecnologia para esse fabricante. Os elementos dependentes de tecnologia são:

- Sinalização SSTL_2 – todos sinais de interface da memória DDR SDRAM são padronizados conforme a norma JEDEC (JEDEC, 2002). Possui nível elétrico de 2,5 V. Esse padrão de interface está presente nos dispositivos FPGA das famílias: Xilinx Spartan-3, -3A e -3E, Virtex-2 e -2 Pro, Virtex-4 e Virtex-5. Quando da implementação do circuito lógico, o projetista deve especificar o tipo de interface, informando à ferramenta de síntese e implementação. Essa informação é adicionada ao arquivo de restrições UCF, utilizado pela ferramenta durante a etapa de implementação do circuito para o dispositivo FPGA;

- Registradores DDR – a escrita e leitura de dados na memória externa passa pelos registradores de dupla amostragem DDR. A ferramenta de síntese da Xilinx pode inferir registradores DDR de entrada para circuitos implementados nos dispositivos Spartan-3, -3A e -3E, Virtex-2 e -2 Pro, Virtex-4 e Virtex-5. Já os registradores de saída não podem ser inferidos e devem ser instanciados no código de entrada utilizando elementos FDDRRSE, FDDRCPE ou ODDR (Virtex-4 e -5) (ISE 8.1I SOFTWARE MANUAL: LIBRARIES GUIDE, 2007);

- Deslocamento em Frequência – um circuito para controle do relógio de sincronismo deve gerar quatro relógios defasados a partir do relógio do sistema. Dependendo da forma em que os relógios são gerados, esse circuito deve ser configurado para atender à todo o circuito lógico. Os dispositivos Xilinx Spartan-3, Spartan-3E e Spartan-3A, Virtex-2 e Virtex-2 Pro, Virtex-4 e Virtex-5 possuem controladores digital de relógio chamados DCM;

- Linha de atraso – o circuito utilizado para defasar o sinal de amostragem DQS em relação dos dados DQ é gerado de diferentes formas nos FPGAs da Xilinx. Os dispositivos fabricados recentemente, como Spartan-3A, Virtex-4 e Virtex-5, possuem internamente aos blocos de ES circuitos de atraso absoluto chamados IDELAY (COSOROABA, 2007). Esse circuito gera o atraso constante num sinal de entrada, com resolução de pico-segundos, independente de fatores variáveis do dispositivo como pressão, temperatura e tensão. Cada bloco de ES dos dispositivos contém um circuito IDELAY, que é utilizado nos controladores de memória DDR implementados com essa tecnologia. Já para dispositivos antigos, como os Spartan-3 e -3E e Virtex-2 e -2 Pro, a linha de atraso é implementada a partir de um vetor de células lógicas combinacionais dos CLBs. As células lógicas são posicionadas manualmente pelo projetista, controlando o atraso de propagação do sinal de entrada pelo número de elementos instanciados. As ferramentas de síntese não são capazes de inferir esse tipo de circuito através de um código de entrada genérico. Portanto, a linha de atraso deve ser instanciada manualmente para cada dispositivo, sendo reprojetada quando o circuito for implementado em outra placa ou FPGA.

Esses recursos relacionados acima não podem ser inferidos por uma ferramenta de síntese, de forma que o projetista deve instanciar como blocos dentro do código de entrada. Agrupar tais elementos num único módulo simplifica a edição do código. Tanto os elementos LUT4 quando os registradores tipo D e DDR usados na interface de ES devem ser instanciados no código HDL como um bloco, com todos sinais descritos e conectados pelo projetista. Os buffers de ES também são instanciados no código HDL, mas também podem ser descritos no arquivo de descrição dos pinos de interface do FPGA. Nesse projeto, optou-se por instanciar os buffers de ES no próprio código HDL. No projeto

para reúso, a portabilidade do IP é definida pela possibilidade de implementar o mesmo circuito em diferentes sistemas a partir da mesma descrição. Os elementos lógicos instanciados limitam a possibilidade de reutilizar o IP em outros projetos. Mas devido à impossibilidade das ferramentas de síntese fazerem a inferência dos elementos lógicos dependentes de tecnologia, a instanciação desses elementos faz-se necessária.

Os elementos lógicos relacionados nessa seção não podem ser descritos diretamente por código HDL genérico. Foram isolados num módulo *firm* para facilitar a alteração do projeto. Outros elementos lógicos presentes no circuito de caminho de leitura de dados do controlador (vide Figura 4.5), como as FIFOs, os contadores gray e os registradores de habilitação de escrita, também são instanciados no código de entrada. Tais elementos lógicos foram substituídos por código HDL genérico possibilitando a síntese por inferência. Dessa forma, o caminho de dados que originalmente era uma mistura de módulo *soft* e *firm*, passou a ser completamente *soft*.

Na próxima seção é feita uma análise das alterações feitas no controlador DDR de um ponto de vista de otimização e desempenho. Após as etapas de síntese lógica e implementação no dispositivo FPGA é feita uma etapa de verificação do projeto. Essa etapa é a análise estática de temporização, utilizada para verificar se o projeto do controlador está de acordo com o especificado. Também é utilizada para verificar se as restrições de temporização são adequadas.

4.4 ANÁLISE ESTÁTICA DE TEMPORIZAÇÃO

As alterações de projeto do controlador de memória requerem que seja feita uma análise rigorosa de seu funcionamento, principalmente com respeito às modificações das restrições de temporização. A substituição das diretivas de restrições de posicionamento dos elementos lógicos por restrições de temporização aumenta as chances de portabilidade do IP. Maior é o esforço de projeto do sistema quando diretivas de posicionamento forçado são utilizadas na implementação do controlador de memória. Nessa seção são reportadas as tentativas de substituição das restrições de posicionamento por diretivas de restrição temporal para implementação do controlador. A análise estática de temporização é utilizada como forma de verificação dos critérios de funcionamento do controlador.

A primeira etapa de verificação estrutural é feita após a síntese do núcleo, através da análise dos caminhos críticos entre registradores. Essa análise permite determinar o período mínimo do relógio de sincronização dos registradores do núcleo. Essa análise é importante para verificar a qualidade do projeto, refazendo blocos internos caso não possa ser atingida a especificação inicial. No caso do controlador DDR implementado nesse trabalho, o resumo das informações de análise estática de temporização é mostrado abaixo:

```
Timing Summary:
```

```
---------------
   Minimum period: 4.653ns (Maximum Frequency: 214.915MHz)
   Minimum input arrival time before clock: 6.608ns
   Maximum output required time after clock: 3.483ns
   Maximum combinational path delay: No path found
```

O período mínimo do relógio para o controlador é de 4,653 ns, ou seja, uma frequência de 214 MHz. O menor tempo de propagação de um sinal de entrada até o registrador é igual a 6,608 ns e o maior tempo de propagação de um sinal registrado até a saída do dispositivo é igual a 3,483ns. Essas informações da síntese são importantes como características de funcionamento do núcleo.

Independente de quanto parametrizável pode ser um núcleo, a sua implementação no sistema físico requer uma configuração que não é parametrizável. Quando o núcleo faz interface com os pinos do dispositivo, cada sinal deve possuir uma diretriz de mapeamento no FPGA, seja de posicionamento ou de temporização. O controlador de memória foi implementado com 64-bits de dado DQ, 8-bits de máscara de dados DM e 8-bits de amostragem DQS, além dos sinais de endereço, relógio e comando. As informações de pinagem do núcleo no dispositivo são passadas para a ferramenta usando o parâmetro LOC no arquivo de restrições, como mostrado abaixo para os sinais de amostragem DQS[7] e de dado DQ[63]:

```
NET "DDR_DQS_io<7>" LOC = AH26;
NET "DDR_DQ_io<63>" LOC = AH29;
```

A implementação da linha de atraso composta de LUT4s será feita a partir do posicionamento forçado dos elementos lógicos. Dessa forma, o tempo de propagação pela linha de atraso pode ser controlado com maior exatidão do que se fosse posicionado automaticamente pela ferramenta de síntese. Como descrito anteriormente na seção 3.1.3, o tempo de propagação total da linha de atraso é composto pelo número de elementos lógicos no caminho do sinal mais o tempo de propagação pelas conexões. O tempo de propagação da lógica e dos sinais na linha de atraso implementada é mostrado na Tabela 4.2, obtida pela análise estática de temporização do circuito de atraso.

O tipo de atraso *Tiopi* é o atraso do buffer de entrada do sinal externo DQS[0]. Já *net* e *Tilo* são respectivamente o tempo de propagação pela conexão entre a lógica e o tempo de propagação da lógica (no caso uma LUT4). A linha de atraso implementada contém seis LUT4s em série, configuradas como multiplexadores 2:1. O tempo total de propagação da linha de atraso tpLA é de 5,373 ns. O sinal IOB_DQS_DELAYED_s[0] é um dos oito sinais de amostragem utilizados para capturar os 64-bits de dado no circuito de leitura do controlador DDR. O sinal de amostragem atraso usado no sincronismo da lógica de captura de dados deve ser roteado usando trilhas locais de distribuição de relógio (*local clock*). Deve-se proibir que a ferramenta de síntese interprete esses sinais de sincronismo como relógios globais e faça o roteamento pelas trilhas de relógio centrais do FPGA.

Tabela 4.2: Análise de temporização da linha de atraso.

Tipo de Atraso	Atraso (ns)	Recurso(s) Lógico(s)
Tiopi	1,427	DDR_DQS_io[0]
net	0,338	DQS_IN_s[0]
Tilo + net	0,254 + 0,305	LUT_one + delay1
Tilo + net	0,275 + 0,212	LUT_two + delay2
Tilo + net	0,275 + 0,057	LUT_three + delay3
Tilo + net	0,254 + 0,024	LUT_four + delay4
Tilo + net	0,275 + 0,037	LUT_five + delay5
Tilo	0,254	LUT_six
net	1,386	IOB_DQS_DELAYED_s[0]
Total	5.373ns (3.014ns logic, 2.359ns route)	

Caso isso ocorra, o tempo de propagação do sinal de amostragem atrasado é acrescido de aproximadamente 5 ns[1].

O sinal de seleção é formado por cinco bits que chaveiam os seis multiplexadores, inserindo ou removendo LUT4s no caminho do atraso. A Tabela 4.3 mostra a relação de configuração do circuito de atraso e o tempo de propagação de DQS até DQSd. Esse cálculo não considera o tempo de propagação Tiopi do buffer de entrada.

Tabela 4.3: Possíveis configurações da linha de atraso.

Sinal de Seleção DQS_controle[4:0]	Número de LUT4s	Atraso (ps)
“11111”	6	2.500
“11110”	5	1.941
“11101”	4	1.454
“11011”	3	1.122
“10111”	2	844
“01111”	1	566

Os elementos lógicos do circuito de captura de dados do controlador devem ser posicionados na fronteira mais próxima do FPGA com a memória externa. No projeto para reúso não foram utilizadas informações de posicionamento forçado, mas sim de limites de tempo de propagação. Durante a etapa de mapeamento, posicionamento e de roteamento do circuito, a ferramenta de síntese utiliza informações de tempo de propagação máximo para determinados sinais. O nome dos sinais internos ao núcleo não é alterado quando ele é parametrizado. Dessa forma, a orientação para o posicionamento dos elementos lógicos é passada à ferramenta de síntese atrelada aos sinais internos. Primeiramente é feita uma análise dos sinais de interesse, que deve sofrer as restrições, que nesse caso está limitado

[1]Resultados medidos experimentalmente: adicional de tempo de propagação do buffer de relógio global de 0,050 ns mais 4,732 ns de roteamento pelo FPGA.

ao circuito de captura de dados. O tempo de propagação interno ao FPGA dos sinais de dado DQ e de amostragem DQS deve ser limitado.

O tempo de propagação dos sinais internamente ao FPGA pode alterar o comportamento do circuito, como discutido anteriormente na seção 3.1.4. Com o posicionamento forçado dos elementos de captura de dados na interface de ES mais próxima entre a memória e o FPGA, o tempo de propagação é garantidamente o mínimo. O posicionamento do circuito feito pelo projetista delega à ferramenta de implementação somente a tarefa de rotear as conexões. A relação entre o tempo de propagação dos sinais de dado deve ser mantida igual em relação ao tempo do sinal de amostragem, usado para sincronizar os registradores de captura. Com o uso de informações temporais para controlar o posicionamento e roteamento do circuito na matriz de células lógicas do dispositivo, não se tem uma garantia dos resultados obtidos. A ferramenta de síntese pode não atingir as especificações temporais para o circuito, o que requer etapas recursivas até chegar à implementação final do circuito.

A ferramenta de síntese e implementação Xilinx ISE possibilita limitar o atraso de propagação entre lógica de três formas diferentes: (1) especificando a relação temporal (ou *offset*) entre dois sinais externos ao FPGA, sendo que um deles é utilizado como sincronismo de relógio de captura e o outro como dado lido; (2) especificando o tempo de propagação máximo entre dois grupos de sinais, ou grupos de lógica como registradores, latches e pinos de ES; (3) definindo o máximo atraso de propagação de um sinal.

4.4.1 Definindo um Offset entre dado e amostragem

A relação de fase entre dois sinais assíncronos externos ao FPGA pode ser especificada utilizando a diretiva OFFSET IN BEFORE. A ferramenta utiliza essa diretiva para determinar uma relação de offset entre um sinal externo que é utilizado para sincronizar um registrador de captura de outro sinal externo. Uma tentativa de limitar o tempo de propagação dos sinais DQ e DQS internamente ao FPGA foi utilizando essa diretiva.

Primeiramente, é criado um grupo de PADS (pinos) para os sinais de dado sincronizados com o amostrador DQS[0], denominado DQ_TMG_0. Em seguida é determinada a relação de offset entre os PADS e o sinal de sincronismo dos registradores DDR_DQS_io[0]. As linhas de código mostradas abaixo são inseridas no arquivo UCF:

```
TIMEGRP "DQ_TMG_0" = PADS ( "DDR_DQ_io<0>" "DDR_DQ_io<1>"
         "DDR_DQ_io<2>" ... "DDR_DQ_io<7>" );
TIMEGRP "DQ_TMG_0" OFFSET = IN 1 ns BEFORE "DDR_DQS_io<0>";
```

A diretiva “OFFSET = IN 1 ns” determina que o sinal de dado está adiantado de um intervalo de 1 ns em relação à transição do relógio externo. O resultado do uso dessa diretiva é que a ferramenta busca alinhar o sinal de dado à borda de transição do sinal de sincronismo, respeitando o tempo de setup do registrador. Como existe um circuito lógico combinacional no caminho do relógio (a linha de atraso), o registrador do dado

é posicionado muito distante da fronteira do FPGA. Aumentando dessa forma o tempo de propagação do sinal de dado. A Figura 4.7 ilustra um exemplo de um sinal de dado com registrador de captura mapeado muito distante do pino de ES, com um tempo de propagação de 4,3 ns.

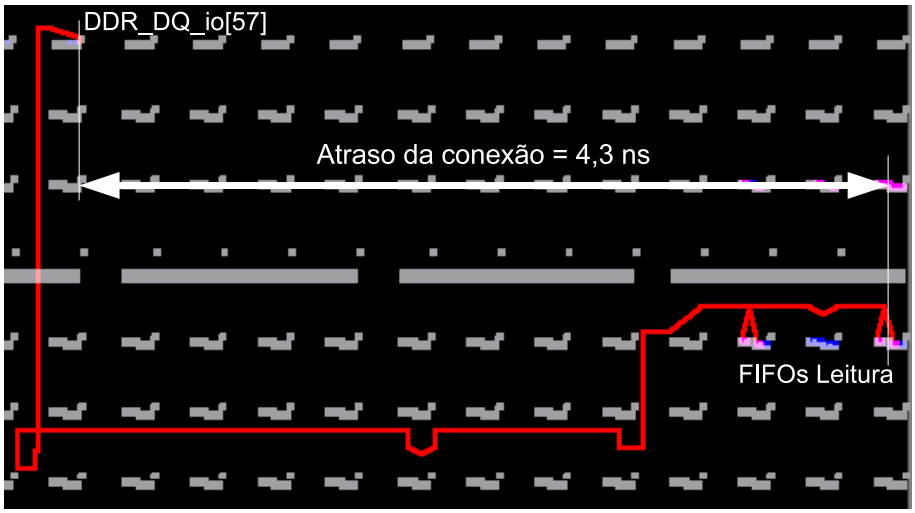

Figura 4.7: Captura do *FPGA Editor*, atraso entre pino de ES e registradores.

Os tempos de propagação dos sinais de amostragem também foram medidos na analise estática de temporização. Os resultados para os atrasos máximo e mínimo dos sinais de amostragem foram, respectivamente, 2.406 ps e 449 ps. Essa variação do tempo de propagação do sinal de sincronismo torna a captura de dados defeituosa.

A definição de atraso usando OFFSET não é eficiente para implementar o controlador de memória. A ferramenta busca aumentar o tempo de propagação do sinal de dado através do roteamento para compensar o atraso da linha de atraso. Para o correto funcionamento dessa diretiva, não deveria haver qualquer tipo de logica entre o sinal de amostragem e a entrada de relógio do registrador de captura. Portanto, outro tipo de restrição de atraso máximo de um sinal deve ser utilizado.

4.4.2 Definindo o atraso máximo entre grupos

Outra forma de controlar o atraso de propagação de um sinal é utilizando a diretiva FROM TO. Dessa forma é possível definir qual é o maior atraso entre grupos de sinais ou de elementos lógicos. A ferramenta utiliza essa diretiva para guiar o processo de mapeamento, posicionamento e roteamento do circuito. As linhas de código abaixo são passadas à ferramenta de síntese, através do arquivo de restrições, para limitar o tempo de propagação:

```
NET "DDR_DQ_io<0>" TNM_NET = DQ_TNM_0;
NET "DDR_DQ_io<1>" TNM_NET = DQ_TNM_0;
...
NET "DDR_DQ_io<7>" TNM_NET = DQ_TNM_0;
```

```
TIMESPEC "TS_DQ_0" = FROM "DQ_TNM_0" TO "RAMS" 350 ps;
```

Os sinais de dado registrados com o mesmo sinal de amostragem são agrupados formando um *TimeGroup*, denominado no exemplo acima de DQ_TNM_0. A instrução TIMESPEC é utilizada para informar que o tempo esperado entre os sinais do grupo DQ_TNM_0 e qualquer RAM é de 350 ps. Com essa informação espera-se que a ferramenta "aproxime" as FIFOs de leitura de dados com a interface do FPGA. Os resultados experimentais de pior caso na implementação do controlador utilizando as restrições acima é mostrado na Tabela 4.4.

Tabela 4.4: Resultado da implementação em FPGA com FROM_TO.

Timegroup	Maior atraso (ns)	Grupo de dados	Pior caso
TS_DQ_0	1,499	0 – 7	DDR_DQ_io[7]
TS_DQ_1	0,753	8 – 15	DDR_DQ_io[15]
TS_DQ_2	1,318	16 – 23	DDR_DQ_io[23]
TS_DQ_3	0,742	24 – 31	DDR_DQ_io[27]
TS_DQ_4	0,812	32 – 39	DDR_DQ_io[32]
TS_DQ_5	0,803	40 – 47	DDR_DQ_io[44]
TS_DQ_6	1,212	48 – 55	DDR_DQ_io[48]
TS_DQ_7	1,293	56 – 63	DDR_DQ_io[61]

Foram criados oito grupos de sinais de dado para cada sinal de amostragem da memória. Pelos resultados apresentados na Tabela 4.4, se pode concluir que o posicionamento e roteamento dos elementos de captura não é satisfatório para implementar o controlador em FPGA. A Figura 4.8 mostra uma captura de tela com o pior de roteamento para os registradores do sinal de dado DDR_DQ_io[7].

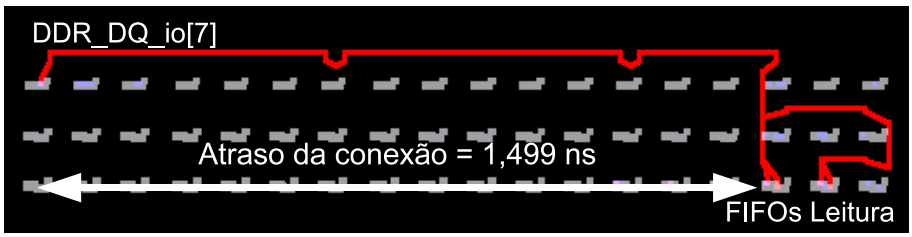

Figura 4.8: Captura do *FPGA Editor*, atraso entre pino de ES e registradores.

4.4.3 Definindo o máximo atraso de um sinal

O atraso máximo de um sinal declarado no código de entrada HDL pode ser definido utilizando-se a diretiva MAXDELAY. A ferramenta de implementação do circuito digital para a tecnologia alvo utiliza essa diretiva, instanciada no arquivo de restrições UCF, como guia para o posicionamento dos elementos lógicos e roteamento das conexões. A

instrução para a ferramenta é passada através de uma única linha de código abaixo, mostrada abaixo:

```
NET "u_DDR_CONTROLLER/IOB_DQ_READ_s<*>" MAXDELAY = 350 ps;
```

Nesse caso, a restrição de posicionamento fica aplicada para o sinal de dado após o buffer de entrada IOB_DQ_READ_s. O asterisco é o caso genérico, que aplica a mesma restrição para todos os 64 bits do sinal de dado. A limitação de 350 ps é o menor valor de atraso na implementação, ou o caso ideal, como mostrado anteriormente na Figura 3.26. Esse valor é somente para a o tempo de propagação da conexão e não considera o tempo de setup do registrador nem o tempo de propagação do buffer de entrada do pino do FPGA. A Tabela 4.5 mostra os resultados de tempo de propagação para os sinais de dado, até os registradores de captura, considerando o melhor e o pior caso para cada grupo de amostragem DQS. Também é mostrado o pior e melhor caso de tempo de propagação para cada sinal de amostragem DQS.

Tabela 4.5: Resultado da implementação em FPGA com MAXDELAY.

Timegroup	Atraso do Dado DQ (ps)		Atraso da Amostragem DQS (ps)	
	Maior	Menor	Maior	Menor
DQS[0]	855	323	1266	1245
DQS[1]	552	326	1294	1251
DQS[2]	544	326	1296	1252
DQS[3]	552	326	1274	1222
DQS[4]	739	326	339	311
DQS[5]	525	323	1227	1224
DQS[6]	638	326	1290	1261
DQS[7]	835	323	1152	995

A variação de propagação dos sinais de amostragem é menor que para os sinais de dado. Isso ocorre devido ao algoritmo de roteamento que utiliza linhas de sincronismo local para chavear registradores localizados nas proximidades do pino de ES. Existe uma diferença de tempo de propagação entre os sinais de amostragem e os sinais de dado. Essa diferença deve ser descontada do tempo de propagação total programado para a linha de atraso. Se o tempo de propagação do dado e do sinal de amostragem fossem iguais, provavelmente a captura funcionaria corretamente com o valor de tSD' (calculado na seção 4.2.3). Os resultados apresentados na Tabela 4.5 são os melhores se comparados com as outras duas técnicas apresentadas anteriormente. Esses resultados apresentam menores tempos de propagação pelo roteamento e também são menos invariantes que nos casos anteriores, usando as diretivas OFFSET IN BEFORE e FROM_TO.

A Figura 4.9 ilustra uma captura de tela com o resultado de roteamento para o menor atraso de propagação obtido (326 ps) para o pino de dado DDR_DQ_io[22]. A ferramenta

posicionou os registradores de captura na interface mais próxima com o pino de ES do dado.

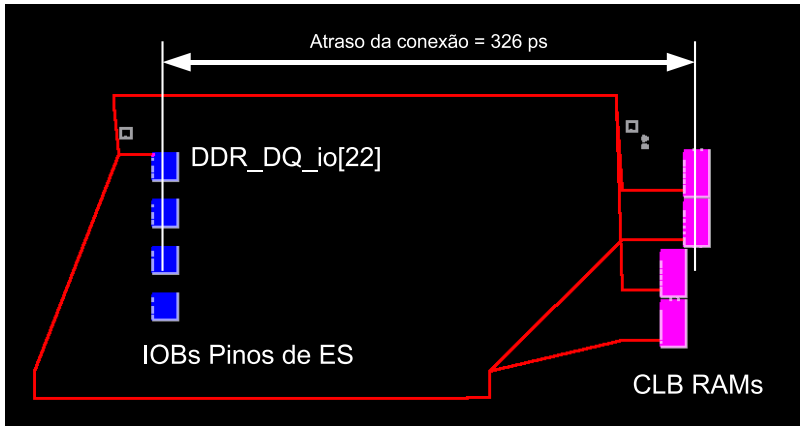

Figura 4.9: Captura *FPGA Editor*, exemplo de menor atraso entre dado e registrador.

O valor de atraso programado para a linha de atraso deve ser modificado a partir dessa análise. Não será considerado o tempo de propagação dos buffers de entrada, que é o mesmo para dados e amostragem (tiopi = 1,427 ns). A diferença existente entre o tempo de propagação do sinal de dado é diferente do tempo de propagação da amostragem. Nesse caso será utilizada a diferença entre a variação média dos sinais de amostragem (tDQSm) menos a variação dos sinais de dado (tDQm). Como o valor de atraso tSD' é o mesmo para todos grupos de amostragem de dado, será feito somente um único cálculo de tSD". Esse cálculo leva em conta todos valores de atraso de propagação, mínimos e máximos, para os oito grupos de amostragem. A variação média dos sinais de amostragem é calculada a partir de (4.1), onde tDQSmax e tDQSmin são valores máximo e mínimo de atraso de propagação para todos os sinais de amostragem.

$$tDQSm = (1/8)\sum_{i=0}^{7}(tDQSi_{max} + tDQSi_{min})/2 \tag{4.1}$$

Para os sinais de dado, a variação média é dada em (4.2).

$$tDQm = (1/8)\sum_{i=0}^{7}(tDQi_{max} + tDQi_{min})/2 \tag{4.2}$$

Esse cálculo resulta que tDQm = 489,94 ps e tDQSm = 1118,69 ps. O novo valor para a linha de atraso é mostrado em (4.3)

$$\begin{aligned} tSD'' &= tSD' + tDQSDQ = tSD' + (tDQm - tDQSm) \\ &= 1406 + (489,94 - 1118,69) = 777,25\ ps \end{aligned} \tag{4.3}$$

O intervalo total de atraso adicionado à amostragem pela linha de atraso é programado através da configuração dos multiplexadores, como mostra a Tabela 4.3. Essa técnica de

geração de atraso com elementos lógicos em série não oferece boa resolução. Portanto deve-se busca uma configuração dos multiplexadores que encontre o valor mais adequado mostrado na tabela. Para o cálculo de tSD" realizado a partir dos resultados apresentados na Tabela 4.5, a melhor configuração para a linha de atraso é com duas LUT4s em série, ou seja, DQS_controle[4:0] = "10111".

Através dessa análise, verifica-se que a implementação do circuito de captura do controlador de memória sobre a plataforma FPGA é possível dentro dos resultados apresentados para a implementação do controlador. Como não existem restrições de posicionamento rigorosas para esse circuito, como as utilizadas anteriormente, pode ser que a ferramenta não seja capaz de posicionar e rotear o circuito corretamente. Essa forma de implementação do controlador é mais simples do ponto de vista de projeto e de portabilidade, mas é suscetível à erros.

A simulação temporal é a etapa seguinte de verificação do controlador. Para isso é utilizado o arquivo SDF gerado após as etapas de posicionamento e de roteamento do circuito no FPGA. Através da simulação usando o arquivo SDF é possível verificar o tempo dos sinais da interface do controlador. Os sinais internos perdem o significado após a geração do arquivo VHDL para simulação, pois o nome dos sinais é alterado. Além disso, essa simulação é muito mais custosa computacionalmente, requerendo muito mais tempo de processamento e dificultando o processo de verificação do núcleo. No caso de circuitos digitais que fazem interface com elementos externos ao FPGA, a forma de verificação completa é a implementação em placa e análise dos sinais lógicos de ES. Essa forma de verificação é mais rápida e eficiente do que a simulação temporal. A última etapa de verificação é a implementação do circuito na placa usando um ambiente gerador de padrões de teste e análise de resultados. Essa etapa é denominada de validação do circuito, que utiliza uma plataforma de desenvolvimento e roda à frequência especificada no início do projeto.

No próximo capítulo será discutida a implementação do controlador numa placa real, o ambiente de validação utilizado e a técnica de geração de padrões de teste.

5 IMPLEMENTAÇÃO E VALIDAÇÃO DO CONTROLADOR

O funcionamento do controlador de memória somente pode ser verificado em placa, num circuito real. Devido às restrições críticas de temporização da interface de dados e da dependência do comportamento do circuito externo ao FPGA, uma implementação em placa pode ter um comportamento diferente do circuito verificado a partir de simulação. Inicialmente foram realizados testes em placa com o controlador gerado pelo MIG, a fim de avaliar a qualidade do ambiente de verificação. Entende-se por ambiente de verificação o módulo que gera os padrões de teste para o controlador e compara com valores esperados. Um módulo de verificação do controlador foi desenvolvido. Esse módulo é utilizado para gerar padrões de dados que, através de escritas e leituras sucessivas, verifica o funcionamento do controlador em placa.

O projeto do IP controlador de memória DDR eliminou as restrições de posicionamento da interface de leitura e substituiu toda a lógica instanciada por código genérico. Essas alterações de projeto necessitam de etapas de verificação e de validação em placa. Durante a verificação, é determinado se as etapas de mapeamento, posicionamento e roteamento do circuito para o dispositivo FPGA foram bem sucedidas. Isso significa que a interface de ES, que implementa o circuito de leitura de dados, não está sujeita à faltas. Também é feita a análise de desempenho e do caminho crítico do circuito resultante da implementação. A validação do IP controlador envolve sua implementação na placa, com o circuito real de verificação, acessando uma memória DDR externa através de escritas, leituras e comparação dos padrões de dados.

Segundo Bushnell e Agrawal (2002), um defeito num sistema eletrônico é uma diferença indesejada entre o hardware implementado e seu projeto. Já um erro é representado por um sinal de saída diferente, produzido por um sistema defeituoso. A falta é a representação de um defeito numa abstração de função do circuito. O defeito é causado por um erro, que é a manifestação de uma falta. A falta é presente no circuito quando existe uma diferença física entre o correto e o incorreto. As faltas podem ser classificadas como permanentes ou aleatórias. As faltas permanentes são caracterizadas por problemas de conexão entre dispositivos numa placa ou entre transistores num chip, dentre outros problemas. Já as faltas aleatórias podem ser ocasionadas por variações de umidade, tem-

peratura e tensão sobre o circuito, por interferência eletromagnética, por violações de temporização em caminhos críticos ou por atividades cósmicas (como incidência de partículas alfa), dentre outras. As faltas aleatórias ocorrem de forma aleatória e possuem um modelo bem conhecido. No caso da implementação do IP controlador de memória, problemas de posicionamento e de roteamento do circuito de ES (descrito anteriormente nas seções 3.1.2 e 3.1.4) podem gerar violações de setup e hold na captura de dados. Isso caracterizaria uma falta com comportamento aleatório presente no circuito de leitura de dado.

Durante esse capítudo do trabalho será descrita a metodologia de validação do IP controlador de memória. Também será feita a análise de algumas técnicas de teste de memória e sua relação com a validação do controlador em placa. Uma arquitetura de teste embarcado no circuito do controlador (*Built-In Self-Test* ou BIST) é proposta para realizar tanto o teste de memória externa quanto a validação da implementação do controlador em placa.

5.1 TESTE DE MEMÓRIA

O aumento crescente de capacidade das memórias baseadas em tecnologia DRAM é causado principalmente pela redução das dimensões físicas dos capacitores utilizados para armazenar cada bit de informação. Para atingir uma cobertura de faltas suficiente o teste de memórias DRAM é orientado para o leiaute físico (BUSHNELL; AGRAWAL, 2002). Memórias DRAM são construídas com capacitores empilhados (*stacked capacitors*) sobre o transistor, que são fabricados em tecnologias de sub-microns (menores que 0,5 μm). Com a evolução das memórias, defeitos gerados por problemas de acoplamento entre células de memória tornaram-se a tendência de defeitos mais comum em memórias DRAM. Isso ocorre pelas alterações tecnológicas de construção dos elementos de acumulação de carga, pela redução das dimensões físicas e pela maior proximidade entre células de armazenamento de carga. A inclusão de linhas e colunas redundantes na matriz de células é uma prática comum na fabricação de memórias. Se não houvesse redundância a produtividade seria nula, pois todo o chip de memória possui defeitos de fabricação. Durante os testes de fabricação do chip de memória é levantado um mapa dos bits com problema. Então, os decodificadores de linha e coluna são refeitos (através de laser ou fusíveis internos) para substituir essas linhas e colunas por outras redundantes que não apresentaram faltas.

O custo do teste por chip de memória deve ser baixo, a fim de manter o preço da memória baixo. O tempo de teste está relacionado diretamente ao custo do teste, quanto maior esse tempo, menor a quantidade de chips testados no mesmo período de tempo. O tempo de teste aumenta exponencialmente com o aumento do número de bits de memória a serem testados. Os algoritmos de teste de memória objetivam alcançar a maior cobertura

de faltas num intervalo menor de execução. Também é necessário que o teste forneça informações de diagnóstico, como o tipo e a localização do defeito na matriz de células.

Testes de memória podem ser aplicados na fábrica, como testes de caracterização e de produção, ou pelo usuário ao longo da vida útil do sistema. Circuitos integrados SoC podem conter um módulo integrado de teste de memória ou mesmo uma rotina implementada pelo processador. Circuitos projetados sem o teste embarcado devem executar uma verificação do sistema por partes, como teste em nível de placa e das interconexões, teste de sistema e teste de memória.

Nesse contexto percebe-se que o teste de memória embarcado no SoC, seja gerado por um módulo específico ou por rotinas do processador, é importante para verificação do funcionamento correto do sistema. Além disso, o uso de partes do próprio SoC para verificação de outras partes é interessante do ponto de vista de economia do teste já que não depende de equipamentos externos.

Uma técnica genérica para teste de memórias externas e as interconexões utilizando a lógica embarcada no chip é apresentada por (CATY et al., 2003). Essa metodologia de teste objetiva uma significativa redução dos custos com teste das placas durante a fabricação. Também é utilizada para melhorar o diagnóstico de defeitos das interconexões entre controlador e memória ocorridas no processo produtivo. Esse trabalho apresenta uma arquitetura chamada E-BIST (*External*-BIST) que é utilizado para gerar diferentes padrões de teste, incluindo algoritmos March. A geração dos padrões de teste é feita de forma programável para testar diferentes tipos de memória, incluindo memórias DDR com largura de barramento de dados de 64-bits.

Outra técnica para o teste da memória externa e das interconexões entre o chip que contém o controlador e a memória é apresentada por (KIM et al., 2004). Nesse trabalho é apresentada uma arquitetura de BIST para memórias tipo DDR, utilizando o controlador para implementar as funções de acesso aos dados em baixo nível. O reúso do controlador de memória reduz o custo do teste embarcado em chip. Esse trabalho classifica memórias externas em três categorias: de latência fixa, com handshake e para ambas. A partir disso é projetada uma arquitetura de BIST compatível com diferentes tipos de memória.

Outra arquitetura de BIST que utiliza algoritmos March para teste da memória DDR e DDR2 é apresentada por (SHEN et al., 2005). Nesse trabalho, a arquitetura proposta utiliza uma estrutura de pipeline para atingir taxas elevadas de frequência no teste de memórias DDR. O teste de memória é realizado *at-speed*.

O uso de algoritmos de teste de memória, como os March, pode ser estendido para testar as interconexões entre o chip e memória. Além disso, a verificação de funcionamento do controlador de memória pode ser feita com o teste de memória. Após as etapas de verificação do projeto do IP controlador e sua integração no SoC, a implementação na placa é seguida por etapas de validação do seu funcionamento. O teste de memória externa pode ser utilizado para validar o controlador implementado em FPGA. Sabendo que a memória

externa não apresenta qualquer tipo de defeito, um padrão de dados é gerado para preencher a memória com informações conhecidas. A validação é feita a partir da leitura do padrão de dados armazenado na memória e comparação com os valores esperados.

5.2 ALGORITMOS MARCH

Algoritmos March são utilizados para detectar e classificar faltas em memórias de diversos tipos, tanto DRAM quanto SRAM. Muitos anos de pesquisa em teste de memória resultaram na criação de diversos algoritmos March de diferentes complexidades. São algoritmos utilizados para verificar a integridade do núcleo de memória, a matriz de células, a lógica de decodificação e o controle da memória. Os algoritmos utilizam operações de acesso aos dados fazendo uma sequência pré-determinada de escritas e leituras na memória. O teste é utilizado para comparar os 1's e 0's lidos da memória. A partir dos resultados de leitura e comparação com o padrão esperado, é feita a análise de integridade da memória.

Algoritmos March podem ser utilizados para detectar faltas em memórias DRAM síncronas com ampla cobertura de faltas. Conforme (VOLLRATH, 2003), o uso de algoritmos March desenvolvidos para realizar o teste de memórias DRAM é eficiente também para memórias do tipo SDR SDRAM. Submetendo a memória à variações de tensão e temporização, foram detectadas faltas com a mesma cobertura que para memórias DRAM. O teste de memórias DDR SDRAM pode ser realizado a partir de algoritmos March, como é mostrado por (SOARES; BONATTO; SUSIN, 2008) e por (SHEN et al., 2005).

A descrição precisa dos testes March foi inicialmente desenvolvida por (GOOR, 1991). A notação utilizada para representar algoritmos March é composta por elementos de endereçamento das operações de escrita e de leitura na memória, como os símbolos $\Uparrow$ para ordem crescente, $\Downarrow$ para ordem decrescente e $\Updownarrow$ para qualquer sentido. Também são utilizados o símbolo *r* para uma operação de leitura e *w* para uma operação de escrita na memória. O padrão utilizado no teste é aplicado para todas as células de memória. Um exemplo de algoritmo simples é o padrão *zero-one* (ou MSCAN) (ADAMS, 2002). A sua representação utilizando a notação de algoritmos March é mostrada na Tabela 5.1, onde o padrão de dados usado no teste é formado por uma palavra e pelo seu complemento, representado nos símbolos $\{y, \bar{y}\}$. Esse algoritmo é composto por quatro elementos de

Tabela 5.1: Algoritmo MSCAN.

M0 :	$\Updownarrow$	(wy)	para todas células, escreve y
M1 :	$\Updownarrow$	(ry)	para todas células, lê y
M2 :	$\Updownarrow$	$(w\bar{y})$	para todas células, escreve $\bar{y}$
M3 :	$\Updownarrow$	$(r\bar{y})$	para todas células, lê $\bar{y}$

marcha, representados de $M0$ até $M3$. Durante a execução de cada elemento é realizada uma varredura completa no espaço de endereçamento de memória.

Qualquer teste de memória deve assumir um "modelo de faltas" para classificar os tipos de faltas encontradas durante o teste de memória. O modelo de falta é uma abstração do efeito físico da falta, representado por algumas categorias de problemas no processo de fabricação das memórias. O algoritmo MSCAN possui uma cobertura de 100% para faltas do tipo *stuck-at* e 100% para faltas de inversão. Não apresenta cobertura para faltas como de decodificação de endereço, de transição, de acoplamento, de acoplamento inverso e de vizinhança, dentre outras. Mais detalhes sobre modelos de faltas pode ser encontrado no capítulo 9 de (ADAMS, 2002).

No princípio do teste de memória, o acesso aos dados era feito por bits, então testes de memória eram orientados para bit. Atualmente memórias são orientadas para palavras (*word oriented*). Da mesma forma, o teste de memórias DDR SDRAM não pode ser feito por células de memória individuais. A interface de acesso ao núcleo DRAM é formada por palavras de 4, 8 ou 16 bits de dado por chip de memória. O teste de memória quando realizado externamente deve ser orientado à palavra de dado. A diferença nesse caso é o padrão de dados utilizado, que deve assumir a possibilidade de interação entre células vizinhas.

Para a verificação da interface de ES do controlador de memória, será utilizado um circuito que implementa o algoritmo de marcha MSCAN orientado à palavra, como apresentado na Tabela 5.1. Esse algoritmo não é suficiente para testar memórias, pela pequena abrangência de faltas, mas pode ser usado para encontrar defeitos de implementação do controlador de memória. O padrão de dados utilizado na verificação do controlador depende do número de bits do grupo de dados DQ controlados por cada amostrador DQS. Para a implementação atual do controlador, a memória externa tem largura de 8-bits de dados por amostrador. O padrão de dados utilizado é mostrado na Tabela 5.2.

Tabela 5.2: Padrão de dado e seu complemento, utilizados na verificação do controlador.

b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	representação
0	1	0	1	0	1	0	1	y
1	0	1	0	1	0	1	0	$\bar{y}$

O acesso aos dados armazenados no núcleo da memória DDR SDRAM é feito por rajadas de tamanhos 2, 4 ou 8. Portanto, o padrão utilizado deve levar em conta a repetição dos bits durante a rajada. Uma memória com saída de dados em tamanho de palavra igual a 8-bits possui um núcleo DRAM com largura igual a 16-bits. Na arquitetura *2n-prefetch*, a palavra lida do núcleo DRAM possui largura igual a 2n-bits, enquanto que na saída do chip de memória a largura é igual a n-bits. Portanto, para verificar a interface de dados do controlador o padrão de dados utilizado no teste é formado por $D_m = y$ e $D_{m+1} = \bar{y}$,

sendo que $D = \{b7, b6, ..., b0\}$. Isso está ilustrado na Figura 5.1.

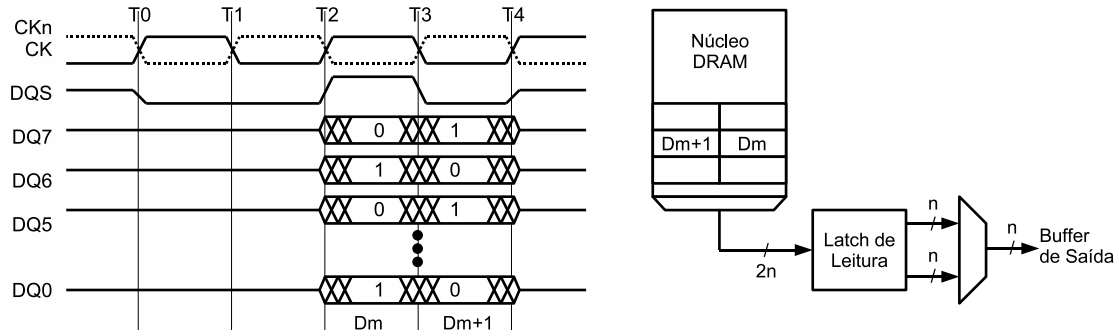

Figura 5.1: Padrão de dados usado no teste do controlador e diagrama de blocos da arquitetura interna da memória DDR SDRAM.

Durante a leitura do padrão de teste escrito na memória, cada bit de dado DQ deve ser alternado (invertido) a cada transição do amostrador DQS. O objetivo principal do teste é verificar se o circuito que implementa o atraso do amostrador e os elementos de captura de dados está implementado corretamente. Portanto, a cada transição de DQS o padrão deve ser invertido para verificar se a amostragem do dado lido está correta. Na Figura 5.1 é ilustrado o diagrama de tempo de uma leitura da memória com rajada de tamanho 2. Isto representa somente um acesso ao núcleo de memória DRAM. Para rajadas de tamanhos 4 e 8, a mesma metodologia deve ser seguida, fazendo a inversão do padrão de dados a cada transição de DQS.

Na próxima seção é detalhada a arquitetura do módulo de BIST utilizado para validação do controlador de memória implementado em plataforma FPGA.

5.3 ARQUITETURA DO BIST

O módulo de hardware utilizado para a validação do controlador na placa é utilizado para gerar a sequência de comandos e o padrão de dados que implementa o algoritmo de marcha. Esse módulo também faz a geração da sequência de endereços para o teste, que pode ser aleatória. A verificação dos dados lidos da memória é utilizada para gerar os sinais de erro de leitura dos dados, para cada grupo de sinais DQ-DQS. O acesso aos dados na memória é feito a partir da ativação de linha e coluna, sendo finalizado com o comando de desativação de linha. O teste completo do controlador também deve verificar o funcionamento correto das funções de ativação e desativação de linha. Durante etapa de validação do controlador em placa, cada acesso é feito por uma sequência completa de comandos ACT + RD/WR + PRE.

A geração de endereços leva em conta duas premissas. (1) Para realizar a verificação da interface de memória do controlador não é necessário percorrer toda a faixa de endereços da memória externa. Um longo teste de memória é suficiente para verificar se o controlador apresenta defeitos de implementação. (2) Algoritmos de marcha como o

MSCAN não necessitam de uma sequência de endereços definida, que pode ser crescente ou decrescente. Portanto, para a verificação do controlador será utilizado um LFSR para geração de endereços aleatórios.

A sequência de endereços gerada por um LFSR obedece à um padrão do tipo pseudo-aleatório, tendo um comportamento aleatório mas repetível. O intervalo de repetição depende do número de bits do gerador de endereços. Caso o LFSR possua uma saída de k-bits, sendo formado por k registradores em série, o intervalo de repetição será de $2^k - 1$ ciclos de relógio. Isso significa que o gerador de padrão terá $2^k - 1$ diferentes estados, sendo que existe somente um padrão com sequência de k '1's consecutivos e $k-1$ '0's consecutivos. Não existe o padrão de k zeros. Por exemplo, o polinômio primitivo de geração de um padrão pseudo-aleatório para $k = 9$ é $x^9 + x^4 + 1$, sendo que o intervalo de repetição é de $2^9 - 1 = 511$ ciclos. A Figura 5.2 ilustra o circuito que com a linha de registradores e conexões de realimentação que implementa o LFSR de 9-bits.

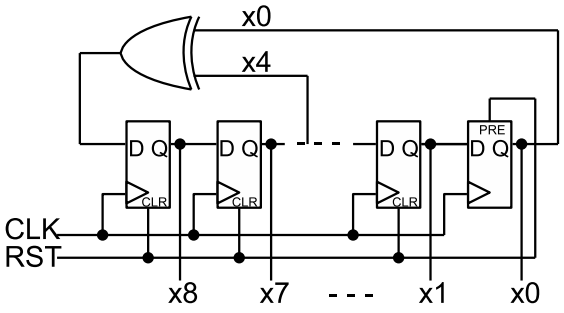

Figura 5.2: Gerador de padrão LFSR.

O tempo total de teste é proporcional ao número de acessos realizados em cada endereço de memória e ao tamanho da faixa total de abrangência de endereços. O número de ciclos necessários para as operações de escrita e de leitura na memória são respectivamente $t_w = 19$ e $t_r = 20$ ciclos de relógio. Esse número é obtido considerando uma rajada de tamanho quatro, mais os ciclos de ACT e PRE para cada acesso e mais os ciclos da interface do usuário. O tempo total de teste, em número de ciclos é dado por (5.1).

$$t = (2t_w + 2t_r)N = (2 \cdot 19 + 2 \cdot 20)511 = 39.858 \tag{5.1}$$

Para um relógio de 100 MHz, o tempo total de teste utilizando um LFSR de 9 bits é igual e $39.858 \cdot 1/100x10^6 = 399\mu s$. Adicionando a esse intervalo o tempo de 200μs de espera para estabilização do circuito DLL da memória, em menos de 600μs é feita a validação do controlador na placa de FPGA.

Os sinais de interface do BIST devem ser parametrizáveis, de forma a ser compatível com as diferentes configurações do IP controlador de memória. Também deve ser não-invasivo, ou seja, não deve alterar o funcionamento do controlador de memória e também não pode interferir na interface com a memória externa, pois esta interface pos-

sui restrições críticas de temporização. O BIST é sincronizado pelo mesmo relógio que o controlador e que a memória externa, caracterizando o teste *at-speed*. Essa metodologia permite o teste do sistema de memória independentemente da implementação do sistema do usuário. O BIST é descrito como um núcleo do tipo soft, podendo ser implementado em outras plataformas FPGA.

O controlador de memória é utilizado para realizar os testes de interface, sendo que o módulo de verificação é conectado à interface do usuário do controlador. A Figura 5.3 ilustra o diagrama de blocos com uma representação simplificada das conexões entre o controlador e o módulo de verificação.

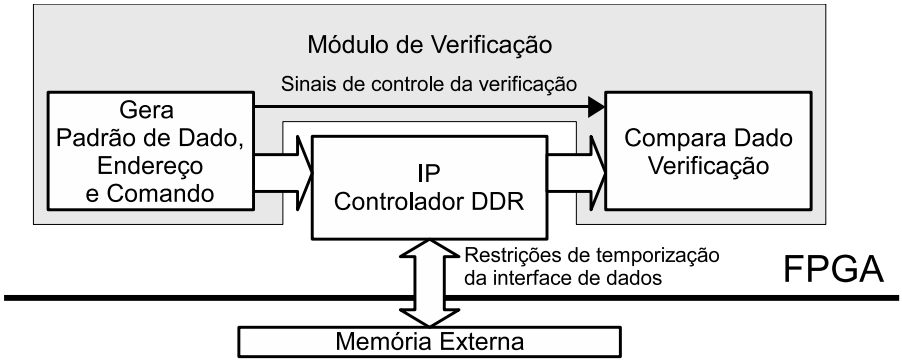

Figura 5.3: Diagrama de blocos do módulo de verificação do controlador.

Dois sub-módulos importantes são responsáveis pela execução do teste: um gerador do padrão de dados, endereço e comandos e o outro, um comparador do padrão lido do controlador. O primeiro sub-módulo gera o padrão de dados, endereço e comando de entrada do controlador e também os sinais de controle para o sub-módulo que compara os dados lidos e gera sinais de estado e erro. O segundo sub-módulo faz a verificação do padrão lido do controlador e gera os sinais de resultado, indicando a ocorrência de defeitos. O controlador, as interconexões e a memória externa representam um único circuito sob verificação (do inglês *Circuit-Under-Verification* ou CUV). O uso do módulo de verificação conectado à interface do usuário não interfere nos sinais com restrições críticas de temporização.

O controle do modulo de verificação deve conter toda a lógica necessária para gerar os sinais de endereço, comando, padrão de dados e controle do teste. Os sinais de endereço para o controlador de memória são gerados diretamente pela saída do LFSR. Um comparador é utilizado para detectar o início da sequência do padrão pseudo-aleatório. A partir da comparação com o início da sequência de dados são gerados os sinais de controle e o padrão de dados, de acordo com o algoritmo March utilizado. A cada nova repetição da sequência, o comando para o controlador deve ser alterado de leitura para escrita. O padrão de dados também é alterado, mas a cada duas repetições da sequência. A Figura 5.4 ilustra um diagrama de blocos com os elementos principais da parte de controle do módulo de verificação.

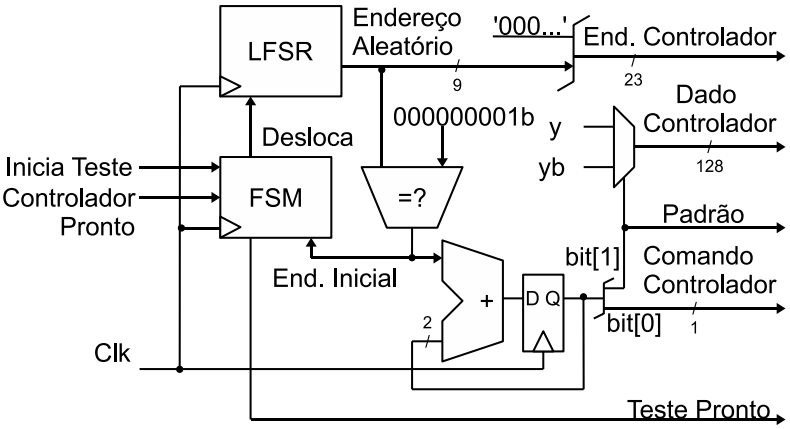

Figura 5.4: Diagrama de blocos da parte de controle do BIST.

Um somador de dois bits é utilizado para gerar os elementos de marcha $M0$ até $M3$. O bit menos significativo (bit[0]) é utilizado para gerar o comando de leitura ou escrita para o controlador. Esse bit altera a cada nova sequência do LFSR. O bit mais significativo, bit[1], é utilizado para controlar a saída de padrão de dados para o controlador. A Figura 5.5 ilustra o diagrama de blocos completo do BIST, contendo o circuito de comparação do padrão lido da memória externa. Os sinais de estado do BIST são utilizados pelo circuito que controla o teste, são eles: inicia teste, teste pronto e compara bytes. O BIST roda a mesma taxa de relógio que o controlador de memória e pode ser parametrizado a partir da largura do barramento de dado e de endereço. Nessa implementação, a largura de dados de entrada e saída é de 128-bits, que representa duas palavras de 64-bits por ciclo do relógio Clk.

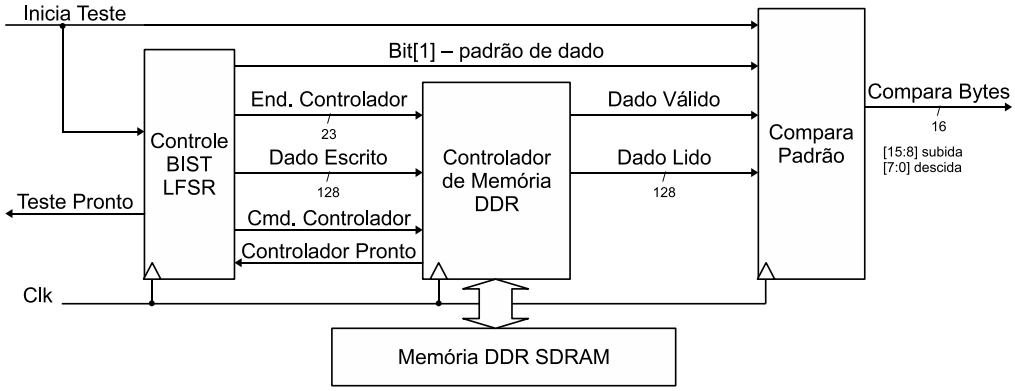

Figura 5.5: Diagrama de blocos do BIST.

O módulo de comparação do padrão lido da memória verifica se duas as palavras de dado lidas por ciclo de relógio são compatíveis com o padrão esperado. O padrão esperado é informado pelo bit[1], sinal “padrão de dados”. A comparação é feita por grupo de bit de dados amostrados pelo mesmo sinal DQS, gerando 16-bits de sinal de comparação. Quando um erro é detectado, o bit referente à palavra com erro é levantado

para ’1’ lógico durante o ciclo de leitura do dado. O sinal de dado válido é utilizado para habilitar a comparação.

Os resultados de síntese do controlador DDR e do BIST para um dispositivo FPGA da família Virtex-2 Pro do fabricante Xilinx são mostrados na Tabela 5.3

Tabela 5.3: Resultados da síntese do controlador DDR.

Sub-módulo	Frequência Máxima (MHz)	Slices	Flip-Flops	LUT4s	Linhas de de Código
Controlador DDR	220	511	1.188	997	3.386
Interface do Usuário	143	138	113	259	759
BIST	225	58	31	115	797
Todo Projeto	143	1.673	1.620	2.317	8.108

5.4 VERIFICAÇÃO FUNCIONAL

O objetivo principal da simulação comportamental é verificar o funcionamento da máquina de controle, verificação dos intervalos de tempo de acesso à memória e o funcionamento do controlador e do BIST. Essa etapa de verificação é feita a partir de *testbeches* escritos em linguagem de descrição de hardware. São utilizados para gerar padrões de teste para o controlador e também para o BIST. A verificação por simulação é feita por etapas, do nível baixo do sistema até o topo. As partes do controlador DDR foram verificadas separadamente, durante o processo de escrita do código do controlador. Em seguida, com o desenvolvimento do BIST, uma nova verificação foi realizada com o BIST integrado ao controlador. Os resultados de simulação apresentados nessa seção foram obtidos com a ferramenta Mentor Modelsim XE (Xilinx Edition).

Essa etapa de verificação se detém somente aos testes funcionais do controlador, para garantir seu funcionamento a partir da descrição inicial. A Figura 5.6 ilustra a forma de onda resultante de simulação funcional do controlador, mostrando algumas das etapas de inicialização da memória.

Para a verificação funcional, utilizou-se um modelo DDR SDRAM funcional descrito em linguagem VHDL, fornecido pelo fabricante Micron. O modelo de memória é conectado à interface de memória do controlador. Um padrão de dados e comandos foi gerado para excitar o controlador e fazer a verificação da sequência de dados lidos da memória. Essa etapa de verificação pode ser muito custosa se feita visualmente, a partir da interface gráfica do simulador. Além disso, a verificação visual é sujeita a erros e depende muito da atenção de quem observa as formas de onda. Para melhorar a verificação, são utilizados comandos de asserção durante o teste, que fazem a verificação dos dados lidos e escritos, e também dos intervalos de tempo necessários para o funcionamento correto da memória. A verificação dos intervalos de tempo como tMRD e tRP (ilustrados na Figura 5.6) são

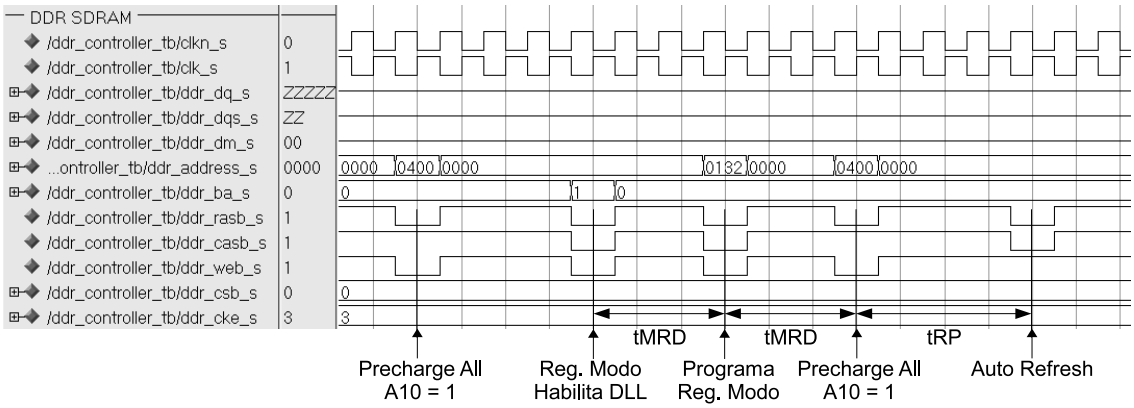

Figura 5.6: Forma de onda de simulação da etapa de inicialização da memória.

verificados com comandos de asserção.

As Figura 5.7 e 5.8 ilustram as etapas de escrita e de leitura na memória, ambas precedidas pela ativação de uma linha e banco de memória e finalizadas pelo comando *precharge all*.

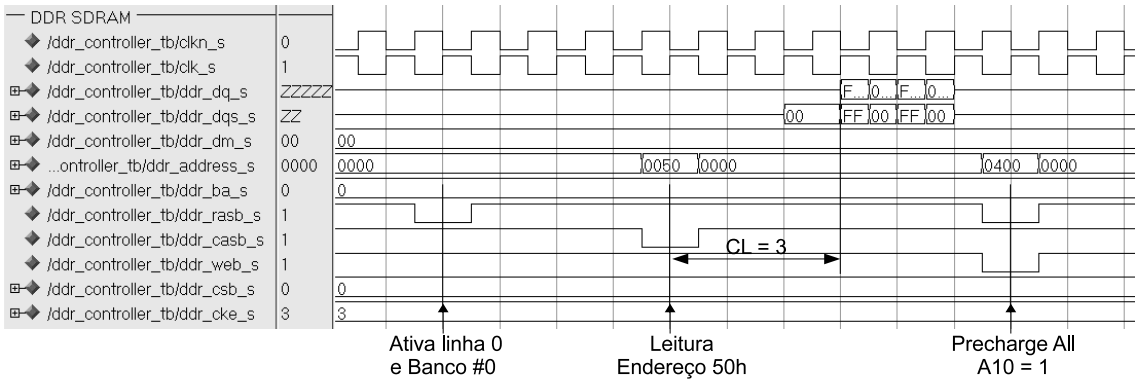

Figura 5.7: Forma de onda de simulação de uma leitura da memória.

Para verificar a funcionalidade do BIST e a detecção de defeitos no circuito controlador mais memória, foram inseridas faltas no código da memória. As faltas simuladas são do tipo *stuck-at* e de decodificação de endereços. A Figura 5.9 ilustra um diagrama de blocos simplificado com a abordagem de simulação do BIST com o controlador. Um circuito de controle do teste (Lógica do sistema) é utilizado para iniciar o teste e coletar os resultados de defeitos encontrados. A lógica compartilha com o BIST o barramento de acesso ao controlador através de um multiplexador 2:1 dos sinais de dado, controle e endereço.

Segundo (BERGERON, 2003), o esforço de verificação consome cerca de 70% do tempo total de projeto de um IP. O número de engenheiros envolvidos nessa etapa é o dobro do número de engenheiros de projeto RTL. Ao final do projeto, a quantidade de código utilizado na verificação é cerca de 80% do total do código. Durante o desenvolvimento do controlador de memória e do BIST, diversas simulações do código HDL foram

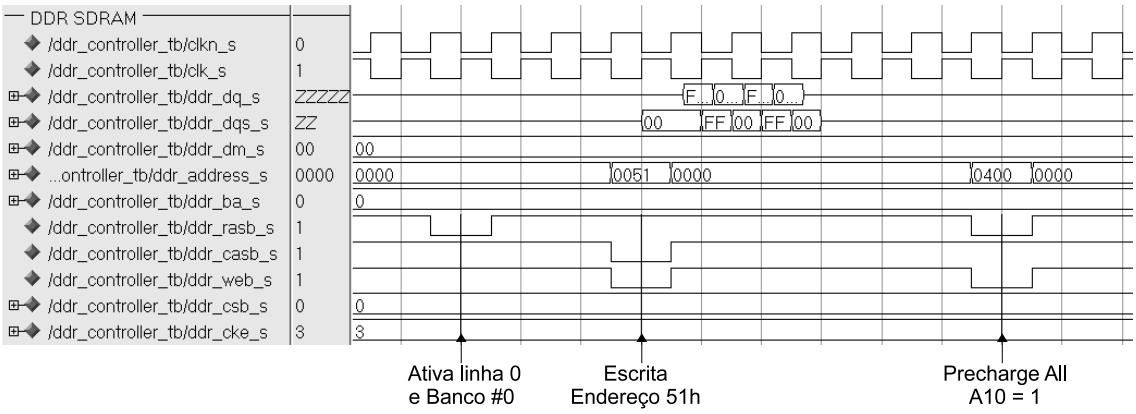

Figura 5.8: Forma de onda de simulação de uma escrita na memória.

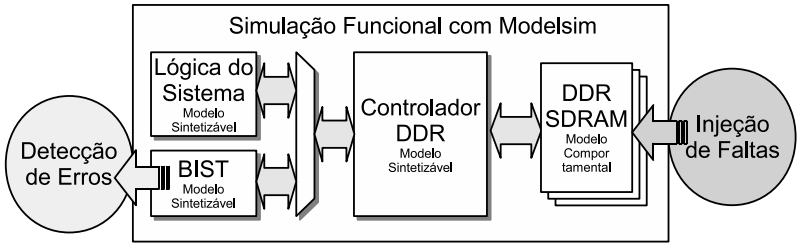

Figura 5.9: Abordagem para simulação do BIST com o controlador.

realizadas para correção de erros do código e para realizar alterações e melhorias.

Após a implementação física do circuito para FPGA, a etapa de simulação temporal é feita para verificar o funcionamento do controlador e do BIST após as etapas de mapeamento, posicionamento e roteamento. O atraso das interconexões é considerado na análise de temporização do circuito. A ferramenta de síntese e implementação gera um relatório dos tempos de propagação da lógica e do roteamento. Esse relatório é utilizado pela ferramenta de simulação para executar a verificação mais realista do funcionamento do circuito.

Na próxima seção são mostrados os resultados de validação do controlador na placa.

5.5 VALIDAÇÃO DO CONTROLADOR EM PLACA

Num sistema processador-memória, além dos defeitos atribuídos à memória existem os defeitos atribuídos às interconexões entre o sistema e a memória externa. Tais defeitos gerados pelas interconexões podem ser classificados em duas categorias: faltas DC ou de tipo stuck que podem ser testadas através de técnicas como *boundary scan*, caso a memória seja compatível com o padrão IEEE 1149.1 (também conhecido como JTAG) e; faltas AC que não podem ser detectadas através de *boundary scan*. Mesmo que a memória do sistema atenda ao padrão de teste tipo IEEE 1149.1, não é necessário testar somente a memória. Para o bom funcionamento do sistema, o teste deve englobar tanto a memória externa quanto as interconexões entre sistema e memória. No caso do controlador de

memória implementado em FPGA, o teste deve englobar a memória, as interconexões e a interface de ES do controlador.

Violações de temporização, tanto de setup quanto de hold, podem ocorrer devido à problemas na implementação do controlador no dispositivo FPGA. Esses problemas podem ocorrer principalmente na interface de dados entre controlador e memória externa. A Figura 5.10 ilustra uma violação de setup, caso o atraso dos sinais do grupo de dado seja excessivo, e uma violação de hold, caso o atraso do amostrador seja excessivo. Isso é caracterizado como uma falta de implementação, onde o atraso de propagação dos sinais externamente e internamente ao FPGA são superiores ao tolerado pelo controlador.

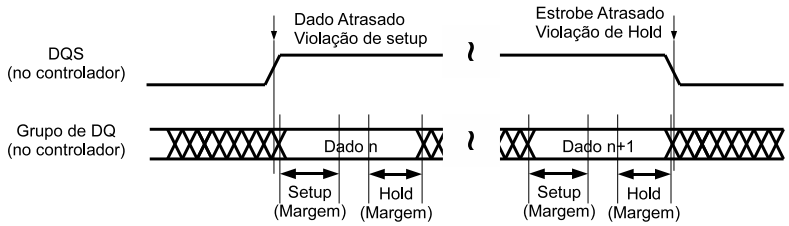

Figura 5.10: Violações de setup e hold na interface de dados.

A ocorrência das faltas pode ser aleatória ou não, dependendo da relação do atraso de propagação dos sinais de dado e de amostragem. A função principal da verificação é avaliar se o controlador foi mapeado e posicionado corretamente no dispositivo FPGA. Os atrasos de propagação dos sinais localizados dentro das faixas aceitáveis para amostrar o dado corretamente.

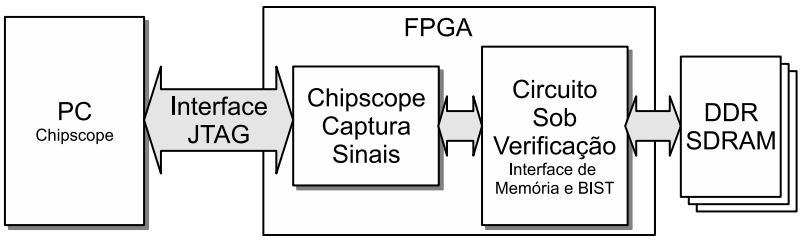

Figura 5.11: Diagrama de blocos do circuito de captura de sinais utilizando ChipScope.

Para a validação em placa, os sinais de erro da comparação do padrão lido da memória foram monitorados utilizando a ferramenta ChipScope do fabricante Xilinx. Essa ferramenta gera um módulo de hardware para ser conectado ao circuito sob verificação, implementado em FPGA. Tal módulo faz a coleta de informações dos sinais selecionados para monitoramento, armazena as informações nas memórias BlockRAM do FPGA e transmite para o computador via interface de programação do FPGA, como ilustra a Figura 5.11.

A Figura 5.12 mostra uma captura de tela com o resultado de uma sequência de padrões de validação no controlador implementado com erro. O controlador DDR que apresenta defeito foi implementado utilizando a restrição de temporização MAXDELAY,

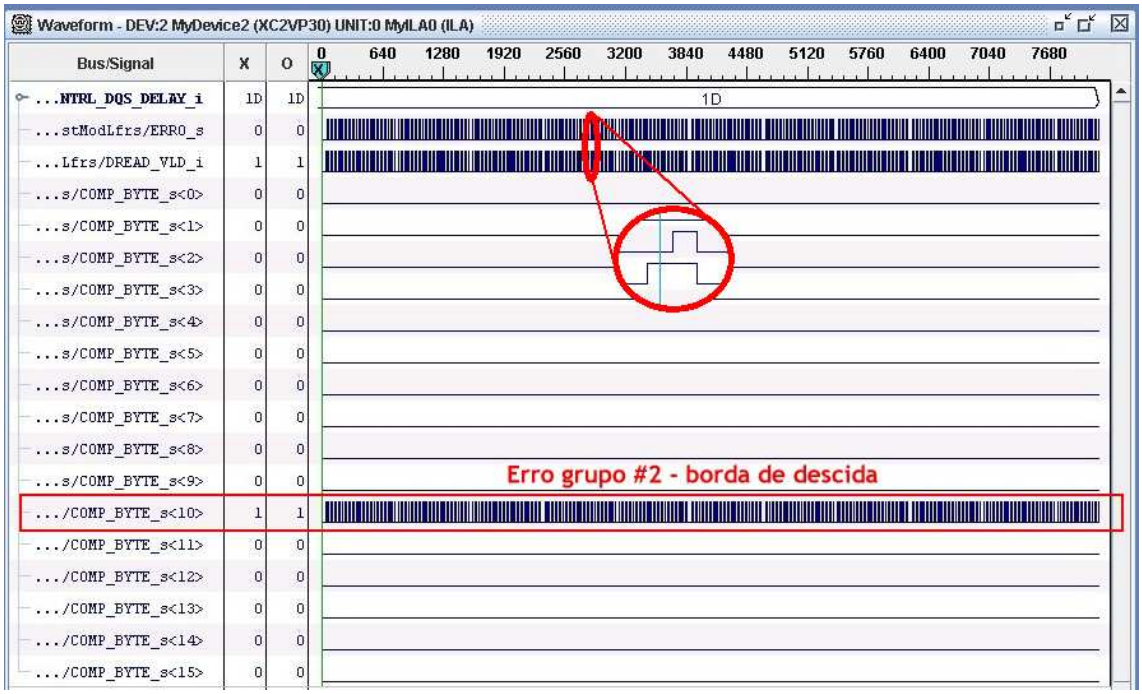

Figura 5.12: Verificação do controlador com defeito de implementação.

como descrito anteriormente na seção 4.4.3. O defeito é causado pela má configuração da linha de atraso. Nesse caso, a linha de atraso está configurada com quatro elementos LUT4 em série (DQS_controle[4:0] = "11101" = 0x1D), para implementar tSD' = 1454 ps. O atraso excessivo faz com que alguns sinais de dado sejam amostrados com violação de hold. Na Figura 5.12, pode-se ver que o grupo de sinais de dado amostrados por DQS[2] apresenta uma falta durante a captura na borda de descida. Como descrito na Tabela 4.5, o sinal de amostragem do grupo #2 é o que possui o maior tempo de propagação dentre todos os grupos.

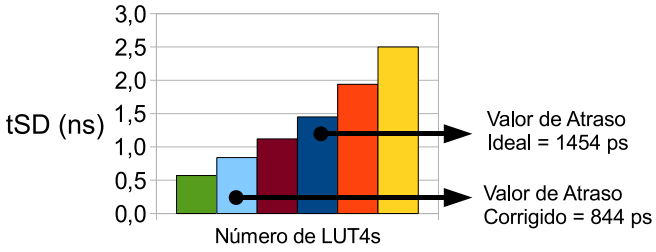

Figura 5.13: Gráfico comparativo entre o número de LUT4s e o atraso.

O ajuste do valor de configuração da linha de atraso corrige o problema da amostragem. A Figura 5.13 ilustra o gráfico contendo a variação do tempo de propagação gerado pela linha de atraso (tSD) numa escala que inicia com uma LUT4 (valor mais à esquerda) até seis LUT4s (valor mais à direita).

Na Figura 5.14 é mostrada a captura de tela da ferramenta ChipScope com a execução do teste para o controlador funcionando corretamente. Nesse caso, a linha de atraso

foi implementada com dois elementos LUT4 em série (DQS_controle[4:0] = “10111” = 0x17), para implementar tSD” = 844 ps. Não são detectados defeitos durante o teste. Nesse caso, o número de LUT4s inseridas na linha de atraso é igual ao calculado em 4.4.3, ou seja, duas LUT4s.

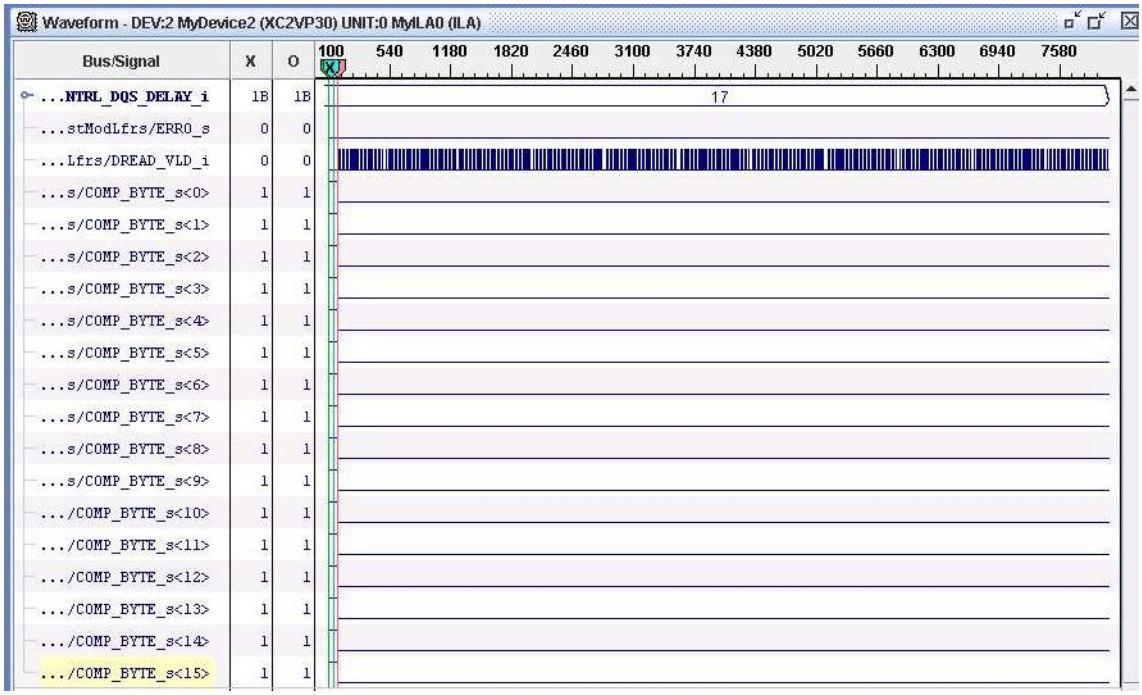

Figura 5.14: Verificação do controlador funcionando corretamente.

A partir desse estudo e da implementação do controlador de memória sobre uma plataforma FPGA, se pode verificar que o projeto do IP controlador vai além da escrita do código de entrada e verificação de caminhos críticos. O sucesso da implementação do controlador num circuito real depende da capacidade da ferramenta em posicionar os elementos de temporização crítica corretamente. O projetista do SoC deve estar atento aos defeitos da implementação e para isso o BIST integrado ao controlador é uma ferramenta fundamental. Foram realizados diversos testes em placa para verificar a implementação do controlador que indicam a eficiência do cálculo do intervalo tSD” a partir dos resultados de análise estática da implementação. A partir dessa análise, é possível determinar se o controlador de memória funcionará corretamente quando implementado no circuito real.

No próximo capítulo desse trabalho são discutidas as conclusões alcançadas e algumas propostas de melhoria para o IP controlador DDR.

6 CONCLUSÕES

Sistemas-em-chip de alto desempenho, utilizados no processamento de aplicações que processam grandes quantidades de informação, necessitam acessar a memória externa em alta velocidade. A implementação eficiente de um controlador de memória é fundamental para o correto funcionamento desses sistemas.

Esse trabalho propôs o projeto e implementação de um controlador de memória DDR SDRAM para sistemas-em-chip prototipados em tecnologia FPGA, com características de reúso e de portabilidade. O projeto e implementação de um controlador de memória externa em tecnologia FPGA oferece como principal desafio a implementação do circuito de captura de dados. Devido às variações de implementação do circuito, os atrasos de propagação internos ao FPGA podem gerar erros na amostragem dos dados. Nesse trabalho foi proposta uma metodologia para implementação automática do controlador utilizando restrições de limite de atraso para sinais de temporização crítica. A implementação de um núcleo de hardware do tipo *firm*, que possui restrições críticas de temporização, é um trabalho complexo. O desafio de implementar uma interface de memória com tamanhas restrições requer inicialmente a compreensão do funcionamento das ferramentas de síntese e de implementação. Também foi desenvolvida uma técnica de verificação da implementação do controlador DDR em FPGA baseada no teste de memória. Como resultado, foi projetado um BIST incluído no controlador de memória, de forma a produzir um IP autoverificável. Essa abordagem traz como principais vantagens a redução da complexidade do teste de sistema, de componentes de memória e também em nível de placa. Também simplifica o teste de memória do sistema ao longo da vida útil do circuito eletrônico. Ao longo desse trabalho foi feito um estudo detalhado do funcionamento de memórias síncronas DRAM e da arquitetura de sistemas com interface de memória externa.

O uso de uma linguagem de descrição de hardware traz grandes vantagens para a implementação de circuitos lógicos. Uma delas é a facilidade de geração de código e, portanto, de circuitos digitais. Outra é a possibilidade de verificação do funcionamento do circuito antes de ser implementado, usando a própria linguagem de descrição como elemento essencial de funcionamento do circuito. Mas a principal vantagem da descrição de hardware é a facilidade de reaproveitamento dos circuitos lógicos, ou o reúso. A

linguagem permite que os circuitos sejam parametrizados em suas dimensões físicas, replicados rapidamente e modificados se necessário. Principalmente modificados para outra arquitetura, dispositivo ou placa. Portanto, diversos fabricantes criam bibliotecas de circuitos lógicos para acelerar o projeto de sistemas maiores através de integração de núcleos de IP.

Quando os elementos lógicos presentes num núcleo são dependentes de tecnologia alvo, o seu reúso é limitado. Cada fabricante cria tecnologias diferentes para implementar circuitos (tanto programáveis como ASICs) e disponibiliza suas próprias ferramentas de implementação. Outras ferramentas que podem ser utilizadas para gerar circuitos para diversos fabricantes utilizam as bibliotecas de dispositivos do próprio fabricante para gerar o circuito. Todas essas ferramentas utilizam linguagens de descrição de código como a entrada de projeto. Se a descrição contém elementos lógicos dependentes da arquitetura, essa ferramenta não poderá fazer a implementação para outro dispositivo diferente. O conhecimento das limitações da ferramenta de síntese é importante para guiar a implementação do circuito no dispositivo alvo e para atingir as especificações de projeto. Nem sempre as ferramentas utilizadas conseguem explorar inteiramente os recursos disponíveis, portanto o projetista deve conhecer meios de guiar a ferramenta para atingir a especificação do projeto.

O desenvolvimento de sistemas-em-chip baseados em IPs permite que partes de hardware sejam rapidamente integradas constituindo um sistema maior. O uso de uma linguagem de descrição de hardware, além de simplificar as etapas de verificação de projeto, permite o reúso dos núcleos. No caso do controlador DDR, os elementos lógicos dependentes da tecnologia utilizados na interconexão com a memória DDR externa, são instanciados a partir das bibliotecas do fabricante. Além disso, os elementos utilizados para implementar a linha de atraso são posicionados em locais específicos do dispositivo, determinados pelo projetista de hardware. Isso caracteriza o núcleo *firm*, que tem roteamento pré-definido e possui elementos dependentes da tecnologia utilizada. Quando um núcleo *firm* é implementado em outro dispositivo, deve-se fazer trabalho de re-engenharia modificando o seu projeto. A modificação do núcleo envolve novas etapas de verificação, o que torna o trabalho de reúso lento e complexo. Estes circuitos podem ser isolados em módulos separados que são reprojetados para cada novo dispositivo. Já a lógica do controlador as máquinas de estado são encapsuladas em módulos do tipo *soft* e reutilizadas em novos projetos.

As alterações no projeto do controlador de memória substituindo os elementos lógicos instanciados de bibliotecas por código genérico facilitam o reúso do IP. O uso das restrições de temporização no lugar de restrições de posicionamento também facilitam o reúso do controlador. Não depende do projetista do sistema-em-chip fazer o posicionamento forçado para cada novo projeto, mas sim orientar a ferramenta para alcançar as especificações do sistema. Como uma medida de quão reutilizável é o código de descrição do IP,

se pode contar o número de linhas que o projetista pode manter inalteradas de um projeto ao seguinte. O esforço de recodificação na adaptação do controlador DDR para um novo projeto é refeito sobre aproximadamente 20% do total das linhas de código fonte. A parte do controlador que contém as restrições de posicionamento forçado e os elementos dependentes da tecnologia representa cerca de 11% do total dos recursos lógicos utilizados pelo controlador.

A constante evolução atual da indústria eletrônica coloca no mercado novos padrões de dispositivos. Isso torna o desafio de implementação maior ainda, pois aliado à dificuldade de projeto está o planejamento de uma forma de adaptação para os novos padrões subsequentes. Como foi mostrado nesse trabalho, em poucos anos os dispositivos de memória DDR SDRAM deram lugar às DDR2 e logo mais será DDR3 dominando o mercado. O projeto do controlador de memória como uma técnica de implementação e de validação é válido e abrange diversos tipos de memória. As interfaces dessas memória sofreram algumas alterações entre gerações mas o princípio de funcionamento é o mesmo. A complexidade de implementação de controladores DDR2 e DDR3 é maior, mas a complexidade das etapas de verificação e de validação do funcionamento é a mesma que para memórias DDR. A verificação do funcionamento envolve as etapas bem definidas de projeto, desde a simulação funcional até a implementação em placa. Cada etapa envolve o tempo de desenvolvimento do próprio ambiente de simulação e teste e também envolve o tempo gasto com a verificação.

O projeto do controlador de memória DDR SDRAM envolve o uso de elementos lógicos dependentes de tecnologia. Além disso, é necessário estabelecer critérios rígidos de posicionamento e de roteamento dos elementos lógicos, principalmente para o circuito que captura dados da memória externa. De outra forma, o controlador implementado em FPGA tende ao erro, devido às faltas presentes no circuito de captura de dados. A integração de um módulo BIST com o controlador é muito útil para permitir o teste automático da implementação IP reutilizando-o em outros sistemas. A abordagem de teste usa o IP controlador DDR, as interconexões entre a memória e FPGA e a memória externa como um circuito único sob verificação. O uso de um módulo BIST é preferível ao uso de um circuito que implemente uma bancada de testes. A remoção da bancada de testes do circuito verificado pode alterar o comportamento do mesmo, modificando atrasos dos sinais de forma que a validação pode ser perdida. Um módulo BIST integrado ao controlador IP permite para testar a sua funcionalidade em cada nova implementação de sistema. Além disso, um teste de memória simples estará disponível para ser realizado ao longo da vida útil do sistema, dispensando a retirada do módulo de memória para seu próprio teste.

Como trabalhos futuros sugere-se a realização de melhorias na parte de controle do controlador de memória. O controle não está preparado para operar com latência de CAS de 2,5 ciclos e não consegue controlar a leitura em rajadas de 2 e 8 palavras. Também deseja-se implementar o controlador em taxas maiores de relógio, como com memórias

DDR de 200 MHz de frequência de barramento. Em sistemas embarcados, a potência dissipada é um parâmetro importante que deve ser minimizado para aumentar a autonomia do sistema. O projeto do controlador de memória DDR SDRAM pode ser orientado para aplicações de baixo consumo de energia, otimizando os acessos aos dados e utilizando o modo de *power-down* da memória externa.

REFERÊNCIAS

ABNT. **NBR 15602-1**: televisão digital terrestre – codificação de vídeo, áudio e multiplexação parte 1: codificação de vídeo. Rio de Janeiro - RJ - Brasil: ABNT, 2007.

ADAMS, R. D. **High performance memory testing**: design principles, fault modeling and self-test. 1.ed. Secaucus, NJ, USA: Kluwer Academic Publishers, 2002.

BERGERON, J. **Writing testbenches**: functional verification of HDL models. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

BONATTO, A. C.; SOARES, A. B.; SUSIN, A. A. DDR SDRAM memory controller validation for FPGA synthesis. In: IEEE LATIN-AMERICAN TEST WORKSHOP, 9., 2008, Puebla, México. **Proceedings...** [S.l.: s.n.], 2008. p.177–182.

BONATTO, A. C.; SOARES, A. B.; SUSIN, A. A. Test Method for External DDR SDRAM memory. In: XIV IBERCHIP WORKSHOP, 2008, Puebla, Mexico. **Proceedings...** [S.l.: s.n.], 2008.

BONATTO, A. C.; SOARES, A. B.; SUSIN, A. A. DDR SDRAM controller IP designed for reuse. In: IP BASED ELECTRONIC SYSTEM CONFERENCE & EXHIBITION, 2008, Grenoble, França. **Proceedings...** Design & Reuse, 2008. p.175 – 180.

BONATTO, A. C.; SOARES, A. B.; SUSIN, A. A. Design and Test of a DDR SDRAM Interface for FPGA systems. In: STUDENT FORUM ON MICROELECTRONICS, 2008, Gramado, Brasil. **Proceedings...** [S.l.: s.n.], 2008.

BUSHNELL, M. L.; AGRAWAL, V. D. **Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits**. Secaucus, NJ, USA: Kluwer Academic Publishers, 2002.

CATY, O.; BAYRAKTAROGLU, I.; MAJUMDAR, A.; LEE, R.; BELL, J.; CURHAN, L. Instruction based BIST for board/system level test of external memories and interncon-nects. In: INTERNATIONAL TEST CONFERENCE, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.140–149.

COSOROABA, A. **White paper**: virtex-4, virtex-5, and spartan-3 generation FPGAs. [S.l.]: Xilinx Inc., 2007.

GEORGE, M. **XAPP802 - Memory Interface Application Notes Overview**. [S.l.]: Xilinx, 2007. Disponível em: <http://www.xilinx.com/memory>. Acesso em setembro 2007.

GOOR, A. J. van de. **Testing semiconductor memories**: theory and practice. New York, NY, USA: John Wiley & Sons, Inc., 1991.

GUPTA, N. **Interfacing Virtex-II Devices With DDR SDRAM Memories for Performance to 167 MHz**. [S.l.]: Xilinx, 2005.

GUPTA, R. K.; ZORIAN, Y. Introducing core-based system design. **IEEE Design & Test of Computers**, Los Alamitos, CA, USA, v.14, n.4, p.15–25, 1997.

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

INTEL. **Excerpts from A conversation with Gordon Moore**: moore's law. [S.l.]: Intel Co., 2005. Disponível em: <ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf>. Acesso em: 08 abr. 2009.

ISE 8.1i Software Manual: libraries guide. [S.l.]: Xilinx Inc., 2007.

ITRS. **International technology roadmap for semiconductors 2007 edition**: design. [S.l.]: ITRS, 2007. Disponível em: <http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf>. Acesso em: 21 jan. 2009.

ITU-T. **Recommendation H.264 – advanced video coding for generic audiovisual services**. Switzerland, Geneva: International Telecommunication Union, 2005.

JEDEC. **JESD8-9B**: stub series terminated logic for 2.5 v (sstl_2) standard. Virginia, USA: JEDEC Solid State Technology Association, 2002. Disponível em: <http://www.jedec.org/download/search/jesd8-9b.pdf>. Acesso em: 12 maio 2008.

JEDEC. **JESD79C**: Double Data Rate (DDR) SDRAM specification. Virginia, USA: JEDEC Solid State Technology Association, 2003. Disponível em: <http://download.micron.com/pdf/misc/JEDEC79R2.pdf>. Acesso em: 5 maio 2008.

JEDEC. **JESD208**: speciality DDR2-1066 SDRAM. Virginia, USA: JEDEC Solid State Technology Association, 2007. Disponível em: <http://www.jedec.org/download/search/JESD208.pdf>. Acesso em: 6 maio 2008.

JEDEC. **JESD79-3A**: DDR3 SDRAM specification. Virginia, USA: JEDEC Solid State Technology Association, 2007. Disponível em: <http://www.jedec.org/download/search/JESD79-3A.pdf>. Acesso em: 7 maio 2008.

JEDEC. **JESD79-2E**: DDR2 SDRAM specification. Virginia, USA: JEDEC Solid State Technology Association, 2008. Disponível em: <http://www.jedec.org/download/search/JESD79-2E.pdf>. Acesso em: 5 maio 2008.

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

KIM, H. C.; JUN, H.-S.; GU, X.; CHUNG, S. S. At-speed interconnect test and diagnosis of external memories on a system. In: INTERNATIONAL TEST CONFERENCE, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.156–162.

KIM, S. D.; SUNWOO, M. H. ASIP Approach for Implementation of H.264/AVC. **Signal Processing Systems**, [S.l.], v.50, n.1, p.53–67, 2008.

LIU, T.-M.; LEE, C.-Y. Design of an H.264/AVC decoder with memory hierarchy and line-pixel-lookahead. **Journal of Signal Processing Systems**, [S.l.], v.50, n.1, p.69–80, jan. 2008.

MICRON. **256Mb**: x4, x8, x16 (DDR) SDRAM Features. [S.l.]: Micron Tecnology, Inc., 2003.

MICRON. **DDR3 Advantages**. [S.l.: s.n.], 2006. Disponível em: <http://download.micron.com/pdf/presentations/ddr3_sdram/ddr3_advantages.pdf>. Acesso em: 15 abr. 2008.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, [S.l.], n.8, p.114–117, abr. 1965.

MOSAID. **The love/hate relationship with DDR SDRAM controllers**: love the cheap memory, hate the complex controller. Ottawa - ON - Canadá: Mosaid – Memorize, 2006.

OLLOQUI, E. P.; TORMO, F. C.; AGULLO, J. S.; BERENGUER, A. H. Implementing high-speed double-data rate (DDR) SDRAM controllers on FPGA. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATION, 14., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.279 – 288.

PATTERSON, D.; ANDERSON, T.; CARDWELL, N.; FROMM, R.; KEETON, K.; KOZYRAKIS, C.; THOMAS, R.; YELICK, K. A case for intelligent RAM: IRAM. **Micro, IEEE**, [S.l.], v.17, n.2, p.34–44, 1997.

RYAN, K. DDR SDRAM functionality and controller read data capture. **Micron Design Line**, [S.l.], v.8, p.11–12, 1999.

SAMSUNG. **Memory market outlook & Samsung's strategy**. [S.l.]: Samsung Electronics Co., Ltd, 2007.

SHEN, S.-C.; HSU, H.-M.; CHANG, Y.-W.; LEE, K.-J. A high speed BIST architecture for DDR SDRAM testing. In: IEEE INTERNATIONAL WORKSHOP ON MEMORY TECHNOLOGY, DESIGN, AND TESTING, 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.52–57.

SHUANG-YAN, C.; DONG-HUI, W.; RUI, S.; CHAO-HUAN, H. An innovative design of the DDR/DDR2 SDRAM compatible controller. In: INTERNATIONAL CONFERENCE ON ASIC, 6., 2005. **Proceedings...** [S.l.: s.n.], 2005. v.1, p.62–66.

SOARES, A. B.; BONATTO, A. C.; SUSIN, A. A. A new march sequence to fit DDR SDRAM test in burst mode. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 21., 2008, New York, USA. **Proceedings...** Association for Computing Machinery – ACM, 2008. p.28 – 33.

STAEHLER, W. T. **Projeto de sistemas digitais complexos**: uma aplicação ao decodificador H.264. 2006. 100p. Dissertação (Mestrado em Ciências da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006.

VOLLRATH, J. E. Testing and characterization of SDRAMs. **IEEE Design and Test of Computers**, Los Alamitos, CA, USA, v.20, n.1, p.42–50, 2003.

XILINX. **UG067 - Xilinx Memory Interface Generator (MIG 007) User Guide**. [S.l.]: Xilinx, 2005.

XILINX. **DS083, Virtex-II Pro and Virtex-II Pro X Platform FPGAs**: complete data sheet. [S.l.]: Xilinx, 2007.

APÊNDICE A *MEMORY INTERFACE GENERATOR*

Nesse apêndice é mostrado o funcionamento da ferramenta da Xilinx para geração automática do controlador de memória DDR e DDR2. Também são discutidas algumas instruções para o uso correto da ferramenta.

A.1 DESCRIÇÃO E MODO DE USO

A versão utilizada nesse trabalho é MIG007_rel6. Em sua interface (Figura A.1) são colocadas as informações para gerar a interface de memória de acordo com as necessidades de projeto. Ele permite a geração de módulos controladores de memória DDR SDRAM. Os controladores gerados são interfaces de baixo nível com funcionamento descrito na seção 3.2.1. Pode ser configurado para gerar controladores DDR ou DDR2 (com ou sem *strobes* diferenciais) com diferentes opções de projeto como: gera interfaces para DIMMs ou para componentes de memória; para sistemas com frequência de operação variando entre 100 MHz e 167 MHz; para famílias de FPGA Spartan3, Virtex-2 e Virtex-2 Pro; com opções de escolha dos bancos de IO da interface com a memória e largura dos barramentos de endereços e de dados.

A interface de memória é gerada através de fontes Verilog ou VHDL existentes na base de dados do programa, de acordo com as opções selecionadas na interface gráfica.

A outra funcionalidade desse programa é o editor de pinos, que permite ao usuário especificar os pinos DQ e DQS do controlador de acordo com sua plataforma. Essa opção permite ao MIG gerar o arquivo UCF com as restrições de mapeamento de elementos lógicos da interface de dados, como descrito na seção 3.1.4. A Figura A.2 mostra como são escolhidos os pinos da interface de memória. A escolha dos pinos não pode ser aleatória e deve ser feita em etapas: primeiro são selecionados os pinos DQS e seus respectivos sinais de dados associados; depois são selecionados um no_dpin para cada DQS da interface; por último são alocados os sinais de relógio, de endereços e de comandos. Quando são selecionados os DQS, os pinos de dados DQ são marcados em cor azul-claro pela ferramenta. Depois de selecionar todos os DQS no projeto são escolhidos os no_dpin. A ferramenta marca com a cor verde os vizinhos do no_dpin selecionado. Nenhum desses

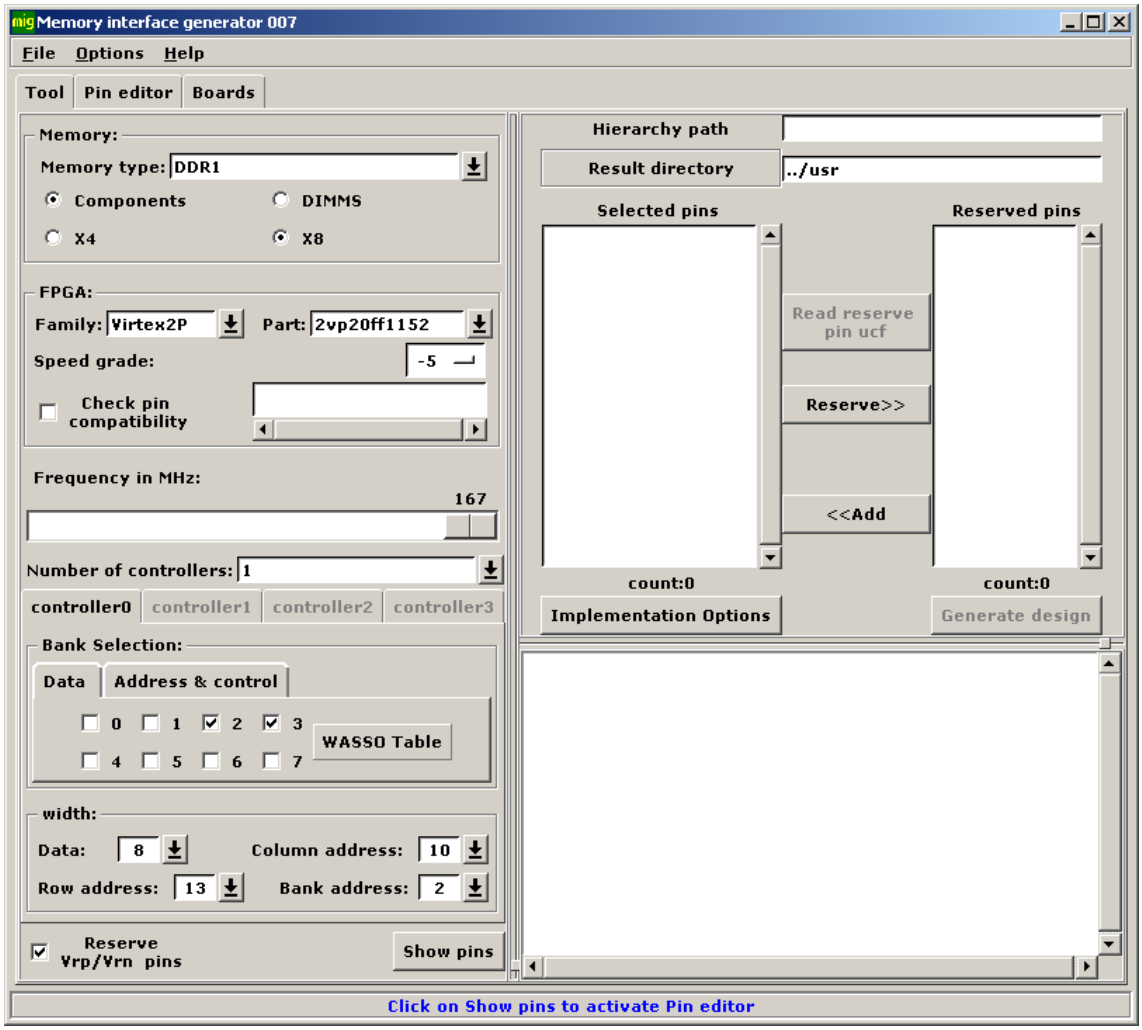

Figura A.1: Interface do *Memory Generator Interface*.

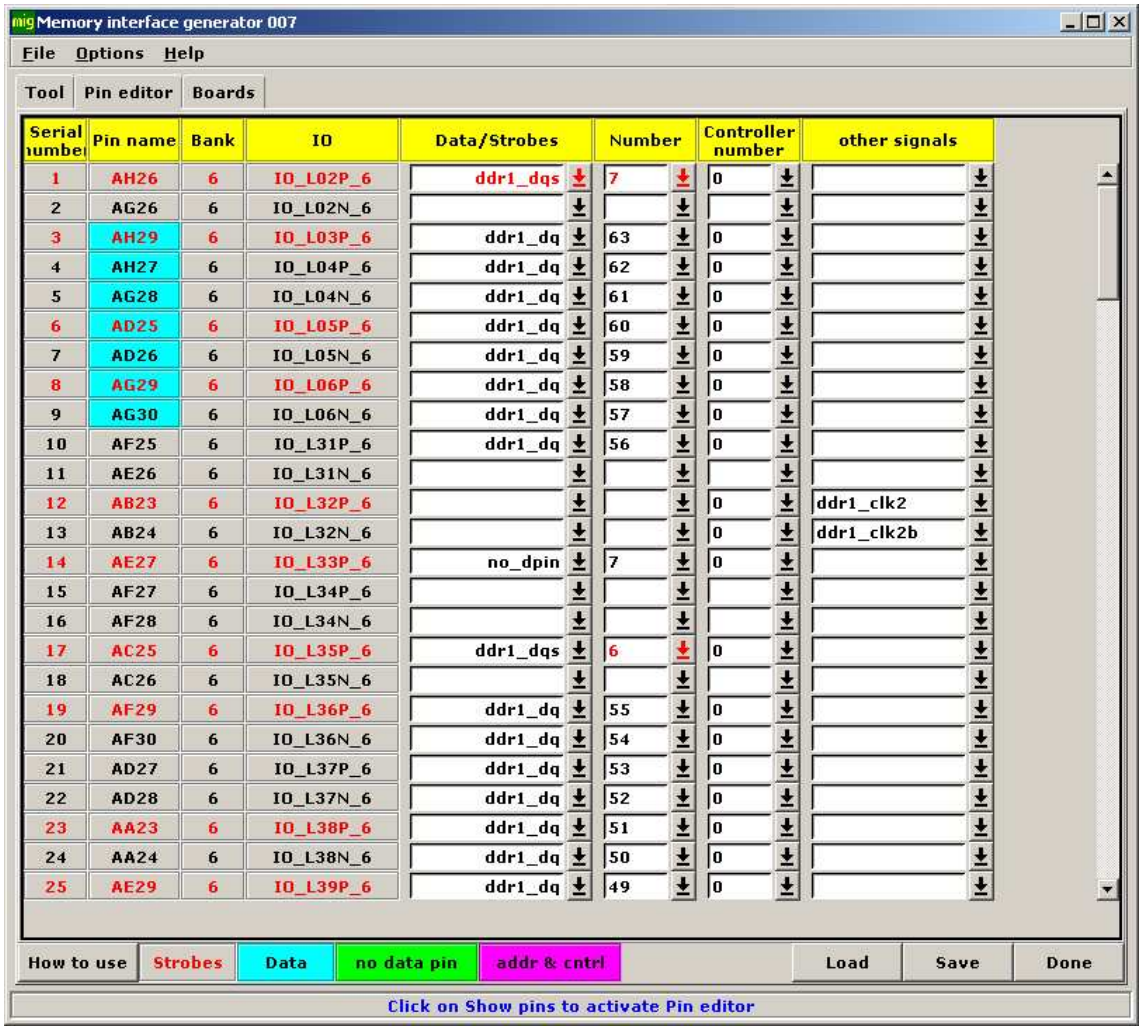

Figura A.2: Editor de pinos do MIG.

vizinhos deve coincidir com um pino DQ ou DQS, caso contrário o arquivo UCF gerado conterá um erro e a ferramenta de síntese não poderá gerar o projeto. Cada no_dpin é usado para alocar os elementos de geração dos ponteiros de escrita de dados nas FIFOS.