UNIVERSIDADE FEDERAL DO PAMPA

RAFAEL VIANNA GARRIDO

PROJETO DE UM CONTROLADOR DE DMA DE ALTA VAZÃO PARA TRANSFERÊNCIA DE DADOS

Bagé
2015

RAFAEL VIANNA GARRIDO

PROJETO DE UM CONTROLADOR DE DMA DE ALTA VAZÃO PARA TRANSFERÊNCIA DE DADOS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Fábio Luís Livi Ramos

Bagé
2015

RAFAEL VIANNA GARRIDO

PROJETO DE UM CONTROLADOR DE DMA DE ALTA VAZÃO PARA TRANSFERÊNCIA DE DADOS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 28 de janeiro de 2015.

Banca examinadora:

_____

Prof. Me. Fábio Luís Livi Ramos
Orientador
Engenharia de Computação - UNIPAMPA

_____

Prof. Me. Gerson Alberto Leiria Nunes
Engenharia de Computação - UNIPAMPA

_____

Prof. Dr. Milton Roberto Heinen
Engenharia de Computação - UNIPAMPA

AGRADECIMENTOS

A minha família, sempre ao meu lado, apoiando-me nas diversas situações, com muito amor e carinho.

Aos Professores Bruno Silveira Neves e Fábio Luís Livi Ramos pelo companheirismo, auxílios e conselhos dados durante a elaboração deste trabalho.

Aos demais professores do curso que, ao longo das disciplinas, ajudaram a formar a maioria das ideias necessárias a este trabalho.

RESUMO

Este trabalho tem como objetivo desenvolver um controlador de DMA (*Direct Memory Access* - Acesso Direto à Memória) em linguagem VHDL (VHSIC *Hardware Description Language* - Linguagem de Descrição de Hardware VHSIC) com foco em alta vazão de dados. O DMA é integrado posteriormente a um sistema composto por uma controladora de memória RAM (*Random-access Memory* - Memória de Acesso Aleatório), parametrizável, com o recurso de *interleaving* de dados e um controlador de dispositivo de E/S (Entrada e Saída de Dados), os quais também são desenvolvidos pelo autor em linguagem VHDL. O projeto é desenvolvido com o auxilio dos *softwares* Altera Quartus II e Altera ModelSim. Ao final do trabalho obteve-se um controlador de DMA que realiza transferências de dados entre uma controladora de memória e um dispositivo de E/S.

Palavras-Chave: DMA, VHDL, *interleaving* de dados, alta vazão de dados.

ABSTRACT

This work aims to develop a DMA (Direct Memory Access) controller in VHDL (VHSIC *Hardware Description Language*) language with focus on high data throughput. The DMA is subsequently integrated into a system consisting of a parameterized RAM (Random-access Memory) memory controller, with feature of data interleaving, and an I/O (In/Out Data) device controller which are also developed by author in VHDL language. The project is developed using Altera Quartus II and Altera ModelSim softwares. A DMA controller performs data transfers between memory controller and I/O device is obtained as final result of this work.

Keywords: DMA, VHDL, memory interleaving, high data throughput.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

AHB - *Advanced High-performance Bus* (Barramento Avançado de Alta Performance)

AMBA - *Advanced Microcontroller Bus Architecture* (Arquitetura de Barramento Avançado de Microcontrolador)

ASIC - *Application-specific Integrated Circuit* (Circuito Integrado de Aplicação Específica)

AXI - *Advanced eXtensible Interface* (Interface Avançada Extensível)

CPU - *Central Processing Unit* (Unidade Central de Processamento)

DMA - *Direct Memory Access* (Acesso Direto à Memória)

DMAC - *Direct Memory Access Controller* (Controlador de Acesso Direto à Memória)

EDA - *Electronic Design Automation* (Automação de Projetos Eletrônicos)

E/S - Entrada e Saída de Dados

FSM - *Finite State Machine* (Máquina de Estados Finitos)

FPGA - *Field-programmable Gate Array*(Arranjo de Portas Programável em Campo)

HDL - *Hardware Description Language* (Linguagem de Descrição de *Hardware*)

IEEE - *Institute of Electrical and Electronics Engineers* (Instituto de Engenharia Elétrica e Eletrônica)

LUTs - *Look-Up Tables*

MIF - *Memory Initialization File* (Arquivo de Inicialização de Memória)

MM2S - *Memory-Mapped to Stream* (Mapeada em Memória para Fluxo)

RAM - *Random Access Memory* (Memória de Acesso Randômico)

RTL - *Register-Transfer Level* (Nível de Transferência de Registradores)

S2MM - *Stream to Memory Map* (Fluxo para Memória Mapeada)

SDF - *Standart Delay Format* (Formato Padrão de Atraso)

VCD - *Value Change Dump*

VHDL - VHSIC *Hardware Description Language* (Linguagem de Descrição de Hardware VHSIC)

VHSIC - *Very High Speed Integrated Circuits* (Circuito Integrado de Alta Velocidade)

SUMÁRIO

1 INTRODUÇÃO

As operações de E/S são responsáveis pela troca de dados entre os dispositivos externos e o processador. Cada um desses dispositivos é conectado ao processador através da conexão de um modulo de E/S, o qual é responsável pelo gerenciamento do dispositivo. Essa conexão é usada para transferência de dados, informações de controle e informações de estado entre o módulo de E/S e o dispositivo externo (STALLINGS, 2003). Para tal comunicação pode-se usar duas técnicas (STALLINGS, 2003):

- E/S Programada: o processador executa um programa e tem controle direto das operações de E/S, incluindo a detecção de estado do dispositivo, o envio de comandos de leitura e escrita e a transferência de dados.
- E/S dirigida por interrupção: o processador envia um comando de E/S e continua a executar outras instruções, sendo interrompido pelo dispositivo quando o mesmo tiver encerrado seu trabalho.

Em ambos os casos o processador age diretamente na transferência dos dados, acarretando duas conseqüências:

- *Overhead* no processador, pois se a transferência for muito grande será consumida uma fração do mesmo ao longo da transferência (PATTERSON, 2005).
- O processador se ocupa de gerenciar a transferência de dados de E/S, tendo de executar várias instruções, como verificar e testar registradores do módulo, E/S mapeada em memória, a cada transferência (STALLINGS, 2003).

Para solucionar tal problema de eficiência, foi desenvolvido um mecanismo que faz a transferência direta dos dados entre a memória e o dispositivo. Este mecanismo é chamado acesso direto à memória (DMA) (PATTERSON, 2005).

Com o controlador de DMA encarregado das transferências de E/S o processador só precisa configurar o controlador de DMA para que a transação seja realizada entre o dispositivo e a memória. Assim, ele pode continuar a execução de outras instruções sem envolver-se diretamente com a transferência. Após o controlador de DMA completar a transferência, uma

interrupção é causada pelo controlador para comunicar ao processador a conclusão da tarefa (PATTERSON, 2005).

A partir desse escopo, surgiu a motivação do desenvolvimento de um controlador de DMA, descrito em VHDL, para realizar transferências de dados com alta vazão entre um dispositivo periférico e a memória.

Tal controlador será uma alternativa de código-aberto (OPENSUSE, 2011) às soluções com código proprietário (UNIVERSIDADE FEDERAL DO PARÁ, 2010), como o AXI DMA *Controller* (Xilinx) (XILINX, 2014) e DMA *Controller Core* (Altera) (ALTERA, 2010).

Sendo o controlador de DMA responsável pela transferência dos dados, pode-se especular um aumento de desempenho nas transferências de dados através da introdução de mais interfaces entre o mesmo e a memória usando, por exemplo, *interleaving* de dados (AMOS, 1999) entre diferentes bancos de memória e aumentando assim a vazão de dados e o desempenho do sistema.

O controlador desenvolvido poderá ser usado em sistemas que demandem muitos acessos a memória ou que realizem transferências de grandes pacotes de dados, tais como um *Proxy* de vídeo (WEI, 2009).

A seção 2 apresenta os principais conceitos utilizados ao longo do presente trabalho. Na seção 3 são apresentados controladores de DMA proprietários e trabalhos relacionados. Na seção 4 são mostrados detalhes dos componentes desenvolvidos. Os resultados e a validação dos componentes desenvolvidos são apresentados na seção 5. Ao final do trabalho, seção 6, serão apresentadas as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Ao longo desta seção serão abordados os principais assuntos referentes ao desenvolvimento deste trabalho.

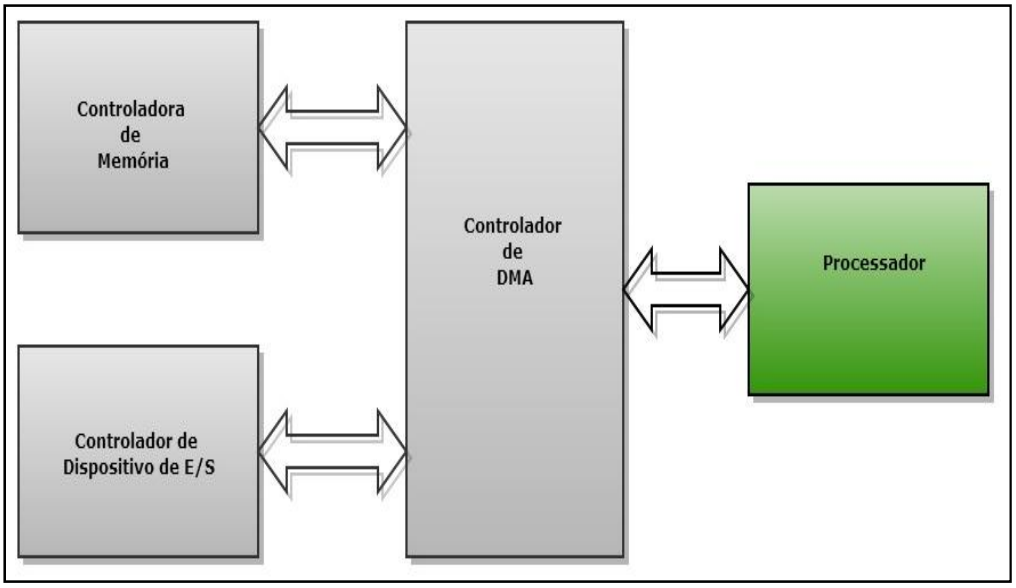
2.1 Acesso direto à memória (DMA - *Direct Memory Access*)

Para dispositivos com alta largura de banda, as transferências consistem principalmente em blocos de dados relativamente grandes. Assim os projetistas de computadores inventaram um mecanismo para desafogar o processador e fazer com que o dispositivo transfira dados diretamente de ou para a memória sem envolver o processador. Esse mecanismo é chamado acesso direto à memória (PATTERSON, 2005).

O controlador de DMA permite que determinados subsistemas dentro de um computador possam acessar a memória do sistema para leitura e/ou escrita independentemente da unidade central de processamento (WADEKAR, 2011).

A seguir, na figura 1, a ilustração de um sistema utilizando um controlador de DMA para realizar transferências entre uma controladora de memória e um controlador de dispositivo de E/S.

Figura 1 - Ilustração de um cenário com controlador de DMA



Fonte: Próprio Autor.

Existem três etapas em uma transferência de DMA (PATTERSON, 2005):

- O processador configura o controlador de DMA fornecendo a identidade do dispositivo, a operação a realizar no dispositivo, o endereço da memória que é a origem ou o destino dos dados a serem transferidos e o número de bytes a transferir.
- O DMA inicia a operação no dispositivo e arbitra o acesso ao barramento. Quando os dados estão disponíveis (do dispositivo ou da memória), ele transfere os dados. O dispositivo de DMA fornece o endereço de memória para a leitura ou a escrita. Se a solicitação exigir mais de uma transferência no barramento, a unidade de DMA gera o próximo endereço de memória e inicia a próxima transferência. Usando esse mecanismo a unidade de DMA pode completar uma transferência inteira, que pode ter milhares de bytes de tamanho, sem incomodar o processador.
- Quando a transferência de DMA termina, o controlador interrompe o processador, que pode então determinar, interrogando o controlador de DMA ou examinando a memória, para verificar se a operação inteira foi concluída com sucesso.

2.2 Protocolo de *handshaking*

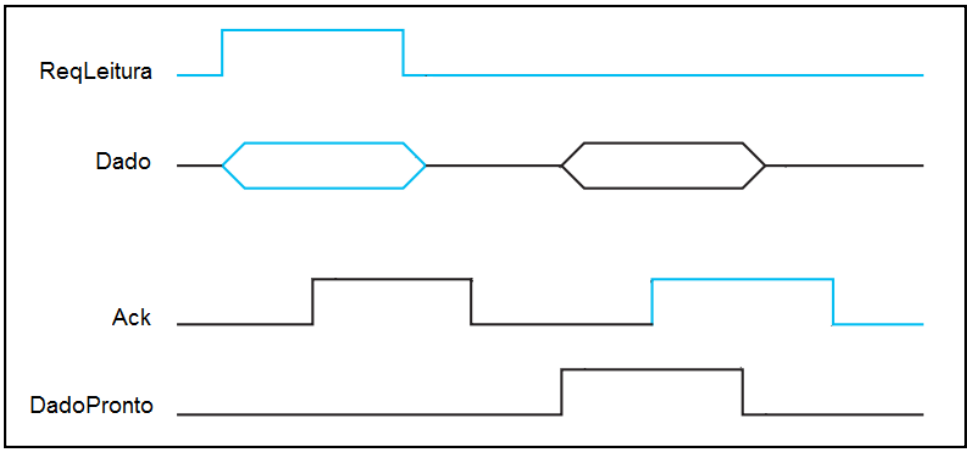
Para coordenar a transmissão de dados entre um emissor e um receptor, um barramento assíncrono utiliza um protocolo de *handshaking.* Um protocolo de *handshaking* consiste em uma série de etapas em que o emissor e o receptor prosseguem para a próxima etapa apenas quando as duas partes concordarem. O protocolo é implementado com um conjunto adicional de linhas de controle. São elas: (PATTERSON, 2005)

- ReqLeitura: usado para indicar uma requisição de leitura.
- DadoPronto: usado para indicar que o dado está disponível nas linhas de dados.
- Ack: usado para confirmar o sinal ReqLeitura ou DadoPronto da outra parte.

Pode-se observar que o sinal ReqLeitura muda para nível alto indicando uma solicitação de leitura. Ao perceber o sinal ReqLeitura, a outra parte levanta o sinal Ack, indicando sua confirmação, a mesma coloca o dado nas linhas de dados, sinal Dado, e levanta o sinal DadoPronto, indicando que o dado solicitado já está disponível nas linhas de dados.

A seguir, na figura 2, a ilustração do protocolo de *handshaking*.

Figura 2 - Ilustração de um protocolo de handshaking



Fonte: PATTERSON, 2005

2.3 *Memory interleaving*

Supondo que exista um processador com um tempo de ciclo T_p diretamente conectado a um único banco de memória com um tempo de acesso de T_m e que, T_m é muito maior que T_p. Então o tempo de acesso do processador a memória vai ser drasticamente reduzido pela unidade de tempo da memória. Se, contudo, houvera um arranjo de N bancos de memória com $N = (T_m/T_p)$, de tal maneira que podem ser acessados em paralelo no mesmo tempo de acesso de um banco, então, desde que N palavras sejam entregues em uma unidade de tempo, o tempo de acesso da memória visto pelo processador - o efetivo tempo de acesso a memória - será $T_p = (T_m / N)$ e, por conseguinte, corresponder ao tempo de ciclo do processador T_p.

Esse arranjo é o que acontece com *memory interleaving,* que em muitas aplicações com memória intercalada fornecem dados a uma taxa que excede os requisitos do processador (AMOS, 1999).

2.4 VHDL - VHSIC *Hardware Description Language*

Um sistema digital pode ser descrito em diferentes níveis de abstração e de diferentes pontos de vista.

Uma maneira de descrever um sistema digital é o VHDL, que é uma linguagem de descrição de *hardware,* inicialmente patrocinada pelo Departamento de Defesa dos Estados Unidos da América como um padrão de documentação de hardware no início de 1980 e em seguida transferida ao IEEE (*Institute of Electrical and Electronics Engineers* - Instituto de Engenharia Elétrica e Eletrônica) (CHU, 2006).

2.5 Parametrização de projeto

O reuso de projetos é um dos principais objetivos no desenvolvimento de um código VHDL. Idealmente, é desejado que o projeto de diversos módulos possa ser compartilhado por diversas aplicações. Partindo de que cada aplicação é diferente, é desejado que um módulo possa ser customizado para a necessidade da aplicação. A customização é normalmente especificada por parâmetros implícitos ou explícitos, e chamada parametrização de projeto. O VHDL provê vários mecanismos para passar e inferir parâmetros e inclui diversos construtores para descrever estruturas replicadas (CHU, 2006).

2.5.1 Parâmetros de largura

Especifica o tamanho (p. e. número de bits) de um dado. Um sistema pode precisar de um ou mais parâmetros para descrever o tamanho de sinais de entrada e saída, assim como os tamanhos dos sinais internos e registradores.

O objetivo principal de parametrizar é descrever o projeto desejado em termos da largura de parâmetros, aí então a mesma descrição em VHDL pode ser usada com diferentes requisitos de tamanho (CHU, 2006).

2.5.2 Replicar estruturas

Muitos circuitos digitais podem ser implementados como uma repetitiva composição de blocos básicos. Desde que seja possível expandir facilmente uma estrutura incrementando o número de iterações, esses circuitos são naturais para a parametrização de projeto (CHU, 2006).

2.6 Pacotes

À medida que um sistema cresce, mais informação é incluída na seção de declaração. Quando um sistema é dividido em vários subsistemas menores, algumas declarações podem estar duplicadas em muitas unidades diferentes do projeto.

O construtor VHDL denominado *package* é um método de organizar essas declarações. Podem-se reunir as declarações comumente utilizadas em um projeto, agrupá-las e armazená-las em um pacote. O código VHDL que desejar usar as declarações só precisa incluir uma cláusula de utilização para ter acesso ao pacote (CHU, 2006).

2.7 FPGA - *Field-programmable Gate Array*

O FPGA (*Field-programmable Gate Array* - Arranjo de Portas Programável em Campo) é um dispositivo semicondutor que pode ser programado depois de fabricado. Em vez de estar restrito a uma função de hardware predeterminada, um FPGA permite a programação de funções e características que se adaptem a novos padrões e seja re-configurado para aplicações específicas mesmo após ser programado. Pode ser usado para implementar praticamente qualquer função lógica que um ASIC (*Application-specific Integrated Circuit* - Circuito Integrado de Aplicação Específica)) poderia realizar (ALTERA, 2015a).

O FPGA consiste de um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar roteamento para comunicação entre elas.

Os blocos lógicos formam uma matriz bidimensional, e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas dos blocos lógicos. Os canais de roteamento possuem chaves de interligação programáveis que permitem conectar os blocos lógicos de maneira conveniente, em função das necessidades de cada projeto, que dá origem ao termo *Place-and-Route* (Local e Rota). Tais blocos contêm LUTs (*Look-Up Tables*), capazes de gerar qualquer função combinacional de *n* variáveis. Além disso, os blocos lógicos possuem *Flip-flops*, capazes de armazenar os resultados gerados pelas lógicas das LUTs (WESTE, 2010).

2.8 *Random-access Memory*

Uma memória de acesso aleatório (RAM), é uma memória que suporta tanto operações de leituras quanto de escritas, com o tempo de acesso para leitura/escrita independente do local físico do dado (MAZUMDER, 1996). Também se caracterizam por serem voláteis, isto é, perdem os dados quando perdem a fonte de energia que às mantêm ativas (WESTE, 2010).

2.9 Simulação *Register-transfer Level* (RTL)

Uma simulação RTL (*Register-transfer Level* - Nível de transferência de Registradores) verifica a funcionalidade do design antes da síntese e do *Place-and-Route*. Essas simulações são independentes de qualquer arquitetura de FPGA escolhido (ALTERA, 2004).

2.10 Simulação *Gate-Level*

A funcionalidade *Place-and-route* no *software* Quartus II produz um design *netlist*, e um arquivo *Standart Delay Format* (SDF - Formato Padrão de Atraso) . A *netlist* consiste da função RTL descrita em uma Linguagem de Descrição de Hardware (HDL - *Hardware Description Language*) mapeada para as primitivas de uma arquitetura específica, como elementos lógicos e elementos de E/S. O arquivo SDF contém as informações de atraso para cada

primitiva da arquitetura e as especificações de roteamento específicas para o design. Juntos, esses arquivos provêm uma simulação precisa do design para a arquitetura do FPGA selecionada (ALTERA, 2004).

2.11 Máquina de estados Finitos

Uma máquina de estados finitos consiste em um conjunto de estados e diretrizes sobre como mudar de estado. As diretrizes são definidas por uma função de próximo estado, que mapeia o estado atual e as entradas para um novo estado (PATTERSON, 2005).

A função de saída especifica o valor dos sinais de saída. Se esta for uma função somente do estado, a saída é conhecida como uma saída Moore. Por outro lado se a saída for uma função do estado e dos sinais de entrada, a saída é conhecida como saída Mealy. Uma máquina de estados finita é chamada Máquina Moore ou Máquina Mealy, se ela conter somente saídas Moore ou saídas Mealy respectivamente (CHU, 2006).

2.12 Análise de Potência

A partir de uma simulação que emula o ambiente em que o componente será exposto, executada sobre a *netlist* gerada pela síntese, um arquivo com o chaveamento dos componentes e interconexões é gerado (geralmente um VCD – *Value Change Dump*).

Após essa etapa, o arquivo com os chavementos é usado, junto da *netlist*, em uma ferramenta específica para medir a potência dissipada durante a operação emulada na simulação do *netlist* (WESTE, 2010).

2.13 ALTSYNCRAM *Memory*

Um dos componentes providos pelo *MegaWizard Pug-In Manager* é a ALTSYNCRAM *Memory* (ALTERA, 2014b). Essa memória é implementada diretamente no FPGA, no modo de uma porta, como foi utilizada no trabalho, pode ser lida ou escrita em um ciclo de *clock* (relógio) (HAMBLEN, 2008).

2.14 RTL *Viewer*

O Quartus II RTL Viewer permite visualizar graficamente a síntese do projeto. Essa visualização não é o projeto final da estrutura, pois ainda não foram feitas as otimizações pelo *software*. Ela mais precisamente representa como o *software* interpreta os arquivos do projeto (ALTERA, 2014c).

2.15 Arquivo de inicialização de memória

Um arquivo de texto com a extensão (.mif) ou (.hex), pode ser criado através do *software* Altera Quartus II e ser usado para especificar o conteúdo inicial de um bloco de memória, ou seja o conteúdo de cada um de seus endereços. Esse arquivo é usado durante a compilação e/ou simulação do projeto (ALTERA, 2012b).

3 TRABALHOS RELACIONADOS

Nesta seção serão apresentados alguns trabalhos e materiais referentes a pesquisas em acervos digitais. Estes tem relação com o controlador de DMA desenvolvido neste trabalho. Alguns dos controladores de DMA pesquisados são de uso específico, desenvolvidos para melhorar o sistema no qual eles são implementados, e outros são controladores de DMA gerais, os quais podem ser integrados a sistemas genéricos. Algumas das ideias para a implementação do controlador de DMA apresentado neste trabalho, bem como para realização de aprimoramentos futuros ao dispositivo proposto, foram inspiradas nas implementações apresentadas.

Embora seja possível encontrar uma grande variedade de trabalhos sobre controladores de DMAs, muitos apresentam as mesmas ideias de implementação, sendo assim, serão apresentados os mais relevantes ao contexto ao qual se destina o controlador desenvolvido no presente trabalho.

Após a apresentação dos trabalhos, será feita uma relação com o contexto no qual o controlador de DMA desenvolvido pelo autor se propõe, avaliando possíveis ideias de implementação.

3.1 *Lightweight* DMA *Management Mechanisms for Multiprocessors on* FPGA

Em (TUMEO, 2008) é apresentado um sistema multiprocessado que adota o acesso direto a memória (DMA) para mover dados entre a memória principal e a memória de cada processador. Cabe salientar que o controlador de DMA em desenvolvimento pelo presente autor não implementa esta funcionalidade, visto que o contexto em que o controlador de DMA será implementado não necessita de transferências no sentido memória para processador. A implementação do controlador de DMA foi uma alternativa escolhida ao uso de caches com complexos protocolos de coerência.

O DMA utilizado no sistema é um DMA simples da empresa Xilinx e não suporta nenhum comando para agendar uma sequência de transferências, isto porque, foi desenvolvido com a intenção de garantir escalabilidade em sistemas multiprocessados e também em FPGAs com poucos recursos.

Para que seja possível esse agendamento de transferências, foi implementada uma fina camada de software capaz de gerenciar uma lista de comandos.

Tais comandos são feitos através de primitivas, que são as possíveis operações do controlador de DMA:

- DMA_get: usada para mover palavras da memória compartilhada para a memória local do processador;
- DMA_put: move palavras da memória local do processador para a memória compartilhada;
- DMA_copy: copia palavras de um local para outro na memória externa;
- DMA_wait: espera o término de uma transferência de DMA.

O buffer circular, então implementado em software, armazena em cada posição:

- Endereço de Origem;
- Endereço de Destino;
- Tamanho da Transferência;
- Variável booleana para indicar se a transferência foi completada.

Ao final de cada transferência é usada uma interrupção para comunicar ao processador do final da mesma.

O buffer usa quatro diferentes índices para seu gerenciamento:

- *Inserted* (Inserido): gerencia as transferências inseridas no buffer pelo processador, apontando para a primeira posição livre no buffer;
- *Launched* (Lançado): são as transferências já iniciadas pelo DMA;
- *Received* (Recebido): transferências que foram confirmadas como completas;
- *Consumed* (Consumido): são as transferências que foram lidas pelo processador.

Para cada uma das primitivas DMA_get, DMA_put ou DMA_copy o gerenciamento do buffer ocorre como descrito a seguir.

Se o buffer circular não estiver cheio, a posição de *Inserted* e *Consumed* forem diferentes, um novo elemento é inserido nele. Caso não haja nenhuma transferência em execução, então a transferência apontada por *Launched* é incrementada e a transferência é iniciada.

Quando uma transferência é completada, uma interrupção é gerada, o índice *Received* é incrementado e a variável booleana é modificada de falso para verdadeiro, indicando que a transferência iniciada pelo DMA está completa.

Se os índices *Launched* e *Inserted* não coincidirem, isso significa que ainda há transferências não executadas e portando o índice *Launched* deve ser incrementado e a próxima transferência iniciada.

3.2 *High Performance Programmable* DMA *Controller* - Intel

O controlador de DMA descrito em (INTEL, 1993) é um controlador de uso geral, de propriedade intelectual da empresa Intel, pode ser integrado a diversos sistemas com microprocessadores, é capaz de realizar transferências entre o dispositivo de E/S e a memória, assim como transferência de dados entre memórias.

São até quatro canais independentes para realizar transferências, expansíveis a qualquer número de canais por meio de cascateando de outros controladores.

Cada canal tem um registrador de 16bits para endereçamento, resultando em 65536 posições de memória endereçadas e outro registrador, também de 16bits, para contagem de palavras transferidas, sendo possível em uma única solicitação de transferência, transferir o máximo de 65536 palavras.

O controlador de DMA opera com basicamente dois ciclos. São eles:

- *Idle Cycle* (Ciclo Inativo): Quando não há requisição de transferência no canal.
- *Active Cycle* (Ciclo Ativo): Estado de atividade do controlador. Este estado contém quatro modos:
    - *Single Transfer Mode* (Modo de Transferência Individual): O controlador irá realizar somente uma transferência.
    - *Block Transfer Mode* (Modo de Transferência em Bloco): O controlador é ativado pelo sinal de requisição de transferência de DMA *e* continua realizando a transferência até completá-la, sendo notificado por um sinal quando o contador de palavras chegar ao final, ou ser notificado por um sinal externo, o qual pode terminar

a transferência. Cabe ressaltar aqui que o sinal de requisição de transferência só precisa permanecer ativo até que o dispositivo reconheça que o controlador de DMA está solicitando uma transferência.

- *Demand Mode* (Modo Demanda): Semelhante ao modo anterior, exceto por continuar realizando transferências até que não haja mais requisições para serem atendidas, ou seja, o sinal de requisição de transferência do DMA estar inativo.
- *Cascade Mode* (Modo Cascata): Usado quando o sistema é estendido para mais de um controlador de DMA. Permite o uso de mais de um controlador de DMA por meio de uma simples expansão do sistema. Para isso alguns sinais do controlador de DMA, o qual irá controlar os demais DMAs cascateados, são ligados aos outros controladores de DMA, criando uma hierarquia de dispositivos. Então o controlador de DMA principal, pode requisitar transferências para os outros controladores de DMA secundários.

3.3 *General Purpose* AHB-AMBA DMA *Controller*

Esse trabalho apresenta um controlador de DMA de uso geral, o qual se integra ao protocolo AMBA-AHB (*Advanced Microcontroller Bus Architecture - Advanced High-performance Bus* - Arquitetura de Barramento Avançado de Microcontrolador - Barramento Avançado de Alta Performance) (ARM, 1999) para realizar as transferências (SHARMA, 2011).

O controlador de DMA tem dois canais capazes de realizar transferências, um para leitura e outro para escrita de dados. Tais transferências podem ser entre a memória e um dispositivo periférico e também entre dois dispositivos periféricos. Após a conclusão da transferência, uma interrupção é gerada para comunicar ao processador da conclusão da transferência.

Existe uma fila de transmissão/recepção para cada canal. Esta fila pode compatibilizar dispositivos com largura de barramentos diferentes e também

serve como um buffer temporário para a comunicação entre os dois dispositivos que estarão realizando a transferência no momento.

Em uma transmissão de dados, no sentido memória para dispositivo periférico, a palavra é lida na memória e então escrita na fila de transmissão e então transferida para o dispositivo periférico. Caso a transferência seja no sentido do dispositivo periférico para a memória, o dado, o qual deve ser transferido, é lido no dispositivo periférico e então, é escrito na fila de recepção sendo escrito logo após na memória.

3.4 DMA *Controller Core* Altera

Controlador proprietário da empresa Altera, de uso geral, é capaz de realizar um volume de transferências de um endereço de origem para um endereço de destino (ALTERA, 2010).

O controlador é composto principalmente por duas portas principais, das quais, uma é utilizada para leitura e outra para escrita e uma porta escrava utilizada para o controle do DMA.

Para configurar uma transferência são informados:

- Endereço de leitura;
- Endereço de escrita;
- Tamanho de uma transferência individual, pode ser de um byte (8-bit), meia palavra (16-bit), palavra (32-bit), palavra dupla (64-bit) ou palavra quádrupla (128-bit);
- Ativação de uma interrupção após conclusão da transferência;
- Ativação da origem ou destino para terminar uma transferência com o sinal de fim de pacote;
- Especificar se a origem e o destino são a memória ou um dispositivo periférico;

O controlador de DMA lê o dado do endereço de origem através da porta *master* (mestre) de leitura e então escreve no endereço de destino através da porta *master* de escrita.

O controlador de DMA também é capaz de realizar transferências em modo *burst* (rajada), onde mais de uma transferência é necessária para

transferir todo o dado solicitado ao controlador. Este modo ocorre de maneira automática, por exemplo, se for configurado o tamanho do dado para uma palavra (32-bit) e for solicitada uma transferência de 64 bytes, o contador de *burst* deverá ser de 16, palavras.

Existe também no controlador de DMA um *buffer*, no estilo fila, entre as portas principais de leitura e de escrita. Ambas as portas, podem controlar o fluxo de dados fazendo com que o periférico possa encerrar a transação do DMA enviando um sinal de fim de pacote ao controlador de DMA.

Quando se está acessando a memória, o endereço para leitura ou escrita pode ser incrementado por (1, 2, 4, 8, 16), após cada acesso, dependendo da largura do dado. Isto ocorre uma vez que a palavra a ser acessada na memória pode estar localizada em mais de uma posição.

3.5 *A Direct Memory Access Controller* (DMAC) IP-*Core using the* AMBA AXI *Protocol*

O controlador de DMA apresentado nesse trabalho é inspirado no Projeto Dalton DMAC (*Direct Memory Access Controller* - Controlador de Acesso Direto à Memória), sendo também compatível com o amplamente utilizado Intel I8237 (CORREA, 2011; INTEL, 1993).

A fim de torná-lo adequado para aplicações embarcadas atuais, o seu padrão de comunicação segue o protocolo AMBA (*Advanced Microcontroller Bus Architecture* - Arquitetura de Barramento Avançado de Microcontrolador) AXI (*Advanced eXtensible Interface* - Interface Avançada Extensível), o que permite uma maior flexibilidade, como menos restrições de tempo e transferência de dados no sistema com carga mínima de CPU (*Central Processing Unit* - Unidade Central de Processamento).

O controlador de DMA atua como uma ponte, onde ele recebe uma quantidade de dados, os armazena em *buffers* internos e encaminha o dado para o destino. Esta operação é repetida até que todo o dado seja transferido.

O controlador de DMA também lida com a prioridade entre todos os dispositivos que requerem acesso de memória, caso duas requisições aconteçam ao mesmo tempo, a de maior prioridade será atendida primeiro.

Os componentes do controlador mantêm informações de direção de transferência, número de palavras a serem transferidas, tamanho de cada palavra, prioridade da transferência, qual o dispositivo para transferir, informações temporárias necessárias para realizar a transferência atual e uma memória para armazenar os dados para serem repassados.

As principais características do controlador de DMA utilizado no trabalho são:

- Registradores: usados para armazenar informações da CPU, tais como direção da transferência, número de palavras transferidas, endereços de transferências e prioridade da transferência.
- Multiplexadores: Em cada transferência do controlador de DMA, o mesmo lê um bloco de palavras do emissor e passa para frente para o receptor. Para realizar a transferência adequadamente, um sinal externo deve ser selecionado entre os sinais provenientes da memória ou a partir de dispositivos. Este é o papel de multiplexadores: geram sinais internos a parir de sinais externos, e, assim, o bloco de controle lida apenas com sinais internos, o que torna seu projeto mais fácil. Por exemplo, se a transferência a ser realizada pelo controlador for no sentido da memória para o dispositivo periférico, primeiramente o controlador irá informar o endereço de leitura na memória e após a memória responder, com a palavra solicitada pelo controlador colocando-a nas linhas de dado de leitura, o controlador de DMA, saberá que os dados estão lá com base nos multiplexadores, que irão gerar tais sinais para informá-lo do que aconteceu.
- *Buffers*: são usados para armazenar um bloco de palavras, este então só será transferido quando o buffer estiver cheio.
- Bloco de controle: Este componente fornece os sinais de controle para o caminho de dados, sendo altamente dependente da forma como os componentes do caminho de dados são organizados. O bloco de controle é composto por duas máquinas de estado: a máquina prioridade e a máquina de transferência. O primeiro administra prioridade entre dispositivos e controla o início da operação, enquanto que o último gere os sinais internos utilizados para controlar o caminho de dados e os sinais externos, utilizado em interrupções e no protocolo AXI.

3.6 LogiCORE IP AXI DMA v7.1

O controlador de DMA a seguir descrito é um controlador proprietário da empresa Xilinx (XILINX, 2014). Tem como característica prover acesso direto à memória com alta largura de banda entre a mesma e um periférico com protocolo AXI4-*Stream*.

Suporta transferências com largura de dado, amparado pelo protocolo AMBA AXI4, de 32, 64, 128, 256, 512 e 1024 bits. Também suporta fluxo (*Stream*) de dados, através do protocolo AXI4-*Stream*, com largura de banda 8, 16, 32, 64, 128, 256, 512 e 1024 bits.

Uma característica interessante neste controlador é a opção espalhar/reunir (*Scatter/Gather*). Tal característica faz com que seja possível, em uma única transferência de memória, transferir mais de um bloco que não esteja em um endereço contíguo de memória.

O controlador de DMA também oferece acesso de alta largura de banda entre a AXI4 *Memory Mapped Interface* e a AXI4-*Stream Interface*. A inicialização, o status e os Registradores de gerenciamento são acessíveis através da AXI4-*Lite Slave Interface*.

O movimento de dados primário de alta velocidade realizado pelo controlador de DMA entre o sistema de memória e o *stream* alvo é através do AXI4 *Read Master* para AXI MM2S *Stream Master*, e do AXI S2MM *Stream Slave* para AXI4 *Write Master*. O controlador de DMA também permite até 16 canais múltiplos de movimentação de dados em ambos os caminhos MM2S e S2MM em modo *Scatter/Gather*.

O canal MM2S (*Memory-Mapped to Stream* - Mapeada em Memória para Fluxo) e o canal S2MM (*Stream to Memory Map* - Fluxo para Memória Mapeada) operam independentemente. O controlador de DMA oferece 4 KB de proteção de fronteira de endereço, particionamento *burst* automático, bem como fornece a capacidade de enfileiramento de vários pedidos de transferência usando quase toda a capacidade de largura de banda do barramento AXI4-*Stream*. Além disso, o AXI DMA oferece realinhamento de dados em nível de byte permitindo que leituras e escritas na memória comecem com um offset de deslocamento.

O canal MM2S suporta um AXI *Control Stream* (Fluxo de Controle AXI) para o envio de dados de aplicativos do usuário para o IP de destino. Para o canal S2MM, um AXI *Status Stream* (Fluxo de Estado AXI) é fornecido para receber os dados de aplicativos do usuário a partir do IP de destino.

A *Engine Scatter/Gather* busca e atualiza o *buffer* de do sistema de memória através da AXI4 *Scatter Gather Read/Write Master Interface*. Uma fila opcional de descritores é oferecida para maximizar a vazão de dados primária.

3.7 Análise sobre os Trabalhos

Analisando os trabalhos descritos, muitas das ideias estão presentes no controlador de DMA desenvolvido neste trabalho, entre elas: o uso de registradores para armazenar os endereços iniciais de cada dispositivo onde deverá ser realizada a transferência, o registrador de tamanho da transferência, que faz com que o controlador de DMA possa realizar uma transferência de tamanho maior que somente um bloco sem a intervenção do processador, um sinal gerado pelo controlador de DMA que pode informar o processador do final da transferência e também o sentido da transferência que pode ser do controlador de dispositivo de E/S para a controladora de memória ou vice-versa.

Algumas das ideias para controle de prioridades de transferências utilizadas em (CORREA, 2011) são bastante interessantes de serem implementadas já que o controlador de DMA desenvolvido no trabalho poderá ter de lidar com múltiplas requisições. Também neste cenário onde múltiplas requisições podem ocorrer, o *buffer* circular utilizado em (TUMEO, 2008), é bastante interessante, pois o processador poderá configurar um número, a ser definido, de requisições sem ter de esperar que a transferência que está em andamento no controlador de DMA termine para que seja possível solicitar uma nova transferência. Cabe ressaltar que em (TUMEO, 2008), o controle do *buffer* é feito em *software*, o que consequentemente, faz com que seja possível usar mais recursos para o gerenciamento do mesmo.

Especulando o uso do controlador de DMA em um *proxy* de vídeo, o qual é um possível alvo para o controlador desenvolvido, a utilização de mecanismos que possam dar suporte a múltiplas requisições torna-se

interessante, pois espera-se de um *proxy* de vídeo, que o mesmo dê suporte a múltiplos usuários, tendo que atendê-los muitas vezes simultaneamente. Nesse caso, se o processador puder configurar várias transferências de DMA, quantas forem necessárias até um número máximo suportado pelo controlador de DMA, pode-se evitar ciclos de ociosidade do processador e também do controlador de DMA, otimizando a entrega dos blocos de vídeo aos usuários finais.

Uma alternativa para quando uma transferência solicitada ao controlador DMA, necessitar de mais de uma transferência entre a controladora de memória e o controlador de dispositivo de E/S é a utilização de *buffers*, como descrito em (CORREA, 2011). Com a implementação de buffers entre o controlador de dispositivo de E/S e a controladora de memória, pode-se reduzir o número de ciclos de *clock* necessários para transferir o bloco inteiro solicitado ao controlador de DMA. Por exemplo, se a transferência de um bloco necessitar de cinco transferências entre o controlador de dispositivo de E/S e a controladora de memória, ao invés de solicitar o dado em um dos dispositivos e logo após escrever no outro, pode-se solicitar cinco vezes o dado a o dispositivo de origem, armazená-lo em um buffer e só após ter o bloco completo, escrevê-lo no dispositivo de destino da transferência. Este caso precisa ser analisado com cuidado, pois um *buffer* que suporte um bloco inteiro, dependendo do tamanho deste bloco, pode consumir muitos recursos do FPGA. Ainda analisando o uso de um *buffer* como intermediário entre os dispositivos que realizam a transferência, cabe salientar que o principal benefício do seu uso é resolver o problema decorrente da diferença de velocidade entre os dispositivos, no qual o dispositivo mais rápido escreve no buffer e não precisa esperar pela disponibilidade do dispositivo mais lento.

Analisando (SHARMA, 2011), mais precisamente as filas de transmissão/recepção de cada canal, estas são uma alternativa para a implementação dos *buffers* necessários para fazer com que o controlador de DMA desenvolvido nesta monografia possa realizar transferências entre dispositivos que tenham diferentes larguras de dados em suas interfaces.

Assim como em (ALTERA, 2010), o controlador de DMA desenvolvido pelo autor desta monografia possui registradores de status que podem ser consultados para saber o status do mesmo. O controlador de DMA também utiliza o método de interrupção para informar ao processador do término de

uma transferência. O modo *burst*, apresentado em (ALTERA, 2010), assemelha-se com o do autor da monografia, diferindo pelo modo como o processador o configura. Ainda em (ALTERA, 2010) o número de transferências necessárias para transferir um bloco inteiro, é calculado automaticamente com base no tamanho do dado de uma transferência individual e assim, calcula-se o número de transferências necessárias para todo o bloco. No trabalho desenvolvido pelo autor da monografia, o processador já deve informar o número de transferências necessárias para transferir o bloco inteiro.

O controlador de DMA apresentado em (INTEL, 1993), possui basicamente dois ciclos primários, sendo semelhante ao desenvolvido pelo autor neste trabalho. Existe o ciclo de ociosidade do controlador de DMA, o qual fica neste estado até que seja solicitada uma transferência ao controlador e o ciclo de atividade. O ciclo de atividade, desenvolvido pelo autor desta monografia, divide-se em dois ciclos: um de configuração do controlador e outro de realização da transferência. Esse ciclo de realização da transferência, assemelha-se com o estado *Block Transfer Mode* implementado pelos autores.

O estado chamado *Demand Mode* cria inspirações para a implementação do DMA pretendida para este trabalho, porém, exige que seja também implementado o *buffler* circular apresentado em (TUMEO, 2008), ou algo semelhante que possa armazenar uma fila de requisições.

Analisando (XILINX, 2014), as funcionalidades mais interessantes, exceto as que já foram mencionadas, são a possibilidade de envio de dados para um endereço IP e o modo *Scatter/Gather*. Este último pode ser utilizado para um melhor aproveitamento da memória do sistema, tal funcionalidade deve ser analisada com um olhar especial na controladora de memória, pois a mesma não suporta o acesso a bancos individuais, como será descrito ao longo do trabalho.

Como já mencionado, as ideias apresentadas nos trabalhos analisados são interessantes e podem acrescer desempenho e funcionalidades ao controlador de DMA que está sendo desenvolvido, porém cada uma tem seu impacto no desempenho do controlador e em sua síntese no FPGA, no qual o sistema será implementado. Cabe então analisá-las profundamente e avaliá-las quanto a sua viabilidade de implementação.

A seguir na tabela 1, uma síntese das principais ideias apresentadas nos trabalhos e suas implementações no presente trabalho. Também são mostradas quais ideias foram implementadas ou não pelo autor.

Tabela 1 - Síntese dos trabalhos

Ideia	**Implementado?**
Movimentação de dados entre memória principal e a memória do processador	Não
Transferências entre controladora de memória e controlador de E/S	Sim
Buffer circular e índices	Não
Endereço de Origem	Sim
Endereço de Destino	Sim
Tamanho da transferência	Sim
Interrupção ao final da Transferência	Sim
Múltiplos canais para transferência de dados	Sim
Ciclos de Operação	Sim
Fila de Transmissão	Não
Buffer Temporário	Não
Ativação da origem ou destino para terminar uma transferência com o sinal de fim de pacote	Não
Especificar se a origem e o destino são a memória ou um dispositivo periférico	Não
Incremento do endereço de leitura/escrita em quantidades variáveis	Não
Prioridade das Transferências	Não
Modo Scatter/Gather	Não
Envio de dados para endereço IP	Não

Fonte: Próprio Autor.

4 ARQUITETURA DO SISTEMA

O sistema composto pelos controladores de DMA, E/S e memória pode ser parametrizado em nível de projeto (CHU, 2006). Os detalhes sobre essa parametrização serão abordados ao longo do texto.

O controlador de DMA é capaz de realizar um número configurável de transferências tanto da controladora de memória para o controlador de E/S, quanto do controlador de E/S para a controladora de memória.

A controladora de memória possui a função de *interleaving* de dados (WEI, 2009), realizando ou não o *interleaving*, dependendo da configuração do sistema.

O dispositivo de E/S será simulado por uma memória RAM, mas manterá as características de um dispositivo de E/S. Também serão abordados os detalhes ao longo do trabalho.

4.1 Parametrização

O sistema pode ser parametrizado nos seguintes pontos:

- Tamanho do bloco: Tamanho em bits de uma única transferência realizada pelo DMA.
- Número de bancos de memória: Quantidade de bancos usado pela controladora de memória para leitura/escrita do bloco transferido pelo DMA.
- Tamanho do dado: Tamanho em bits do dado lido/escrito em cada banco da controladora de memória.
- Tamanho do endereço: Tamanho em bits do endereço de acesso da controladora de memória e do controlador de E/S.

A parametrização completa do sistema não se resume aos itens citados acima e será abordada novamente quando utilizada.

Uma restrição importante para a parametrização do sistema é que o tamanho do bloco deve ser um múltiplo de (tamanho do dado x número de bancos). Esse detalhe será abordado na seção 4.3.1

4.2 Controlador de E/S

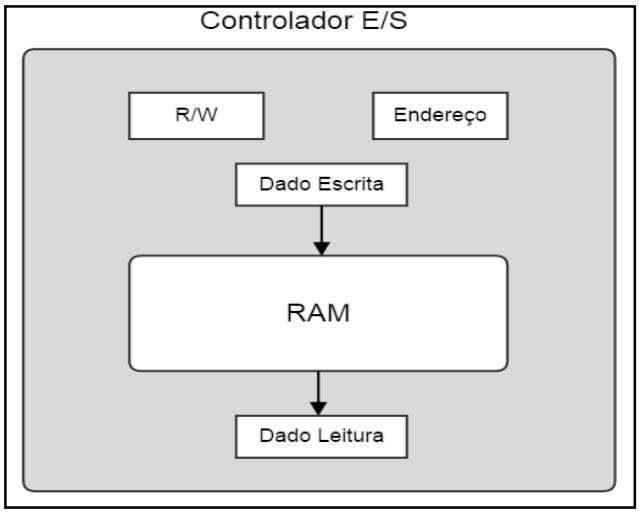
Como citado anteriormente, para dar suporte ao desenvolvimento e a uma posterior validação do controlador de DMA, o dispositivo de E/S, em não se tratando do foco central deste trabalho, foi simulado por uma memória RAM. O modelo utilizado foi Altera ALTSYNCRAM de uma porta. Embora a memória RAM utilizada seja uma memória síncrona, a comunicação com o controlador de DMA é feita de maneira assíncrona, através do protocolo *handshaking*. O modo como o controlador executa o protocolo e o próprio protocolo serão abordados no capitulo 4.4, que trata o controlador de DMA.

A interface de comunicação do controlador de E/S com o controlador de DMA é parametrizado com a largura do tamanho do bloco, que é o tamanho único de suas transferências, ou seja, a cada operação de leitura/escrita no controlador de E/S são transferidos o número de bits do tamanho do bloco. O espaço de endereçamento, ou tamanho do endereço, também é parametrizado, significando que o controlador de E/S terá a capacidade de 2^(Tamanho do endereço) blocos.

Para inicializar a memória do controlador foi usado um arquivo de inicialização (ALTERA, 2012b).

A seguir, na figura 3, uma ilustração do componente desenvolvido.

Figura 3 - Controlador de E/S



Fonte: Próprio Autor.

4.3 Controladora de memória

A memória utilizada pela controladora é uma memória RAM de uma porta, modelo Altera ALTSYNCRAM, que lê e escreve dados em um ciclo de *clock*.

A escolha do componente deu-se em função da simplicidade de funcionamento e fácil adequação ao sistema proposto.

4.3.1 Parametrização

A controladora de memória também é parametrizada, assim como o controlador de E/S, pelo tamanho do endereço de leitura/escrita. A interface de comunicação com o controlador de DMA é parametrizado conforme o tamanho do bloco, que também é o tamanho de uma única transferência entra a controladora de memória e o controlador de DMA.

Como citado anteriormente, o tamanho do bloco deve ser um múltiplo de (tamanho do dado x número de bancos). Isso acontece pois a controladora não consegue acessar individualmente cada um dos bancos de memória, se houver mais de um no caso. Se o tamanho do bloco não for um múltiplo, o dado nem sempre começará pelo primeiro banco de memória e esse cenário não é tratado pela controladora de memória.

Outro componente da controladora, também parametrizado é o número de bancos de RAM, que são gerados através do comando *for generate*.

4.3.2 *Memory Interleaving*

Um recurso utilizado pela controladora para melhorar o desempenho nas leituras/escritas é a distribuição do dado em diversos bancos de memória paralelos. Ao realizar a transferência de um bloco na controladora, desde que a mesma seja parametrizada com mais de um banco de memória, a leitura/escrita do bloco é feita em paralelo nos bancos.

Supondo um cenário em que o tamanho do bloco seja de 128 bits, que existam quatro bancos de memória e que cada banco tenha 32 bits de largura,

ao custo de uma leitura/escrita em paralelo a cada banco a controladora realiza a transferência. A figura 4 ilustra a suposição.

Figura 4 - Ilustração do *interleaving*

Fonte: Próprio Autor.

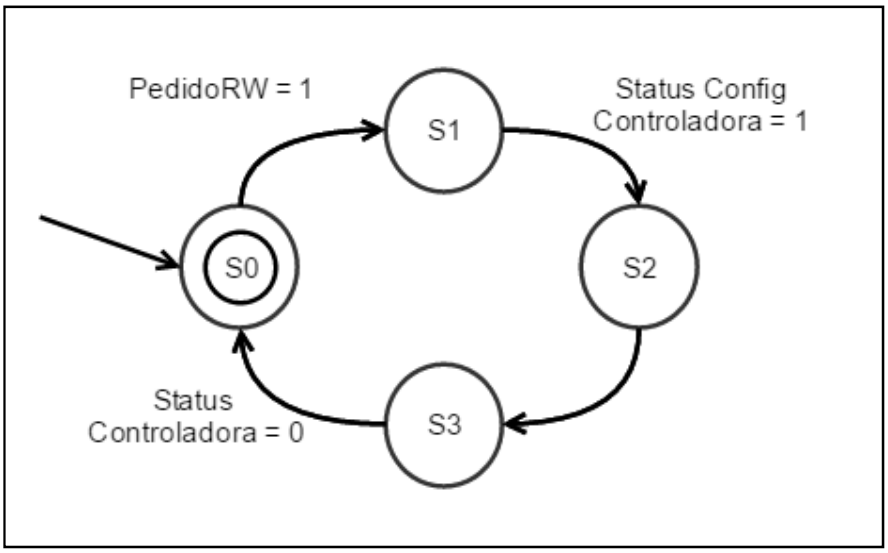
A controladora também tem pode ler/escrever um tamanho de bloco que necessite de mais de uma leitura/escrita aos bancos em paralelo. Na situação descrita anteriormente, se o bloco tiver um tamanho de 256 bits, serão precisas duas leituras/escritas em paralelo aos bancos. Mesmo nesse cenário são lidos/escritos em uma transferência quatro vezes mais dados do que uma transferência para um banco de largura de 32 bits.

Outro fator importante tratado pela controladora é o endereçamento dos dados, o qual é abstraído do controlador de DMA. Seguindo o exemplo anterior em que o bloco tem um tamanho de 256 bits, serão necessários dois endereços de cada banco para armazenar o dado, mas tanto o controlador de DMA quanto o processador não precisam mudar a maneira de endereçar os dados, pois a controladora fará essa tarefa. Essa funcionalidade será abordada mais profundamente ao longo da seção.

4.3.3 Unidade de Controle

A unidade de controle, responsável por comandar o dispositivo é uma Máquina de Estados Finita e funciona da forma como mostrada na figura 5.

Figura 5 - FSM da controladora de memória.



Fonte: Próprio Autor.

A seguir, na tabela 2 as transições de estados da controladora de memória.

Tabela 2 - Transição de estados da controladora de memória

Estado Atual	Estado Futuro	Condição	Sinais Gerados
s0	s1	PedidoRw = 1	
s1	s2	StatusConfig Controladora = 1	MudaPedidoRw MudaStatusConfigControladora
s2	s3		MudaStatusControladora MudaStatusConfigControladora
s3	s0	Status Controladora = 0	

Fonte: Próprio Autor.

A máquina inicia no estado s0, mudando para o estado s1 quando o sinal PedidoRW muda para 1. Quando a máquina está no estado s1 é ativado o sinal MudaPedidoRw, o qual abaixa o nível do sinal PedidoRW, nesse estado também é ativado o sinal MudaStatusConfigControladora. Ao perceber o sinal StatusConfigControladora em nível alto a máquina de estados muda para o estado s2. No estado s2 é ativado o sinal MudaStatusConfigControladora, mudando para nível baixo o sinal StatusConfigControladora, também é ativado o sinal MudaStatusControladora fazendo com que o sinal StatusControladora

fique em nível alto. Nesse ponto a máquina de estados muda para o estado s3 na qual realiza as leituras/escritas e permanece no mesmo até que sejam realizadas todas as leituras/escritas necessárias, quando a mesma atinge o total, o sinal StatusControladora muda para nível baixo, isso ocorre fora da máquina de estados e a mesma ao perceber tal mudança vai para o estado s0.

A máquina de estados acima, pode ser considerada uma máquina de Moore, pois os sinais de saída dependem somente do estado em que a máquina se encontra.

4.3.4 Leitura/Escrita de um bloco

A leitura/escrita de um bloco na controladora de memória ocorre como descrita ao longo da seção.

4.3.5 Configuração da controladora

Ao receber um pedido de leitura/escrita, primeiramente a controladora verifica se não existe uma transferência em andamento ou se a mesma não está sendo configurada para realizar uma transferência. Isso permite que a controladora trabalhe em um cenário que exista mais de um dispositivo que possa solicitar um dado. Caso a controladora esteja ocupada no momento, o pedido será ignorado e deverá ser feito novamente quando a mesma não estiver ocupada.

Se a controladora não estiver ocupada existem dois cenários possíveis de transferências:

- Leitura de um bloco - deve ser informado o endereço de leitura e o tipo de operação, no caso uma leitura.
- Escrita de um bloco - deve ser informado o bloco a ser escrito, o endereço para escrita e o tipo de operação, nesse caso uma escrita.

Lembrando que o endereço a ser informado, nos dois casos, é um endereço que não leva em consideração a existência ou não do *interleaving*. Como o endereçamento é abstraído do controlador de DMA o número de leitura(s)/escrita(s) necessárias para a transferência de um bloco é calculada a nível de projeto, através da seguinte fórmula:

$$\text{Nro. Leitura(s) ou Escrita(s)} = \frac{\text{Tamanho do bloco}}{\text{Nro. de bancos de RAM x Tamanho do dado}}$$

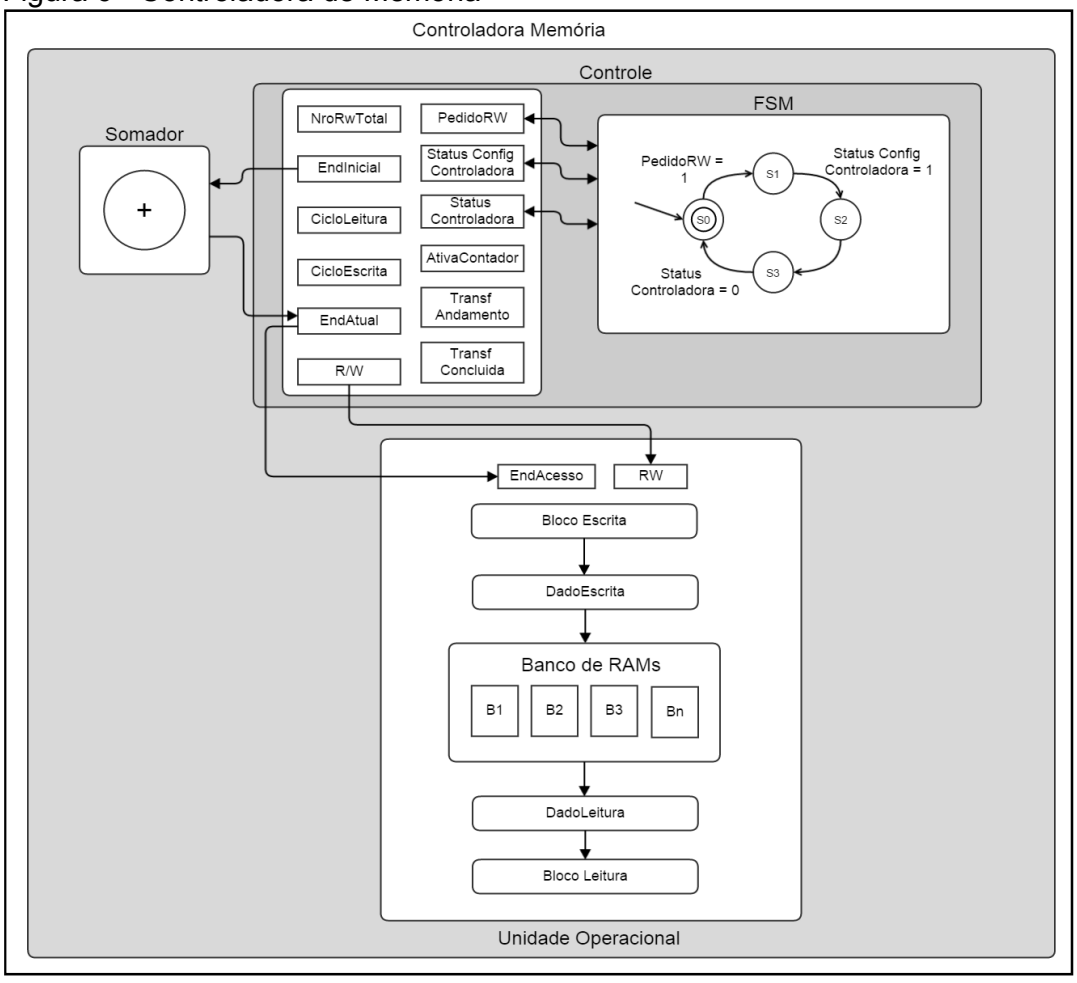
Conhecendo o número de leitura(s) ou escrita(s) necessárias para cada bloco pode-se calcular o endereço onde reside o bloco no(s) banco(s) de memória(s), através da seguinte fórmula:

$$\text{Endereço inicial} = \text{Nro. Leitura(s) ou Escrita(s)} \times \text{Endereço Leitura ou Escrita}$$

O endereço inicial será o endereço onde inicia o bloco solicitado à controladora de memória.

A seguir, na figura 6, um diagrama que ilustra, considerando algumas abstrações, o componente desenvolvido.

Figura 6 - Controladora de Memória



Fonte: Próprio Autor.

4.3.6 Execução da transferência

Após a configuração da controladora, a mesma entra em um estado de ocupada e começa a realizar a transferência.

Em um cenário onde são necessárias mais de uma leitura/escrita no(s) banco(s) de memória a controladora deve gerar o próximo endereço de acesso após a conclusão de cada operação nos bancos.

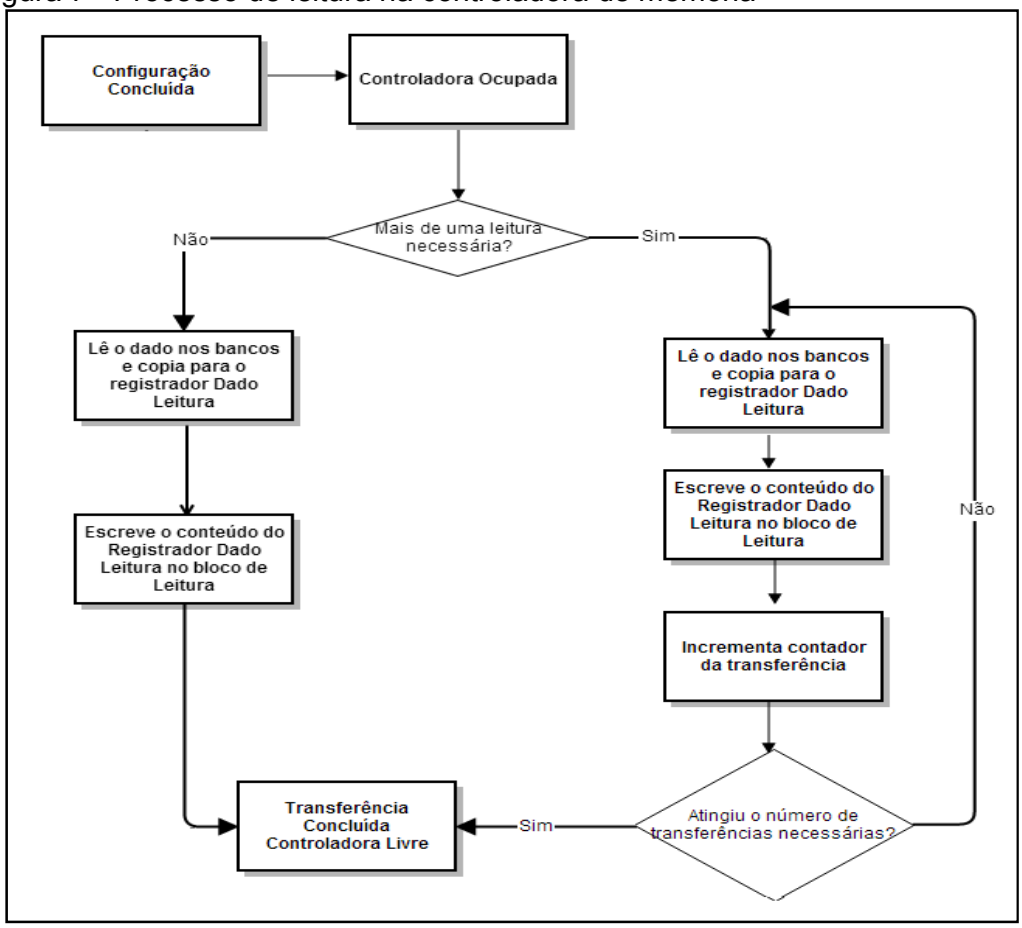
4.3.6.1 Leitura na controladora

Quando a operação solicitada para a controladora é uma leitura, após o dado ser lido no(s) banco(s) de memória, ele é escrito em um registrador de dado de leitura, que possui o tamanho de (tamanho do dado x número de bancos de memória) pois esse é o tamanho da interface de acesso aos bancos de memória pela controladora. Caso seja necessária apenas uma leitura para entregar o dado completo solicitado à controladora de memória, o conteúdo desse registrador é copiado para um registrador de bloco de leitura e então para a interface de leitura da controladora.

Se for necessária mais de uma leitura nos bancos de memória para entregar o dado solicitado à controladora, a mesma, a cada leitura nos bancos de memória, escreve o conteúdo do registrador de dado de leitura em uma posição diferente do bloco de leitura, de modo que ao final da operação na controladora, o bloco de leitura contém o dado solicitado.

O processo de leitura ocorre de maneira simplificada como mostrado na figura 7.

Figura 7 - Processo de leitura na controladora de memória



Fonte: Próprio Autor.

4.3.6.2 Escrita na controladora

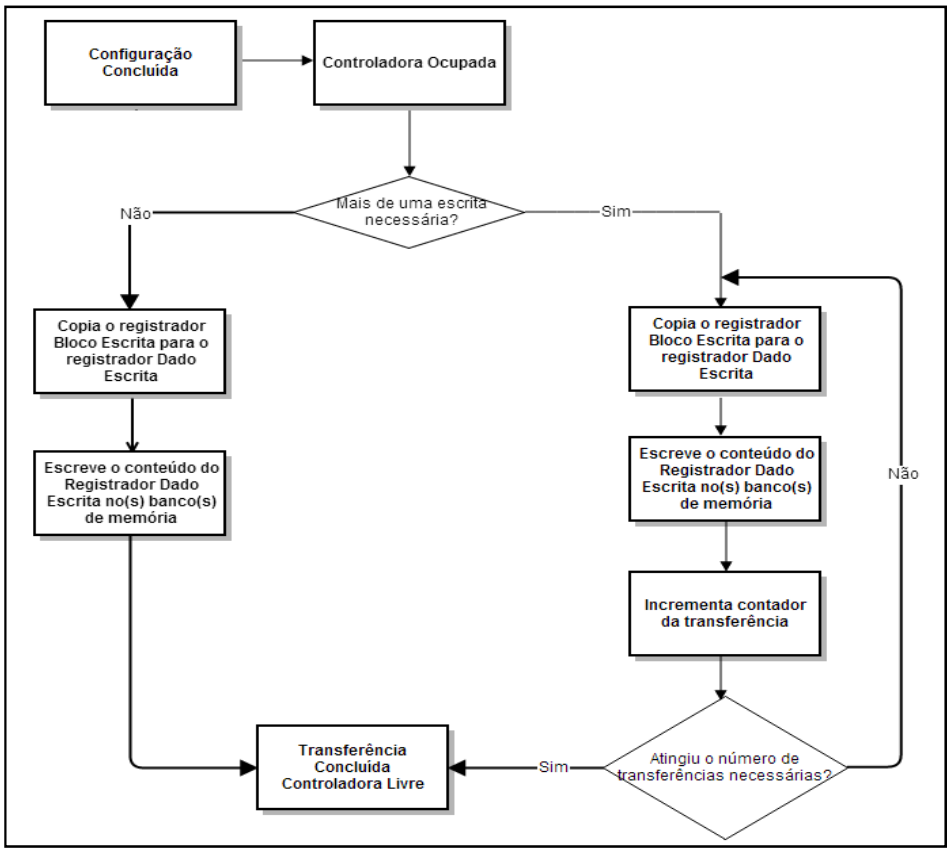
Uma operação de escrita é semelhante a uma leitura, diferindo basicamente na ordem das operações.

O bloco de escrita é copiado para um registrador interno da controladora e após isso, se necessário, deve ser dividido de modo que seja do tamanho de (Tamanho do dado x Número de bancos de memória) e copiado para um registrador de dado de escrita.

Após a escrita do conteúdo desse registrador na memória, se necessário é copiada a próxima parte do registrador de bloco de escrita para o registrador de dado de escrita, e assim escrito novamente nos bancos de memória.

O processo de escrita ocorre de maneira simplificada como mostrado na figura 8.

Figura 8 - Processo de escrita na controladora de memória.



Fonte: Próprio Autor.

4.3.7 Conclusão da transferência

Após todo o bloco ser lido ou escrito na controladora é gerado um sinal para comunicar a conclusão da transferência, a controladora muda o estado para livre e está apta a receber novos pedidos.

4.4 Controlador de DMA

O controlador de DMA realiza transferências de blocos, com tamanho parametrizável pelo tamanho do bloco, entre controlador de E/S e a controladora de memória. As transferências podem ser de um ou mais blocos e nos dois sentidos, E/S para memória e memória para E/S.

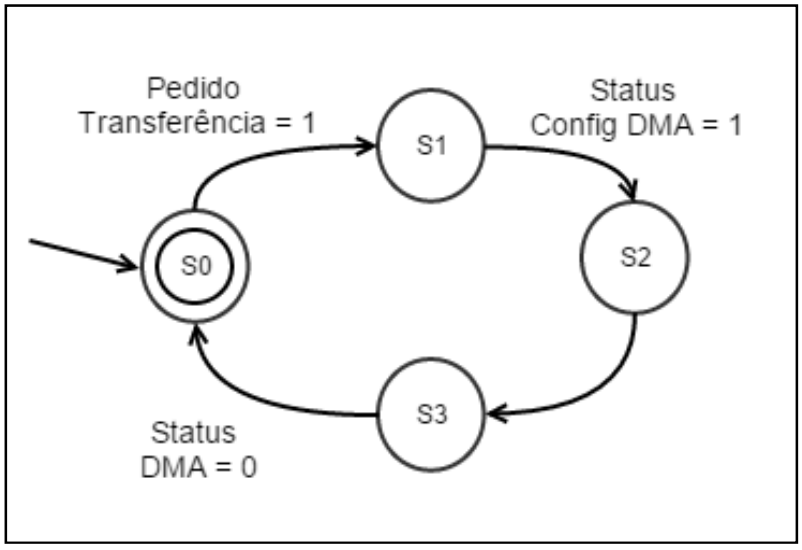
Para comunicação com o controlador de dispositivo de E/S é utilizado o protocolo de *handshaking.* A comunicação com a controladora de memória ocorre como descrita na seção 4.3.

Para simplificar o entendimento, algumas vezes durante essa seção, será omitido o termo controlador quando o autor estiver se referindo aos controladores de DMA, E/S e a controladora de memória.

4.4.1 Unidade de Controle

Assim como a controladora de memória, o controlador de DMA é operado por uma máquina de estados finita, a qual opera como mostrado na figura 9.

Figura 9 - FSM do controlador de DMA



Fonte: Próprio Autor.

Ao receber um pedido de transferência, sinal PedidoTransferencia em nível alto, o controlador de DMA muda do estado inicial s0 para o estado s1. Estando no estado s1, o sinal MudaPedidoTransferencia é ativado, abaixando o sinal PedidoTransferencia, o sinal MudaStatusConfigDMA também é ativado, fazendo com que o sinal StatusConfigDMA passe para nível alto, e o estado mude para s2. No estado s2 são ativados os sinais MudaStatusConfigDMA e MudaStatusDMA, resultando em uma mudança para nível baixo do sinal StatusConfigDMA e uma mudança para nível alto do sinal StatusDMA, assim a máquina de estados passa para o estado s3. Durante o estado s3, o controlador de DMA realiza as transferências solicitadas, quando o controlador

atinge o número total de transferências, um sinal externo muda o valor de StatusDMA para 0, fazendo com que a FSM mude para o estado s0, indicando o fim da transferência.

A seguir, na tabela 3 as transições de estados do controlador de DMA.

Tabela 3 - Transição de estados do controlador de DMA

Estado Atual	Estado Futuro	Condição	Sinais Gerados
s0	s1	Pedido Transferencia = 1	
s1	s2	Status ConfigDMA = 1	MudaPedido Transferencia MudaStatusConfigDMA
s2	s3		MudaStatusDMA MudaStatusConfigDMA
s3	s0	StatusDMA = 0	

Fonte: Próprio Autor.

Assim como a máquina de estados da controladora de memória, a máquina de estados do controlador de DMA, pode ser considerada uma máquina de Moore.

4.4.2 Protocolo de *Handshaking*

O controlador de DMA implementa o protocolo de comunicação assíncrono *handshaking* para comunicar-se com o controlador de E/S, embora o dispositivo de E/S seja simulado por uma memória RAM, decidiu-se utilizar esse protocolo para uma simulação mais realista.

O protocolo constituí-se de três sinais:

- RLE: usado para indicar uma requisição de leitura ou escrita;
- DP: usado para indicar que o dado está disponível na linha de dados;
- Ack: usado para confirmar o conhecimento do sinal ReqLeitura ou DadoPronto da outra parte;

Uma transferência entre o controlador de DMA e o controlador de E/S acontece em sete passos. Conforme citado na seção 4.2, o controlador de E/S implementa alguns passos do *handshaking*.

A tabela 4 mostra onde é implementado cada um dos passos.

Tabela 4 - Passos do *handshaking*

PASSO	CONTROLADOR DMA		CONTROLADOR E/S	
	TIPO DA TRASFERÊNCIA		TIPO DA TRASFERÊNCIA	
	DMA PARA E/S	E/S PARA DMA	DMA PARA E/S	E/S PARA DMA
1			X	X
2	X	X		
3			X	X
4	X			X
5		X	X	
6	X			X
7		X	X	

Fonte: Próprio Autor.

4.4.2.1 DMA para E/S

Em uma transferência do controlador de DMA para o controlador de E/S, os passos acontecem como descrito na tabela 5.

Tabela 5 - Passos do *handshaking* para uma transferência no sentido DMA - E/S

PASSO	Descrição	Ack	RLE	DP
1	E/S vê RLE, lê endereço, levanta Ack	0	1	0
2	DMA vê Ack, baixa RLE	1	1	0
3	E/S vê RLE, baixa Ack	1	0	0
4	DMA põe dado, levanta DP	0	0	0
5	E/S vê DP, lê dado, levanta Ack	0	0	1
6	DMA vê Ack, baixa DP	1	0	1
7	E/S vê DP, baixa Ack	1	0	0

Fonte: Próprio Autor.

4.4.2.2 E/S para DMA

Quando a transferência é do controlador de E/S para a controlador de DMA, os passos acontecem como descrito na tabela 6.

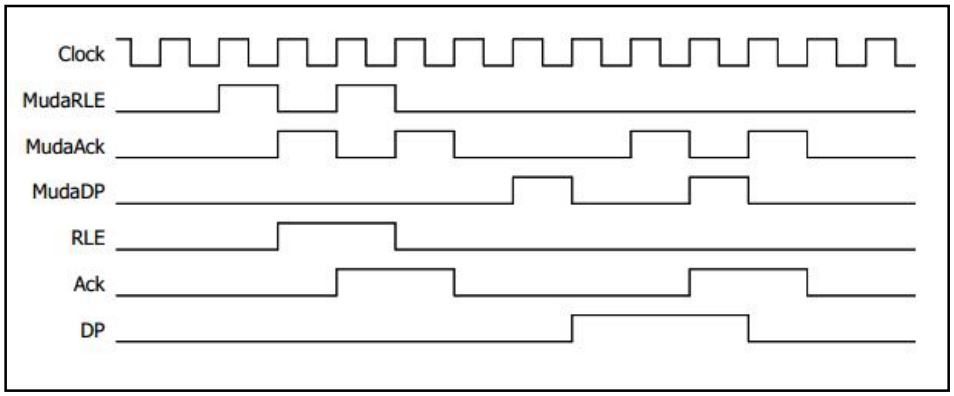
Tabela 6 - Passos do *handshaking* para uma transferência no sentido E/S - DMA

PASSO	Descrição	Ack	RLE	DP
1	E/S vê RLE, lê endereço, levanta Ack	0	1	0
2	DMA vê Ack, baixa RLE	1	1	0
3	E/S vê RLE, baixa Ack	1	0	0
4	E/S põe dado, levanta DP	0	0	0
5	DMA vê DP, lê dado, levanta Ack	0	0	1
6	E/S vê Ack, baixa DP	1	0	1
7	DMA vê DP, baixa Ack	1	0	0

Fonte: Próprio Autor.

A seguir, na figura 10 a ilustração do *handshaking* citado nas tabelas 5 e 6.

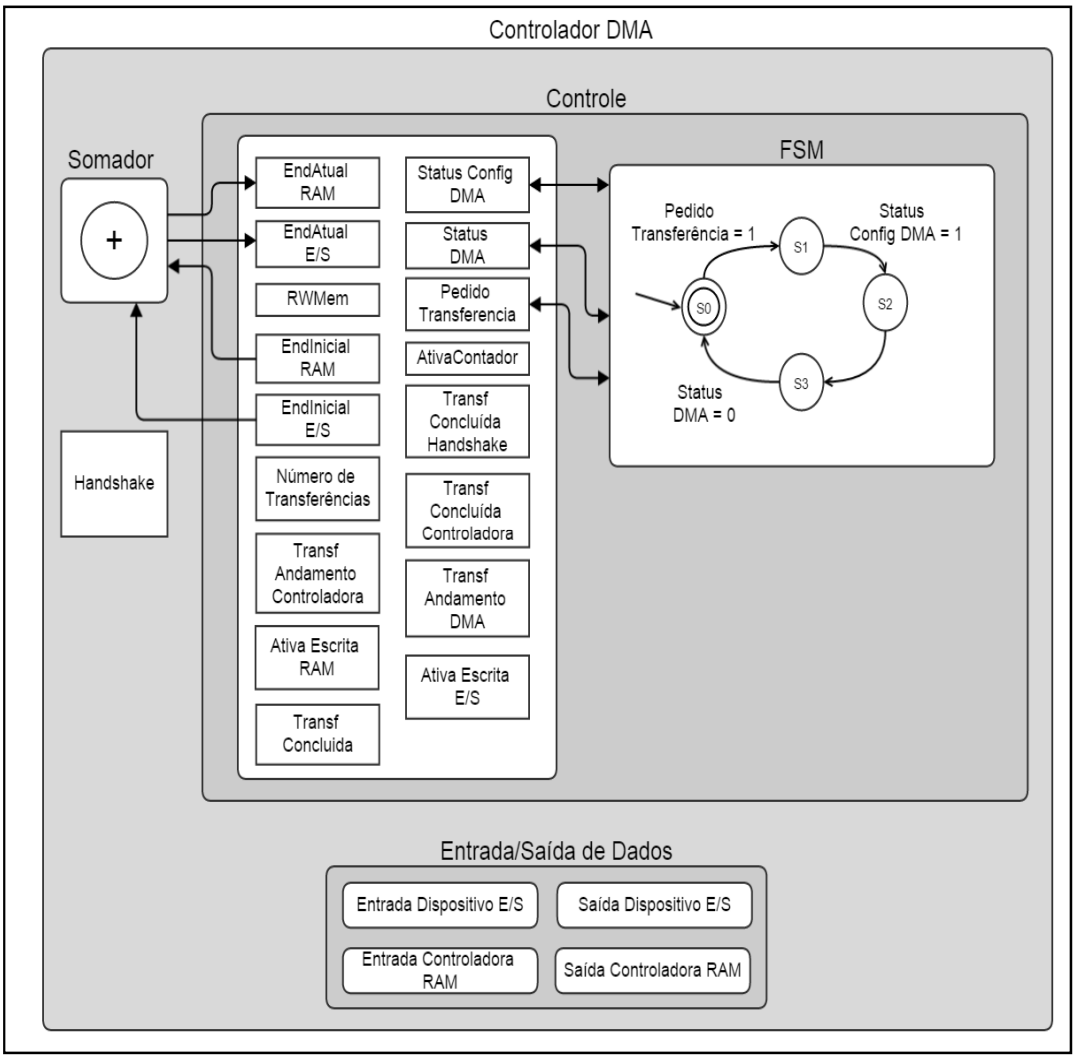
Figura 10 - Ilustração do *Handshaking*



Fonte: Próprio Autor.

A seguir, na figura 11, uma ilustração do componente desenvolvido, considerando algumas abstrações.

Figura 11 - Ilustração do controlador de DMA



Fonte: Próprio Autor.

4.4.3 Realizando uma transferência

Assim como já citado, o controlador de DMA pode realizar um número configurável de transferências do tamanho de um bloco. Para solicitar uma transferência o controlador de DMA deve ser configurado conforme a seguir.

4.4.3.1 Configuração do DMA

Ao ser solicitado para realizar uma transferência o controlador de DMA precisa estar inativo, isto é, não pode haver nenhuma transferência em

andamento e também não pode estar sendo configurado. Desse modo o DMA pode ser utilizado em um cenário em que haja mais de um dispositivo que possa solicitar uma transferência.

O controlador de DMA precisa ser configurado com as seguintes informações:

- Endereço inicial na controladora de memória: endereço da controladora de memória onde começa o dado a ser transferido;
- Endereço inicial no controlador de E/S: endereço do controlador de E/S onde começa o dado a ser transferido;
- Sentido da transferência: se a transferência será da memória para a E/S ou da E/S para a memória;
- Tamanho da transferência: indica de quantos blocos é a transferência;

Após ser configurado, o controlador de DMA, entra em estado de ocupado, não podendo realizar novas transferências até que complete a atual.

Cabe reforçar que o endereço inicial da controladora de memória ignora a existência ou não do *interleaving*, a controladora de memória abstrai o endereçamento do controlador de DMA.

4.4.3.2 Memória para E/S

Se a transferência solicitada para o controlador de DMA for no sentido da controladora de memória para a controladora de E/S, a operação acontece como a seguir:

- O controlador de DMA solicita a leitura de um bloco para a controladora de memória e aguarda pela resposta da mesma;
- Após ser informado pela controladora de memória da conclusão da operação, o controlador de DMA inicia uma transferência com o controlador de E/S através do protocolo de handsha*king*, informando ao controlador de E/S o endereço de escrita do bloco no controlador. Um sinal é gerado para ativar a escrita no controlador, isso ocorre porque o dispositivo de E/S é simulado por uma memória RAM.
- Após ser concluída a transferência com o controlador de E/S, se o tamanho da transferência for maior do que um bloco, o controlador de

DMA gera o próximo endereço de acesso à controladora de memória, assim como o endereço de acesso ao controlador de E/S. Repetem-se os passos descritos anteriormente até que se atinja o tamanho total da transferência;

4.4.3.3 E/S para memória

No caso de uma transferência no sentido controlador de E/S para controladora de memória, a operação se dá como descrito abaixo:

- O controlador de DMA inicia o protocolo de *handshaking* com o controlador de E/S solicitando a leitura de um bloco.
- Após ser concluído o protocolo de *handshaking*, o controlador de DMA inicia uma transferência com a controladora de memória, solicitando a escrita de um bloco.
- Depois de concluída a transferência com a controladora de memória, se o tamanho da transferência for maior do que um bloco, o controlador de DMA gera o próximo endereço de acesso no controlador de E/S e na controladora de memória e realiza uma nova transferência, até que o tamanho total da mesma seja atingido.

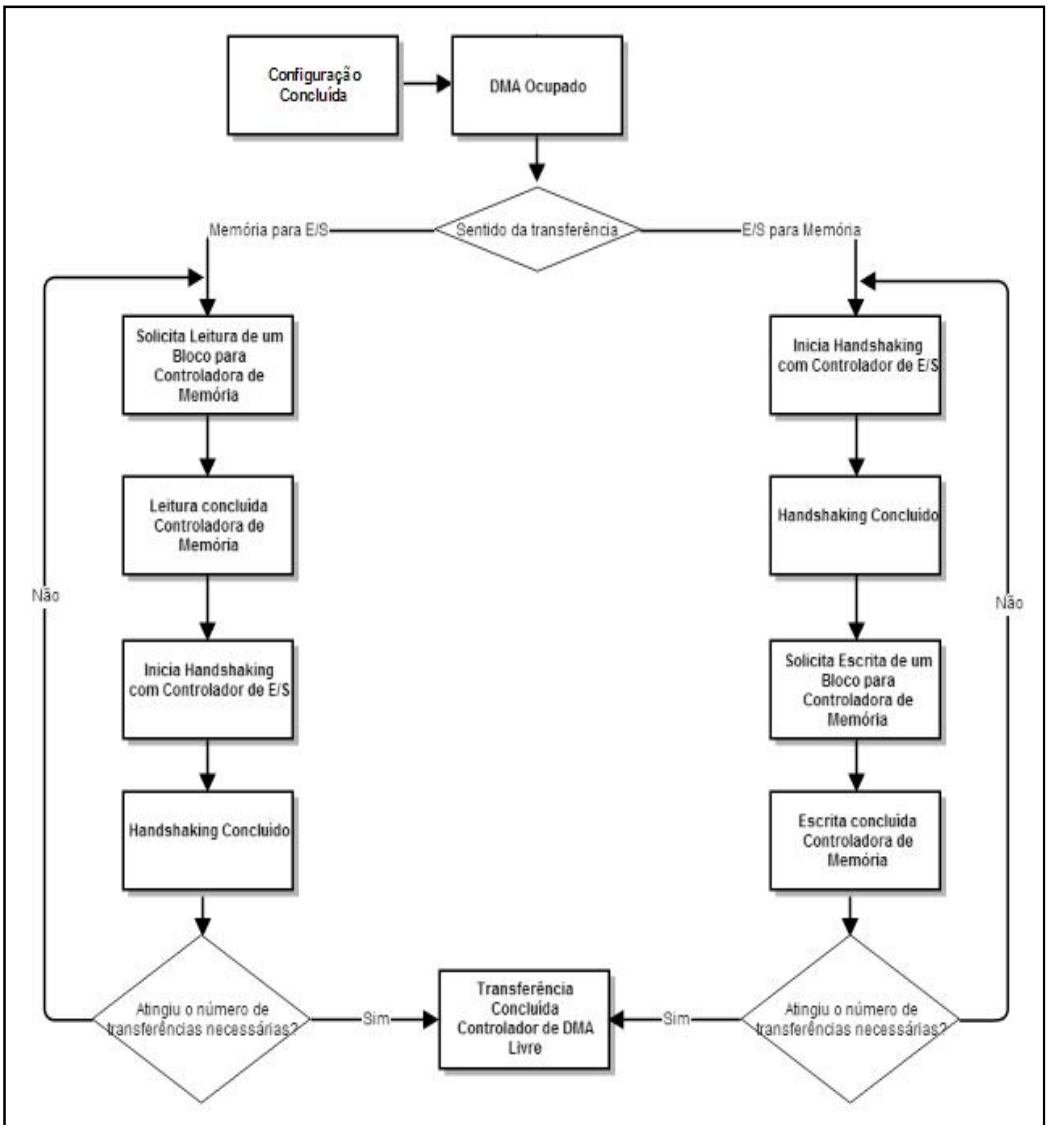
4.4.3.4 Conclusão da transferência

Ao completar toda a transferência solicitada ao controlador de DMA, o mesmo entra em estado de inativo e gera um sinal informando da conclusão da tarefa.

Uma transferência do controlador de DMA ocorre de maneira simplificada como mostrado na figura 12

.

Figura 12 - Transferência do controlador de DMA



Fonte: Próprio Autor.

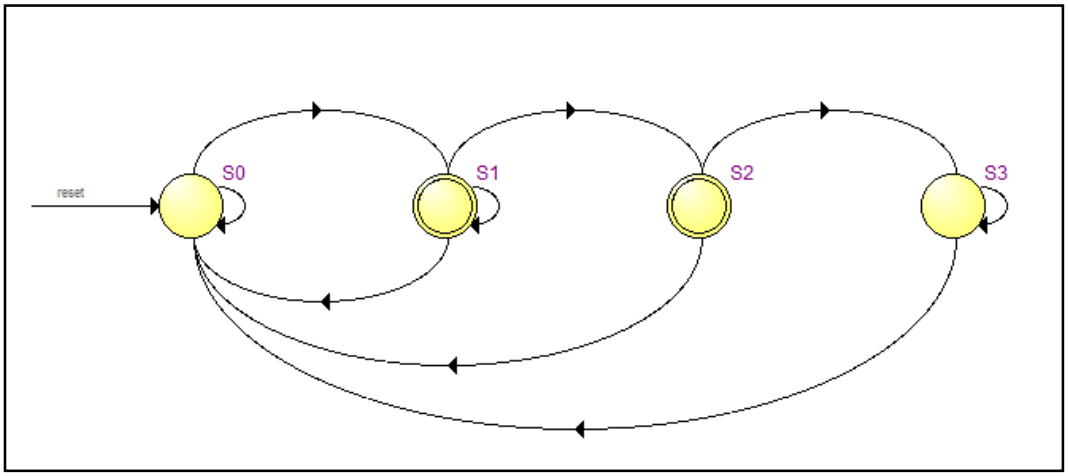
5 VALIDAÇÃO E RESULTADOS

Após o desenvolvimento dos componentes, a ferramenta Quartus II pode otimizar os mesmos, tais otimizações serão mostradas a seguir.

5.1 Unidade de controle - Controladora de memória

A máquina de estados da unidade de controle, responsável por comandar o dispositivo, foi otimizada como mostra a figura 13.

Figura 13 - Unidade de controle da controladora de memória otimizada.



Fonte: Próprio Autor.

A seguir, na tabela 7, as transições de estados da controladora de memória, também fruto de otimizações.

Tabela 7 - Transições de estados da controladora de memória

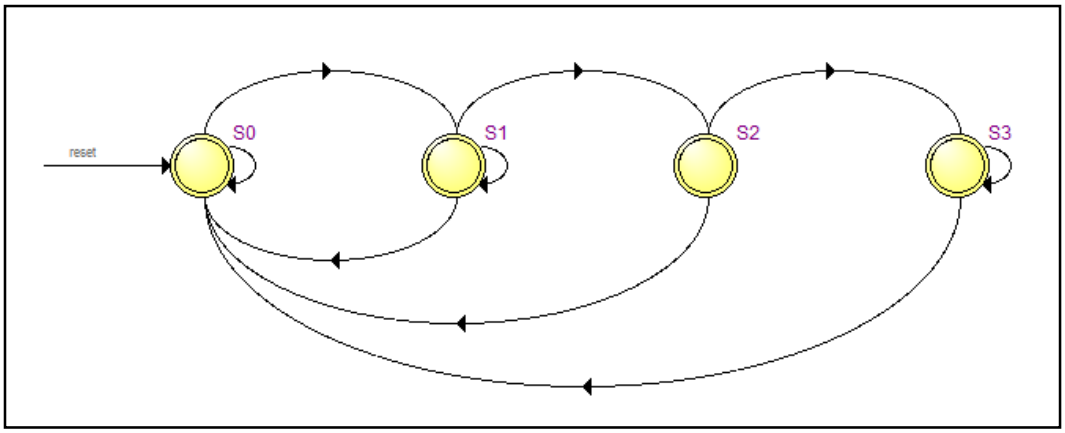
Estado Atual	Estado Futuro	Condição	Sinais Gerados
s0	s0	(!PedidoRWOut) + (PedidoRWOut).(reset)	MudaStatusConfig Controladora MudaPedidoRw
s0	s1	(PedidoRWOut).(!reset)	
s1	s0	(reset)	
s1	s1	(!StatusConfigControladora). (!reset)	
s1	s2	(StatusConfigControladora). (!reset)	
s2	s3	(!reset)	MudaStatusControladora
s2	s0	(reset)	
s3	s3	(StatusControladora).(!reset)	
s3	s0	(!StatusControladora) + (StatusControladora).(reset)	

Fonte: Próprio Autor.

5.2 Unidade de Controle - Controlador de DMA

Assim como a controladora de memória, a máquina de estados do controlador de DMA também foi otimizada pela ferramenta, como pode ser observado na figura 14.

Figura 14 - Unidade de controle do controlador de DMA otimizada.



Fonte: Próprio Autor.

A seguir, na tabela 8, as transições de estados, também fruto de otimizações.

Tabela 8 - Transições de estados do controlador de DMA

Estado Atual	Estado Futuro	Condição	Sinais Gerados
s0	s1	(PedidoTransferencia).(!reset)	MudaStatusConfigDMA MudaPedidoTransferencia
s0	s0	(!PedidoTransferencia) + (PedidoTransferencia).(reset)	
s1	s2	(process_1).(!reset)	
s1	s1	(!process_1).(!reset)	
s1	s0	(reset)	
s2	s3	(!reset)	MudaStatusConfigDMA MudaStatusDMA
s2	s0	(reset)	
s3	s3	(StatusDMA).(!reset)	
s3	s0	(!StatusDMA) + (StatusDMA).(reset)	

Fonte: Próprio Autor.

5.3 Simulações RTL da Controladora Memória

A seguir serão mostrados os resultados de simulações RTL feitas através do software Altera ModelSim. Como a memória lê um dado de qualquer tamanho da mesma maneira, serão usados cinco bits para o tamanho do dado para facilitar a visualização.

A controladora foi parametrizada da seguinte maneira:

- Tamanho do bloco: 30 bits;
- Número de bancos de memória: três;
- Tamanho do dado: cinco bits;
- Tamanho do endereço: oito bits

A partir desses dados a controladora foi parametrizada. Haverá um *interleaving* de três bancos de memória, sendo necessárias duas operações nos bancos para ler/escrever o bloco solicitado.

5.3.1 Leitura na controladora de memória

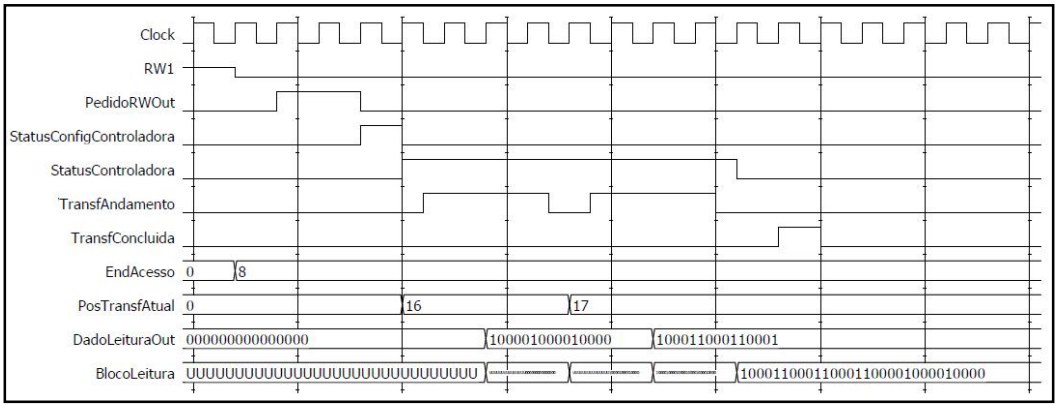
Uma leitura na controladora de memória ocorre, de maneira simplificada, como descrito a seguir:

- A controladora recebe um pedido de operação, sinal PedidoRW;

- É verificado o tipo de transferência, neste caso uma leitura, sinal RW1 em nível baixo;
- É lido o endereço de acesso, sinal EndAcesso;
- Como citado anteriormente o bloco de leitura é armazenado em dois endereços nos bancos de memória, então neste caso o acesso ao endereço oito, solicitado para leitura, deverá ser feito nos endereços dezesseis e dezessete, através do sinal PosTransfAtual.
- O dado lido nas posições citadas, sinal DadoLeituraOut será respectivamente armazenado na posição menos significativa e mais significativa do bloco de leitura, como mostrado no sinal BlocoLeitura.
- Após ler todo o bloco na controladora, a mesma, ativa o sinal de transferência concluída, como pode ser visto através do sinal TransfConcluida.

A figura 15 ilustra esta descrição.

Figura 15 - Leitura na controladora de memória



Fonte: Próprio Autor.

5.3.2 Escrita na controladora de memória

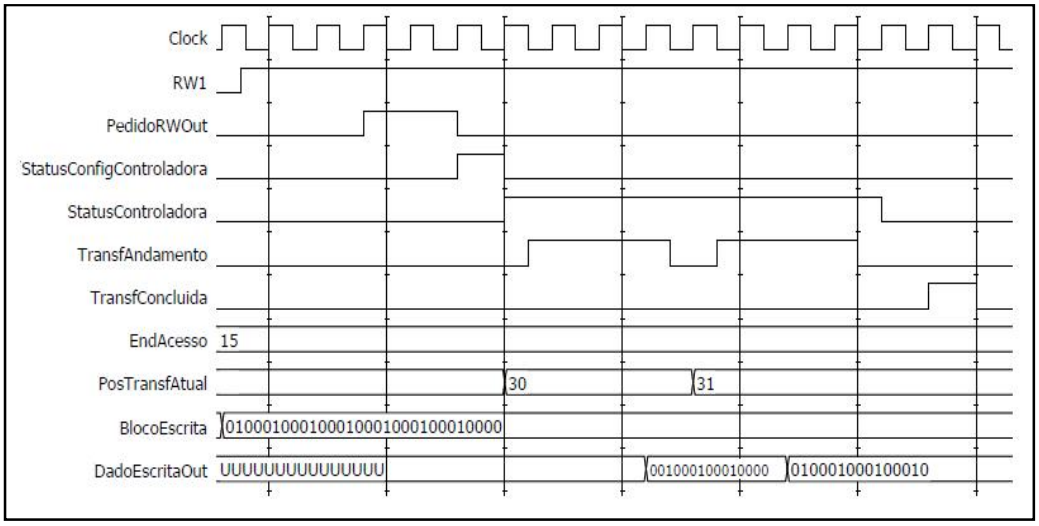
Uma escrita na controladora de memória ocorre, de maneira simplificada, como descrito a seguir:

- A controladora recebe um pedido de operação, sinal PedidoRW;
- É verificado o tipo de transferência, neste caso uma escrita, sinal RW1 em nível alto;

- Como no cenário anterior serão necessárias duas escritas nos bancos para completar a operação de escrita do bloco, sinal BlocoEscrita, no endereço quinze, sinal EndAcesso. A escrita nos bancos então será feita nos endereços 30 e 31, como pode ser visto no sinal PosTransfAtual.
- Analogamente a uma leitura, o bloco a ser escrito, terá sua parte menos significativa escrita na posição 30 e a parte mais significativa na posição 31, que pode ser verificado pelo sinal DadoEscritaOut.
- Após a escrita completa do bloco é gerado um sinal para avisar da conclusão da transferência, como pode ser visto no sinal TransfConcluida.

  A figura 16 ilustra o cenário descrito.

Figura 16 - Escrita na controladora de memória



Fonte: Próprio Autor.

5.4 Simulações RTL do controlador de DMA

A seguir serão mostrados os resultados de simulações funcionais, feitas no software Altera ModelSim. Assim como nos testes da controladora de memória, será usada a largura de dado de 5 bits.

O sistema DMA - E/S - memória, foi parametrizada da seguinte maneira:

- Tamanho do bloco: 30 bits;
- Número de bancos de memória: três;

- Tamanho do dado: cinco bits;
- Tamanho do endereço: oito bits

A partir desses dados o sistema foi parametrizado de modo que há um *interleaving* de três bancos de memória, sendo necessário duas operações nos bancos para ler/escrever o bloco solicitado.

5.4.1 Memória para E/S

O controlador de DMA é configurado para realizar a seguinte transferência:

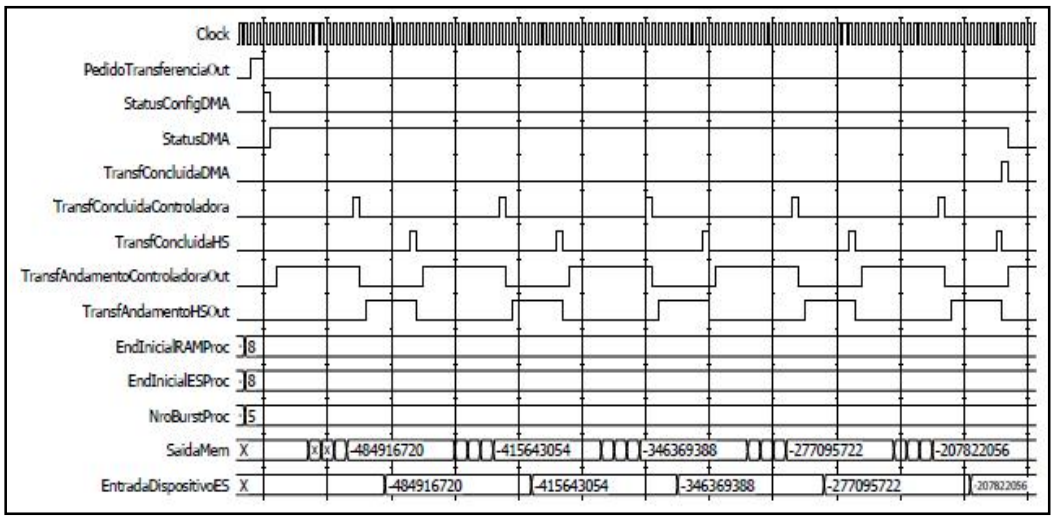
- Sentido da transferência: Memória para E/S;
- Endereço inicial na controladora de memória: oito;
- Endereço inicial n controlador de E/S: oito;
- Tamanho da transferência: cinco blocos;

  A transferência ocorre como descrito a seguir:
- O controlador de DMA recebe uma solicitação de transferência, sinal PedidoTransferenciaOut;
- Após ser configurado como descrito anteriormente, o controlador entra no estado de ocupado, sinal StatusDMA;
- O controlador de DMA solicita um bloco para a controladora de memória e, quando a mesma avisa o controlador de DMA de que já tem o dado solicitado, através do sinal TransfConcluidaControladora, o bloco está disponível para leitura, sinal SaidaMem;
- De posse do bloco lido na controladora de memória, o controlador de DMA inicia o protocolo de *handshaking* com o controlador de E/S, escrevendo o bloco na porta de entrada do controlador de E/S, sinal EntradaDispositivoES. Chegando ao final dos passos do protocolo de *handshaking*, é gerado um sinal que avisa ao controlador de DMA da conclusão do mesmo, sinal TransfConcluidaHS;
- Como a transferência total solicitada ao DMA é de cinco blocos, são feitas mais quatro transferências e então ao atingir o número total solicitado, o sinal StatusDMA muda para nível baixo, indicando que está livre para realizar novas transferências e é gerado um sinal avisando da

conclusão da transferência solicitada ao DMA, sinal TransfConcluidaDMA;

O resultado da simulação é mostrado na figura 17.

Figura 17 - Simulação de transferência da Memória para E/S



Fonte: Próprio Autor.

5.4.2 E/S para Memória

O controlador de DMA é configurado para realizar a seguinte transferência:

- Sentido da transferência: E/S para Memória;
- Endereço inicial na controladora de memória: quinze;
- Endereço inicial no controlador de E/S: dez;
- Tamanho da transferência: cinco blocos;

A transferência ocorre como descrito a seguir:

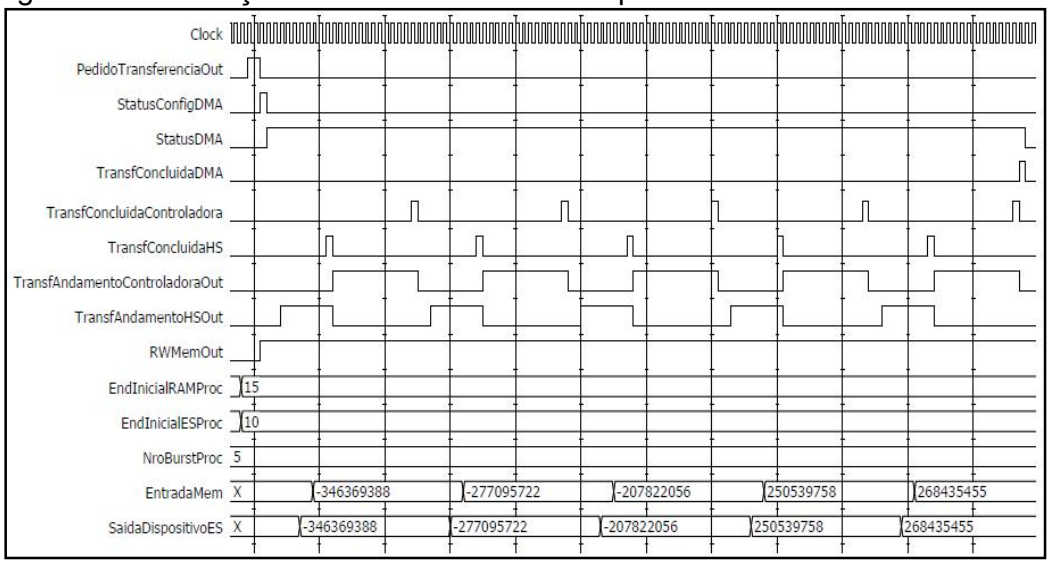
- O controlador de DMA recebe uma solicitação de transferência, sinal PedidoTransferenciaOut;
- Após ser configurado como descrito anteriormente, o controlador entra no estado de ocupado, sinal StatusDMA;
- É então iniciado o protocolo de *handshaking* com o controlador de E/S e após concluído o protocolo, sinal TranfConcluidaHS, o controlador de DMA tem o dado, sinal SaidaDispositivoES. Neste momento é solicitada

a escrita do bloco na controladora de memória, os dados são colocados na porta de entrada da controladora, sinal EntradaMem, e após a conclusão, a mesma avisa o DMA através do sinal TransfConcluidaControladora;

- Como a transferência total solicitada ao DMA é de cinco blocos, são feitas mais quatro transferências e então ao atingir o número total solicitado, o sinal StatusDMA muda para nível baixo, indicando que está livre para realizar novas transferências e é gerado um sinal avisando da conclusão da transferência solicitada ao DMA, sinal TransfConcluidaDMA;

  O resultado da simulação é mostrado na figura 18.

Figura 18 - Simulação de transferência da E/S para Memória



Fonte: Próprio Autor.

5.5 Simulações *Gate-Level*

Após o desenvolvimento dos dispositivos, foi feita a síntese em um FPGA alvo. Embora não tenha sido feita a síntese física no FPGA, foi feita uma simulação *Gate Level* (Nível de Portas), a qual é a transformação dos dispositivos desenvolvidos em linguagem VHDL para portas lógicas. Estas são o produto final da síntese, as quais serão programadas no FPGA.

O FPGA escolhido como alvo foi Cyclone IV GX, modelo EP4CGX150CF23C7 (ALTERA, 2014a), esse modelo foi escolhido por ter uma capacidade maior de memória do que alguns outros disponíveis e também por poder ser usado sem a necessidade de uma licença comercial da ferramenta Quartus II.

A frequência de *clock* usada para as simulações foi de 100 MHz, embora em algumas simulações pudesse ser aumentado para 110 MHz, outras necessitavam de uma frequência de *clock* menor, em torno de 100 MHz então, por motivos práticos, manteve-se a frequência em 100 MHz.

Para avaliar os dispositivos desenvolvidos foram testadas transferências de pacotes do tamanho de 50 Kbits. Nesse momento foram encontradas limitações da versão gratuita da ferramenta ModelSim, o que tornou o tempo de algumas simulações muito grande, por isso a escolha de pacotes de 50 Kbits.

A escolha dos cenários buscou abranger o maior número possível de configurações do sistema. Foram simulados cenários com um banco de memória até um número de 32 bancos e um tamanho de bloco de 32 bits até 2048 bits.

Algumas configurações criaram no sistema um *interleaving* de dados, aumentando o tamanho do bloco que seria transferido pelo controlador de DMA. Em algumas configurações também, a controladora de memória precisava escrever o bloco transferido em mais de uma linha dos bancos de memória.

A seguir, na tabela 9 os cenários simulados com os resultados de vazão, total de elementos lógicos consumidos no FPGA e potência consumida pelo sistema completo, composto por controladora de memória, controlador de DMA e controlador de E/S.

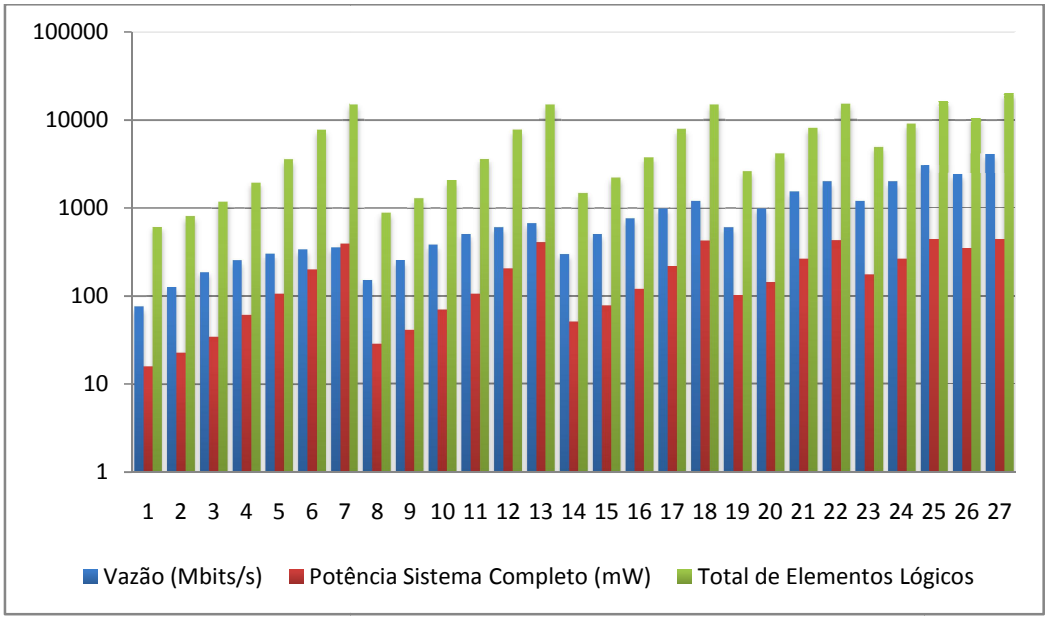
Tabela 9 - Cenários simulados e resultados obtidos

Cenário	Tamanho de Bloco	Número de Bancos memória	Vazão (Mbits/s)	Potência Consumida (mW)	Total Elementos Lógicos
1	32	1	76,3	15,9	611
2	64	1	127,1	23,05	809
3	128	1	190,7	34,72	1183
4	256	1	254,2	61,98	1957
5	512	1	305,0	106,4	3541
6	1024	1	338,9	204,24	7763
7	2048	1	358,8	402,78	15139
8	64	2	152,5	28,71	896
9	128	2	254,2	42,33	1286
10	256	2	381,2	70,16	2066
11	512	2	508,2	107,59	3587
12	1024	2	609,7	205,5	7783
13	2048	2	677,4	413,98	15169
14	128	4	305,0	51,64	1471
15	256	4	508,2	78,36	2242
16	512	4	762,0	120,8	3776
17	1024	4	1015,6	218,76	7885
18	2048	4	1218,3	426,99	15190
19	256	8	609,7	102,69	2625
20	512	8	1015,6	144,71	4160
21	1024	8	1522,1	265,26	8297
22	2048	8	2027,7	431,52	15431
23	512	16	1218,3	177,34	4926
24	1024	16	2027,7	265,24	9024
25	2048	16	3036,6	437,83	16197
26	1024	32	2431,7	345,55	10557
27	2048	32	4042,1	440,26	19784

Fonte: Próprio Autor

No gráfico 1, a seguir, a comparação dos cenários simulados da tabela 9:

Gráfico 1 - Cenários simulados em escala logarítmica.



Fonte: Próprio Autor

Analisando o gráfico 1, nota-se um aumento na vazão, potência consumida e total de elementos lógicos a medida que se aumenta o tamanho do bloco, fixando-se o número de bancos de memória. Porém aumentando-se o número de bancos de memória pode-se obter uma vazão maior com um consumo de potência menor e menos elementos lógicos consumidos pelo FPGA, mesmo com um tamanho de bloco menor. Por exemplo, se compararmos o cenário 24, em que se têm uma vazão de 2027,7 Mbits/s e um consumo de 265,24 mW para um tamanho de bloco de 1024 bits e 16 bancos de memória, com o cenário 18 em que temos uma vazão de 1218,3 Mbits/s e um consumo de 426,99 mW para um tamanho de bloco de 2048 bits, nota-se que embora aumentando o tamanho de bloco a vazão é menor e o consumo de potência é maior. O mesmo se aplica ao total de elementos lógicos.

Os resultados foram obtidos através da realização das transferências, citadas anteriormente, de pacotes de 50 Kbits.

A seguir, na tabela 10, os resultados de potência consumida pelos dispositivos individualmente.

Tabela 10 - Potência consumida pelos dispositivos

Cenário	Sistema Completo (mW)	Controladora de Memória (mW)	Controlador de E/S (mW)	Controlador de DMA (mW)
1	15,9	4,51	3,91	0,38
2	23,05	5,1	8,56	0,47
3	34,72	5,52	17,05	0,52
4	61,98	6,84	33,97	0,79
5	106,4	8,54	68,31	1,29
6	204,24	15	145,34	6,92
7	402,78	24,36	294,27	14,1
8	28,71	9,24	8,56	0,55
9	42,33	9,53	17,12	0,59
10	70,16	11,22	34,05	0,86
11	107,59	13,74	67,91	1,14
12	205,5	21,29	140,15	4,93
13	413,98	33,09	302,12	20,76
14	51,64	18,38	17,13	0,61
15	78,36	18,96	34,18	0,91
16	120,8	22,67	68,22	1,47
17	218,76	27,82	140,52	5,13
18	426,99	43,76	296,25	17,23
19	102,69	36,63	34,11	0,79
20	144,71	39,11	69,03	1,91
21	265,26	55,14	151,04	9,13
22	431,52	61,48	292,32	15,46
23	177,34	72,34	68,6	3,19
24	265,24	74,75	140,9	5,32
25	437,83	87,41	274,47	9,53
26	345,55	143,14	142,52	5,87
27	440,26	192,11	158,9	18,24

Fonte: Próprio Autor

Observando a tabela 10, nota-se que o controlador de DMA é o componente responsável pelo menor consumo de potência. Isso ocorre porque diferentemente da controladora de memória e do controlador de E/S, o controlador de DMA não sintetiza nenhum banco de memória, somente alguns registradores necessários para sua lógica de funcionamento.

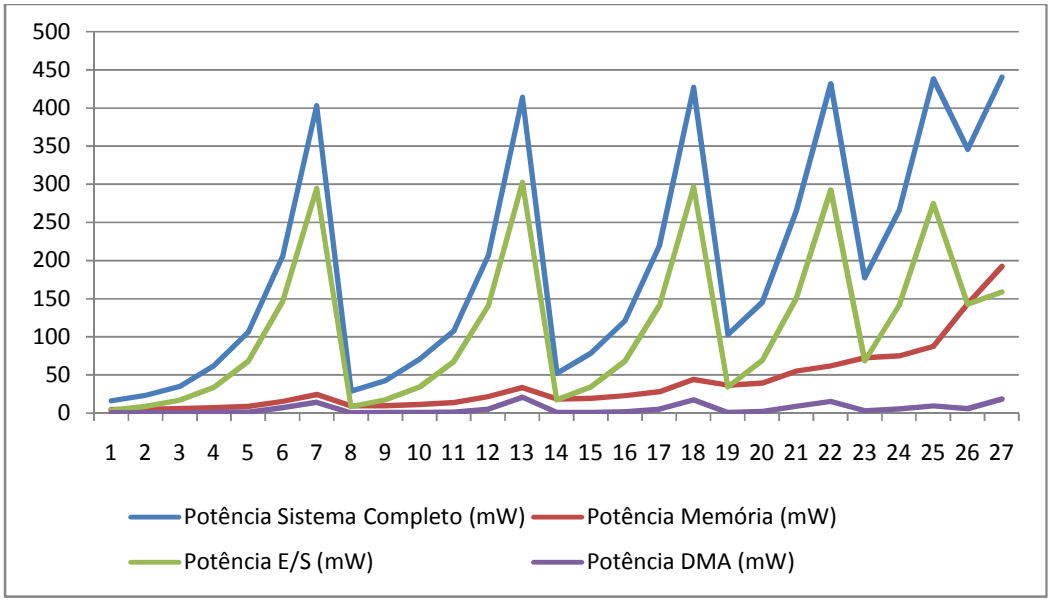
O controlador de E/S, é na maioria dos cenários o maior consumidor de potência, isso ocorre porque o mesmo é emulado por uma memória RAM e essa memória deverá ter a mesma largura do tamanho do bloco. Esse impacto

no consumo de potência pode ser notado observando-se que à medida que cresce o tamanho de bloco o consumo aumenta.

A potência consumida pela controladora de memória também cresce à medida que o tamanho de bloco aumenta, mas não pelo motivo descrito no caso do controlador de E/S. No caso da controladora de memória, parte do consumo é feito por registradores que devem ter a largura do tamanho de bloco, além de registradores necessários ao seu funcionamento lógico. Porém o maior responsável pelo consumo no componente é o número de bancos de memória, esse fator pode ser melhor observado no gráfico 2.

No gráfico 2, a seguir, a comparação dos consumos de potência da tabela 10:

Gráfico 2 - Comparação do consumo de potência dos dispositivos



Fonte: Próprio Autor

A seguir na tabela 11, os dados de consumo do FPGA:

Tabela 11 - Dados de consumo do FPGA

Cenário	Bits de Memória	Elementos Lógicos	Funções Combi-nacionais	Registra-dores Lógicos Dedica-dos	Registra-dores
1	131.072	611	264	502	502
2	196.608	809	296	694	694
3	327.680	1.183	331	1.078	1.078
4	589.824	1.957	431	1.846	1.846
5	1.114.112	3.541	606	3.382	3.382
6	2.162.688	7.763	5.084	7.478	7.478
7	4.259.840	15.139	9.895	14.646	14.646
8	262.144	896	295	790	790
9	393.216	1.286	361	1.174	1.174
10	655.360	2.066	427	1.942	1.942
11	1.179.648	3.587	625	3.478	3.478
12	2.228.224	7.783	5.055	7.574	7.574
13	4.325.376	15.169	9.888	14.742	14.742
14	524.288	1.471	359	1.366	1.366
15	786.432	2.242	489	2.134	2.134
16	1.310.720	3.776	619	3.670	3.670
17	2.359.296	7.885	5.103	7.766	7.766
18	4.456.448	15.190	9.854	14.934	14.934
19	1.048.576	2.625	487	2.518	2.518
20	1.572.864	4.160	1.257	4.054	4.054
21	2.621.440	8.297	5.098	8.150	8.150
22	4.718.592	15.431	9.967	15.318	15.318
23	2.097.152	4.926	1.254	4.822	4.822
24	3.145.728	9.024	6.377	8.918	8.918
25	5.242.880	16.197	10.475	16.086	16.086
26	4.194.304	10.557	6.373	10.454	10.454
27	6.291.456	19.784	14.569	17.623	17.623

Fonte: Próprio Autor

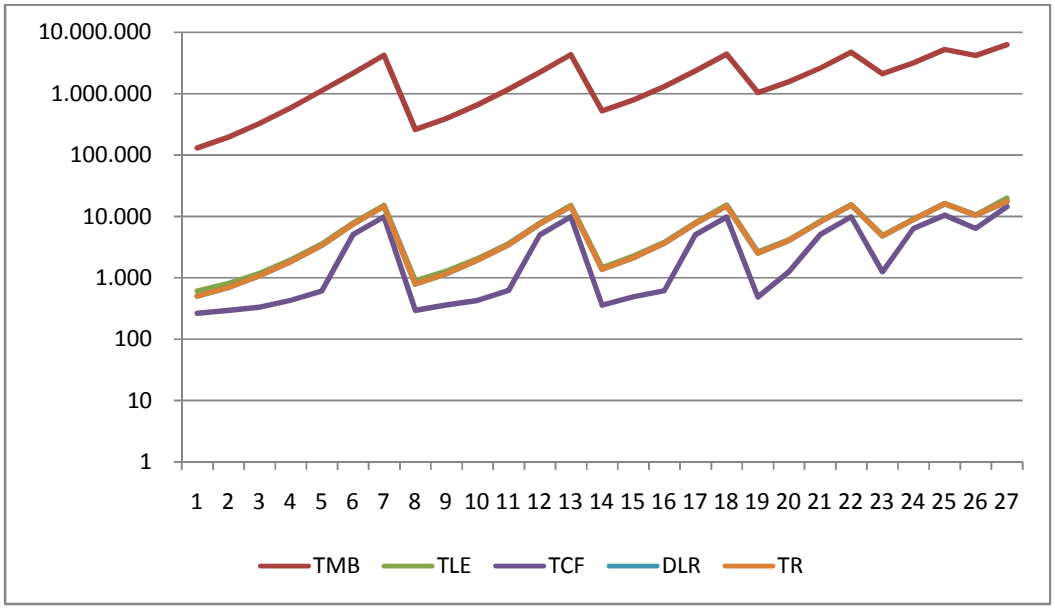
Analisando o total de bits de memória, os mesmos tem relação direta com o tamanho do bloco e o número de bancos de memória. O primeiro é responsável por gerar a largura da memória do controlador de E/S, o qual deve ter a mesma largura do tamanho do bloco. O segundo é influenciado pelo número de bancos de memória e pelo tamanho do dado de cada banco.

O total de elementos lógicos, registradores lógicos dedicados e total de registradores têm estreita relação com o total de bits de memória, conforme pode ser analisado no gráfico 3.

Infere-se que o total de funções combinacionais, além de poder ser explicado também pelo consumo de bits de memória, pode ter uma relação com o controlador de DMA.

A seguir, o gráfico 3, que ilustra o consumo do FPGA pelos dispositivos:

Gráfico 3 - Comparação do consumo do FPGA pelos dispositivos, em escala logarítmica.



Fonte: Próprio Autor

6 CONCLUSÕES E TRABALHOS FUTUROS

Os objetivos do trabalho foram atingidos, ao final deste trabalho obteve-se um sistema composto por um controlador de dispositivo de E/S, uma controladora de memória com alta vazão de dados e um controlador de DMA que realiza transferências entre os dois dispositivos citados.

Para tal a interface que conecta os módulos do sistema composto pela controladora de memória, controlador de DMA e controlador de E/S, pode ser parametrizado, podendo ser ajustado para a configuração onde tenha o melhor desempenho. Esse desempenho pode ser ajustado visando um ou mais dos três possíveis objetivos:

- Maior/menor vazão de dados;
- Maior/menor potência consumida pelo sistema;
- Maior/menor consumo de recursos do FPGA;

A controladora de memória desenvolvida possibilita o acesso de alta vazão, realizando o *interleaving* do bloco de leitura/escrita nos bancos. Foi constatado um *overhead* na controladora, necessário para o *interleaving* dos dados.

O dispositivo de E/S comporta-se de maneira adequada, embora simulado por uma memória RAM.

A maior taxa de transferência obtida no sistema foi de 4042,1 Mbits/s, com a seguinte configuração:

- Tamanho de bloco: 2048 bits;
- Número de bancos de memória: 32;

Resultando em um consumo de:

- Potência: 440,26 mW;
- Total de bits de memória: 6291456;
- Total de elementos lógicos: 19784;
- Total de funções combinacionais: 14569;
- Total de registradores: 17623;

Esse cenário também foi o que obteve o maior consumo de potência, bits de memória, elementos lógicos, funções combinacionais e registradores.

A menor taxa de transferência obtida no sistema foi de 76,3 Mbits/s, com a seguinte configuração:

- Tamanho de bloco: 32 bits;
- Número de bancos de memória: um;

Resultando em um consumo de:

- Potência: 15,9 mW;
- Total de bits de memória: 131072;
- Total de elementos lógicos: 611;
- Total de funções combinacionais: 264;
- Total de registradores: 502;

Esse cenário também foi o que obteve o menor consumo de potência, bits de memória, elementos lógicos, funções combinacionais e registradores.

Surgem como inspirações para trabalhos futuros algumas das ideias apresentadas na tabela 1, seção 3.7:

- Implementação de filas de transmissão como possibilidade de compatibilizar dispositivos com largura de interfaces diferentes;
- Implementação de *buffers* circulares para que o controlador de DMA possa armazenar um certo número de solicitações de transferências, diminuindo assim o número de ciclos ociosos entre transferências e necessitando menor atenção do processador;
- Implementando-se os *buffers* circulares, pode-se especular introduzir prioridades nas transferências.

Como aperfeiçoamento dos componentes desenvolvidos, é possível tentar reduzir o *overhead* na controladora de memória, melhorando o tempo para leitura/escrita dos dados e também o *overhead* no controlador de DMA, melhorando o tempo de transferência dos blocos;

REFERÊNCIAS

ALTERA. **Quartus II Handbook:** Verification, v.3, 2004. Disponível em: <http://users.ece.gatech.edu/~hamblen/UP3/qts_qii5v3.pdf>. Acesso em: 07 jan. 2015.

_____. **Quartus II Handbook Version 9.1:** Embedded Peripherals. San Jose v.5. 2010 Disponível em: <http://users.ece.gatech.edu/~hamblen/UP3/qts_qii5v3.pdf>. Acesso em: 07 jan. 2015.

_____. **About the MegaWizard Plug-In Manager.** 2012a. Disponível em: <http://quartushelp.altera.com/12.1/mergedProjects/hdl/mega/mega_view_megawiz.htm>. Acesso em: 18 set. 2014.

_____. **Memory Initialization File (.mif).** 2012b. Disponível em: <http://quartushelp.altera.com/12.0/mergedProjects/reference/glossary/def_mif.htm>. Acesso em: 17 set. 2014.

_____. **Cyclone IV FPGA Family:** Lowest Cost, Lowest Power, Integrated Transceivers. 2014a. Disponível em: <http://www.altera.com/literature/ug/ug_ram_rom.pdf>. Acesso em: 07 jan. 2015.

_____. **Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT):** User Guide. San Jose. 2014b. Disponível em: <http://www.altera.com/literature/ug/ug_ram_rom.pdf>. Acesso em: 07 jan. 2015.

_____. **Quartus II Handbook Volume 1:** Design and Synthesis. San Jose v.1. 2014c. Disponível em: <http://www.altera.com/literature/hb/qts/qts_qii51013.pdf>. Acesso em: 07 jan. 2015.

_____. **FPGAs.** 2015a. Disponível em: <http://www.altera.com/products/fpga.html>. Acesso em: 07 jan. 2015.

_____. **ModelSim-Altera Software**. 2015b. Disponível em: <http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html>. Acesso em: 07 jan. 2015.

_____. **Quartus II Web Edition Software.** 2015c. Disponível em: <http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html>. Acesso em: 07 jan. 2015.

AMOS, R. Omondi.. **The Microarchitecture of Pipelined and Superscalar Computers.** 1. ed. Norwell: Kluwer Academic Publisher Boston, 1999. 271 p.

ARM. **AMBA™ Specification.** 1999. Disponível em: < http://www-micro.deis.unibo.it/~magagni/amba99.pdf>. Acesso em: 07 jan. 2015.

CORREA, Ilan et al. A Direct Memory Access Controller (DMAC) IP-Core using the AMBA AXI protocol. **26th South Symposium on Microelectronics**, Novo Hamburgo, p. 167-171, abril. 2011. Disponível em: <http://www.inf.ufrgs.br/sim-emicro/images/Anais_SIM2011.pdf>. Acesso em: 04 jan. 2015.

CHU, Pong P. **Rtl Hardware Design using VHDL:** Coding for Efficiency, Portability and Scalability. 1. ed. New Jersey: John Wiley & Sons, 2006. 669 p.

HAMBLEN, James O.; HALL, Tyson S.; FURMAN, Michael D.. **Rapid Prototyping of Digital Systems:** SOPC Edition. 2. ed. New York: Springer Science + Business Media, 2008. 428 p.

INTEL. **8237A High Performance Programmable DMA Controller (8237A-5).** 1993. Disponível em: <http://read.seas.harvard.edu/cs261/hwref/8237A.pdf>. Acesso em: 04 jan. 2015.

MAZUMDER, Pinaki; CHAKRABORTY, Kanad. **The Testing and Testable Design of High-Density Random-Access Memories.** 1. ed. Norwell: Kluwer Academic Publisher Massachusetts, 1996. 386 p.

OPENSUSE. **Software Livre e de Código Aberto.** 2011. Disponível em: <http://pt.opensuse.org/Software_Livre_e_de_C%C3%B3digo_Aberto#O_qu.C3.AA_.C3.A9_ C.C3.B3digo_Aberto.3F>. Acesso em: 19 set. 2014.

PATTERSON, David A.; HENNESSY, John L.. **Organização e projeto de computadores:** A interface hardware/software. 3. ed. Rio de Janeiro: Elsevier, 2005. 484 p.

SHARMA, Chetan. General Purpose AHB-DMA Controller. **Int. J. Comp. Tech. Appl.**, Índia, v. 2, p. 248-252, mar./abr. 2011. Disponível em: <http://www.ijcta.com/documents/volumes/vol2issue2/ijcta2011020203.pdf>. Acesso em: 04 jan. 2015.

STALLINGS, William . **Arquitetura e organização de computadores**. 5. ed. São Paulo: Prentice Hall, 2003. 786 p.

TUMEO, Antonino et al. Lightweight DMA Management Mechanisms for Multiprocessors on FPGA. **Application-Specific Systems, Architectures and Processors,** Leuven, p. 275-280, jul. 2008. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4580191>. Acesso em: 04 jan. 2015.

UNIVERSIDADE FEDERAL DO PARÁ. **Software Proprietário**. 2010. Disponível em: <http://www.ufpa.br/dicas/progra/proaqui.htm>. Acesso em: 19 set. 2014.

WADEKAR, Ameya; SWAPNIL, Sapre; LOHANI**,** Rajesh. Design and Implementation of a Universal DMA Controller. **International Conference and Workshop on Emerging Trends in Technology***,* Mumbai, p. 1189-1190, fev. 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=1980281&dl=ACM&coll=DL&CFID=362910279&CFTOKEN=33277679>. Acesso em: 18 de set. 2014.

WEI, Tu et al. Proxy Caching for Video-on-Demand Using Flexible Starting Point Selection. **IEEE Transactions on multimedia**, Buffalo, v. 11, n. 4, p. 716-729, jun. 2009. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=04895726>. Acesso em: 19 set. 2014.

WESTE, Neil H.E.; HARRIS, David M. **CMOS VLSI Design:** A Circuits and Systems Perspective. 4. ed. Boston: Addison-Wesley, 2010. 864 p.

XILINX. **LogiCORE IP AXI DMA v7.1:** Product Guide**.** 2014. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf>. Acesso em: 19 set. 2014.

APÊNDICE A - Ferramentas de Projeto

Altera Quartus II

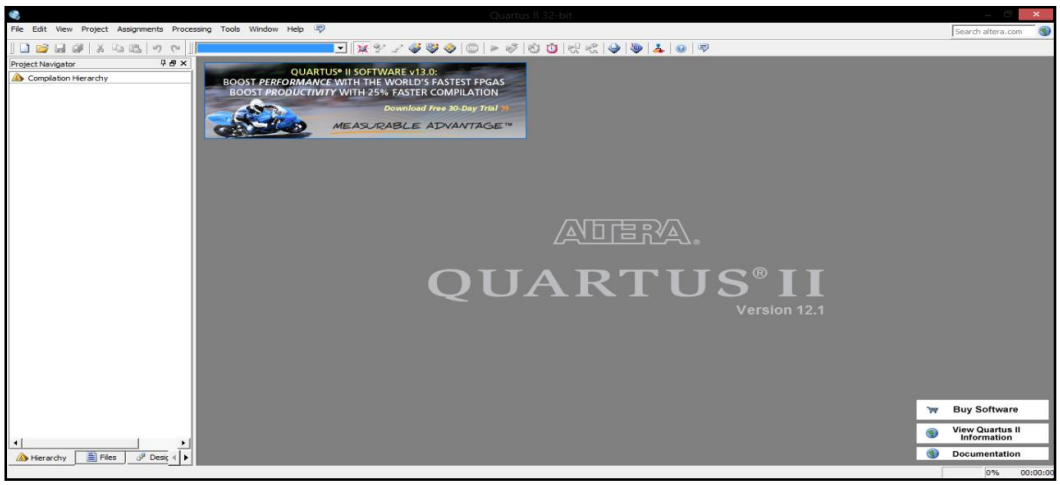
O projeto de circuitos digitais pode ser automatizado através do uso de ferramentas, uma delas é o Altera Quartus II (ALTERA, 2015c), utilizada na descrição, compilação e programação de sistemas

Algumas funcionalidades importantes utilizadas durante o trabalho foram:

- Editor de texto apto a receber linguagem de programação como VHDL.
- Compilador em linguagem VHDL.

A figura 19 mostra a tela inicial do software.

Figura 19 - Tela inicial do software Altera Quartus II



Fonte: Próprio Autor.

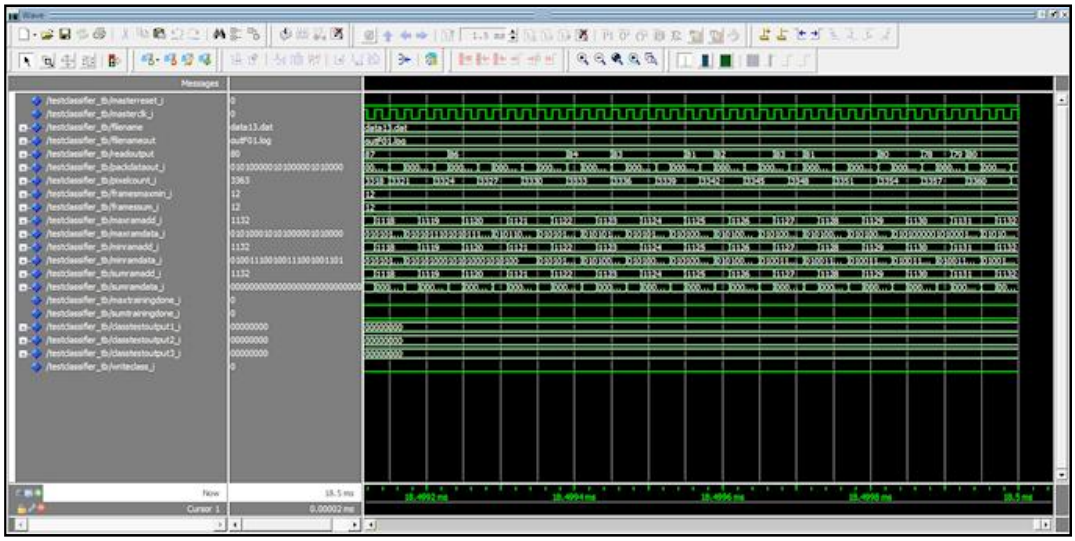
MegaWizard Pug-In Manager

Permite a criação e modificação de componentes completos em um alto nível de abstração. Fornece acesso a um banco de dados de componentes já prontos facilitando e diminuindo o tempo de projeto (ALTERA, 2012a).

Altera ModelSim

Para simulação de projetos VHDL, pode ser usado o software Altera ModelSim (ALTERA, 2015b), que é instalado juntamente com o Altera Quartus II. Este software proporciona simulações em forma de onda, como pode ser visto na figura 20.

Figura 20 - Simulação em forma de onda, *software* Altera ModelSim



Fonte: Próprio Autor.