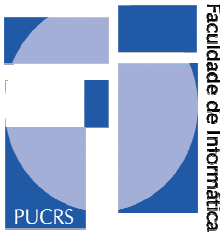
Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Curso de Bacharelado em Ciência da Computação



Implementação do GNU Chess em uma Plataforma Multiprocessada com Arquitetura de Comunicação baseada em Rede Intrachip

Trabalho de Conclusão II

Autores:

Celso Marasca Soccol

Giovani Zucolotto

Orientador:

Ney Laert Vilar Calazans

Porto Alegre, Novembro de 2006.

Resumo

Do ponto de vista de projetos de hardware para jogos, observa-se uma tendência à utilização de máquinas com diversos processadores concorrentes, organizados em *processadores multi-núcleos* (em inglês, *multi-core processor*s), tal como o processador Xeon da Intel ou o processador Cell utilizado no Playstation 3. Tendo em vista esta tendência e o crescente mercado de jogos, é interessante que o desenvolvimento de software comece a se adaptar a estes novos padrões de arquitetura. Este trabalho é uma investigação e porte do jogo GNU Chess para uma plataforma multiprocessada usando um meio de comunicação recentemente proposto, as redes intrachip.

Palavras Chave: multi-processamento, jogos, redes intrachip (NoCs), GNU Chess.

Abstract

From the point of view of game hardware design, it is possible to observe a trend towards the use of machines containing several concurrent processors. These are organized in multi-core processors as the Intel Xeon processor or the Cell processor, used in the Playstation 3 game console. In view of this trend and of the increasing success of game markets, it's interesting that the development of software starts to adapt to these new architecture standards. This work is an investigation and port of the GNU Chess game to a multiprocessing platform using a recently proposed new type of communication midia, intrachip networks.

Key words: multiprocessed, games, intrachip network (NoCs), GNU Chess.

Índice Analítico

1 Introdução 1
1.1 Motivação 1
1.1.1 Múltiplos Processadores Intrachip 2
1.1.2 Evolução de Console de Jogos 2
1.1.3 Evolução de Suporte a Jogos em Computadores Pessoais 3
1.1.4 Conclusão da Motivação 3
1.2 Objetivos 4
1.3 Estrutura do Trabalho 5
2 GNU Chess 6
2.1 Descrição do GNU Chess 6
2.1.1 Estrutura de arquivos da distribuição GNU Chess 6
2.1.2 Representação do tabuleiro 8
2.1.3 Estruturas de dados relevantes 12
2.1.4 Gerador de movimentos 14
2.1.5 Busca por jogadas 14
2.1.6 Avaliação de jogadas 14
2.1.7 Livro de jogadas 15
2.1.8 Tabela *Hash* 15
2.2 Heurísticas no jogo de xadrez 15
2.3 Implementação das Heurísticas 17
2.3.1 Minimax 17
2.3.2 Poda Simples 19
2.3.3 Alpha-beta poda 19
3 Revisões e Classificações de jogos 21
3.3.1 Jogos computacionais 22
3.3.2 A Relação Jogador-Jogo em Jogos Computacionais 26
3.3.3 A Estrutura Interna de um Jogo 27
3.3.4 A Engine de um Jogo Computacional 28
3.3.5 A Equipe de Implementação de Jogos 29
3.3.6 Estudos de Caso 30
4 Arquitetura Alvo 33
4.1 Arquitetura de Hardware 33
4.2 Noc Hermes 34
4.2.1 Roteador 36
4.2.2 Interface Externa 37
4.3 Processador Plasma 38
4.3.1 Histórico 38
4.3.2 Arquitetura MIPS 39
4.3.3 Organização do Plasma 41
4.3.4 Modificações na arquitetura original do Plasma 42
4.3.5 Compilador MIPS-GCC 43
4.4 Módulo de vídeo 43
4.4.1 Interface VGA 46
4.4.2 Controlador de Acesso à Memória DDR 47
4.4.3 Leitura e escrita no módulo de vídeo 50
4.5 Módulo de Teclado 51
4.6 Organização da infra-estrutura 53
5 Porte do GNU Chess para a Arquitetura Alvo 54

5.1 Modificações realizadas no código do GNU Chess 54
5.1.1 Porte de software 54
5.1.2 Porte de hardware 55
6 Desenvolvimento de Hardware e Software 57
6.1 Wrappers 57
6.1.1 Protocolo padrão dos wrappers 57
6.1.2 Plasma - Hermes 59
6.1.3 Teclado – Hermes 59
6.1.4 Módulo de vídeo – Hermes 61
6.2 Software para Comunicação 63
6.2.1 Plasma – Teclado 63
6.2.2 Plasma – Módulo de Vídeo 64
6.2.3 Plasma – Plasma 65
7 Validação 68
7.1 Validação dos Módulos 68
7.1.1 Validação do Módulo de Vídeo e wrapper 68
7.1.2 Validação do módulo de teclado e wrapper 69
7.2 Validação do Sistema 70
8 Conclusões e Trabalhos Futuros 71
9 Referências bibliográficas 73

Lista de figuras

Figura 1 – Um *chipset* KT400 (North Bridge) conectado diretamente ao processador (AMD Athlon Processor), à memória, à porta AGP e ao chipset VT0235 (South Bridge).... 4
Figura 2 – Posições em um tabuleiro de xadrez denominadas usando-se uma combinação de letras e números. ... 9
Figura 3 - Representação de *bitboard* de peões. As posições com o valor "1" são as que contém peões. As posições vazias contem o valor "zero".... 10
Figura 4 - Representação de *bitboard* de peões depois de sofrer um deslocamento de oito bits. As posições vazias contem o valor "zero".... 10
Figura 5 – Representação do tabuleiro de um exemplo de uma partida de xadrez . As posições com o valor "um" representam posições ocupadas por uma peça qualquer. As posições vazias contem o valor "zero".... 11
Figura 6 – Negação do tabuleiro da Figura 5. Todas as posições antes livres agora estão marcadas com o valor "um". As posições vazias contem o valor "zero".... 11
Figura 7 – Operação "e" bit a bit aplicado entre Figura 4 e Figura 6 resultando em todas as posições possíveis para os peões representados na Figura 3. As posições vazias contem o valor "zero".... 12
Figura 8 – Descrição das estruturas de dados (a) `Board`, (b) `HashSlot` e (c) `PawnSlot`.... 13
Figura 9 - algoritmo do Minimax sendo aplicado ao Jogo da Velha. O computador (que tem seu turno representado pela cor azul) joga com o xis e o jogador humano (que tem seu turno representado pela cor vermelha) joga com a bola. As setas de volta representam a jogada que foi selecionada em cada ponto do jogo [KAM06].... 18
Figura 10 – Poda Simples: já que um nodo vencedor foi encontrado, não é mais necessário continuar procurando. O computador (que tem seu turno representado pela cor azul) joga com o xis e o jogador humano (que tem seu turno representado pela cor vermelha) joga com a bola. As setas de volta representam a jogada que foi selecionada em cada ponto do jogo [KAM06].... 19
Figura 11 – Alpha-beta poda sendo aplicado a um jogo arbitrário. O computador tem seu turno representado pela cor azul e o jogador humano seu turno representado pela cor vermelha. A seta de volta representa a jogada que foi selecionada. Pode-se perceber que quando a jogada do jogador humano superou em pontuação a jogada do computador ($\alpha \geq \beta$), a busca parou [KAM06].... 20
Figura 12 – Exemplo de tela do jogo *Monkey Island*.... 23
Figura 13 – Exemplo de tela do jogo *World of Warcraft*.... 24
Figura 14 – Exemplo de tela do jogo de simulação de construção de cidades *Sim City*.... 25
Figura 15 – Tela ilustrativa do *Jogo da Vida*.... 25
Figura 16 – Exemplo de tela do jogo de Poker online *Texas Hold`Em*.... 26
Figura 17 – Exemplo de cena do jogo *Diablo*.... 31
Figura 18 – Exemplo de tela do jogo *Chess Master 10th*.... 32
Figura 19 – Diagrama da plataforma inicialmente proposta.... 33
Figura 20 - Exemplo de estrutura de um sistema com comunicação baseada em NoCs.... 35
Figura 21 – Exemplo de algumas topologias para uso em NoCs. - (a) Malha 2D 3x3; (b) Toro 2D 3x3; (c) Hipercubo 3D.... 35
Figura 22 – Chave de roteamento da NoC [MEL03].... 36
Figura 23 – Sinais da interface de comunicação entre um núcleo de processamento e um porta Local da NoC.... 37
Figura 24 – Formas de onda extraídas da simulação de transmissão de dados em uma porta local.... 38
Figura 25 – Diagrama de blocos da arquitetura MIPS I [PAT96].... 39
Figura 26 – Diagrama de blocos do processador Plasma.... 42
Figura 27 - Processador Plasma com novas entidades e novos registradores mapeados em memória.... 43
Figura 28 – Diagrama de blocos do sistema de vídeo conectado as interfaces de vídeo (VGA) e memória RAM (DDR RAM) da plataforma Xilinx ML-401.... 44
Figura 29 – Controlador de VGA seus sinais.... 45
Figura 30 – Sinal de sincronismo horizontal.... 47
Figura 31 – Sinal de sincronismo vertical.... 47
Figura 32 – Diagrama de blocos do controlador DDR conectado à memória DDR da placa ML-401.... 48
Figura 33 – Escrita na memória.... 49
Figura 34 – Leitura da memória.... 50
Figura 35 – Processo de escrita no módulo de vídeo.... 51
Figura 36 – Processo de leitura no módulo de vídeo.... 51
Figura 37 – Diagrama de blocos do controlador de teclado.... 52
Figura 38 – Forma de ondas representando a transmissão de um *scancode* pelo PS2.... 52
Figura 39 - Organização da plataforma desenvolvida neste trabalho.... 53
Figura 40 – Exemplo de mensagem recebida por um *wrapper* conectado a rede de interconexão.... 57
Figura 41 – Diagrama do formato das mensagens.... 59
Figura 42 – Forma de ondas extraída de simulação do *wrapper do controlador de teclado*.... 60
Figura 43 – Formas de onda mostrando o recebimento de uma requisição de leitura pelo *wrapper* de teclado.... 60

Figura 44 – Formas de onda mostrando o envio de uma mensagem de resposta do *wrapper* de teclado. 61
Figura 45 – *Wrapper* do controlador de vídeo. 62
Figura 46 – Forma de ondas mostrando a chegada de uma mensagem destinada ao módulo controlador de vídeo. 63
Figura 47 – *Wrapper* do controlador de vídeo armazenando dados recebidos através da rede em *buffer* local. 63
Figura 48 – Função responsável pela leitura dos comandos vindos do teclado. 64
Figura 49 – Representação de um peão de dimensões de 8 X 8 pixels 65
Figura 50 – Fluxo de comunicação entre Tarefa 1 e Tarefa 2 para a busca de jogada a partir de dois pontos da árvore de jogadas. 67
Figura 51 – Formas de onda mostrando um exemplo de escrita de uma mensagem no módulo de vídeo. 69
Figura 52 - Formas de onda mostrando o início de escrita de pacotes na memória de vídeo. 69
Figura 53 – Mensagem contendo o caractere “a”. 70
Figura 54 – Tela que mostra a execução do comando “show board” 70
Figura 55 – Tempo de escrita de uma linha gráfica na memória de vídeo. 72
Figura 56 – Tempo de leitura de uma linha gráfica 72

Lista de tabelas

Tabela 1 - Tipos de dados em linguagem C, tamanhos em bytes e mnemônicos do montador. 40
Tabela 2 - Tipos das instruções da arquitetura MIPS. 41
Tabela 3 - Quadro comparativo de diferenças entre a arquitetura MIPS e o processador Plasma. 42
Tabela 4 - Cores e respectivos sinais RGB. 46

1 Introdução

Jogos acompanham a humanidade há muitos anos. A atividade de jogar se desenvolveu junto com a evolução das sociedades. Jogos são atividades que envolvem uma ou mais pessoas, um conjunto de recursos como tabuleiros, tabelas, gráficos e que estão associados a um conjunto de regras. Um pouco mais formalmente pode-se definir um *jogo* como uma tripla J = <P, M, R>, onde P é o *conjunto de jogadores*, os agentes de ações, especificados pelo conjunto de regras (R) e utilizando os recursos do jogo (M); os jogadores podem ser pessoas, máquinas ou processos executando em um computador; M é o conjunto de *meios* ou *recursos do jogo* que viabilizam a realização deste, estando disponíveis para uso pelos jogadores (P) de acordo com o que é previsto pelas regras (R). Exemplos de recursos de jogos são tabuleiros, tabelas, peças, gráficos, imagens, ou até idéias; R é um *conjunto de regras* que controla as ações dos jogadores e o uso dos recursos (M) por estas ações, além da definição dos critérios para determinar a vitória de um jogador ou de um conjunto de jogadores no jogo.

Jogos têm como objetivo exercitar habilidades, o convívio social através de desafios ou simplesmente a diversão. Atualmente esta atividade dá origem a uma vasta gama de produtos comerciais que movimentam grandes somas de recursos e sustentam uma florescente indústria de jogos. Jogos computacionais, de uma forma breve, são jogos que utilizam como meio equipamentos eletrônicos como consoles de jogos, computadores ou outros. A Electronic Arts, por exemplo, uma das maiores empresas no ramo, faturou aproximadamente 3 bilhões de dólares em vendas de jogos computacionais no ano de 2002 [BET03]. O mercado de jogos computacionais é um dos que mais tira proveito deste cenário.

A acirrada disputa por novos consumidores força indústrias concorrentes à constantemente inovar técnicas, efeitos e, de forma geral, a atratividade dos jogos. Arquiteturas de computadores específicas fomentam estas inovações e assim, acabam contribuindo para aumentar o consumo de jogos. Como exemplo da expansão desse mercado, pode-se encontrar jogos associados também a diversos produtos que não foram destinados originalmente para este fim, como celulares e PDAs, entre outros.

1.1 Motivação

Várias são as motivações para o presente trabalho. Discutem-se aqui três temas que funcionam como delimitadores da forma do presente trabalho: (*i*) a tendência de uso de múltiplos processadores em um único chip; (*ii*) a evolução de consoles de jogos comerciais; (*iii*) a evolução de recursos de suporte a jogos em computadores pessoais.

1.1.1 Múltiplos Processadores Intrachip

Há uma tendência atual de empregar mais de um processador para implementar a unidade central de computadores e centrais de jogos. Estes múltiplos processadores interagem através de uma arquitetura de comunicação como, por exemplo, um barramento. A principal justificativa desta tendência é que a maneira tradicional de aumentar a velocidade de um único processador – aumentar o número de cálculos efetuados por unidade de tempo, diminuindo o tamanho dos transistores e aumentando a freqüência do relógio – dá cada vez menos resultados nas tecnologias submicrônicas atuais. Isto é devido, por exemplo, à elevada geração de calor por processadores monolíticos [MOO06].

Os transistores utilizados hoje possuem dimensões extremamente reduzidas. Mesmo estando no estado de cortado, correntes infinitesimais sempre atravessam o dispositivo. Este vazamento de corrente aquece os circuitos. Somado à corrente de troca de estado para conduzindo ou cortado, isto pode produzir uma dissipação excessiva de calor. Se processos de fabricação continuassem neste caminho, no ano de 2015 os processadores produziriam mais watts por milímetro quadrado do que a superfície do Sol [MOO06].

Nos últimos dois anos, a velocidade de processadores virtualmente estagnou, e as maiores empresas fabricantes anunciaram versões “dual-core” de seus processadores. Uma das vantagens destes é oferecer recursos para que o sistema operacional lide com tarefas que exigem maior paralelismo no processamento, o que aumenta consideravelmente o desempenho para processamento multitarefa, já que cada núcleo processador tem a sua própria interface independente com o barramento principal e sua própria memória *cache*.

1.1.2 Evolução de Console de Jogos

Alguns exemplos de uso de multiprocessamento em jogos computacionais são os videogames de sétima geração, tais como o Playstation 3, da Sony, o XBox 360 da Microsoft e o Wii da Nintendo.

O Playstation 3 faz uso do processador Cell (Cell BroadBand Engine Architecture), que conta com um processador Power PC similar aos usados pela Apple em seus computadores de mais alto desempenho e outros oito núcleos processadores, chamados de “processadores sinergéticos”, desenvolvidos para realizar tarefas multimídia. Um barramento especialmente desenvolvido interconecta os nove processadores alcança uma velocidade de transferência de dados de até 300 Gbits/s, segundo alegam seus proponentes [MOO06].

O XBox 360 também utiliza multiprocessamento, com o seu processador Xenon de três núcleos processadores baseados na arquitetura do Power PC, produzido pela IBM [MIC06].

1.1.3 Evolução de Suporte a Jogos em Computadores Pessoais

Jogos fazem uso de tecnologia de ponta, motivando o desenvolvimento de componentes melhores e mais rápidos. Pode-se notar isso na rápida evolução que ocorreram das placas de vídeo para computador nos últimos anos.

Além da tecnologia das placas de vídeo melhorar, novas formas para conectá-las ao computador foram desenvolvidas para melhorar ainda mais o seu desempenho. Um exemplo disso é a porta AGP (Advanced Graphics Port), que é um canal de alta velocidade ponto-a-ponto da placa de vídeo com a placa mãe. A AGP foi criada para que a placa de vídeo pudesse ter um acesso dedicado e de mais alta velocidade ao processador principal e à memória, conectando-se diretamente ao barramento principal do computador.

Antes da AGP, o padrão de conexão utilizado para vídeo era o PCI (Peripheral Component Interconnect), que serve para conectar não só placas de vídeo, mas também *modems*, placas de som, controladoras SCSI, placas de rede, e outras. Como o PCI é um barramento, todos esses periféricos competem pela transmissão de dados. Então com a evolução da demanda por alto desempenho gráfico atingiu-se um ponto onde o barramento PCI tornou-se o gargalo para operações de vídeo gerando a necessidade de criar interfaces específicas para tais dispositivos, dada a necessidade de uma grande vazão de dados diretamente com o processador e a memória.

A Figura 1 ilustra a conexão dos dispositivos em uma placa mãe que possui AGP. O fato da porta AGP estar conectada na North Bridge, no mesmo nível que o processador e memória mostra a importância dada a aplicações gráficas (principalmente jogos) no mercado de hoje.

Neste trabalho, não serão utilizados barramentos, apesar da vazão de dados que estes podem alcançar (o barramento do processador Cell, por exemplo, alega alcançar 300 Gbits/s de vazão). Será utilizada uma rede intrachip (em inglês, *Network on Chip* ou NoC), para explorar esse meio de transmissão de dados entre dispositivos e também para aproveitar, e ampliar os estudos já existentes no grupo de pesquisa GAPH (Grupo de Apoio ao Projeto de Hardware). Além disto, deseja-se aproveitar também as vantagens do uso de uma NoC (ver Seção 4.2).

1.1.4 Conclusão da Motivação

Além das motivações descritas nas Seções 1.1.1 a 1.1.3, a implementação de jogos é um assunto que faz uso de grande parte dos conhecimentos adquiridos ao longo do curso de Ciência da Computação, sendo de grande importância tanto comercial quanto acadêmica a nível mundial, justificando a proposta do presente trabalho.

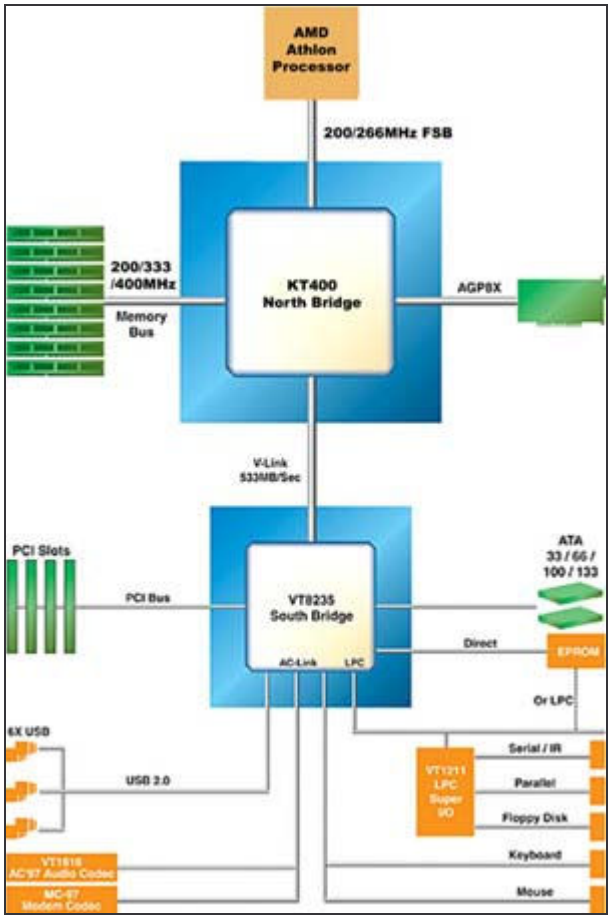


Figura 1 – Um *chipset* KT400 (North Bridge) conectado diretamente ao processador (AMD Athlon Processor), à memória, à porta AGP e ao chipset VT0235 (South Bridge).

1.2 Objetivos

Os objetivos maiores deste trabalho são o desenvolvimento de uma plataforma que permita a execução do código do GNU Chess [GNU06] e o porte do mesmo para o código objeto do processador PLASMA [OPE06] fazendo uso de multiprocessamento.

A plataforma desenvolvida apresenta características e funcionalidades semelhantes as características de máquinas dedicadas a jogos computacionais encontradas no mercado. Esta plataforma faz uso de uma plataforma desenvolvida em trabalho paralelo, chamada provisoriamente de MPSoC [WOS05] acrescida de módulo de vídeo (ver Seção 4.4) e módulo de teclado (ver Seção 4.5).

O código do GNU Chess foi modificado de modo a possibilitar sua compilação com a utilização do compilador MIPS-GCC (ver Seção 4.3.5) e execução no processador PLASMA usado pela plataforma MPSoC [WOS05].

O módulo de teclado foi desenvolvido de modo a ser conectado em um dos nodos da NoC contida na plataforma com o objetivo de fazer a interface entre o usuário e a NoC possibilitando a entrada de dados por parte de usuário com o uso de um teclado.

O módulo de vídeo foi desenvolvido visando possibilitar a geração de imagens em um monitor padrão VGA a partir de dados enviados pela NoC de forma a transmitir ao usuário informações visuais sobre andamento do jogo.

Concomitantemente ao desenvolvimento dos módulos da plataforma e ao porte do código do GNU Chess foram desenvolvidos *wrappers* para possibilitar a comunicação entre a NoC e os módulos. Adicionalmente ao processo de porte do GNU Chess foram desenvolvidos processos que permitissem a interação entre os módulos de hardware desenvolvidos, plataforma utilizada e o software do jogo de xadrez.

Posteriormente foram feitos testes de cada módulo individualmente e de todo o sistema fazendo a validação do projeto em simulação.

1.3 Estrutura do Trabalho

Os capítulos seguintes mostram o desenvolvimento deste projeto, as decisões tomadas e alterações do projeto. O Capítulo 2 apresenta o software desenvolvido bem como são descritas as estruturas de dados e a lógica que o jogo utiliza para a geração de jogadas e interação com o jogador. No Capítulo 4 a arquitetura de hardware é apresentada com os módulos que a compõem e suas características. O porte do software de jogo e questões específicas sobre o funcionamento deste em conjunto com a arquitetura de hardware são abordadas no Capítulo 5. O capítulo seguinte, Capítulo 6, aborda questões pertinentes ao desenvolvimento de hardware e software do projeto e como estes se comunicam para possibilitar a execução do jogo. Os testes para validação de módulos e da plataforma como um todo são mostrados no Capítulo 7. Por fim, no Capítulo 8 são apresentadas as conclusões e feitas sugestões para trabalhos futuros.

2 GNU Chess

Neste Capítulo será descrita a estrutura básica do GNU Chess.

2.1 Descrição do GNU Chess

GNU Chess é uma implementação de um jogo de xadrez completo de domínio público sob a licença da GNU GPL (GNU General Public License). É completo, no sentido de que possui todos os elementos necessários para um jogo de xadrez computacional incluindo: entrada e saída de dados, estrutura de modelagem do universo (tabelas de posições de peças) e algoritmos de geração de jogadas. Além disso, o GNU Chess permite a consulta a livros de jogadas e suporta interfaces visuais Xboard [GNU06].

Este trabalho utiliza o código da versão 5.07 do GNU Chess compilada com o MIPS-GCC 4.0.

2.1.1 Estrutura de arquivos da distribuição GNU Chess

Esta Seção lista e explica brevemente cada arquivo que compõe a versão 5.07 do código do GNU Chess. São os arquivos:

atak.c: implementa algoritmos que verificam as possibilidades de ataques das peças dispostas na configuração do tabuleiro como este se encontra.

book.c e book.h: implementa algoritmos que fazem geração de jogadas baseadas em jogadas feitas por jogadores humanos e armazenadas em arquivos chamados de livro de jogadas para melhorar a qualidade das mesmas na inicialização e finalização das partidas.

cmd.c: implementa algoritmos que validam os comandos inseridos pelo usuário e fazem chamada a função que os executa presente no arquivo move.c.

debug.c: implementa rotinas de depuração do código do GNU Chess.

epd.c: implementa rotinas para armazenamento de históricos de jogos utilizando o padrão “EPD” (ver Seção 2.1.7).

eval.c e eval.h: implementa rotinas que avaliam cada peça do tabuleiro levando em consideração sua classe (peão, cavalo, bispo, torre, rainha ou rei) e a posição em que cada uma se encontra para possíveis jogadas com cada uma delas.

genmove.c: implementa rotinas que fazem modificações nas estruturas do código nas quais a configuração do tabuleiro e dados sobre a partida estão armazenadas conforme as jogadas vão sendo realizadas durante a partida.

getopt1.c, getopt.c e getopt.h: contém partes necessárias do código da biblioteca GNU C para a compilação do GNU Chess, caso o sistema em que o GNU Chess estiver sendo compilado não possua a mesma.

hash.c: implementa rotinas de acesso às tabelas *hash* do GNU Chess.

hung.c: implementa rotinas que calculam as peças que estão em perigo de sofrer um ataque.

init.c: implementa rotinas de inicialização do GNU Chess.

input.c: implementa rotinas que permitem entrada de dados por parte do usuário. Estas rotinas podem ser adaptadas dependendo da interface da entrada de dados, que pode ser textual, visual (gráfica) ou através de uma conexão a um módulo externo como é o caso deste trabalho (ver módulo de teclado na Seção 4.5 e modificações no código do GNU Chess na Seção 5.1).

iterate.c: implementa uma iteração na árvore de buscas de jogadas do algoritmo de geração de jogadas.

lexpgn.c e lexpgn.h: parsers de arquivos de livros de jogadas no padrão PGN.

main.c: implementa o laço principal no qual o jogo irá ser executado.

move.c: implementa rotinas que executam as jogadas definidas pelo usuário ou pelo algoritmo gerador de jogadas.

null.c: implementa rotinas que fazem uma jogada nula, ou seja, nenhuma peça é movida, apenas as estruturas de dados são atualizadas.

output.c: implementa as rotinas que geram saídas de dados para o usuário, seja visual ou textual. Estas rotinas podem ser adaptadas dependendo da interface de saída de dados. No caso deste trabalho a rotina foi modificada de forma a se comunicar com o módulo de vídeo (ver módulo de vídeo na Seção 4.4 e modificações no código do GNU Chess na Seção 5.1)

pgn.c: implementa rotinas que salvam o histórico de partidas em arquivos texto no padrão PGN descrito na Seção 2.1.7.

players.c: implementa rotinas que armazenam e mostram informações sobre os jogadores como: quantas partidas cada um perdeu, ganhou ou empatou, nome dos jogadores, etc.

ponder.c: implementa rotina que além usar a rotina de busca de jogadas na árvore, adiciona uma verificação por jogadas em livros de jogadas.

quiesce.c: implementa rotina que busca pelos valores de α e β, necessários para o algoritmo da heurística Alpha-Beta poda (ver Seção 2.3.3).

random.c: implementa gerador aleatório de 32 bits.

repeat.c: implementa rotina que verifica repetições de jogadas na tabela *hash*.

search.c: implementa rotina de busca de jogadas na árvore de busca.

solve.c: implementa rotina que busca por jogadas em arquivos do padrão EPD.

sort.c: implementa rotinas que selecionam uma jogada seguindo critérios como: jogadas baseadas na tabela *hash* (que armazena melhores jogadas), se a jogada vai capturar alguma peça adversária e histórico do jogo.

test.c: implementa rotinas que testam a velocidade de execução de algumas partes do código do GNU Chess como avaliação de peças e geração de jogadas além de implementar rotinas que ler arquivos no padrão EPD e testar possíveis jogadas.

ttable.c: implementa rotinas que buscam ou armazenam dados nas tabelas *hash* existentes.

util.c: implementa algumas rotinas úteis de propósito mais geral como por exemplo, uma rotina que conta o número de bits com valor "um" em um *bitboard*.

common.h: contém a declaração de todas as funções, variáveis (declaradas como "extern") e a maior parte das constantes. Este arquivo é incluído por todos os outros arquivos do GNU Chess.

config.h: contém as definições das opções que são selecionadas durante a fase de compilação do código.

GCint.h: definição de macros.

inlines.h: contém funções declaradas de forma estática ("static inline").

version.h: contém a versão do código do GNU Chess.

2.1.2 Representação do tabuleiro

Existem várias maneiras de representar uma configuração do tabuleiro em um jogo de xadrez. A mais natural é uma matriz bidimensional de oito por oito elementos (o tamanho de um tabuleiro de xadrez). Cada elemento da matriz teria o valor da peça que ocupa a posição correspondente do tabuleiro. As duas vantagens desta representação são sua fácil implementação e facilidade de avaliação de valor das peças. A desvantagem é a dificuldade de cálculo de jogadas, como por exemplo, cálculo de todas as jogadas possíveis e verificação se o rei está em cheque.

A representação utilizada pelo GNU Chess é o *bitboard*. Esta representação consiste em um vetor tipo *long long* (GNU C) de 64 bits, que é uma estrutura utilizada para representar peças em um tabuleiro, não só para jogos de xadrez, mas também para vários outros jogos de tabuleiro computacionais, como damas e outros.

As posições em um tabuleiro de xadrez são denominadas usando uma combinação de letras e números, onde cada coluna é representada por uma letra (de A a H) e cada linha é representada por um número (de 1 a 8). A posição A1 que, normalmente é o canto inferior esquerdo, é representada pelo bit 0 do vetor de 64 bits e a posição H8 (canto superior direito) é representada pelo bit 63. Esta representação é mostrada na Figura 2. Os valores dentro da tabela representam o bit de cada posição do tabuleiro no *bitboard*.

8	56	57	58	59	60	61	62	63
7	48	49	50	51	52	53	54	55
6	40	41	42	43	44	45	46	47
5	32	33	34	35	36	37	38	39
4	24	25	26	27	28	29	30	31
3	16	17	18	19	20	21	22	23
2	8	9	10	11	12	13	14	15
1	0	1	2	3	4	5	6	7
	A	B	C	D	E	F	G	H

Figura 2 – Posições em um tabuleiro de xadrez denominadas usando-se uma combinação de letras e números.

No jogo de xadrez, cada jogador possui seis tipos de peças diferentes, elas são: peão, torre, cavalo, bispo, rainha e rei. Uma matriz bidimensional de doze *bitboards* (tamanho dois por seis) seria suficiente para mostrar a posição de todas as peças de um tabuleiro: seis para as brancas e seis para as pretas. Por exemplo, pode-se ter acesso à posição de todos os cavalos pretos do tabuleiro da seguinte maneira:

```
BitBoard bb[PRETO][CAVALO]
```

O vetor "bb" do tipo *bitboard* de 64 bits terá o valor "um" nas posições do tabuleiro onde houver cavalos pretos, e no resto, terá o valor "zero".

Esta representação permite que o gerador de jogadas avalie rapidamente todas as jogadas possíveis para um grupo de peças. Peões movimentam-se uma posição à frente da atual de cada vez. Assim, utilizando-se apenas dois *bitboards*, pode-se visualizar todas as possíveis posições que todos os peões de um jogador podem assumir na próxima jogada (não incluindo ataques) com as seguintes operações: movimenta-se todos os peões do jogador, uma posição à frente da atual. Isto pode ser feito com um deslocamento à direita de oito posições no *bitboard* que representa os peões do jogador. Se por exemplo, os peões do jogador ocupam as toda a linha 2 do tabuleiro (ver Figura 2), eles estariam ocupando do bit 8 ao bit 15 do vetor *bitboard* que os representa, após um deslocamento à direita, os peões passariam a ocupar toda a linha 3 do tabuleiro, ou seja, do bit 16 ao bit 23 do vetor *bitboard*. Logo após, verifica-se quais destas novas posições estão vazias no *bitboard* que contém a posição de todas as peças da partida. Isto pode ser visualizado da Figura 3 a Figura 7.

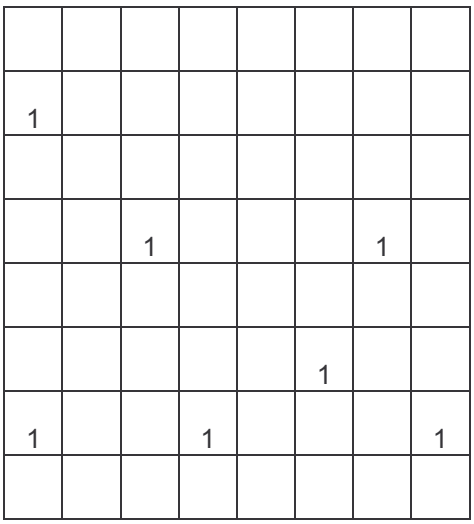


Figura 3 - Representação de *bitboard* de peões. As posições com o valor "1" são as que contém peões. As posições vazias contem o valor "zero".

Na Figura 4 é feito um deslocamento de oito bits em relação à Figura 3, ou seja, todos os peões foram movidos uma posição à frente.

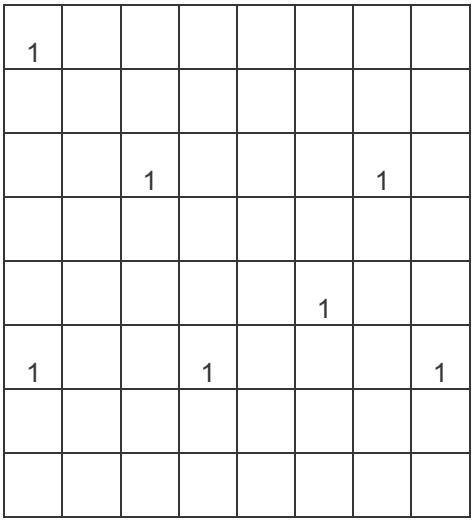


Figura 4 - Representação de *bitboard* de peões depois de sofrer um deslocamento de oito bits. As posições vazias contem o valor "zero".

A Figura 5 mostra uma a representação de um *bitboard* de todo o tabuleiro de uma partida qualquer na qual os peões da Figura 3 estão inclusos.

1				1	1		
1			1				
	1				1	1	
		1		1		1	
				1	1		
1		1	1		1	1	
1			1				1

Figura 5 – Representação do tabuleiro de um exemplo de uma partida de xadrez . As posições com o valor "um" representam posições ocupadas por uma peça qualquer. As posições vazias contem o valor "zero".

A Figura 6 mostra o tabuleiro da Figura 5 negado, ou seja, todas as posições vazias foram marcadas com o valor um e todas as posições com valor um ficaram vazias.

	1	1	1			1	1
	1	1		1	1	1	1
1		1	1	1			1
1	1		1		1		1
1	1	1	1			1	1
	1			1			1
	1	1		1	1	1	
1	1	1	1	1	1	1	1

Figura 6 – Negação do tabuleiro da Figura 5. Todas as posições antes livres agora estão marcadas com o valor "um". As posições vazias contem o valor "zero".

A Figura 7 mostra um *bitboard* com todas as posições que todos os peões do jogador podem assumir na próxima jogada. Este *bitboard* é resultado de uma operação "e" bit a bit nos *bitboards* representações na Figura 4 e Figura 6.

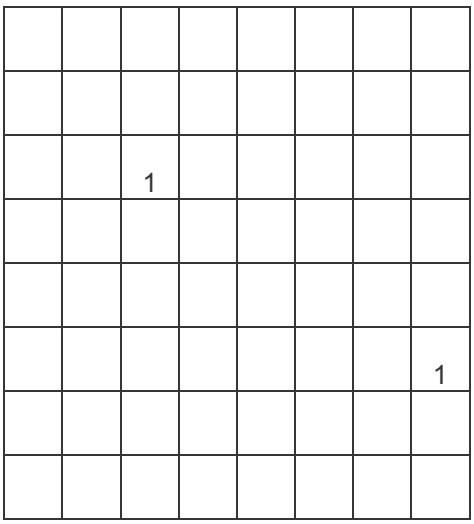


Figura 7 – Operação "e" bit a bit aplicado entre Figura 4 e Figura 6 resultando em todas as posições possíveis para os peões representados na Figura 3. As posições vazias contem o valor "zero".

Se houvessem dois *bitboards*, um chamado de "PeõesBrancos" para representar todos os peões brancos, e outro chamado de "TodasPosições" para representar todas as posições ocupadas do tabuleiro, a expressão (em linguagem C) **(PeõesBrancos >> 8) & ~TodasPosicoes** poderia ser usada para representar todas as possíveis posições que os peões brancos podem assumir na próxima jogada. A representação dos peões brancos sofreu um deslocamento de oito bits (ou seja, foi movida uma posição à frente da atual) e a representação de todas as posições de peças contidas no tabuleiro foi negada. A operação "e" bit a bit destas duas representações resultará em um novo *bitboard* contendo todas as posições possíveis para os peões brancos na próxima jogada.

De maneira semelhante, é possível calcular todas as posições possíveis para cavalos e o rei. Para gerar jogadas com peças que se deslocam ao longo de colunas, linhas ou diagonais inteiras, é necessário um pouco mais de informação, para isso representações como o *bitboard rotacionado* (explicado em detalhes na Seção 2.1.4) são utilizadas.

2.1.3 Estruturas de dados relevantes

Como já foi mencionado na Seção 2.1.2, a principal estrutura de dados do GNU Chess é o *bitboard*.

Outra estrutura de dados importante é o *Board*, definida no arquivo *common.h* do GNU Chess, que tem como objetivo descrever a posição das peças e revelar informações importantes sobre o jogo, como a posição de todas as peças inimigas e amigas de cada jogador, se o rei e a torre de cada jogador estão em roque, proporcionar fácil acesso à posição dos reis para definir se estão em cheque, entre outras.

A estrutura *Board* do GNU Chess está representada na Figura 8(a).

```
typedef struct
{
   BitBoard b[2][7];
   BitBoard friends[2];
   BitBoard blocker;
   BitBoard blockerr90;
   BitBoard blockerr45;
   BitBoard blockerr315;
   short ep;
   short flag;
   short side;
   short material[2];
   short pmaterial[2];
   short castled[2];
   short king[2];
} Board;
```

(a)

```
typedef struct
{
        HashType key;
        int move;
        int score;
        uint8_t flag;
        uint8_t depth;
} HashSlot;
```

(b)

```
typedef struct
{
        KeyType pkey;
        BitBoard passed;
        BitBoard weaked;
        int score;
        int phase;
} PawnSlot;
```

(c)

Figura 8 – Descrição das estruturas de dados (a) `Board`, (b) `HashSlot` e (c) `PawnSlot`.

A matriz “b” representa todas as peças no tabuleiro. A matriz “friends” representa as peças amigas (do lado do jogador), as matrizes que começam com o nome “blocker” fazem parte da geração de jogadas e verificação de peças bloqueadas usando para isso o método *Rotated bit-board*, explicado na Seção 2.1.4. A variável “ep” é setada com o valor da posição quando um peão movimenta-se duas casas em uma jogada. A variável “flag” é relacionada a privilégios de tipos de jogadas da torre (como se pode ou não fazer *roque*). A variável “side” armazena o lado que está jogando no momento (lado de peças pretas ou brancas). A matriz “material” armazena o total de peças que não estão mais na partida (sem incluir o rei e peões), assim como “pmaterial” armazena o total de peões que não estão mais na partida de cada um dos lados. A matriz “castled” identifica se o rei está em cheque para cada um dos lados e a matriz “king” mostra a posição do rei de cada um dos lados.

Uma outra estrutura de dados relevante no GNU Chess é *hashslot*, utilizada pela tabela *General hash* (ver Seção 2.1.8). A estrutura *hashslot* está representada na Figura 8(b).

O tipo *HashType* é um vetor de 64 bits. O campo *key* é uma chave hash de 64 bits, baseada na profundidade da árvore de busca na qual a jogada foi encontrada. Quando uma jogada melhor é encontrada nesta profundidade, este registro é substituído e transportado para a segunda posição da tabela. O inteiro *move* armazena a melhor jogada da profundidade corrente da árvore. O inteiro *score* armazena o valor (em pontos) da jogada. O vetor de oito bits *flag* faz referencia a informações do algoritmo alpha-beta poda (ver Seção 2.3.3). O vetor de oito bits *depth* armazena a profundidade da árvore de busca na qual a jogada foi encontrada.

A estrutura *PawnSlot* é utilizada pela tabela *Pawn hash* (ver Seção 2.1.8) e está representada na Figura 8(c).

O tipo KeyType é um vetor de 32 bits. O campo *pkey* é uma chave de 32 bits baseada na disposição de peões no tabuleiro. O *bitboard passed* é um vetor de 64 bits que representa todos os peões "passados". Um peão "passado" é um peão que está em uma posição na qual nada pode impedi-lo de chegar à 8ª casa do tabuleiro. O *bitboard weaked* é um vetor de 64 bits que representa todos os peões que estão prestes a sofrer um ataque. O inteiro *score* armazena o valor em pontos da disposição de peões. O inteiro *phase* mostra a fase em que o jogo se encontra (inicio, meio, fim, etc).

2.1.4 Gerador de movimentos

O método utilizado é o *bitboard rotacionado*, que é uma variação do *bitboard*. O *bitboard rotacionado* torna algumas tarefas de verificação mais eficientes. As posições de todos os *bitboards* do jogo são rotacionadas em 45 graus, 90 graus para direita ou esquerda para melhor visualizar as jogadas possíveis de peças que se movimentam em colunas ou linhas inteiras (torres) ou em diagonais inteiras (bispos). No *bitboard* normal (zero grau), pode-se visualizar mais facilmente jogadas de peças que se movimentam em colunas, como peões (uma posição à frente) ou torres (quando se movimentam nas colunas), e rei (uma posição para frente ou para trás).

Utilizando o *bitboard rotacionado* a 90 graus, pode-se visualizar mais facilmente as jogadas em linhas (torres) mais facilmente, rotacionando o tabuleiro a 45 graus, pode-se visualizar mais facilmente jogadas nas diagonais do jogo (bispos).

2.1.5 Busca por jogadas

Tem por objetivo buscar na árvore de jogadas, a melhor jogada possível que o computador pode realizar dada configuração do tabuleiro no momento da busca.

A busca por jogadas utiliza o algoritmo Alpha-beta poda (explicado na Seção 2.3) com uma pequena modificação que consegue um pouco mais de desempenho chamada PVS (Principal Variation Search).

2.1.6 Avaliação de jogadas

A avaliação de jogadas tem por objetivo avaliar se vale a pena ou não fazer uma jogada em um determinada configuração de peças no tabuleiro, para isso, cada peça é avaliada individualmente de forma numérica através de uma pontuação.

A avaliação é feita através de uma série de rotinas contidas no arquivo *eval.c* do GNU Chess. Estas atribuem pontos a cada peça do jogo. Exemplos são as rotinas *ScoreP(), ScoreN() e ScoreB()*. As rotinas avaliam cada peça do tipo correspondente (P - *Pawn* (Peão), N – *Knight* (Cavalo), B –

Bishop (Bispo)), de acordo com suas posições no tabuleiro e suas oportunidades de jogadas, atribuindo uma pontuação a elas e decidindo se é ou não interessante fazer uma jogada com cada uma destas peças.

2.1.7 Livro de jogadas

Livro de jogadas são arquivos que descrevem uma coleção de boas jogadas definidas através da experiência de jogadores humanos. Estes arquivos podem estar em vários formatos conhecidos. Dois deles são aceitos pelo GNU Chess: EPD e PGN.

PGN (do inglês Portable Game Notation) [PGN06] é um padrão desenvolvido para representação de partidas de xadrez em arquivo no formato ASCII.

O padrão EPD (do inglês Extended Position Description)

[EPD06] permite que se use um arquivo para armazenar configurações tabuleiros de partidas de xadrez junto com informações sobre a mesma .

Um livro em formato ASCII pode ser guardado no arquivo *book.dat* para consultas de jogadas durante o jogo.

2.1.8 Tabela *Hash*

É uma área de memória onde informações sobre posições são guardadas. Existem duas tabelas nesta versão: *General hash* e *Pawn hash*. A tabela *General* hash é utilizada para guardar as melhores jogadas em cada altura da árvore de busca, com o objetivo de aumentar a velocidade da busca de jogadas. A tabela *Pawn hash* é utilizada para guardar as melhores jogadas de peões, levando em consideração disposição dos peões no jogo e momento da partida (ver Seção 2.1.3). As tabelas de *hash* são utilizadas para aumentar a velocidade das jogadas durante uma partida. Segundo os desenvolvedores do GNU Chess, a velocidade de execução pode aumentar de 25% a 50% no meio do jogo e muito mais do que isso perto do fim do jogo devido ao uso destas tabelas. A tabela *General hash* tem como valor padrão 1024 *hashslots* e a tabela *Pawn hash* 512 *pawnslots* (ver Seção 2.1.3).

2.2 Heurísticas no jogo de xadrez

A via que muitos autores consideram privilegiada para o xadrez computacional é a "força bruta", isto é, o recurso intensivo à capacidade de processamento da máquina para realizar cálculos exaustivos. Em termos puramente lógicos, a abordagem por busca exaustiva é simples: para cada posição, são determinadas todas as seqüências de jogadas legais que podem seguir-se, contando

passo a passo com todas as respostas possíveis do adversário. Em seguida, escolhe-se a linha de jogo que leva mais seguramente ao estado final desejado (vitória) ou a uma posição intermediária favorável, assumindo que o adversário fará em cada momento o melhor lance à sua disposição. Por isso, o fator Nps ("nós por segundo", *nodes per second*, número de posições diferentes analisadas por segundo) é muitas vezes considerado a principal medida de força de um computador de xadrez. Essa força, para um mesmo programa, varia com o poder de cálculo da máquina. O problema é que, havendo no xadrez um número muito grande de posições legais e respectivas combinações em jogos válidos, a busca exaustiva torna-se impraticável. Calculou-se que uma análise exaustiva que antecipe uns meros cinco lances requer a análise de seiscentos trilhões de posições. O *Deep Blue* levaria cerca de oitocentas e vinte horas para fazer essa análise. Temos, portanto, sérias limitações ao mero uso da força bruta. Então, o que fazer?

Dadas as limitações da "força bruta", o xadrez computacional tem de recorrer a heurísticas. Heurística é uma regra prática, baseada em experimentos ou na experiência de quem a usa, empregada para resolver problemas obtendo algum grau de otimalidade na solução, evitando explorar todas as soluções possíveis para encontrar o resultado ótimo absoluto. A implementação das heurísticas para o jogo de xadrez começa pela definição dos aspectos a serem avaliados com base em padrões de posicionamento das peças, continua com a criação de algoritmos para reconhecer tais padrões e passa para a definição do peso que cada fator terá na avaliação global. A função de avaliação posicional materializa todo o processo, definindo como será atribuído um único valor numérico a cada posição com base em todos os elementos considerados. Por exemplo, o *Deep Blue II,* computador da IBM que venceu o mestre enxadrista Kasparov em uma partida de xadrez, usava uma "função de avaliação rápida" (para os aspectos mais relevantes e mais fáceis de identificar) e uma "função de avaliação lenta" (para os aspectos mais sutis): se a primeira fornecesse um resultado claramente de boa qualidade, a segunda era dispensada.

Tudo o que possa ser importante e não esteja previsto, ou esteja sub valorizado na função de avaliação torna-se "invisível" para o jogo de xadrez. As situações de maior risco são as posições instáveis, aquelas em que o balanço material ou posicional pode ser alterado abruptamente: um rei está em xeque, uma peça foi tomada, há peões avançados na sexta ou sétima linha (um peão que atinja a oitava linha pode ser "promovido" a uma peça à escolha do jogador, constituindo assim uma forte ameaça).

O que acontece é que a distinção entre posições mais ou menos desejáveis é em geral pouco clara. Por exemplo, considere-se os peões dobrados. "Peões dobrados" são peões da mesma cor que ocupam uma mesma vertical do tabuleiro, tendo assim pouca mobilidade e sendo facilmente atacados e bloqueados. Estes se constituem, em geral, um ponto fraco. Mas podem, em certos casos, ajudar a controlar o centro do tabuleiro.

Amorim [AMO02] identifica vários tipos de posições que enganam sistematicamente os programas de xadrez. Um caso é a incapacidade do programa para distinguir entre uma troca de peças desfavorável e um sacrifício. Chama-se "sacrifício" à entrega deliberada de uma peça em troca de uma vantagem posicional, relevante, mas por vezes sutil e cujo valor só será revelado mais tarde e em função da situação concreta do jogo.

Outro caso é o de uma peça "encravada". Uma peça "encravada" não pode deixar a casa onde se encontra, pois se o fizer expõe uma peça de valor superior - que está protegendo-a por interposição - à captura. Se a peça exposta à ameaça é o rei, retirar a peça "encravada" é ilegal. Em outros casos, a peça encravada pode sair, mas com perda grave de material. Assim, uma peça encravada está na prática reduzida à condição de muro contra uma ameaça do adversário, mas um programa de xadrez normalmente considera-a em termos formais como se ela continuasse a ter poder de agir sobre várias casas do tabuleiro.

Outra situação ambígua para um programa de computador é a de uma cadeia de peões fechada. Uma "cadeia de peões fechada" é uma estrutura de peões que se protegem mutuamente e que têm um efeito de barreira, dividindo o tabuleiro em dois campos (todas as peças do adversário estão do outro lado da barreira e não podem furá-la), de tal modo que o rei da mesma cor fica inacessível às peças de que dispõe o adversário. Ora, perante uma cadeia de peões fechada acontece que as jogadas de preparação para uma ruptura vantajosa parecem inofensivas ao computador, porque os seus efeitos estão para lá do seu horizonte de análise.

É para incorporar conhecimento deste tipo em heurísticas que se recorre à experiência e sensibilidade de mestres enxadristas humanos que identifiquem indicadores de desequilíbrio.

2.3 Implementação das Heurísticas

Alguns algoritmos e heurísticas utilizados em jogos de xadrez e outros serão explicadas a seguir.

2.3.1 Minimax

É um método de decisão da melhor jogada possível que o computador poderia realizar a partir do estado atual do jogo ignorando quanto tempo demoraria para calcular. Este método é empregado em jogos baseado em turnos, ou seja, a cada turno um jogador fará sua jogada, sendo um deles normalmente, o computador. O método enumera todas as alternativas de jogadas possíveis através de uma estrutura de árvore e seleciona a melhor utilizando alguns critérios. Para que o computador decida uma jogada no seu turno, ele deve prever qual jogada o seu oponente fará durante o próximo

turno, como medida de segurança, assume-se que o oponente sempre fará a melhor jogada possível para si.

Para aplicar este método é necessário criar uma maneira de avaliar as disposições de peças ou do tabuleiro de um jogo numericamente com (por exemplo) +1 sendo uma boa disposição do jogo para si, -1 sendo uma boa disposição do jogo para o oponente e com valores dentro deste intervalo sendo disposições não tão boas para os dois jogadores. Se for assumido que +1 significa que o jogador A ganhou, -1 significa que o jogador B ganhou e 0 significa um empate, então o jogador A tentará maximizar a pontuação do jogo e o jogador B tentará minimizá-la, daí o nome do método. A Figura 9 mostra o método sendo utilizado em um Jogo da Velha.

A árvore do Minimax cresce exponencialmente a cada jogada, sendo que mesmo em jogos simples, ou seja, com possibilidades reduzidas de jogadas, executá-lo até completar todas jogadas possíveis seria muito lento.

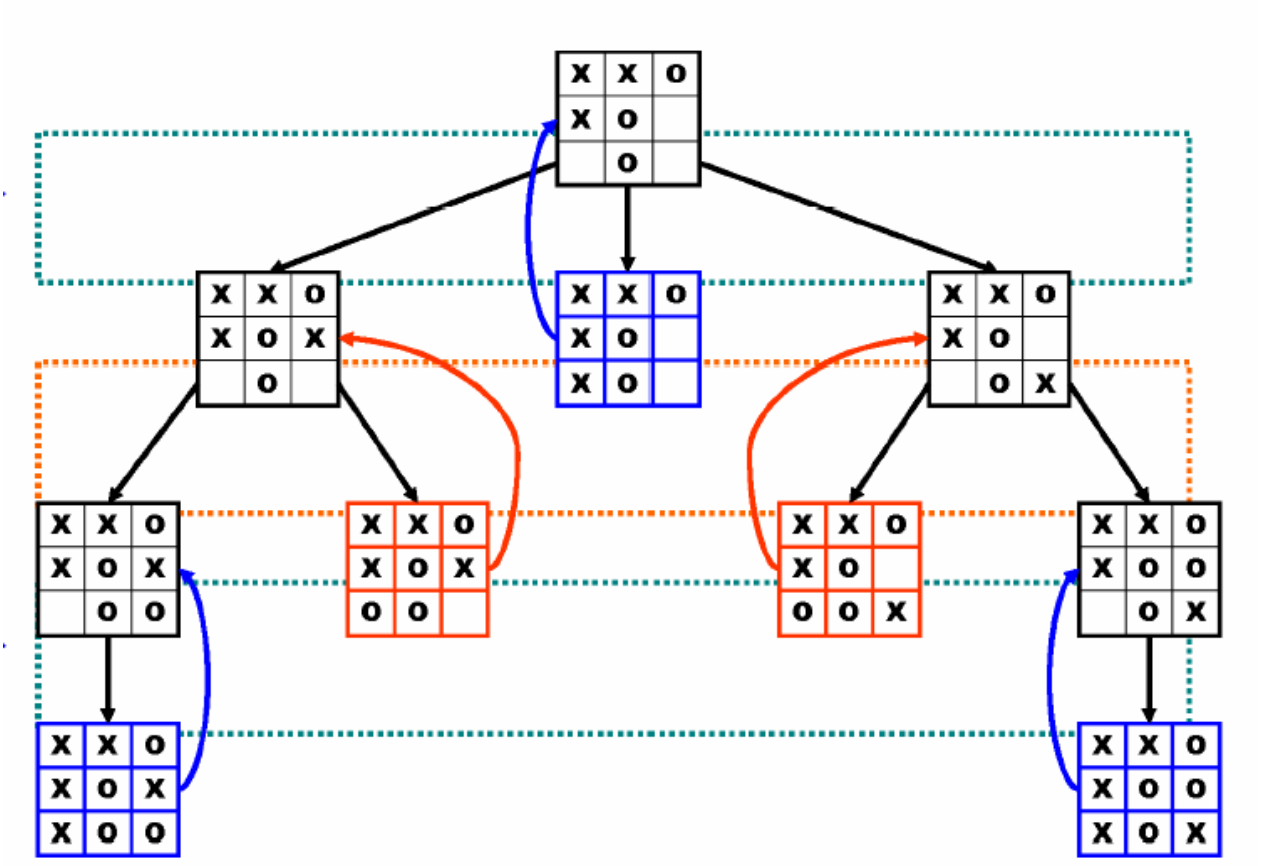


Figura 9 - algoritmo do Minimax sendo aplicado ao Jogo da Velha. O computador (que tem seu turno representado pela cor azul) joga com o xis e o jogador humano (que tem seu turno representado pela cor vermelha) joga com a bola. As setas de volta representam a jogada que foi selecionada em cada ponto do jogo [KAM06].

2.3.2 Poda Simples

É uma otimização do Minimax. O Minimax procura uma boa jogada em uma sub-árvore de cada vez, quando uma boa jogada é encontrada em uma delas, não é mais necessário procurar em outras sub-árvores, encerrando a busca. No momento em que o computador encontrou a jogada vencedora no exemplo da Figura 9, não seria mais necessário buscar em outras sub-árvores. A Figura 10 representa o uso deste método no Jogo da Velha.

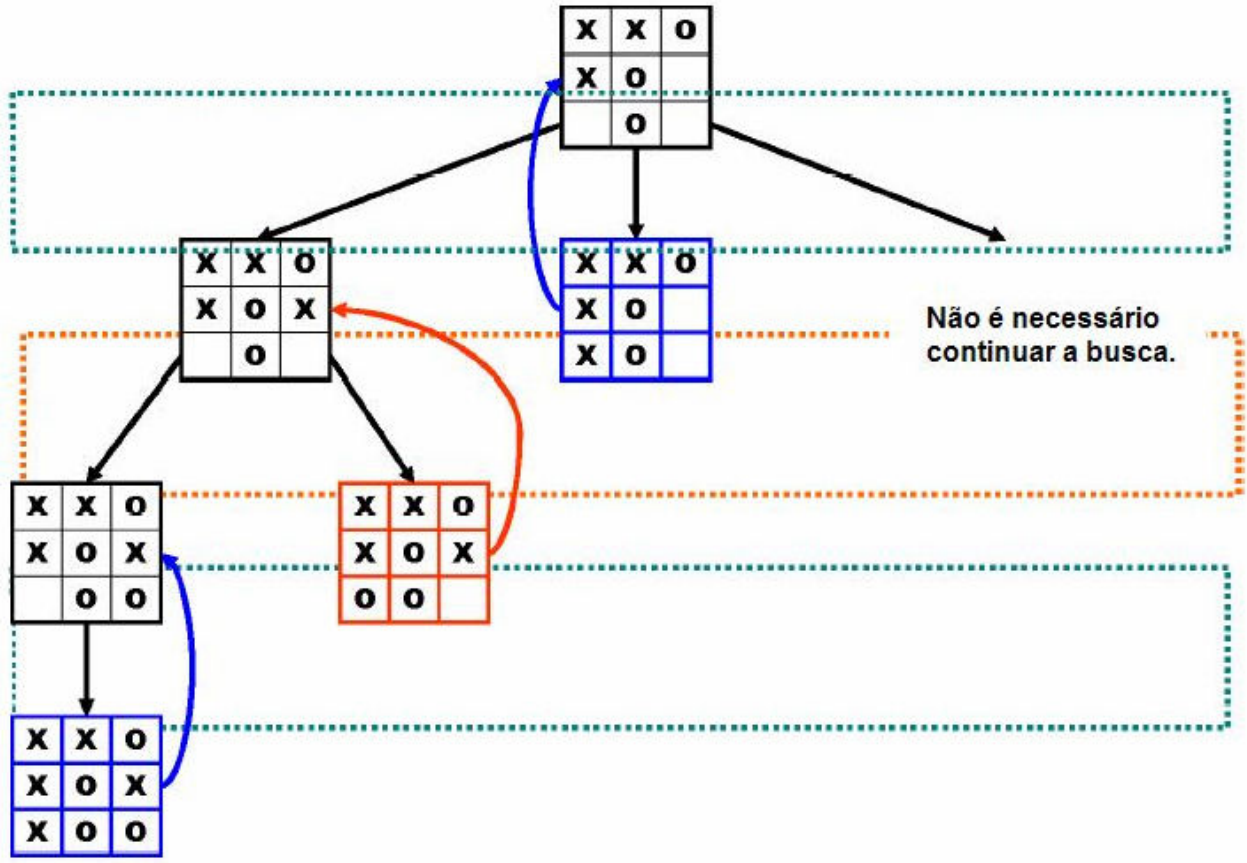


Figura 10 – Poda Simples: já que um nodo vencedor foi encontrado, não é mais necessário continuar procurando. O computador (que tem seu turno representado pela cor azul) joga com o xis e o jogador humano (que tem seu turno representado pela cor vermelha) joga com a bola. As setas de volta representam a jogada que foi selecionada em cada ponto do jogo [KAM06].

2.3.3 Alpha-beta poda

É um algoritmo de busca que reduz ainda mais que a poda simples o número de nodos que precisam ser avaliados na árvore de pesquisa pelo método Minimax mantendo o melhor resultado encontrado ao longo da busca pela sub-árvore. α é a melhor jogada que o computador pode conseguir e β é a melhor jogada que o jogador humano pode conseguir. Se $\alpha \geq \beta$, ou seja, o jogador humano tem uma jogada melhor que o computador, não é mais necessário procurar nesta sub-árvore. α sempre começa com valor –2 e β sempre começa com valor +2. A Figura 11 mostra este método sendo aplicado a um jogo arbitrário.

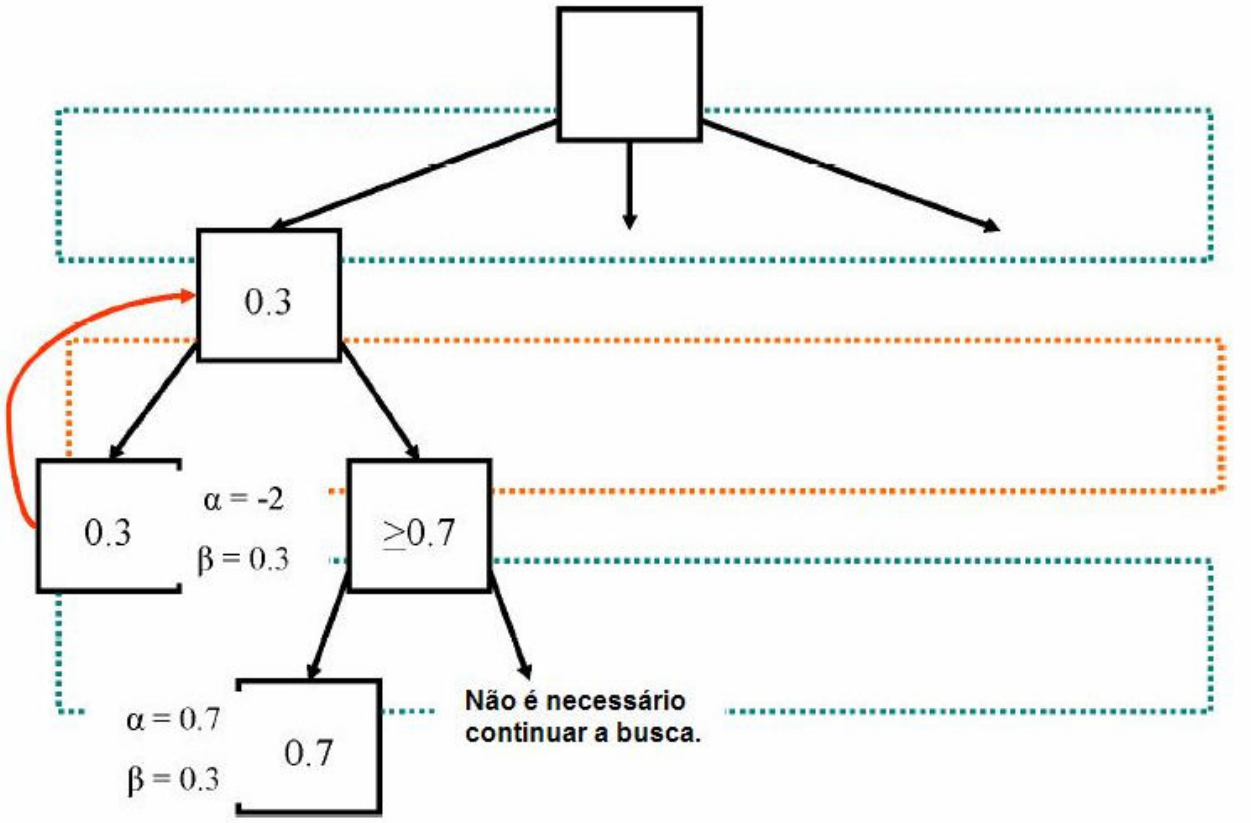


Figura 11 – Alpha-beta poda sendo aplicado a um jogo arbitrário. O computador tem seu turno representado pela cor azul e o jogador humano seu turno representado pela cor vermelha. A seta de volta representa a jogada que foi selecionada. Pode-se perceber que quando a jogada do jogador humano superou em pontuação a jogada do computador ($\alpha \geq \beta$), a busca parou [KAM06].

3 Revisões e Classificações de jogos

Segundo Crawford [CRA97] a atividade de jogar pode ser classificada em cinco grandes grupos conforme os meios utilizados para a realização dos jogos: jogos de tabuleiro, cartas, habilidades atléticas, jogos infantis e jogos computacionais.

Um jogo computacional é um software executado em uma plataforma eletrônica que fornece meios para o jogador poder interagir com o mesmo. Estes jogos freqüentemente são classificados em gêneros. Alguns dos gêneros mais comuns podem ser classificados, segundo Betke [BET03]: (*i*) jogos de estratégia – são jogos que exigem raciocínio e planejamento estratégico para superar os adversários (e.g. *Commandos* da Eidos Interactive); (*ii*) jogos de aventura – jogos em que o usuário deve resolver enigmas e charadas para prosseguir (e.g. *The Curse of Monkey Island* da Lucas Arts); (*iii*) *role-playing games* – jogos nos quais o jogador deve interagir através de um personagem com características próprias geralmente escolhidas no início do jogo e que evoluem em seu decorrer (e.g. *Ragnarok* da Gravity Corp.); (*iv*) ação – jogos onde o jogador deve ter reações rápidas para sobrepujar os desafios. Jogos de tiro são comuns nesse gênero (e.g. *Half Life* da Valve Games); (*v*) simuladores – jogos que almejam simular alguma situação da vida real. São bastante usados, também, para fins educacionais (e.g. *Flight Simulator* da Microsoft).

Títulos pertencentes a estes gêneros são encontrados em diferentes plataformas como jogos para: plataformas dedicadas (*Street Fighter 2* para *arcades*), plataformas portáteis (jogo da serpente para celulares e *palmtops*), videogames caseiros (*God of War* para Playstation2), computadores pessoais (*Quake 3*) e mainframes como o *DeepBlue* da IBM.

Existem diferentes formas de classificar jogos. Cris Crawford em 1982 propôs uma classificação de jogos na tentativa de organizar o trabalho de desenvolvimento de jogos [CRA97]. Em seu trabalho, Crawford definiu cinco classes que juntas englobariam todos os jogos. São estas classes:

- jogos de tabuleiro – são jogos onde os jogadores dispõem de um conjunto de peças geralmente associadas pertencentes diretamente a cada jogador. Estas peças são dispostas em um suporte dividido em setores onde os jogadores devem movê-las buscando conquistar territórios ou eliminar as peças do adversário.
- Jogos de cartas – usualmente, baralhos tradicionais são compostos por 52 cartas divididas em 13 valores associados a quatro naipes. Os objetivos deste tipo de jogo são fazer combinações de cartas definidas pelas regras que levem o jogador à vitória. Como o número de combinações é muito grande e geralmente excede a capacidade de raciocínio dos jogadores, muitos destes jogos ganham um perfil

mais intuitivo.

- Jogos atléticos – nesta classe de jogos, as proezas físicas são mais importantes que as habilidades mentais dos competidores. É importante estabelecer uma diferenciação entre competição e jogo já que ambos são facilmente confundidos neste âmbito. Nos jogos é preciso haver a interação direta entre os participantes enquanto que na competição o desempenho de um não necessariamente afeta o do outro. Desta forma, uma corrida pode ser classificada como uma competição visto que os envolvidos estão correndo contra o relógio. Em uma partida de vôlei, por outro lado, os integrantes de um time influenciam nas ações do time adversário.
- Jogos infantis – nestes jogos são desenvolvidas habilidades sociais. Estes não envolvem elementos complexos, mas desafiam a criança a atingir seus limites. Seus objetivos são, basicamente o desenvolvimento de atividades recreativas.

Jogos computacionais – são jogos que são desenvolvidos com a utilização de uma plataforma eletrônica. Existem inúmeros gêneros e formas de classificar estes jogos. Estes podem ser encontrados em diversas plataformas desde *Arcades* dedicados, passando a consoles caseiros, computadores, computadores portáteis e celulares.

3.3.1 Jogos computacionais

Em sua classificação Crawford enquadra os jogos computacionais em duas categorias distintas. A primeira é chamada de jogos de habilidade e ação onde a característica predominante é jogabilidade em tempo real. Nestes, a coordenação motora e o tempo de reação do jogador são as principais habilidades exigidas. A maioria das pessoas associa todos os jogos computacionais a esta categoria. Como exemplo desta categoria pode-se enquadrar todos os jogos de *Arcade* e quase todos os jogos do antigo console ATARI 2600.

Citaremos aqui algumas subcategorias definidas para a categoria Habilidade e Ação:

- Jogos de combate – apresentam um confronto direto. O objetivo do jogador é destruir o inimigo controlado pelo computador ou por outro jogador. O desafio está em evitar as investidas do inimigo e manter-se vivo o máximo que puder. Um exemplo desta subcategoria é a série *Quake*, onde o jogador é um soldado que deve matar soldados adversários que podem ser controlados pelo computador ou por outras pessoas.
- Jogos de labirinto – o jogador de se locomover em um labirinto e completar algum objetivo específico enquanto é perseguido por inimigos. Como exemplo pode-se citar o jogo *Pac-man*, no qual o jogador é perseguido por fantasmas ligeiramente

mais lentos que o jogador enquanto tenta percorrer todas as partes do labirinto.

- Jogos de esportes – são jogos que tentam imitar esportes. O jogador controla um competidor ou um time que deve vencer um competidor ou time adversário que pode ser controlado pela máquina ou outro jogador. Existem diversos esportes representados em jogos eletrônicos como futebol, basquete, vôlei, skate, surf, snowboard, entre outros. Pode-se citar como exemplo o *Fifa Soccer*, com o qual o jogador controla um time de futebol.
- Jogos de corrida – o jogador deve competir corridas contra o computador ou outros usuários. Nestes jogos, comumente deve-se completar um circuito no menor tempo possível. O jogador pode comandar um carro, moto, bicicleta ou uma pessoa. Um exemplo desta subcategoria é a série *Need for Speed*.

A segunda categoria chamada de Jogos de Estratégia, nos quais o raciocínio é mais exigido do que a habilidade motora. Pode-se dizer que a principal diferença entre jogos de estratégia e jogos de habilidade e ação está na ênfase na habilidade motora.

Algumas subcategorias para a categoria Jogos de Estratégia são descritas abaixo:

- Jogos de aventura – o jogador deve percorrer um mundo complexo em busca de algum objetivo, atravessando obstáculos adquirindo itens, conversando com outros personagens e resolvendo quebra cabeças. Como exemplo desta subcategoria pode-se citar o jogo *Monkey Island*, ilustrado na Figura 12.

Figura 12 – Exemplo de tela do jogo *Monkey Island*.

- Jogos de Representação (RPG do inglês *Role Playing Games*) – o jogador interage com o mundo de fantasia através de um personagem. Os mais tradicionais apresentam um cenário de fantasia medieval com aventuras em castelos e cavernas coletando uma grande quantidade de itens como espadas, armaduras, amuletos e tesouros. Os objetivos destes jogos são bastante diversificados e o enredo é muito importante. Geralmente nesta subcategoria os personagens possuem poderes

extraordinários associados a suas características. Suas características evoluem no decorrer do jogo. O diálogo e a cooperatividade entre personagens tem papéis importantes para o sucesso dos jogadores. Um exemplo destes jogos é o *World of Warcraft* no qual jogadores ao redor do mundo jogam uns com os outros através da Internet, ilustrado na Figura 13.

Figura 13 – Exemplo de tela do jogo *World of Warcraft*.

- Jogos de azar – são jogos baseados em probabilidades. O desafio do jogador é avaliar suas chances de acordo com o ambiente corrente. Nesta sub categoria existem vários jogos de baralhos.

Andrew Rollings e Ernest Adams também definiram todas estas classificações e acrescentaram mais algumas, que serão apresentadas a seguir:

- Simulações de construções e gerenciamento – são jogos que tratam de processos. O objetivo do jogo não é destruir um inimigo, mas construir algo dentro de um contexto de um processo em andamento. Quanto mais o jogador entender e controlar o processo, mais sucesso terá na construção. Como exemplo, coloca-se o jogo de maior sucesso desta categoria, o simulador de gerenciamento de cidades *Sim City* da empresa Maxis, ilustrado na Figura 14, que provou que um jogo de computador não precisa grandes doses de ação e violência para fazer sucesso.

Figura 14 – Exemplo de tela do jogo de simulação de construção de cidades *Sim City*.

- Jogos de vida artificial – Vida Artificial é uma ramificação de pesquisas em ciência da computação, assim como é a Inteligência Artificial. Vida Artificial envolve a modelagem de processos biológicos, freqüentemente usados para simular ciclos de vida de seres. O mais famoso jogo é chamado simplesmente de *Life* ilustrado na Figura 15, foi criado pelo matemático John Conway. Ele descreve um autômato celular, simulando seres que "vivem" em uma matriz. Tudo o que eles fazem é viver, reproduzir e morrer de acordo com três regras básicas. Outros exemplos de jogos de Vida Artificial é Dogs, da Nintendo, onde o jogador deve cuidar de um animal de estimação.

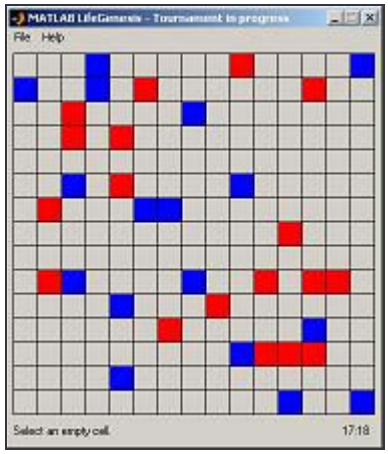


Figura 15 – Tela ilustrativa do *Jogo da Vida*.

- Jogos online – não é necessariamente uma classificação, mas uma tecnologia. Devido ao avanço das tecnologias de rede e da Internet, tornou-se viável que vários jogadores participassem de uma mesma partida em suas próprias casas. Atualmente a maioria dos jogos possui uma versão online, que oferece a possibilidade de enfrentar ou se associar a outro jogador humano ao invés de jogar apenas contra o computador. Um exemplo deste tipo de jogo, que pode ser de qualquer classificação, é o *Texas Hold`em* da empresa MindViz ilustrado na Figura 16.

Figura 16 – Exemplo de tela do jogo de Poker online *Texas Hold`Em*.

3.3.2 A Relação Jogador-Jogo em Jogos Computacionais

Na tentativa de fazer uma análise e organizar o processo de criação de jogos, Crawford traçou alguns elementos comuns a estes. Em sua pesquisa foram definidos quatro elementos fundamentais. O primeiro elemento abordado foi a Representação. Um jogo representa uma parcela da realidade. Esta realidade representada deve ser fechada (completa, não deve depender de referências do mundo externo) [CRA97]. A representação deve ter um sistema formal de regras que devem cobrir cada situação possível com que o jogador possa se deparar evitando inconsistências. Além disso, as regras de interação entre elementos do ambiente devem ser claras e bem definidas. Ao contrário dos simuladores, que buscam representar a realidade da forma mais fiel possível, em jogos a realidade é deliberadamente estilizada. Isto se dá porque seu objetivo é educacional ou o entretenimento e não uma avaliação de habilidades (como é o caso dos simuladores).

O elemento seguinte é a interação. Interação nos permite agir na realidade representada. Nela podemos observar as conseqüências de nossas ações. Isto infere ao jogo um elemento social ou interpessoal. O desafio do jogo muda através da interação diferente dos enigmas que são desafiadores até o momento em que se descobre sua solução.

Um terceiro elemento intrínseco a todos os jogos é o Conflito. O conflito surge do desafio feito ao jogador quando este tenta superar obstáculos dinâmicos ou ativos (obstáculos passivos ou estáticos são característicos de enigmas ou desafios atléticos [CRA97]) que o impedem de atingir seu objetivo. Existem vários casos em que, ao invés de o conflito estar ligado à disputa entre jogadores, estes unem-se contra um terceiro elemento. Este é o caso dos jogos massivos de representação on-line (*MMORPGs* do inglês Massive Multiplayer On-line RPG) onde os jogadores formam grupos cooperativos para enfrentar os monstros da realidade representada.

O elemento final de um jogo é a Segurança. Sendo um jogo uma representação da realidade com fins educacionais ou de entretenimento é importante salientar que a segurança é fundamental. Em um jogo podemos viver experiências fisiológicas de perigo e conflito sem sofrer as conseqüências destas ações.

Com isso define-se o que é um jogo. Jogos são representações da realidade com regras definidas que propiciam a interação entre os jogadores ou com o computador. Os jogos dificultam a tarefa do jogador de atingir seu objetivo com desafios dinâmicos que reagem a suas ações, disto surge o conflito. No jogo podemos vivenciar situações que na vida real poderiam culminar em resultados desastrosos como dano físico ou crises econômicas. Na seção seguinte estão alguns estudos de caso de jogos condizentes com as descrições feitas nesta sessão e na anterior.

3.3.3 A Estrutura Interna de um Jogo

Os jogos computacionais apresentam alguns elementos internos comuns. Estes elementos variam muito para cada jogo, mas, de uma forma geral, podem ser divididos em seis grupos: Modelagem do Universo, Física do Jogo, Inteligência Artificial (I.A.), Acesso a Periféricos, Seqüenciamento do Jogo e Artes Gráfica e Sonora. Nesta fase do Trabalho de Conclusão, não foi encontrada documentação que definisse objetivamente estes elementos. Esta classificação é proposta aqui, sendo parte dela baseada em descrições sobre equipes de desenvolvimentos de jogos tais como aparecem em [BET03] e [ROL00].

A modelagem do universo é uma parte fundamental para a criação de um jogo. Nela são definidas as estruturas que representarão logicamente o universo. É nesta parte que são determinados o sistema de coordenadas para localização de recursos do jogo dentro do ambiente do jogo. Este mesmo sistema pode ser usado para localizar os jogadores ou suas imagens no universo.

A Física, por sua vez, é a parte do programa onde são estabelecidas funções de interação física entre os objetos. A Física é responsável por simular leis da física semelhante ou de forma distintas do que ocorre no mundo real. Dependendo do enredo do jogo estas leis são deliberadamente alteradas ou ignoradas. Neste momento, são definidas regras como a ação da gravidade nos objetos, detecção de colisão, trajetória de projéteis, entre outros. O jogo *Penumbra*, por exemplo, tem por objetivo explorar o uso de uma elaborada física no ambiente de jogos [FRI06].

A Inteligência Artificial é um conjunto de algoritmos que regem o comportamento dos adversários ou personagens controlados pelo equipamento eletrônico onde o jogo está sendo executado. Além de controlar o comportamento dos personagens a Inteligência Artificial pode inferir uma característica de adaptabilidade aos antagonistas. Alguns algoritmos permitem que os antagonistas mudem seu comportamento conforme o estilo do jogador. O jogo *SiN Episodes* publicado pela Ritual Entertainment apresenta uma Inteligência Artificial adaptativa, ou seja ela

adapta-se as habilidades do jogador tornando-se mais difícil se o jogador for habilidoso ou mais fácil caso contrário.

Além dos componentes que definem o universo e que permitam a criação de um universo consistente é também importante ressaltar a parte de comunicação com o usuário. O sistema deve oferecer ao jogador um meio para que este último possa expressar sua vontade e possa interagir com o jogo. O Acesso a Periféricos deve definir um conjunto de programas ou programa que viabilize esta interação. Este é responsável por estabelecer a comunicação entre jogador e o jogo através da troca de informações. Estes periféricos recebem comandos do usuário que influenciarão o ambiente de jogo e as conseqüências destas influências deve ser informada de alguma forma ao jogador (geralmente através de imagens de vídeo).

O Seqüenciamento do Jogo define características como estória, enredo, missões, fases e seqüência de níveis com os quais o jogador irá se deparar. No Seqüenciamento dos Jogos também são definidos os diálogos dos personagens e eventos que irão ocorrer no desenvolvimento da estória.

A Arte Gráfica e Sonora define os elementos visuais, efeitos sonoros e músicas do jogo. A Arte Gráfica é responsável pela criação da identidade visual do jogo. Nela estão definidos os menus, imagens dos personagens (sejam desenhos em duas dimensões ou modelos tridimensionais), efeitos de partículas, reflexos e luminosidade, em fim, tudo que será visualizado. A parte sonora é responsável por todos efeitos sonoros desde falas a sons do ambiente e músicas até distorções sofridas pelo ambiente.

Estes componentes podem estar associados á *engine* do jogo explicada na Seção 3.3.4. Isto, porém, não é uma regra. Geralmente encontramos na *engine* a modelagem do universo e regras de física, mas esta pode englobar outros elementos.

3.3.4 A Engine de um Jogo Computacional

O principal componente de um jogo é a *engine*. A *engine* é literalmente o motor do jogo. Ela dita as regras de como os objetos serão gerados na tela, faz o controle da física associada e é responsável por outras funções importantes. A *engine* pode ser dividida em alguns elementos básicos:

- *Rendering*: é o componente responsável por gerar as imagens que serão mostradas ao usuário através de um dispositivo gráfico de saída. É uma parte muito importante do jogo, pois é o principal meio pelo qual o jogador recebe informações do mesmo.
- Interface com o usuário: através dos dispositivos de entrada e saída o jogador pode

informar ao sistema ações que deseja executar, e receber informações por parte do jogo, sejam visuais, sonoras ou outras.

- Física: é responsável pelo comportamento do jogo e como os personagens e objetos interagem com o ambiente.
- Inteligência artificial: é responsável pelas ações dos elementos que não estão sob o controle direto do jogador como, por exemplo, os personagens "não jogáveis" (do inglês *Non-playable Characters* - NPCs). Juntamente com o *rendering*, esse item é responsável por "dar vida" ao jogo e determinar o seu sucesso ou fracasso comercial.
- Sistema de áudio: responsável pelos efeitos sonoros e músicas que serão utilizados no decorrer do jogo.

Existem empresas especializadas na criação de *engines* como a *Havok*, que produziu a *engine Havok* e *Havok 2*, utilizada em vários jogos populares como *Half-Life 2*, *Age of Empires* e *Medal of Honor*.

Vários jogos diferentes possuem a mesma *engine*, o que os diferencia é a arte gráfica, os diferentes tipos de efeitos sonoros, a história e o estilo.

As empresas interessadas em produzir um jogo podem produzir também a *engine* ou comprá-la. Se for comprada, a empresa poderá concentrar-se apenas em produzir o jogo em si (personagens, enredo, arte gráfica e sons, *scripts* de inteligência artificial), como os objetos serão mostrados na tela, como a inteligência artificial executa os *scripts*, como a física age são tarefas da *engine*.

3.3.5 A Equipe de Implementação de Jogos

Jogos computacionais modernos de alto desempenho são produtos complexos, desenvolvidos por equipes de pessoas. Dada a complexidade do projeto e implementação destes produtos, existem várias etapas de desenvolvimento de um jogo. Conforme definido em [BET03] existem quatro equipes essenciais no desenvolvimento destes jogos: (*i*) a equipe de projeto principal responsável pela concepção do jogo. Esta equipe é comandada pelo projetista principal do jogo. Este é como o diretor de um filme, define os conceitos do jogo e decide o que será incluído ou cortado da versão final; (*ii*) a equipe da *mecânica do jogo* é responsável pela parte de programação necessária ao desenvolvimento do jogo. As pessoas envolvidas nesta etapa são responsáveis por fazer toda a programação do jogo e de programas auxiliares como, por exemplo, editores de cenários. Elas desenvolvem a *engine,* a inteligência artificial, os gráficos, a física e outros componentes. (*iii*) a equipe de *desenvolvimento de missões e níveis* é responsável por criar os estágios do jogo e elaborar as missões que deverão ser cumpridas pelos jogadores. (*iv*) a equipe de *escritores de estórias e*

diálogos é responsável por desenvolver a estória e enredo do jogo. Também é tarefa desta criar os diálogos de interação entre os personagens.

Uma tarefa importante na confecção de jogos, é a modelagem do universo do jogo. É comum que esta tarefa seja responsabilidade dos programadores da mecânica apesar de ser uma tarefa com características próprias. O programador que modela o universo deve ter certo domínio de vetores, matrizes, trigonometria, cálculo e álgebra. Este é responsável por definir regras de trajetória de projéteis no ambiente, por exemplo. Outro exemplo é o cálculo de colisão de objetos, estes partes da física do jogo. Assim a Inteligência Artificial é importante em jogos. Através desta os antagonistas dos jogos terão reações convincentes às ações do jogador. Com uma boa inteligência artificial é possível oferecer um desafio interessante aos jogadores.

A interface entre jogo e jogador é feita pelo programador de interfaces. Ele é responsável por desenvolver todo o sistema de menus de navegação e painéis de informação no jogo como barras de nível de energia, munição, lista de itens ou outras informações importantes ao jogador.

Os sons, música e diálogos do jogo são responsabilidades do programador de áudio. Este deve tornar possível a utilização de falas, músicas de fundo e efeitos sonoros pelos programadores da *mecânica do jogo*. Também é responsabilidade dele a programação de distorções e efeitos que os sons sofrem em relação ao ambiente.

Para facilitar o processo de criação de jogos são definidas ferramentas auxiliares. Estas ferramentas não são diretamente partes integrantes dos jogos. Elas são usadas para inúmeras tarefas como criação de níveis, edição de som, aplicação de texturas, modelagem 3D de personagens, aplicação de animações e inúmeras outras necessidades dependendo do projeto do jogo. Apesar de não ser uma parte notória para a maioria dos consumidores de jogos computacionais, várias grandes empresas de desenvolvimento de jogos mantêm setores especializados para esta função. Mais de 50% dos programadores da Id Software (responsável por títulos de sucesso como *Quake* e *Doom*) estão envolvidos na tarefa de desenvolvimento de ferramentas auxiliares [BET03].

3.3.6 Estudos de Caso

Os jogos denominados RPGs (Role-Playing Games ou Jogos de Interpretação de Papéis) surgiram na década de 70 como jogos de tabuleiro, sendo o exemplo mais conhecido nessa categoria o *Dungeons and Dragons*, hoje comercializado pela empresa Wizards® ou o *Diablo*, comercializado pela empresa Blizzard North.

Num RPG, o jogador incorpora um personagem em uma história de fantasia que vai se desenvolvendo gradativamente. Normalmente, o jogador possui diversas escolhas quando da criação do seu personagem. Essas escolhas refletem no perfil que o personagem terá durante o jogo. Esse perfil é constituído de diversos atributos (tais como raça, classe, aparência, armas

preferenciais, feitiços, etc.) e outras características que evoluem à medida que o jogador vai avançando no jogo (tais como força, agilidade, destreza, inteligência, sabedoria, carisma, nível de experiência do jogador, etc). Os atributos influenciam os resultados das batalhas, ações e objetivos que o jogador irá enfrentar no decorrer do jogo.

A experiência é um fator vital em um RPG, pois o jogador normalmente inicia com uma quantidade baixa ou nula desse atributo. À medida que vai derrotando inimigos e alcançando metas, o valor de sua experiência vai aumentando, o que torna possível o jogador aumentar o seu nível e, conseqüentemente, aumentar seus atributos, tornando-se cada vez mais evoluído. A figura 1 ilustra alguns aspectos do desenvolvimento de um RPG geralmente discutidos durante o desenvolvimento desse tipo de jogo [KOR05].

Diablo é um RPG para PC desenvolvido pela empresa Blizzard North. *Diablo* retrata a matança de hordas de monstros, tais como esqueletos vagando em masmorras e calabouços, coletando ouro e itens mágicos até liquidar o chefe final, o mal supremo. O conceito chave por trás de *Diablo* é fazer com que a interface seja a prioridade número um, não a história, nem o tamanho do jogo, não o número de diferentes personagens, tampouco uma mecânica de RPG muito complexa. A Figura 17 mostra uma cena do jogo.

Figura 17 – Exemplo de cena do jogo *Diablo*.

Neste jogo existe um amplo sistema de itens, magias, evolução do personagem e antagonistas. A *engine* gerencia um vasto número de relações entre os elementos citados para determinar o resultado das ações dos personagens. Neste jogo, como explicado anteriormente, o foco não está na interação entre personagens, e ele não possui um mundo vasto, características marcantes em um RPG. Mesmo assim, *Diablo* é classificado como um RPG, dado que possui alguns aspectos principais desta categoria, como evolução de personagens, uma ficha típica do estilo mostrando os atributos do personagem, grande número de itens e magias a serem coletados e usados, além de possuir ambiente e personagens que são típicos de RPGs (guerreiros ou feiticeiros em um calabouço

ou masmorra medieval). A cada novo jogo é apresentado ao jogador um conjunto de labirintos gerados randomicamente. Estes labirintos apresentam adicionalmente um conjunto de missões menores também selecionadas aleatoriamente.

Jogos de tabuleiro dispõem como meios o uso de peças e uma base ou suporte divididos em regiões. Por exemplo, no caso do jogo de xadrez existem dois conjuntos de 16 peças (8 peões, 2 torres, 2 cavalos, 2 bispos, uma rainha e um rei). Cada tipo de peça movimenta-se de maneira única sobre o tabuleiro dividido em 64 casas (8x8). O objetivo do jogo é encurralar o rei adversário.

Existe uma grande quantidade de implementações de jogos de xadrez. Dentre estes, pode-se citar a série *Chess Master* da Ubisoftware, um exemplo de tela é mostrado na Figura 18.

Figura 18 – Exemplo de tela do jogo *Chess Master 10th*.

4 Arquitetura Alvo

Este capítulo apresenta a arquitetura desenvolvida neste trabalho. A Seção 4.1 apresenta a arquitetura de hardware da plataforma desenvolvida. A Seção **Erro! Fonte de referência não encontrada.** apresenta a arquitetura de software e faze definições sobre características de jogos, relações entre jogos e jogadores e cita as equipes envolvidas na produção de jogos.

4.1 Arquitetura de Hardware

No início deste trabalho foi proposta uma arquitetura de hardware. Esta arquitetura apresenta um conjunto de módulos interligados por uma rede de conexão visando prover recursos para a implementação de um jogo de xadrez. Foi definido que a plataforma deveria ser composta por dois processadores Plasma, um módulo de vídeo e um módulo que permitisse a entrada de dados possibilitando a interação com o usuário. A Figura 19 apresenta uma ilustração da arquitetura definida na proposta.

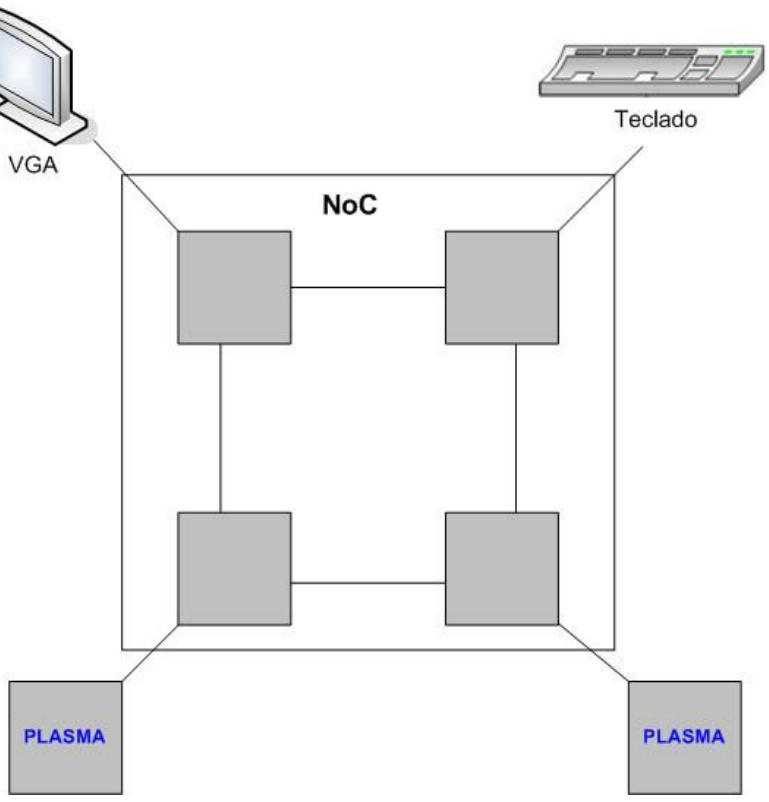


Figura 19 – Diagrama da plataforma inicialmente proposta.

No decorrer do desenvolvimento do projeto foi percebido que o formato da arquitetura proposta deveria sofrer adaptações. Estas adaptações foram definidas para atender novas necessidades que, até então, não haviam sido levantadas.

Em alto nível, a arquitetura primeiramente sugerida atende perfeitamente as necessidades do jogo visto que esta possui duas unidades de processamento, um sistema de entrada de dados e um sistema de geração de imagens. Estes componentes comunicam-se através da rede de interconexão Noc Hermes [MEL03] que já foi utilizada em outros projetos no GAPH.

A arquitetura original foi idealizada para ser executada em uma plataforma de prototipação. Até o presente momento não foi possível fazer a prototipação da implementação deste projeto apesar das simulações feitas em software. Isto se deve ao fato de que alguns dos módulos integrantes da plataforma implementados neste trabalho não foram ainda testados em uma plataforma de prototipação. Além disso, módulos desenvolvidos em trabalhos paralelos a este e dos quais este trabalho depende ainda não foram prototipados. A tarefa de testar e prototipar estes módulos e posteriormente prototipar a arquitetura integralmente demandou um esforço que não foi possível de executar em tempo hábil à entrega deste trabalho.

4.2 Noc Hermes

Uma forma de resolver alguns dos problemas inerentes a barramentos é a utilização de redes intra-chip (em inglês Networks on Chip, ou NoCs). NoCs são redes internas a um circuito integrado e tiram proveito de conceitos já largamente utilizados em telecomunicações e sistemas distribuídos, como a transferência de informações através do uso de camadas de protocolos, para implementar arquiteturas de comunicação mais escaláveis e modulares que barramentos.

De uma forma geral, NoCs podem ser representadas por grafos compostos por dois tipos de nodos: de processamento e de chaveamento. Nodos de processamento são dispositivos quaisquer conectados à NoC, tais como microprocessadores, USB, memórias, entre outros. Os nodos de chaveamento, por sua vez, são responsáveis por transportar informações entre nodos de processamento, conforme exemplificado na Figura 20. Os nodos de chaveamento possuem filas (do inglês, FIFO – First In First Out) nas entradas, nas saídas, ou centralizadas. Estas filas são espaços de memória preenchidos por informações em tráfego de um nodo para outro, quando necessário. Denominam-se estas filas usualmente de *buffers*.

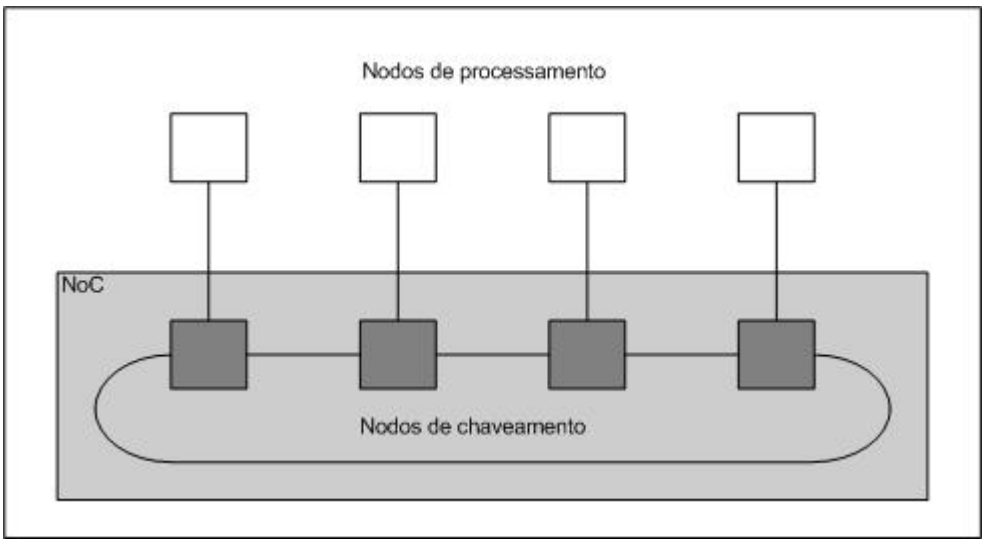


Figura 20 - Exemplo de estrutura de um sistema com comunicação baseada em NoCs.

Cada ligação entre os nodos (de roteamento/processamento e roteamento/ roteamento) é denominado enlace (em inglês *link*). A Figura 21 mostra algumas topologias que podem ser usadas para interconectar os nodos de chaveamento de uma NoC.

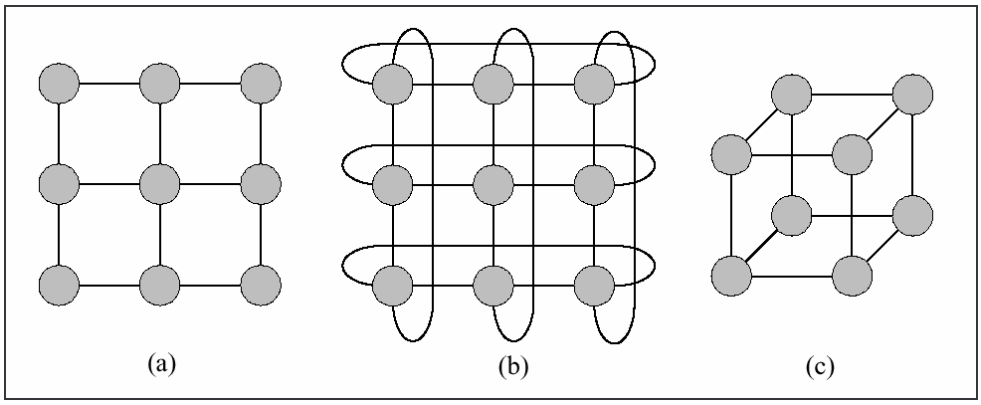


Figura 21 – Exemplo de algumas topologias para uso em NoCs. - (a) Malha 2D 3x3; (b) Toro 2D 3x3; (c) Hipercubo 3D.

Sistemas muliprocessados em chip (em inglês *Multi Processor Sistems on Chip* - MPSoCs) estão se tornando cada vez mais complexos, incorporando um número cada vez maior de processadores e, portanto, necessitando de muito mais velocidade de transferência de dados. A decisão mais lógica a se tomar, portanto, seria pensar em maneiras mais eficientes para conectá-los do que barramentos, como por exemplo, NoCs.

Barramentos são eficientes mas possuem limitações. À medida que a complexidade de SoCs e que a demanda por mais velocidade de transmissão de dados aumenta, é necessário que o gargalo provido pelo uso dos barramentos seja superado.

NoCs permitem concorrência na comunicação, maior nível de abstração (com a utilização de módulos de encapsulamento denominados *wrappers*), modularidade e mais escalabilidade. Porém ocupam mais espaço em um chip.

Analisando economicamente, a substituição de barramentos em SoCs por NoCs deverá produzir as seguintes vantagens [ART05]:

- Aumentar o desempenho do SoC.
- Reduzir o "time to market" do SoC.
- Reduzir o risco de projeto do SoC.

4.2.1 Roteador

As NoCs são compostas por diversos nodos denominados roteadores ou chaves de roteamento. Estas chaves têm a função de receber mensagens e repassá-las a uma roteador adjacente até que esta atinja seu destino.

As chaves de roteamento são compostas por portas de comunicação que ligam uma chave a outra ou a um módulo de processamento. Na rede utilizada neste trabalho, cada roteador é composto por cinco portas: East, West, North, South e Local. A porta local destina-se a conectar um nodo de processamento à NoC, as demais conectam os roteadores entre si. Cada porta possui uma fila para armazenar os *flits*. A Figura 22 mostra a ilustração de uma chave de roteamento.

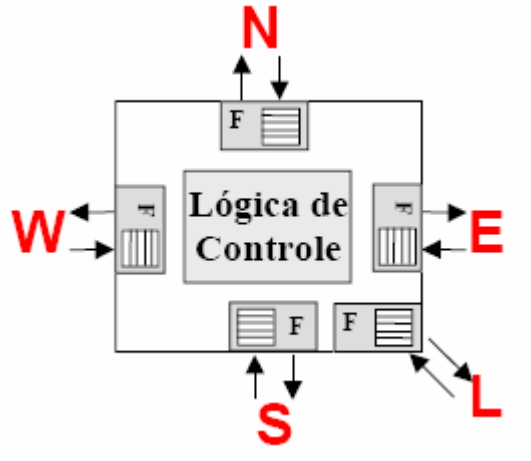


Figura 22 – Chave de roteamento da NoC [MEL03].

Cada chave possui um endereço XY. Este endereço indica a posição da chave na rede de interconexão, onde X é a posição horizontal e Y a posição vertical. Este endereço é utilizado para direcionar os pacotes das mensagens pela rede.

Ao receber um novo pacote cada uma das filas faz uma requisição de chaveamento ao árbitro da chave. O árbitro então escolhe a requisição de maior prioridade, caso ocorram requisições simultâneas, e encaminha a requisição selecionada à lógica de chaveamento. É verificada a possibilidade de chaveamento desta requisição. Sendo possível é feito o chaveamento e o árbitro é informado. O árbitro informa à fila que gerou a requisição que o chaveamento foi feito e esta começa a transmitir os dados.

Maiores informações sobre as chaves de roteamento, lógica de controle, regras de arbitragem e outras questões inerentes a NoC Hermes são encontradas em [MEL03].

4.2.2 Interface Externa

A interface de comunicação entre os nodos de processamento com a NoC é feita através da porta Local do roteador. Esta porta Local apresenta um canal de entrada e um de saída como mencionado na seção 4.2.1.

A rede utilizada neste trabalho apresenta oito sinais para comunicação com os núcleos de processamento, quatro para o canal de entrada e quatro para o canal de saída. A Figura 23 mostra ilustra uma interface de comunicação com um núcleo de processamento, em seguida é apresentada um explanação sobre cada sinal.

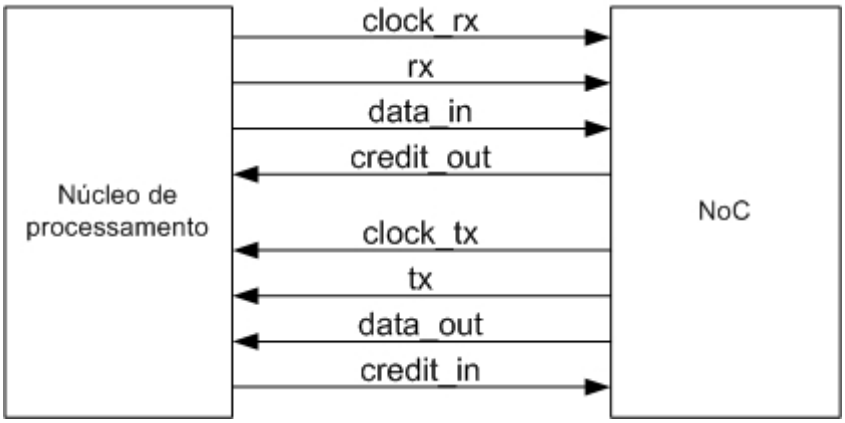


Figura 23 – Sinais da interface de comunicação entre um núcleo de processamento e um porta Local da NoC.

Nesta interface os sinais *clock_rx* e *clock_tx* têm a função de manter a sincronia entre os dois módulos para a transmissão dos pacotes de uma mensagem. O sinal *clock_rx* faz a sincronia quando o núcleo de processamento envia pacotes para a NoC enquanto o sinal *clock_tx* faz a sincronia no fluxo de dados com sentido oposto. Os sinais *data_in* e *data_out* carregam os dados da transmissão. Em cada interação da comunicação estes sinais repassam um *flit* de dados para o nodo adjacente. De forma análoga aos primeiros dois sinais descritos anteriormente o sinal *data_in* transmite os dados do núcleo de processamento para a NoC enquanto sua contraparte se encarrega da transmissão de dados no sentido oposto. Os sinais *rx* e *tx* por sua vez indicam quando há dados validos nos sinais *data_in* e *data_out* respectivamente. Por fim, os sinais *credit_out* e *credit_in* informam ao módulo adjacente quando este está apto a receber dados. O sinal *credit_out*, no caso da NoC, informa ao núcleo de processamento quando há espaço disponível na fila da porta Local para a recepção de dados. A Figura 24 mostra a recepção de dados de um núcleo de processamento em uma porta local da NoC. Em seguida é feita a explicação do processo de transmissão de dados do exemplo mostrado.

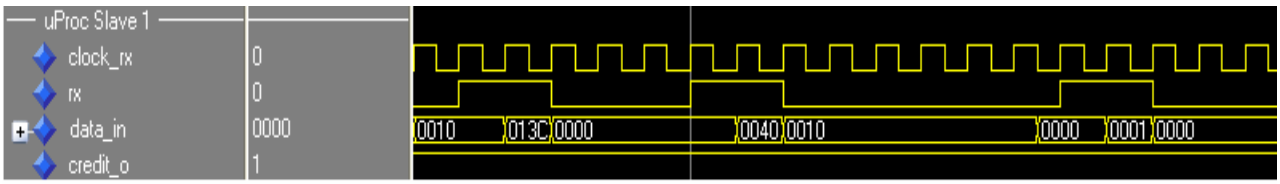


Figura 24 – Formas de onda extraídas da simulação de transmissão de dados em uma porta local.

Inicialmente é verificado o sinal *credit_out* está em nível lógico alto, o que indica que a porta está apta a receber dados. Em seguida os dados são recebidos através do sinal *data_in* a cada pulso de subida do sinal *clock_rx* sempre que o sinal *rx* está em nível lógico alto.

4.3 Processador Plasma

O processador MIPS, inicialmente proposto no projeto MIPS em 1984 liderado por John Hennessy na Universidade de Stanford, assim como o SPARC, derivado do projeto RISC liderado por David Patterson da Universidade de Berkeley, utilizam arquitetura RISC (*Reduced Instruction Set Computer*) [SWE99]. A grande maioria dos processadores RISC referenciam estes dois projetos seminais [PAT96].

Escolheu-se neste trabalho usar o processador PLASMA. Este processador é uma implementação baseada na arquitetura MIPS, atende as necessidades deste trabalho e seu código é aberto. Sendo assim, a escolha do processador PLASMA é uma opção interessante. Nesta Seção será mostrado um breve histórico do desenvolvimento da arquitetura MIPS. Também será descrito p funcionamento desta e, posteriormente serão feitas comparações entre a arquitetura MIPS e a organização do processador Plasma.

4.3.1 Histórico

O processador MIPS original foi o R2000, que operava com um processador de ponto flutuante associado, o R2010. Na geração seguinte foi incorporado um *pipeline* de cinco estágios. Os processadores são de 32 bits. Na mesma geração do R3000 é criado o R3500 que combina o R3000 e o R3010 (processador de ponto flutuante) em um chip, de maneira análoga ao i486 da Intel que já possuía um processador de ponto flutuante.

Tradicionalmente, os microprocessadores MIPS utilizam um *pipeline* de cinco estágios com exceção da família R4000. Esta família também introduziu uma arquitetura de 64 bits. Variações do R4000 foram usadas na produção de PDAs e *video games* pela NEC [PAT96].

A escolha do processador MIPS para uso neste trabalho foi baseada nos seguintes fatores: *(i)* a familiaridade do grupo com a arquitetura e conjunto de instruções; *(ii)* a existência da implementação PLASMA que é, para todos os efeitos, um processador MIPS I com algumas

restrições, descritas na Seção 4.3.3; *(iii)* a existência de projetos paralelos usando o PLASMA em andamento no GAPH [WOS05] [CAR06]; *(iv)* o MIPS é um processador muito usado em sistemas embarcados, em particular em consoles de jogos, como o Playstation e Nintendo64, entre outros.

4.3.2 Arquitetura MIPS

Um processador MIPS é composto por uma unidade de processamento de inteiros e por co-processadores auxiliares. Além do processador principal, que executa operações sobre inteiros, um processador MIPS possui um co-processador que executa operações de ponto flutuante, chamado de *co-processador1*, e um co-processador para tratamento de interrupções e exceções, o *co-processador 0*. A Figura 25 apresenta um diagrama de blocos simplificado da arquitetura MIPS [PAT96].

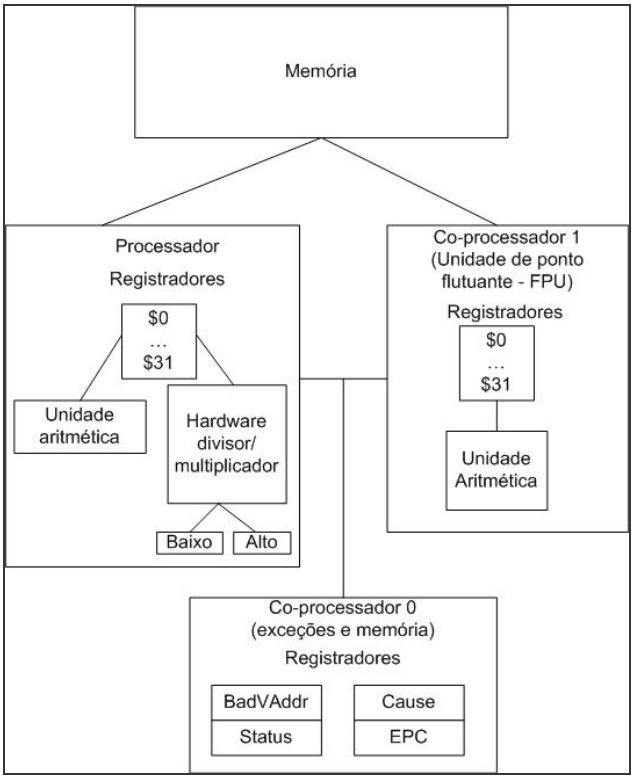


Figura 25 – Diagrama de blocos da arquitetura MIPS I [PAT96].

A arquitetura MIPS disponibiliza 32 registradores (de **$0** ao **$31**) de 32 bits (exceção à família R4000, que tem registradores de 64 bits) para uso geral. O registrador **$0** sempre contém o valor zero, o registrador **$31** pode armazenar o endereço de retorno de uma subrotina. Com exceção de **$0**, todos os demais são de uso geral. Apesar de se poder usar qualquer dos registradores de uso geral da mesma forma, existem convenções de uso e nomes que serão mostradas a seguir.

Os registradores **$1** (**$at**), **$26** (**$k0**) e **$27** (**$k1**) são reservados para o montador e para o sistema operacional [SWE99]. Os registradores **$2** e **$3**, também denominados **$v0** e **$v1**, respectivamente, são utilizados para retorno de valores para funções. A passagem de parâmetros

para funções é feita através dos registradores **$4** ao **$7** (**$a0** ao **$a3**) e da pilha de dados. Os registradores **$8** ao **$15**, **$24** e **$25** (**$t0** ao **$t9**) são usados para dados temporários que não necessitam ser preservados entre chamadas de subrotinas (chamados de *caller-saved*). Os registradores **$16** ao **$23** (**$s0** ao **$s7**) são usados para dados que necessitam serem armazenados durante uma subrotina e recuperados ao final da mesma (denominados *callee-saved*). O registrador **$29** (**$sp**) pode ser usado como ponteiro de pilha (*stack pointer* em inglês). Ele está sempre apontando para o topo da pilha. O registrador **$30** (**$fp**) é o ponteiro de *frame*. Pode ser utilizado como registrador *callee-saved*. O registrador **$28** (**$gp**) é um ponteiro global utilizado para apontar para um bloco de memória de 64Kbytes na estrutura que armazena constantes e variáveis globais. O endereço de retorno é armazenado no registrador **$31** (**$ra**) após a execução de uma instrução *jal*.

Processador MIPS pode operar tanto no esquema *little-endian* como no esquema *big-endian*.

A arquitetura MIPS é do tipo load/*store*, ou seja, as instruções de acesso à memória apenas realizam uma leitura ou uma escrita na memória, sem operar sobre os dados manipulados. As instruções lógicas e aritiméticas ou de modificação de dados são feitas sempre sobre registradores. O endereçamento de dados na memória é sempre feito do mesmo modo. O endereço resultante para leitura ou escrita é sempre obtido da soma do conteúdo de um registrador com um deslocamento. Este deslocamento costuma ser um valor de 16 bits em complemento de 2 (entre –32768 a 32767). O registrador utilizado como operando nestas instruções serve como base para o cálculo de endereço.

As operações de *load* e *store* podem armazenar de 1 a 8 bytes de dados por vez. O tipo do dado a ser armazenado determina o espaço de memória a ser ocupado. A Tabela 1, ilustra os tipos possíveis, tamanhos e instruções para acessá-los.

Tabela 1 - Tipos de dados em linguagem C, tamanhos em bytes e mnemônicos do montador.

Tipo em C	Tamanho em bytes	Mnemônico do *assembler*
long long	8	“d”
int	4	“w”
long	4	“w”
short	2	“h”
char	1	“b”

As operações disponibilizadas para o programador em linguagem de montagem da arquitetura MIPS podem ser divididas em duas classes. *Pseudo-instruções* são operações não executáveis diretamente pelo hardware do processador, sendo convertidas para uma seqüência de instruções

efetivamente implementadas em hardware. Alguns exemplos de *pseudo-instruções* são: *bgt*, *blt*, *bge* e *ble*.

As instruções propriamente ditas são aquelas executadas diretamente pelo hardware do processador. Estas podem ser divididas em três formatos como mostrado na Tabela 2 onde *opcode* é o código da instrução, *rs* é o registrador de origem, *rt* o registrador alvo e *rd* o registrador destino. *Função* é a função a ser executada, *shamt* é a quantidade de bits a serem deslocados em operações de deslocamento de bits (*shift* em inglês), *imediato* é o valor a ser utilizado pela instrução e *endereço* a ser acessado na memória de dados.

Tabela 2 - Tipos das instruções da arquitetura MIPS.

Tipo	**Bits 31 a 0 – formato (bits)**					
R	*opcode* (6)	*rs* (5)	*rt* (5)	*rd* (5)	*shamt* (5)	*função* (6)
I	*opcode* (6)	*rs* (5)	*rt* (5)	*imediato* (16)		
J	*opcode* (6)	*endereço* (26)				

4.3.3 Organização do Plasma

A CPU do Plasma suporta todas as instruções do MIPS I ™ do modo de usuário com duas exceções. Operações de *load* e *store* não alinhadas não são suportadas, devido ao fato de estas serem patenteadas. Exceções também não são suportadas.

Esta CPU foi implementada em VHDL com um *pipeline* de três estágios. A Figura 26 ilustra o diagrama de blocos do Plasma, conforme descrito em [OPE06].

A arquitetura MIPS utiliza duas memórias distintas com interfaces também distintas, uma para dados e outra para instruções. Estudos do código do processador Plasma mostram que este utiliza apenas uma memória, ou seja, a arquitetura original MIPS é uma arquitetura Harvard enquanto que a organização do Plasma, apesar de ser baseada no MIPS, é uma arquitetura Von Neumann. Estas são as principais diferenças entre a arquitetura MIPS e a implementação do Plasma. A Tabela 3 mostra as principais diferenças entre a arquitetura MIPS e o processador Plasma detectadas até o momento.

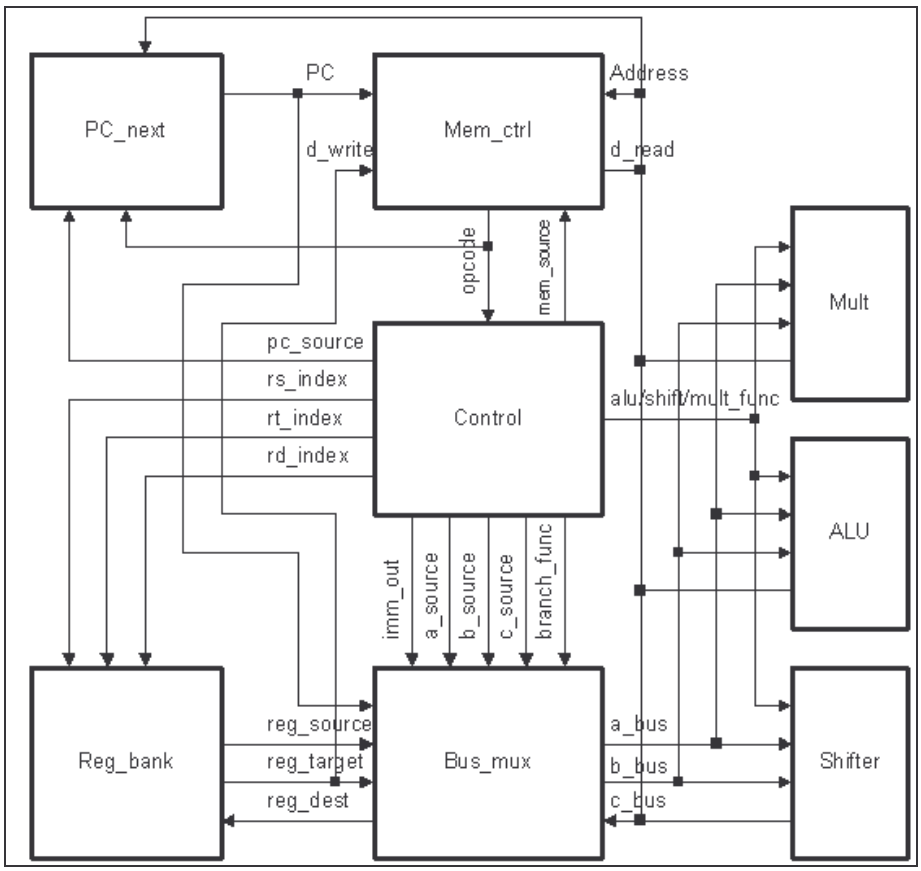


Figura 26 – Diagrama de blocos do processador Plasma.

Tabela 3 - Quadro comparativo de diferenças entre a arquitetura MIPS e o processador Plasma.

Arquitetura MIPS I	Implementação Plasma
Organização Harvard	Organização Von Neuman
Pipeline de 5 estágios ou mais	Pipeline de 3 estágios
Com suporte a exceções	Sem suporte a exceções
Acesso à memória de forma desalinhada possível (LWL, LWR, SWL, SWR)	Sem suporte a acessos desalinhados à memória (LWL, LWR, SWL, SWR são patenteadas)

4.3.4 Modificações na arquitetura original do Plasma

A plataforma utilizada neste trabalho é a plataforma desenvolvida por [WOS05] acrescida de módulo de teclado e módulo de vídeo. Esta plataforma faz uso do processador Plasma a modificado de forma a poder executar múltiplas tarefas em uma mesma CPU através de um mecanismo de paginação [WOS05]. Estas modificações dizem respeito à criação de novos mecanismos, exclusão de entidades e registradores mapeados em memória, bem como inclusão de novas entidades e novos registradores mapeados em memória. A Figura 27 mostra o processador Plasma alterado com novas entidades e novos registradores.

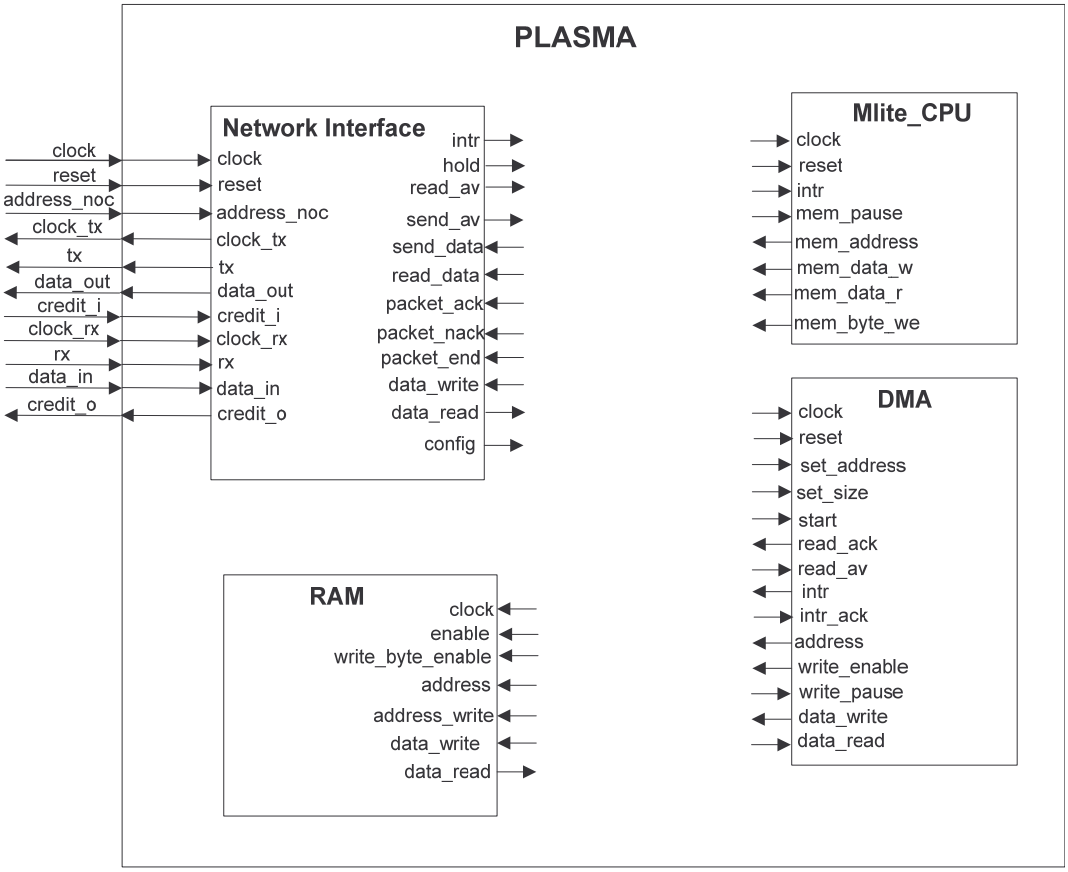


Figura 27 - Processador Plasma com novas entidades e novos registradores mapeados em memória.

4.3.5 Compilador MIPS-GCC

Compilador de linguagem C que gera código objeto para o processador MIPS. Este compilador não possui bibliotecas que dependem de sistema operacional como no caso do compilador GCC para a plataforma x86, por exemplo. Algumas das bibliotecas que não estão presentes neste compilador são: *stdlib*, *stdio*, *pthreads* entre outras.

4.4 Módulo de vídeo

O sistema de vídeo tem por finalidade exibir na tela as informações gráficas do jogo. Este sistema é composto pela interface VGA e pelo controlador de acesso à memória do tipo DDR. Este será gerenciado por um módulo que acessa os dados da memória DDR através de seu controlador e exibe-os na tela através da interface VGA. A Figura 28 mostra o diagrama de blocos do módulo que foi desenvolvido.

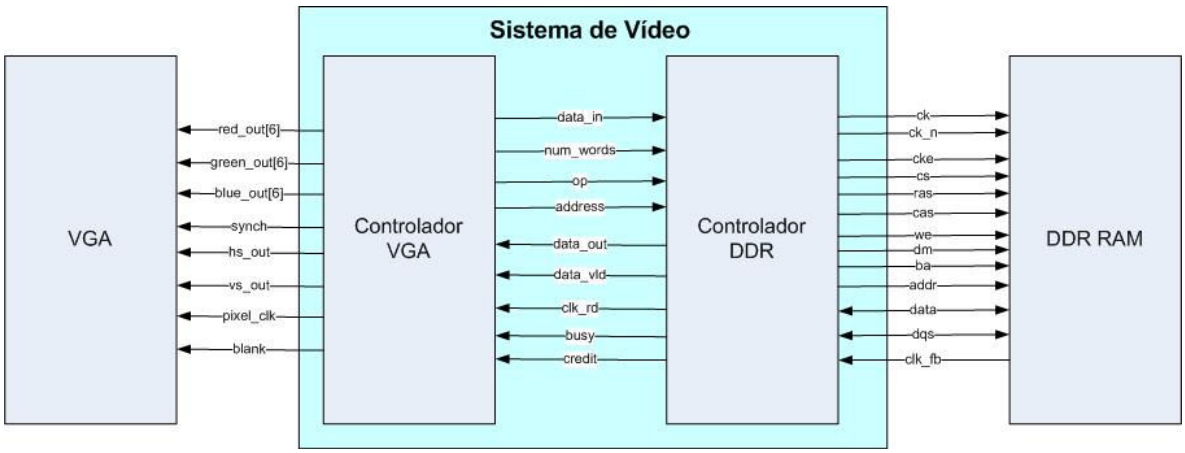


Figura 28 – Diagrama de blocos do sistema de vídeo conectado as interfaces de vídeo (VGA) e memória RAM (DDR RAM) da plataforma Xilinx ML-401.

Existem três interfaces distintas no controlador de VGA: (*i*) interface com *wrapper* da controladora VGA; (*ii*) interface com a saída VGA, e; (*iii*) interface com a memória DDR. A interface com o *wrapper* de VGA é composta por quatro sinais que possibilitam a recepção de dados que serão utilizados para compor as imagens no monitor VGA conectado ao console. Estes sinais são descritos a seguir:

- Tx: destina-se a sinalizar ao controlador de vídeo que existe uma seqüência de dados a serem escritos na memória de vídeo;
- Ack_tx: informa ao *wrapper* de vídeo que a primeira palavra foi escrita e que as demais palavras podem ser transmitidas na vazão de uma palavra por ciclo de relógio;
- Data_in_wrapper: sinal de 32 bits que recebe as palavras transmitidas pelo *wrapper*.
- Address_in_wrapper: endereço da memória de vídeo a partir do qual as palavras transmitidas pelo *wrapper* devem ser gravadas.

A interface com a saída de vídeo VGA é composta por um conjunto de seis sinais. Estes sinais geram informações sobre sincronismo de desenho vertical e horizontal, períodos de branco (em inglês *blank*) e composição das cores de cada pixel. A seguir são apresentados todos os sinais que compõem esta interface:

- Rgb – conjunto de sinais que definem a cor de cada ponto no monitor;
- Hs_out – sinal de sincronização horizontal do canhão de elétrons;
- Vs_out – sinal de sincronização vertical do canhão de elétrons;
- Pixel_clock – sinal de relógio para sincronização de impressão dos pixels;
- Blank – quando está em nível lógico baixo, o sinal rgb deve estar desativado para o retorno do canhão de elétrons;

A interface de comunicação com a memória DDR é composta por 13 sinais conectados à memória DDR que é utilizada como memória de vídeo. A descrição destes sinais não será abordada neste trabalho visto que isto foge do escopo do presente trabalho. Na Figura 29 é apresentada uma ilustração do controlador VGA com os sinais necessários para a comunicação com seu *wrapper*, memória DDR e interface VGA.

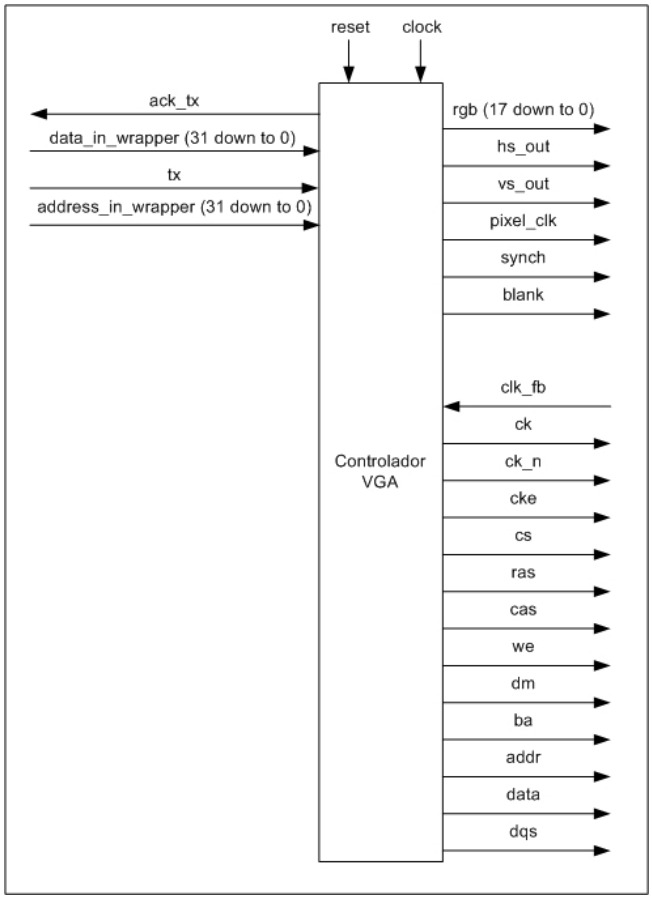


Figura 29 – Controlador de VGA seus sinais.

Neste trabalho a área visível da tela é composta por 640 pixels de largura por 480 pixels de altura. Estas dimensões foram escolhidas por serem um padrão muito utilizado e se adequa as necessidades do projeto.

Optamos por utilizar pixels de 8 bits, sendo os três bits menos significativos destinados a cor vermelha os três seguintes a cor verde e os dois mais significativos para a cor azul. Nesta configuração é possível gerar 256 cores, que possibilitam apresentar o ambiente gráfico do jogo de forma clara.

Sendo a tela composta por 640 pixels de largura e cada pixel sendo composto por 8 bits, cada linha ocupa 160 palavras de 32 bits da memória. Assim, uma tela inteira, que possui 480 linhas ocupa 76800 palavras (160 X 480).

Até o presente momento o módulo de vídeo está prototipado na plataforma de prototipação ML-401, porém com problemas na leitura de dados da memória DDR.

4.4.1 Interface VGA

É o principal componente usado pelo jogo para passar informações ao jogador durante uma partida. Este componente funciona juntamente com a memória DDR. Os dados a serem exibidos na tela são lidos de uma área reservada na memória. Estes dados serão fornecidos à memória por um dos processadores da arquitetura do jogo. Este mesmo processador é responsável por atualizar os dados na memória durante o decorrer do jogo. Os dados gerados para o monitor seguem o formato VGA.

O tamanho visível padrão da tela no formato VGA é de 640 x 480 pixels. A imagem deve ser atualizada 60 vezes por segundo (60Hz).

O formato VGA tem três sinais de cor denominados *RGB* (*red*, *green* e *blue* em inglês). Cada um destes sinais é responsável por controlar um canhão de elétrons que irão gerar as cores na tela do monitor. Os monitores de cristal líquido não funcionam da mesma forma, porem suportam este formato. A combinação destes sinais é a base para a geração das cores do padrão. Os valores do sinal RGB para as cores apresentadas são mostrados na Tabela 4. Na plataforma de prototipação ML-401 cada um dos sinais RGB que se conectam com o monitor são definidos por 6 bits permitindo uma boa dose de variedade de cores.

Tabela 4 - Cores e respectivos sinais RGB.

Cor	Sinal RGB
preto	000
branco	111
vermelho	100
azul	001
amarelo	110
ciano	011
rosa	101
verde	010

A tela começa a ser desenhada a partir do canto superior esquerdo e prossegue preenchendo a linha até o canto direito com o movimento do canhão de elétrons. Em seguida o canhão retorna para o canto esquerdo e preenche a linha seguinte até desenhar a tela inteira e retornar ao início. Para tanto, são necessários dois sinais adicionais para sincronismo, um vertical e outro horizontal para que o canhão se posicione no início da nova linha e o feixe de elétrons seja desligado enquanto o

canhão se posiciona. Ambos são ativados em nível lógico zero. A Figura 30 e a Figura 31 mostram os sinais gerados e seus tempos para uma tela de resolução 640 x 480.

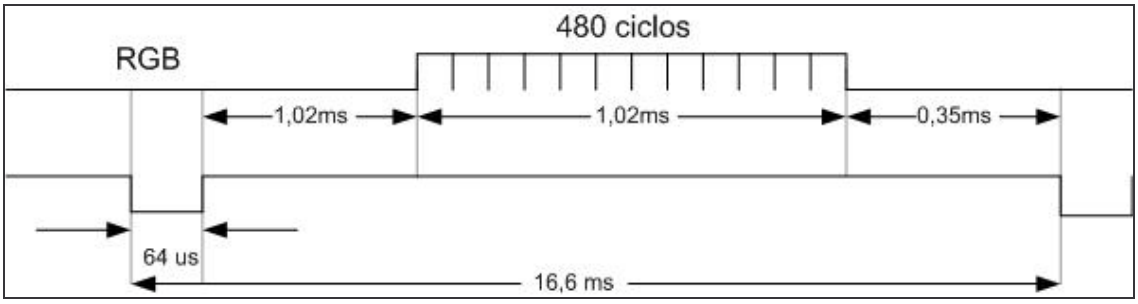


Figura 30 – Sinal de sincronismo horizontal.

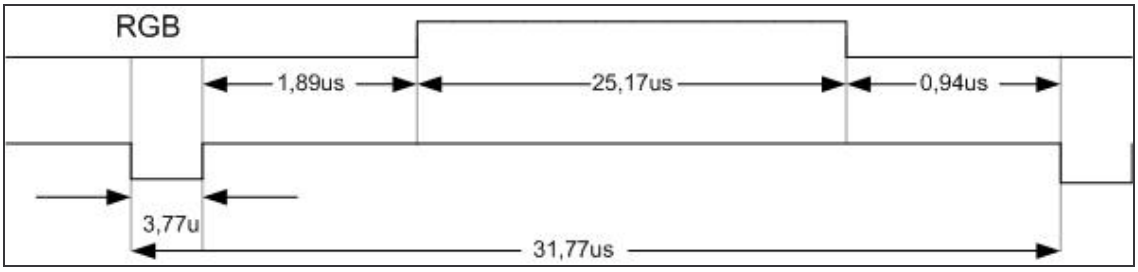


Figura 31 – Sinal de sincronismo vertical.

Se os sinais de sincronismo forem gerados fora do momento correto, o monitor usado irá desligar ou apresentar alguma mensagem indicando que o sinal não foi reconhecido. Os monitores normalmente possuem um diodo emissor de luz que liga com a cor verde se reconheceu o sinal, e cor laranja caso contrário.

É possível alterar a resolução da tela mudando a freqüência do sinal *pixel clock*. É possível gerar um número maior de cores e atingir resoluções muito mais altas do que 640 x 480 *pixels* em formatos de vídeo mais recentes que foram criados a partir do formato VGA.

4.4.2 Controlador de Acesso à Memória DDR

Utilizou-se o controlador de memória DDR desenvolvido por Everton Carara do Grupo de Apoio ao Projeto de Hardware. Este controlador visa abstrair as complicações de acesso à memória do tipo DDR, tratando esta de forma semelhante a uma memória RAM comum. A plataforma ML-401 possui dois chips de memória DDR SDRAM HYB25D256160BT(L)-7 ×16 ligados em paralelo de maneira a formar uma memória com palavras de 32 bits. A Figura 32 ilustra a interface do controlador e sua ligação com a memória.

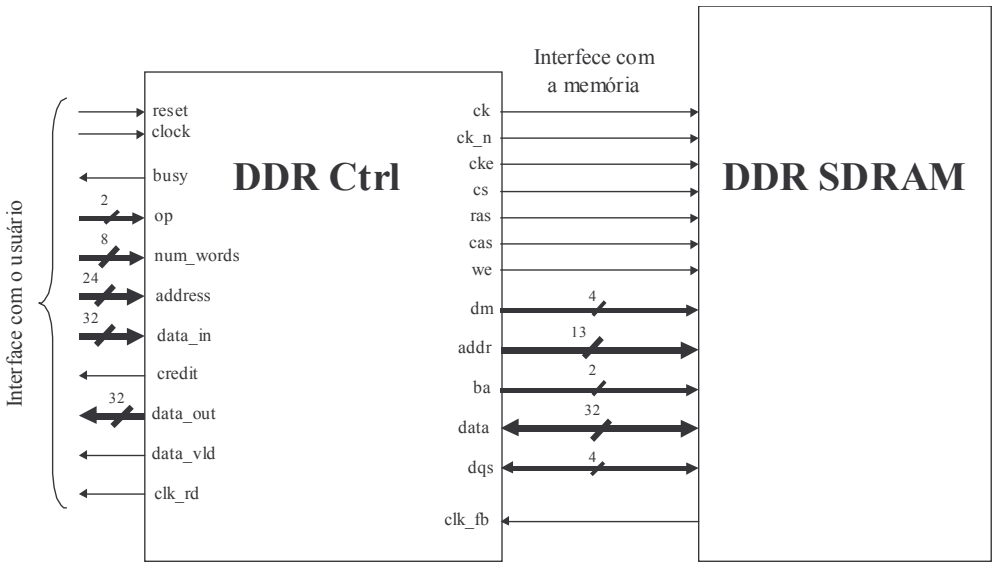


Figura 32 – Diagrama de blocos do controlador DDR conectado à memória DDR da placa ML-401.

Descrição da interface com o usuário.

- *reset*: sinal de inicialização do controlador. Ativo em '1'.
- *clock*: clock do controlador.
- *busy*: em nível '1' indica que o controlador está ocupado executando a inicialização da memória, *refresh* ou leitura/escrita. Em nível '0' indica que o controlador está apto a realizar uma operação de leitura/escrita.
- *op*: operação a ser executada. Estas operações possíveis são: 1 - *READ*, 2 – *WRITE* e 0 - *IDLE*.
- *num_words*: número de palavras a serem lidas/escritas em uma operação (1 – 255).
- *address*: endereço inicial de leitura/escrita.
- *data_in*: palavra a ser escrita na memória.
- *credit*: em nível alto indica que o controlador está apto a receber palavras na entada *data_in*.
- *data_out*: palavra lida da memória.
- *data_vld*: em nível '1' indica que a palavra em *data_out* é válida.
- *clk_rd*: clock para a leitura das palavras na saída *data_out*.

A Figura 33 ilustra uma escrita em rajada de 4 palavras na memória a partir do endereço 0x100 usando o controlador DDR. As entradas em negrito devem ser geradas pelo usuário. As palavras a serem escritas são: 1, 2, 3 e 4.

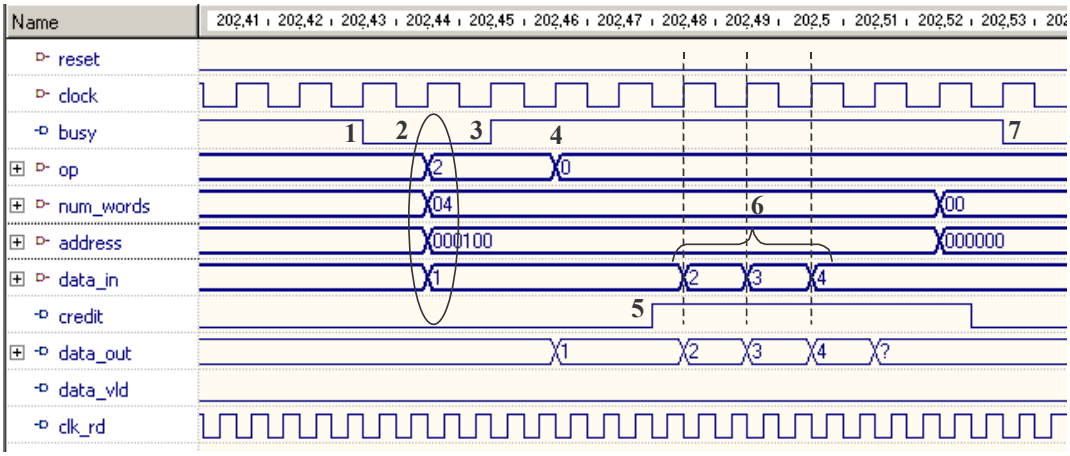


Figura 33 – Escrita na memória.

O controlador coloca a saída *busy* em nível baixo, indicando que está apto a realizar uma leitura ou escrita na memória.

1. Uma escrita é solicitada ao controlador através da entrada *op* (*op = 2*). O número de palavras a serem escritas (*num_words*) é 4 a partir do endereço (*address*) *0x100* e a primeira palavra a ser escrita (*data_in)* é 1.
2. O controlador indica através da saída *busy* que a solicitação foi atendida.
3. A entrada *op* deve voltar para *idle* (*op = 0*), pois a solicitação de escrita já foi atendida.
4. O controlador indica através da saída *credit* que na próxima borda de subida da entrada *clock* a próxima palavra a ser escrita na memória deve ser colocada na entrada *data_in*.
5. A cada borda de subida da entrada *clock* uma nova palavra a ser escrita na memória é colocada na entrada *data_in*.
6. O controlador indica através da saída *busy* que a operação está concluída e novas solicitações podem ser feitas.

O resultado da escrita é o seguinte:

MEM[0x100] = 1

MEM[0x101] = 2

MEM[0x102] = 3

MEM[0x104] = 4

A Figura 34 ilustra a leitura em *burst* de 4 palavras da memória a partir do endereço 0x100. As entradas em negrito devem ser geradas pelo usuário. As palavras a serem lidas correspondem às escritas na Figura 34, ou seja: 1, 2, 3 e 4.

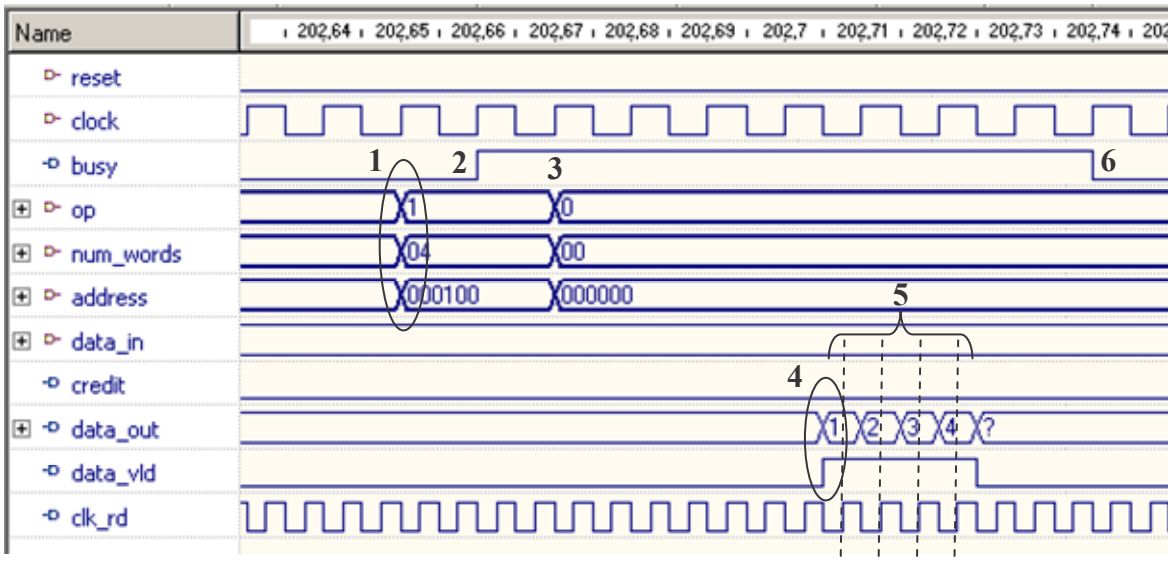


Figura 34 – Leitura da memória.

1. Uma leitura é solicitada ao controlador através da entrada op (op = 1). O número de palavras a serem lidas (num_words) é 4 a partir do endereço (address) 0x100 .
2. O controlador indica através da saída busy que a solicitação foi atendida.
3. A entrada op deve voltar para idle (op = 0), pois a solicitação de leitura já foi atendida.
4. A saída data_vld indica que a palavra na saída data_out é válida. A cada borda do clock (descida/subida) uma nova palavra estará válida na saída data_out.
5. As palavras em data_out devem ser lidas na borda de subida da saída clk_rd.
6. O controlador indica através da saída busy que a operação está concluída e novas solicitações podem ser feitas.

O controlador DDR e as informações expostas nesta Seção foram extraídas de documento de comunicação interna escrito por Carara [CAR06].

4.4.3 Leitura e escrita no módulo de vídeo

A leitura e escrita na memória integrante do módulo de vídeo ocorrem em momentos específicos baseados no estado do sinal *blank*. O sinal *blank* sinaliza quando o canhão de elétrons do monitor está escrevendo uma linha na tela ou reposicionando-se para escrita de uma nova linha.

Quando o sinal *blank* está em nível lógico baixo (momento em que o canhão RGB está desligado para reposicionamento na tela), sempre ocorre a leitura de uma linha inteira da tela. Esta linha será escrita na tela do monitor na próxima borda de subida de *blank*.

A escrita na memória de vídeo ocorre quando o sinal *tx* está em nível lógico alto (há dados disponíveis na entrada do módulo). O módulo de vídeo aguardará até a próxima borda de subida de *blank* (quando nenhuma leitura estará sendo realizada) para começar a armazenar os dados. A

Figura 35 mostra o processo de escrita no módulo de vídeo enquanto a Figura 36 mostra o processo de leitura.

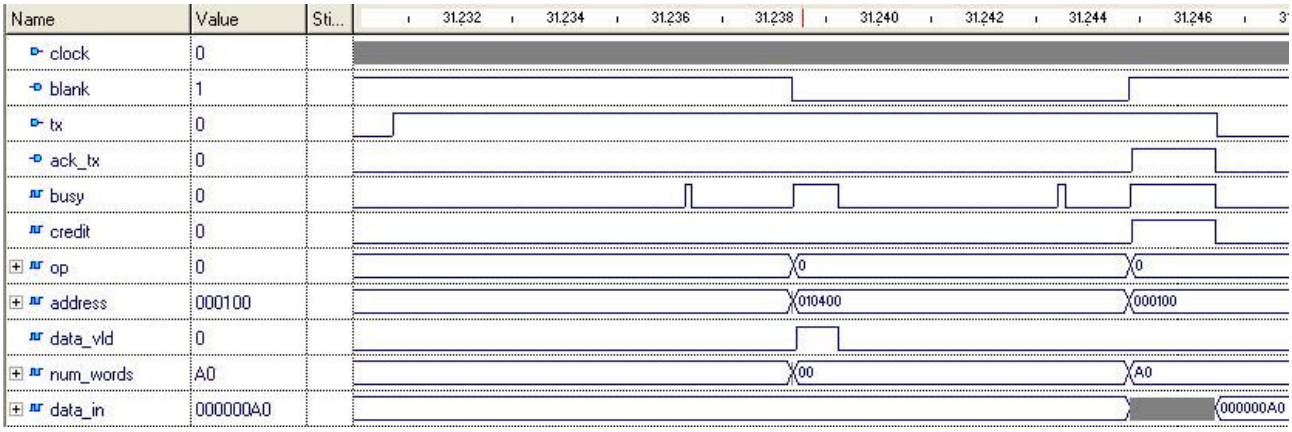


Figura 35 – Processo de escrita no módulo de vídeo.

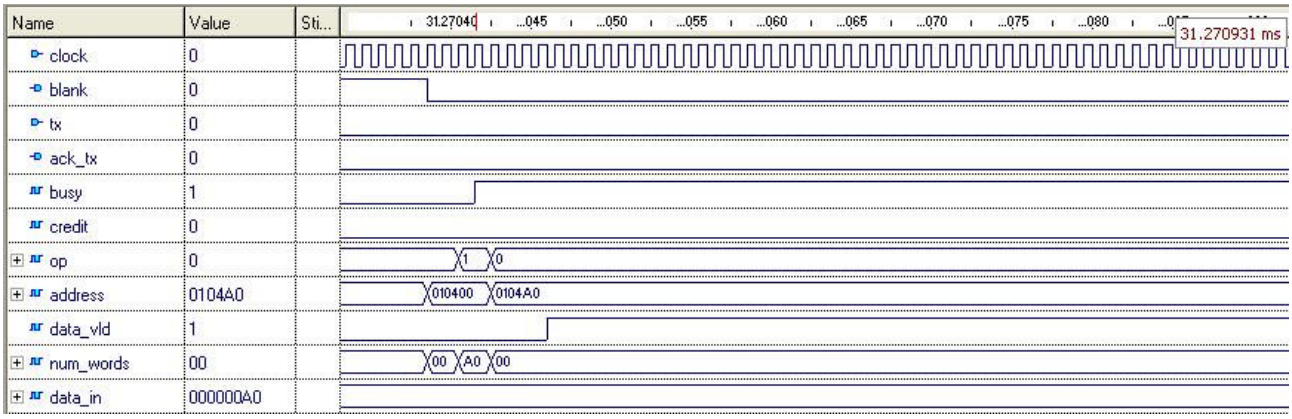


Figura 36 – Processo de leitura no módulo de vídeo.

4.5 Módulo de Teclado

Na proposta para este trabalho foram sugeridos alguns componentes que, em um primeiro momento, seriam úteis à plataforma como teclado, mouse e, possivelmente, *joystick*. Com o decorrer do estudo foi realizada a definição de alguns elementos do trabalho como o jogo, que é um jogo de xadrez, descrito na Seção 2.1. Verificou-se que a criação de um controlador de teclado e funções de interface com o programa principal do jogo são suficientes para implementar a interface básica com o usuário para entrada de dados. O controlador de entrada de dados para a plataforma de jogos desenvolvida utiliza o conector PS2. Este controlador é conectado aos sinais de freqüência e de dados da porta PS2 conectada ao FPGA. Estes sinais são denominados *kb_clk* e *kb_data* respectivamente e recebem informações a partir do teclado (componente de entrada pelo qual o usuário enviará comandos ao jogo). A Figura 37 mostra o diagrama de blocos do módulo, e os sinais de entrada são explicados logo abaixo.

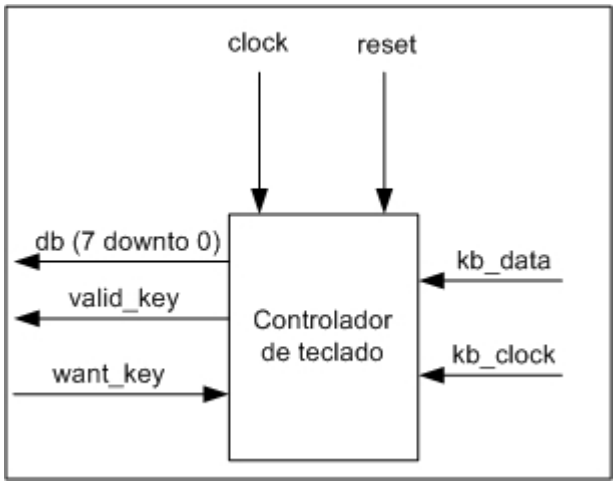


Figura 37 – Diagrama de blocos do controlador de teclado.

Cada tecla pressionada gera 8, 16 ou 24 bits recebidos do teclado pelo sinal *kb_data*. Cada 8 bits representa um *scancode* sendo que cada tecla gera um, dois ou três *scancodes*. O PS2, ao qual conecta-se o teclado, repassa estes *scancodes* ao console através do sinal *kb_data* em 11 ciclos do sinal *kb_clock* alinhados na borda de descida do sinal *kb_clock*. Quando *kb_data* permanece em nível lógico '1' nenhum dado está sendo transmitido. No primeiro ciclo de transmissão o sinal *kb_data* vai para nível lógico '0' sinalizando um *bit de início*. Este *bit de início* é seguido pelos oito bits de *scancode* referentes à tecla pressionada e um bit de paridade. O 11º bit sinaliza o fim da transmissão. Este bit apresenta nível lógico alto e permanece assim até que uma tecla seja pressionada novamente e uma nova transmissão ocorra. Este processo é ilustrado pela Figura 38.

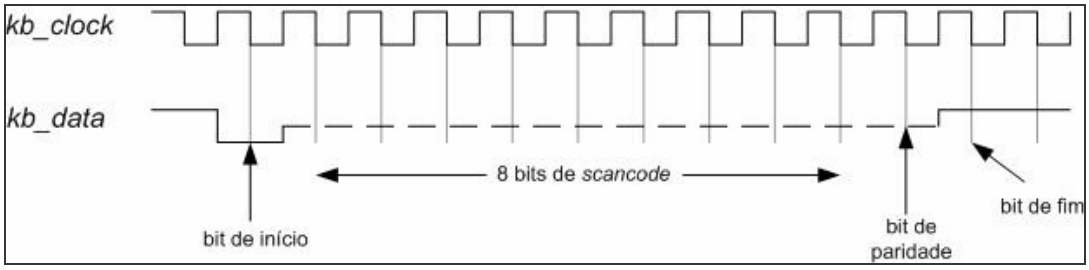


Figura 38 – Forma de ondas representando a transmissão de um *scancode* pelo PS2.

Ao pressionar uma tecla o teclado imediatamente envia os *scancodes* referentes a ela da maneira descrita anteriormente. Ao manter-se esta tecla pressionada, o processo de envio do referido *scancode* se repetirá a cada 100ms no que se denomina de função "autorepeat".

O componente desenvolvido neste trabalho recebe as informações do PS2, converte os *scancodes* para código ASCII transmitindo este dado ao sistema através do sinal *db*. O sinal *valid_key* informa ao sistema quando um novo caractere está disponível para leitura. O sinal *want_key* foi adicionado ao controlador de teclado para facilitar os testes da parte de entrada de dados da plataforma. Este sinal indica quando o sistema está apto a receber um novo caractere.

O módulo de teclado está atualmente prototipado. Todas as teclas, com exceção de teclas especiais (print screeen, pause, window, etc) são reconhecidas.

4.6 Organização da infra-estrutura

Foi utilizada a plataforma desenvolvida por [WOS05], a qual é composta por uma NoC 3x2 na qual estão conectados três processadores Plasma nos endereços 1x0, 1x1 e 2x0. A esta plataforma foram acrescentados dois módulos. O módulo de teclado foi conectado ao endereço 0x1 e o módulo de VGA ao endereço 0x0. A Figura 39 mostra a organização da plataforma utilizada neste trabalho.

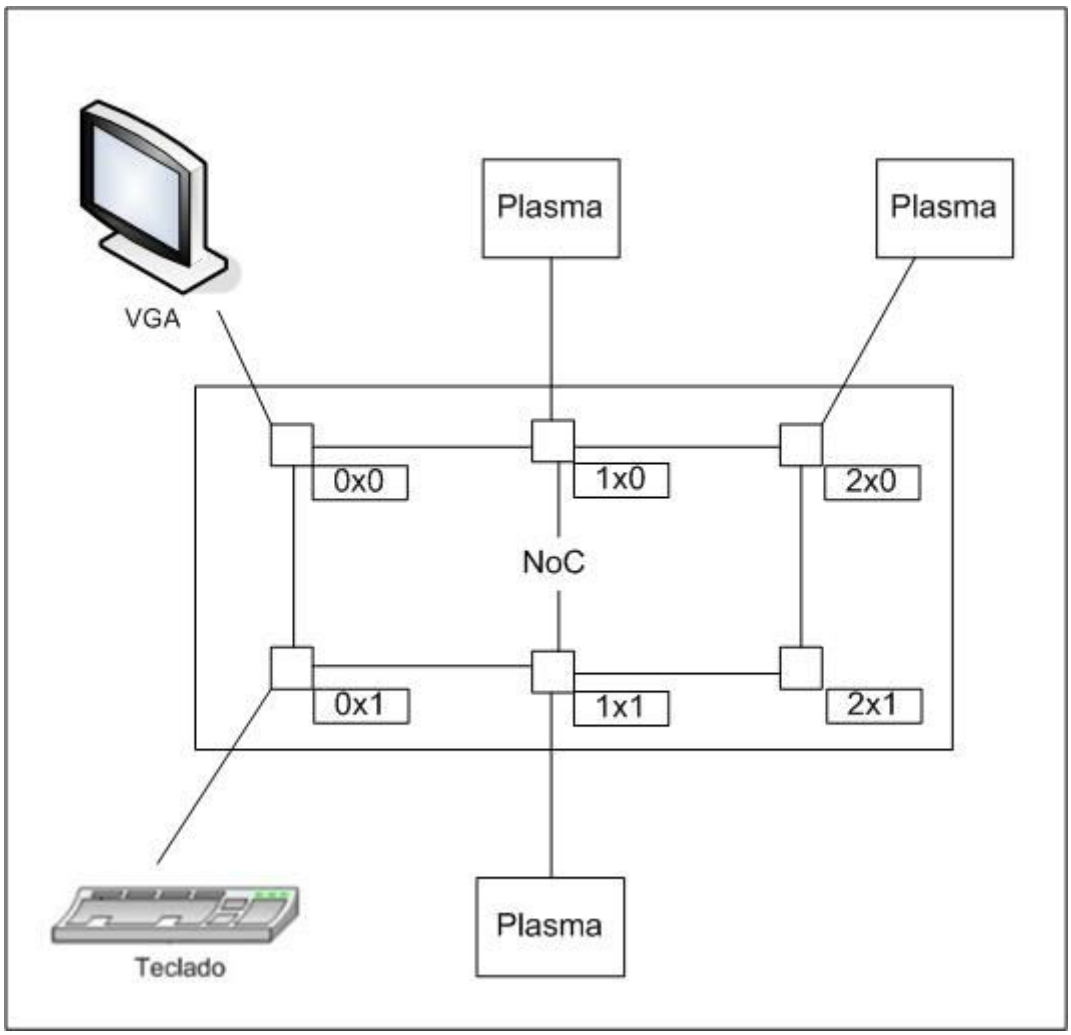


Figura 39 - Organização da plataforma desenvolvida neste trabalho.

A tarefa principal do GNU Chess foi alocada no processador conectado ao endereço 1x0. O processador conectado ao endereço 1x1 seria utilizado para executar a tarefa auxiliar de busca explicada na Seção 6.2.3 que não foi implementada até então. O processador Plasma conectado a porta 2x0 não foi utilizado nesta implementação. O módulo de vídeo foi conectado a porta 0x0. O módulo de teclado foi conectado a porta 0x1.

5 Porte do GNU Chess para a Arquitetura Alvo

Este Capítulo apresenta o processo de porte do código do GNU Chess para a arquitetura definida neste trabalho, incluindo modificações no código do GNU Chess e alterações na arquitetura alvo.

5.1 Modificações realizadas no código do GNU Chess

As modificações realizadas no código do GNU Chess foram feitas tendo em vista que o programa deveria executar em uma arquitetura com recursos como memória para armazenamento de dados bem menor que na arquitetura na qual ele foi desenvolvido para executar originalmente.

5.1.1 Porte de software

O GNU Chess foi desenvolvido de forma a ser compilado em qualquer versão recente do GCC para a arquitetura x86 no momento da escrita deste trabalho, então, outras modificações tiveram que ser feitas para que o mesmo fosse compilado com o MIPS-GCC, que é um compilador C para o processador MIPS (ver Seção 4.3.5). Este compilador não fornece bibliotecas como *stdio*, *stdlib*, entre outras. Portanto, as chamadas de funções tiveram que ser cuidadosamente modificadas no código. Funções como *printf*, *memset*, *isspace*, *toupper*, *atoi* e todas as funções de manipulação de *strings* como *strtol, strlen* foram substituídas por versões implementadas no decorrer do trabalho, e funções como *malloc* foram cuidadosamente removidas. O GNU Chess faz uso de paralelismo através de *threads* para a rotina que implementa a entrada de dados. Isto também teve que ser retirado do código, sendo que na sua nova versão, o programa pára e fica a espera de dados de entrada. Isso não compromete de maneira nenhuma o desempenho, já que a *thread* de entrada serve apenas para que o código seja melhor utilizado no caso de acoplá-lo a uma interface gráfica ou outra interface diferente da padrão.

Optou-se por retirar do código do GNU Chess o que não era necessário para uma versão que fosse executada na plataforma MPSoC, economizando assim espaço de memória em detrimento do desempenho. Foram retiradas as funções de acesso a livros de jogadas, já que não seria conveniente ter arquivos de livros para jogadas de melhor qualidade se o estudo deste trabalho não envolve qualidade de jogo. Também foi removida a tabela *hash* geral, pois esta tabela necessitava ser alocada dinamicamente, ocupando um grande espaço e proporcionava apenas melhoras no desempenho do algoritmo de busca.

As rotinas de entrada e saída do código do GNU Chess tiveram que ser alteradas de forma que este código pudesse comunicar-se os módulos componentes da plataforma.

As modificações citadas anteriormente, necessárias para o porte, foram realizadas em aproximadamente 21 dias.

O processo de porte foi feito observando-se três passos:

1. Modificação e/ou remoção de código.
2. Compilação e teste na plataforma x86.
3. Compilação e teste na plataforma alvo.

5.1.2 Porte de hardware

A plataforma MPSoC teve que sofrer modificações para acomodar o código do GNU Chess. A principal alteração foi referente à memória, que teve que ser drasticamente aumentada de 8Kb para 2MB. Outra alteração foi do número de páginas e por conseqüência, a diminuição do número de programas que podem ser executados simultaneamente em um único processador. Na plataforma original, existem quatro páginas de 2Kb cada em cada processador Plasma. Cada página permite que um programa seja executado independentemente dos programas das outras páginas, ou seja, pode-se executar quatro programas simultaneamente, sendo a primeira página destinada ao *kernel* do processador e três para o usuário. O número de páginas foi alterado para duas de 1Mb cada, sendo a primeira reservada para o *kernel* desenvolvido em trabalho paralelo [WOS05] e a segunda destinada ao código do GNU Chess. O aumento foi necessário devido ao tamanho do código do GNU Chess.

A alteração foi feita nos seguintes arquivos do processador Plasma:

- *Ram.vhd*: o valor da constante ADDRESS_WIDTH foi alterado de 15 para 21. Esta constante define o tamanho da memória total de cada processador definida por 2 elevado a ADDRESS_WIDTH. Este valor é dividido pelo número de páginas contidas no processador. Também mudou-se o endereço inicial da segunda página do processador definida na variável *index*, de 2048 para 262144. O cálculo para se chegar a estes valores é: ((2 ** ADDRESS_WIDTH) / NUMERO_DE_PAGINAS) / 4. O valor constante 4 é utilizado para o valor de index seja dado em palavras (as instruções do MIPS são palavras de 4 bytes).
- *Mlite_cpu*: os bits que indicam o endereço da página ativa da tarefa no processador no sinal *mem_address* foi modificado em número (de dois para um) devido à redução do número de páginas em cada processador e deslocado (do bit dezessete para o bit vinte) devido ao aumento do tamanho de cada página.
- *reg_bank*: a variável *page*, que armazena o número de *bits* que indexam as páginas foi modificado de dois (indexa quatro páginas) para um (indexa duas páginas).

Alterações no arquivo de kernel da MPSoC:

- *kernel.c*: o tamanho da memória destinada a tarefa 1 do processador (que será executado na segunda página do processador) foi aumentado de 2048 bytes para 1048576 bytes.

6 Desenvolvimento de Hardware e Software

Este capítulo trata da fase de desenvolvimento do projeto que se destina a integrar a parte de hardware e software. Inicialmente serão discutidos os elementos de hardware que fazem a interface entre os módulos que constituem o console de jogo e a rede de interconexão Noc.

6.1 Wrappers

Wrappers, neste projeto, são módulos de hardware destinados a converter os sinais transmitidos no protocolo da rede Hermes para os protocolos dos módulos que compõem a arquitetura de hardware definida neste trabalho. No decorrer desta Seção serão definidos os diferentes tipos de *wrappers* desenvolvidos e os protocolos que estes implementam.

6.1.1 Protocolo padrão dos wrappers

Para que seja possível a comunicação entre os módulos definidos no capítulo 4 deve-se definir um protocolo de comunicação que será utilizado pelos *wrappers* que ligam os módulos à NoC.

Para que a comunicação entre as chaves que compõem a rede, ver seção 6.1.1 para maiores informações, foi definido o protocolo de nível de rede exposto a seguir: as mensagens são compostas por *flits* de 16 bits. O primeiro *flit* que compõe a mensagem traz informações sobre o endereço da chave à qual a mensagem se destina. Através da análise deste *flit* cada roteador que compõe a rede pode definir se a mensagem deve ser passada adiante ou se esta deve ser transmitida para o módulo que está conectado à sua porta local.

O segundo *flit* que compõe indica o número de *flits* que ainda estão por vir e que compõem a mesma mensagem. Chamamos este de *flit* de tamanho da mensagem. O restante da mensagem é chamado de *payload*. A Figura 40 mostra um exemplo de mensagem recebida por um *wrapper* conectado a rede de interconexão.

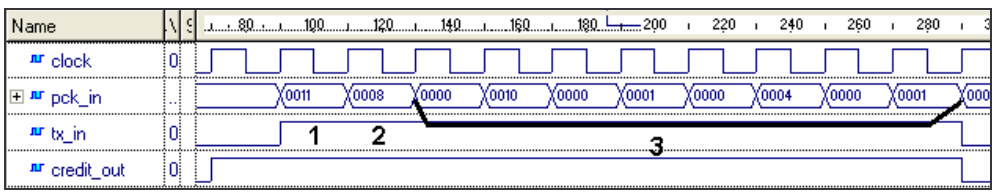


Figura 40 – Exemplo de mensagem recebida por um *wrapper* conectado a rede de interconexão.

Em *1* pode-se ver o sinal *credit_out* em nível lógico alto indicando que o *wrapper* está pronto para receber uma nova mensagem. Neste momento é recebido o *flit* que contém o endereçamento do nodo da rede ao qual se destina esta mensagem, no caso o nodo 11 em hexadecimal. Juntamente com a chegada do pacote que contém o endereço de destino da mensagem o sinal *tx_in* indica que

este pacote contém dados válidos. No ciclo de relógio seguinte, indicado pelo sinal *clock*, é recebido o *flit* que informa o tamanho do *payload* mostrado em *2*. Por fim são recebidos os demais *flits* que compõem o *payload* da mensagem mostrados em *3*.

Em um nível acima ao nível de rede é implementado um segundo protocolo que atende as necessidades de comunicação entre as múltiplas tarefas executadas em cada módulo do sistema desenvolvido por [WOS05]. Este protocolo consiste basicamente de mensagens contendo três serviços distintos definidos como: (*i*) mensagem de requisição de dados (em inglês *request message*) identificada pelo valor 10 em formato hexadecimal; (*ii*) mensagem de entrega de dados (em inglês *delivery message*) identificada pelo valor 20 em formato hexadecimal e; (*iii*) mensagem de resposta nula (em inglês *no message*). Este último identificado pelo valor 30 em formato hexadecimal.

A primeira mensagem listada, *request message*, destina-se a solicitar dados a um módulo conectado à rede. O segundo tipo de mensagem, *delivery message*, tem por objetivo transmitir dados gerados por um módulo a outro. Este último tipo de mensagem pode ser gerado como resposta a uma *request message* anterior. As mensagens do tipo *no message* são usadas como resposta a uma mensagem de requisição de dados, porém estas não carregam dado algum. As mensagens *no message* indicam que não há mensagens a enviar.

As mensagens pertencentes a este protocolo apresentam uma configuração básica de oito *flits* organizados em pares, ou seja, quatro pacotes de 32 bits. O primeiro pacote identifica o serviço da mensagem. O segundo pacote traz o endereço do módulo que gerou esta mensagem. O pacote seguinte indica para qual tarefa a mensagem se destina e, por fim, o último mostra qual tarefa originou a mensagem.

Adicionalmente, as tarefas pertencentes à classe de *delivery message* apresentam mais dois grupos de *flits* após o pacote que indica a tarefa de origem da mensagem. O primeiro grupo é o pacote que indica o tamanho da mensagem de serviço que virá em seguida. Este pacote indica o número de pacotes que compõem a mensagem do serviço. Por fim, o último conjunto de pacotes, traz os dados que compõem a mensagem do serviço. A Figura 41 mostra um diagrama com a estrutura dos grupos de mensagens citados anteriormente.

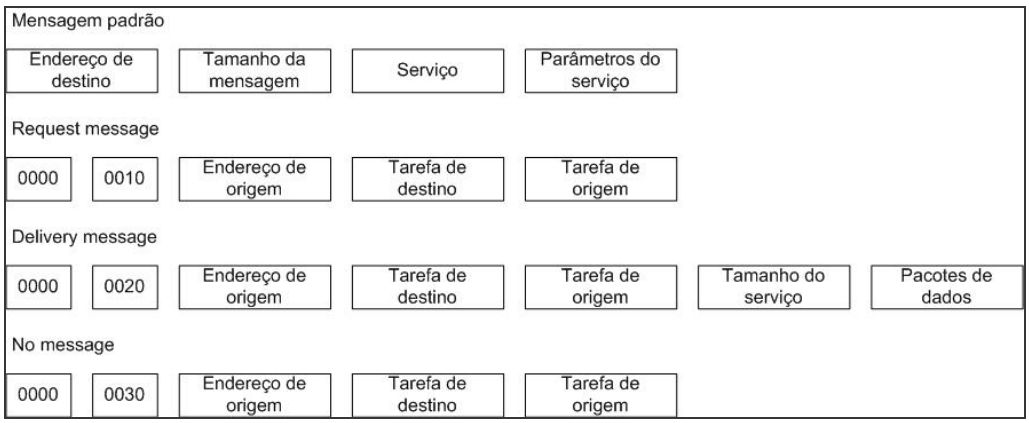


Figura 41 – Diagrama do formato das mensagens.

Existem ainda duas outras tarefas distintas que se destinam à alocação de tarefas nos processadores presentes na arquitetura. Estas tarefas são chamadas de *Task_Alocation* e *Allocated_task*. As mensagens destinadas à alocação de tarefas não serão abordadas neste trabalho.

6.1.2 Plasma - Hermes

A interface entre a NOC Hermes e o Plasma é implementada por um *wrapper* desenvolvido por Everton Carara do Grupo de Apoio ao Projeto de Hardware, o qual chamaremos nesta Seção de *wrapper* de comunicação. O *wrapper* de comunicação servirá de base para os *wrappers* desenvolvidos durante a segunda parte do Trabalho de Conclusão.

O *wrapper* de comunicação para pacotes de controle é composto por duas máquinas de estado. A primeira é responsável pelo recebimento de pacotes vindos da NoC e a segunda é responsável pela emissão de pacotes através da rede.

Este wrapper implementa os protocolos definidos na seção 6.1.1 e destina-se a tornar possível a comunicação do processador Plasma com os demais módulos integrantes da implementação.

6.1.3 Teclado – Hermes

No presente trabalho foi desenvolvido um módulo responsável por conectar o controlador de teclado com a rede de conexão utilizada. Este módulo denominado *Wrapper de teclado* tem a finalidade de interagir com as mensagens vindas de um dos processadores enviando dados que o usuário insere no sistema a través do teclado.

O *wrapper de teclado*, basicamente, é composto por duas máquinas de estados. A primeira recebe os dados da controladora de teclado. Em seguida o wrapper armazena o dado recebido em um *buffer* que será utilizado por uma segunda máquina de estados. Após armazenar a informação recebida do teclado a primeira máquina de estados informa a segunda que o *buffer* contém um caractere válido através do sinal *buffer_valido*. A partir de então esta máquina aguarda que a

máquina responsável por montar a mensagem que será enviada à NoC. Este cíclo é repetido indefinidamente. A Figura 42 mostra a forma de ondas extraída da simulação de teste do *wrapper de teclado*.

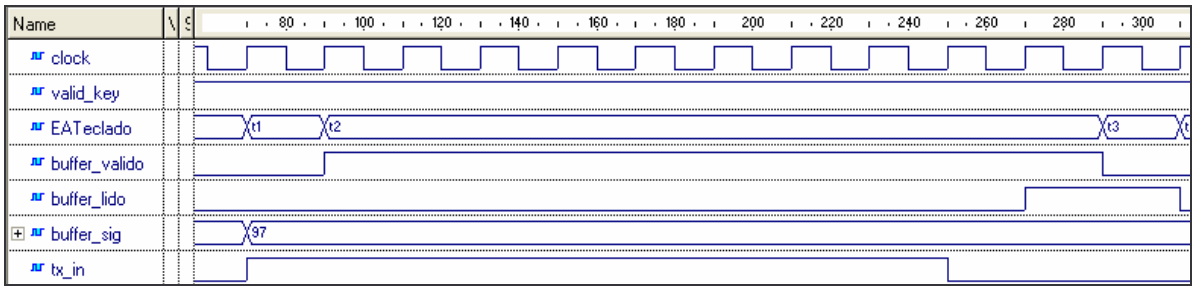


Figura 42 – Forma de ondas extraída de simulação do *wrapper do controlador de teclado*.

Na forma de ondas apresentada na Figura 42 o *wrapper do controlador de teclado* recebe o caractere "a" (97 em inteiro conforme no formato ASCII) quando entra no estado *t1* (representado pelo sinal *EATeclado*). Em seguida o sinal *buffer_valido* informa à segunda máquina de estados que existe um caractere válido em *buffer_sig*. A máquina fica aguardando o sinal *buffer_lido* no estado *t2* para continuar o processo.

Por sua vez, a segunda máquina de estados cumpre a função de receber requisições originadas do processador e compor mensagens de resposta contendo dados que informam a última tecla pressionada. Para tanto, devemos observar que o esta máquina comunica-se com uma tarefa específica que está sendo executada em um dos processadores. O funcionamento desta tarefa e a forma como o *wrapper* e a tarefa se comunicam será explicada posteriormente.

Esta segunda máquina responde apenas a mensagens de requisição. Ao receber a solicitação de dados através da rede, a máquina de estados verifica se há dados válidos no *buffer*. Certificando-se da validade dos dados a través do sinal *buffer_valido*, controlado pela primeira máquina de estados, a mensagem começa a ser composta com as informações obtidas a través da mensagem de requisição. A Figura 43 mostra as formas de onda extraídas da simulação. Em seguida é dada uma explicação do funcionamento desta máquina da forma como são gerados os sinais.

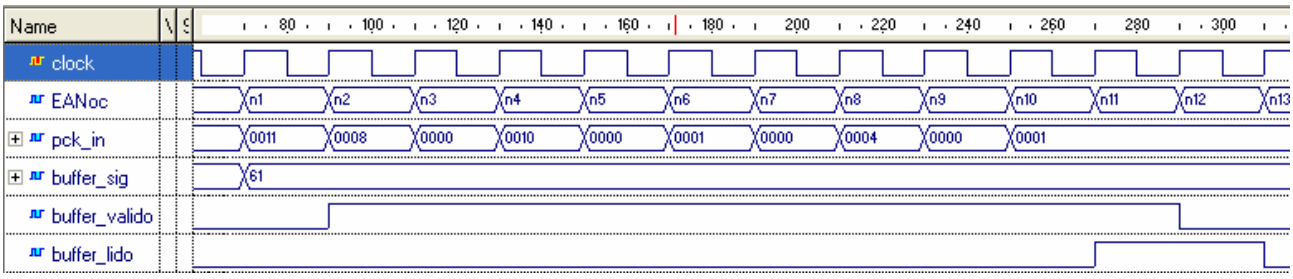


Figura 43 – Formas de onda mostrando o recebimento de uma requisição de leitura pelo *wrapper* de teclado.

O sinal denominado *EANoc* indica o estado em que se encontra a máquina de estados de recepção e envio de pacotes do *wrapper* de teclado. Quando este sinal encontra-se em *n1* o *wrapper* recebe o primeiro pacote da mensagem contendo o endereço do nodo ao qual a mensagem se

destina. Em seguida, no estado *n2*, é recebido o pacote que contém o tamanho da mensagem. Nos estados *n3* e *n4* é recebido o serviço da mensagem. O endereço do processador que originou a mensagem é recebido em *n5* e *n6*. Nos demais estados até o estado *n10* são armazenados o identificador da tarefa de origem da mensagem e os pacotes contendo o identificador da tarefa à qual a mensagem se destina. Por fim o estado *n11* lê o valor contido no *buffer* e sinaliza à primeira máquina que este já pode buscar um novo caractere através do sinal *buffer_lido*. O estado *n12* destina-se a aguardar que o sinal *buffer_valido* apresente nível lógico baixo evitando a possibilidade de múltiplas leituras de um mesmo dado do *buffer*.

Nos estados *n13* até o *n26* o *wrapper* de teclado envia a mensagem de resposta à requisição recebida. A mensagem de resposta destina-se à tarefa que gerou a mensagem de requisição e leva a informação da última tecla pressionada. Esta mensagem de resposta obedece ao protocolo definido em 6.1.1. Na Figura 44 pode-se ver uma mensagem de resposta contendo o caractere "g", representado em hexadecimal segundo o padrão ASCII. Este valor é enviado nos estados *n25* e *n26* da máquina de estados.

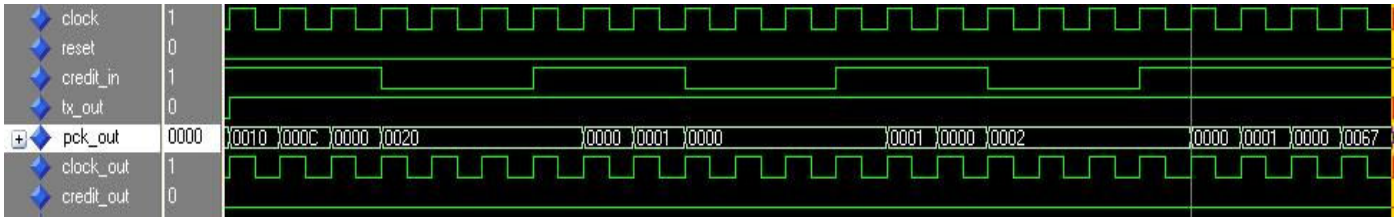


Figura 44 – Formas de onda mostrando o envio de uma mensagem de resposta do *wrapper* de teclado.

6.1.4 Módulo de vídeo – Hermes

O *wrapper* de vídeo é responsável pela comunicação entre a rede Hermes e o módulo de vídeo descrito na seção 4.4. Este *wrapper* recebe os dados enviados pelo processador Plasma responsável por gerar as imagens do jogo.

O *wrapper* de VGA é composto por uma interface de comunicação com o controlador de vídeo e uma interface para comunicação com a NoC. A interface de comunicação com o controlador de vídeo é formada pelos sinais *ack_tx_vga*, *data_vga*, *tx_vga* e *address_vga*. A interface com a NoC e seu funcionamento seguem as definições expostas na Seção 6.1.1. A Figura 45 mostra o *wrapper* do controlador de vídeo e seus sinais de comunicação.

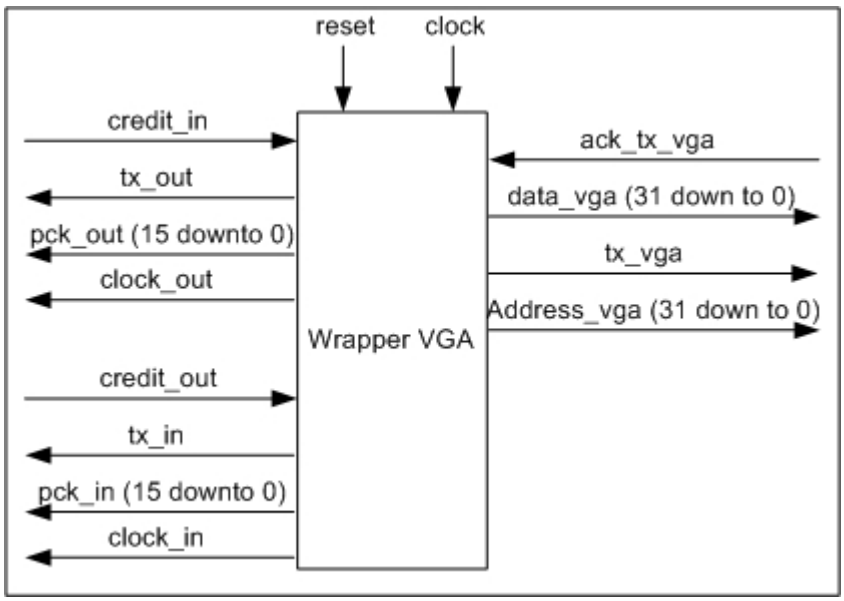


Figura 45 – *Wrapper* do controlador de vídeo.

Para que ocorra a comunicação entre a Noc e o módulo de vídeo o *wrapper* do controlador de vídeo fica constantemente aguardando mensagens geradas por um dos processadores que compõem o sistema. A mensagem recebida deve estar identificada como um serviço de entrega (*delivery message*). Mensagens que não estejam rotuladas como *delivery messages* serão ignoradas. A Figura 46 mostra o início de uma mensagem destinada ao módulo controlador de vídeo. A parte de *carga* das mensagens destinadas ao módulo de vídeo, que são recebidas pelo respectivo *wrapper*, deve ser composta por dois elementos: (*i*) - o endereço de memória a partir do qual serão gravados os dados, e; (*ii*) – os bytes com as informações que formarão a imagem na tela.

Nesta implementação decidiu-se que o processador enviaria para o módulo de vídeo mensagens contendo informações para preencher exatamente uma linha no monitor, ou seja, 160 palavras de 32 bits. Uma implementação alternativa poderia modificar este padrão buscando melhorias de desempenho. Para maiores informações sobre possíveis modificações veja o capítulo 8.

Nesta Figura 46 são mostrados os primeiros pacotes da mensagem. Estes pacotes contêm o endereço do módulo destino, o tamanho da mensagem, o serviço da mensagem, o endereço do processador de origem, o identificador da tarefa de destino, o identificador da tarefa de origem, o tamanho do serviço e o endereço a partir do qual os dados serão escritos. A mensagem recebida segue o padrão definido em 6.1.1.

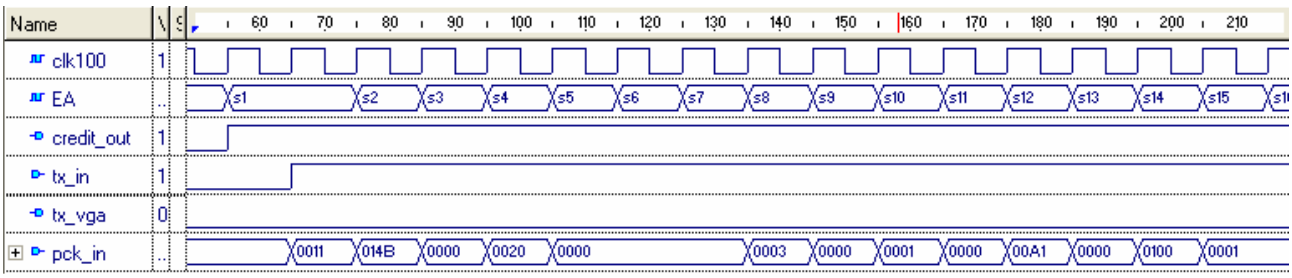


Figura 46 – Forma de ondas mostrando a chegada de uma mensagem destinada ao módulo controlador de vídeo.

Na seqüência da mensagem mostrada na Figura 46 serão recebidos os pacotes que contêm as 160 palavras com os dados necessários para preencher uma linha no monitor conectado ao console de jogo. Estes dados são recebidos pelos estados *s15* e *s16*. A Figura 47 mostra a chegada dos primeiros quatro pacotes contendo dados de vídeo sendo armazenados no sinal *buffer_sig* do *wrapper* do controlador de vídeo.

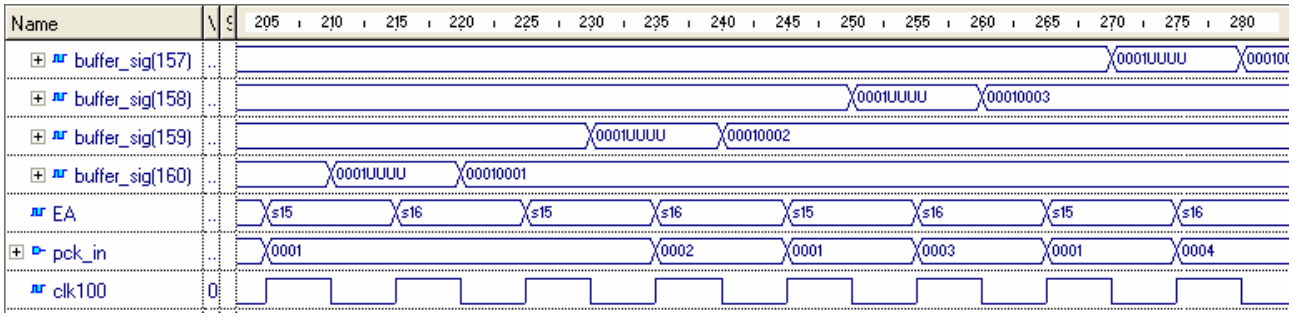


Figura 47 – *Wrapper* do controlador de vídeo armazenando dados recebidos através da rede em *buffer* local.

6.2 Software para Comunicação

Nesta seção serão apresentadas as funções desenvolvidas para implementar a comunicação entre o jogo e os módulos da plataforma desenvolvida.

6.2.1 Plasma – Teclado

A comunicação entre processador e teclado ocorre através da rede utilizando os *wrappers* de teclado e do processador Plasma, descritos na seção 6.1, e de funções descritas em C. Esta seção destina-se a descrever o software que possibilita a comunicação da tarefa principal com o teclado.

O processo de leitura de dados do teclado está inserido no código da tarefa principal e visa comunicar-se com a tarefa virtual implementada pelo *wrapper* de teclado. Primeiramente este processo gera uma mensagem de requisição de serviço através da Noc. Esta mensagem é endereçada ao *wrapper* de teclado.

Após o envio da mensagem, este processo fica trancado aguardando uma mensagem de resposta contendo o código ASCII da última tecla pressionada. Ao receber esta resposta o processo armazena o caractere recebido em um *buffer* juntamente com os demais que tenha recebido

anteriormente. Este processo se repete até que o caractere recebido seja o caractere referente ao da tecla *enter*. Ao receber uma mensagem indicando que a tecla *enter* foi pressionada o processo verifica o conteúdo do *buffer* gerando um comando a ser repassado para o processo de validação do mesmo.

Para fins de validação do sistema foi desenvolvido um módulo denominado *InputTexto* (veja a Seção 7.1.2 para maiores informações) que conecta-se diretamente ao *wrapper* de teclado. Este módulo lê um arquivo de texto caractere a caractere. Este módulo repassa os caracteres lidos para o *wrapper* de teclado no formato ASCII de maneira semelhante ao controlador de teclado descrito na Seção 4.5. A seguir é apresentada a função responsável por ler os caracteres digitados pelo jogador. A Figura 48 mostra o código da função responsável pela leitura dos comandos vindos do teclado.

```
void ReadLine(char _linha[50])
{
  int i, iTemp;
  char ch;
  Message msg;
  i = 0;
  ch = 0;
  strcpy(_linha,"");

  while (ch != '\n')
  {
    while(!ReadPipe(&msg, 2));
    if(itoa(msg.msg[0]) != "13") {
      _linha[i++] = (char)msg.msg[0];
    }
  }
  _linha[i] = '\0';
}
```

Figura 48 – Função responsável pela leitura dos comandos vindos do teclado.

6.2.2 Plasma – Módulo de Vídeo

Ao receber o comando "show board" a tarefa principal do sistema de executar a função responsável por gerar os dados (função chamada de *ImprimeTela*) que serão enviados ao módulo de vídeo. A função *ImprimeTela* deve percorrer o tabuleiro de jogo, representado pela estrutura *board* (ver Seção 2.1.3), verificando quais peças encontram-se em quais casas do tabuleiro e gerar os dados que serão enviados posteriormente.

A função *ImprimeTela* deve gerar e enviar mensagens contendo uma linha da imagem a ser mostrada de cada vez. Este é o formato que o *wrapper* do controlador de vídeo espera receber através da NoC (ver Seção 6.1.4). A Figura 49 faz a representação de um peão de 8 X 8 pixels que será utilizado para exemplificar o formato dos pacotes mostrados em seguida.

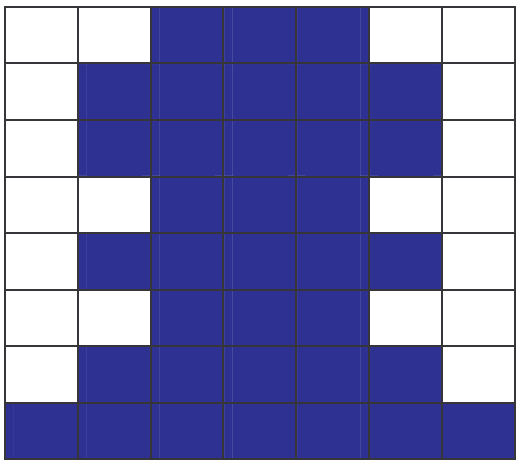


Figura 49 – Representação de um peão de dimensões de 8 X 8 pixels.

Para gerar a imagem de um peão de 8 X 8 pixels é preciso enviar 8 mensagens (uma mensagem por linha) para o módulo de vídeo. Neste exemplo serão utilizados pacotes com duas palavras para fins de esclarecer o processo.

Neste trabalho foi definido que cada pixel é definido por uma seqüência de 8 bits, logo deve-se observar que cada palavra carrega informações suficientes para a geração de quatro pixels. Cada mensagem destinada ao módulo de vídeo carrega uma linha gráfica (640 pixel agrupados em 160 palavras de 4 pixels cada). Para formar a primeira linha da imagem do peão é preciso montar duas palavras de dados. A primeira palavra é montada informando que os primeiros dois pixels da imagem devem ser pintados de branco e os dois remanescentes pintados de azul. Esta palavra então ficaria com o valor 7711 em formato hexadecimal conforme a especificação feita na Seção 4.4.1. A palavra que define as cores da segunda parte da linha deve informar que o primeiro pixel deve ser pintado com a cor azul enquanto os dois pixels seguintes devem ser pintados de branco. Neste exemplo o último pixel da segunda palavra não aparece na imagem e será definido como contendo a cor preta. Desta foram, a segunda palavra que define as cores dos pixels da primeira linha contem o valor 1770 no formato hexadecimal.

6.2.3 Plasma – Plasma

A comunicação entre os processadores Plasma integrantes do console de jogo é feita utilizando-se diretamente as funções de comunicação entre tarefas desenvolvidas por [WOS05]. Com estas funções é possível trocar informações entre os dois processadores e fazer com que ambos funcionem de maneira cooperativa para a execução do software adaptado do GNU Chess.

No contexto dessa Seção denomina-se o processador que contém o código integral do software do jogo de Mestre. O processador auxiliar para a escolha de uma jogada chama-se Escravo.

O Mestre executa a tarefa principal identificada pelo valor 1 e chamada aqui de Tarefa 1. O Escravo por sua vez executa uma tarefa que auxilia no processamento da árvore de jogadas e recebe

requisições da Tarefa 1. Esta segunda tarefa sendo executada no Escravo é definida como Tarefa 2. Ambas as tarefas devem manter uma cópia das estruturas de dados utilizadas para representar o tabuleiro (ver Seções 2.1.2 e 2.1.3) ou, em uma implementação alternativa, acessar uma memória comum visível às duas tarefas.

O processo de comunicação entre Escravo e Mestre inicia-se quando a Tarefa 1 recebe um comando que possa alterar o estado do jogo. Os comandos que podem alterar o estado de um jogo são os comandos que fazem a movimentação de uma peça por parte do jogador ou o comando *go* que força o software a executar uma jogada. No caso em que o jogador move uma peça a Tarefa 1 deve informar a Tarefa 2 o movimento feito para que esta atualize os dados da estrutura de dados local que representa o tabuleiro.

Quando o comando *go* é recebido pela Tarefa 1 esta deve iniciar a busca por uma jogada em uma árvore de jogadas. Ao iniciar esta busca a Tarefa 1 e a Tarefa 2 devem iniciar a busca por jogadas em pontos distintos da árvore de jogadas. Para que isto ocorra a Tarefa 1 deve enviar uma mensagem à Tarefa 2 definindo a partir de qual ponto da árvore esta tarefa deve iniciar a busca por jogadas.

Ao final da busca feita pela Tarefa 1 esta deve requisitar que a Tarefa 2 envie melhor jogada que esta encontrou em sua busca. A Figura 50 mostra o fluxo de comunicação entre as tarefas 1 e 2 no processo de busca por uma jogada a partir de dois pontos distintos da árvore de Jogadas.

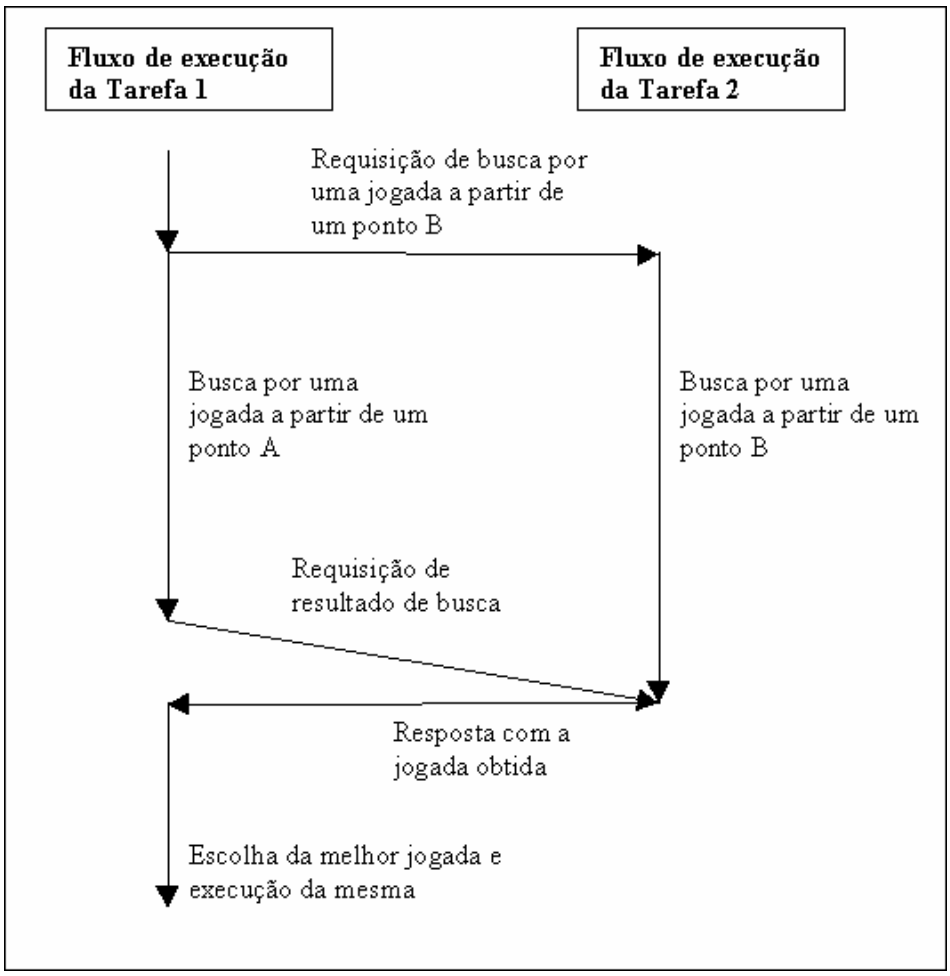


Figura 50 – Fluxo de comunicação entre Tarefa 1 e Tarefa 2 para a busca de jogada a partir de dois pontos da árvore de jogadas.

7 Validação

Este Capítulo trata da fase de validação do sistema desenvolvido neste trabalho. Aqui são mostrados os processos de validação dos módulos de hardware individualmente e do sistema como um todo. A validação do sistema é feita com trechos do software desenvolvido para o ser executado em conjunto com a parte de hardware fazendo assim uma validação geral.

7.1 Validação dos Módulos

Nas seções seguintes são mostrados os processos adotados para validar os módulos desenvolvidos. Nelas são mostrados os resultados dos testes extraídos das ferramentas utilizadas no processo.

7.1.1 Validação do Módulo de Vídeo e wrapper

Para validar o módulo de vídeo foi desenvolvido um gerador de sinais na linguagem VHDL. Este gerador de sinais conecta-se aos fios do módulo de vídeo que são ligados ao *wrapper* do controlador de vídeo.

Para validar do módulo de vídeo, o gerador de sinais produz uma seqüência de 160 valores de 32 bits (um linha da tela) que são passados para o sinal *data_in_wrapper* do módulo de vídeo. A cada ciclo de relógio é gerado um novo valor a ser inserido em *data_in_wrapper*. Este novo valor é igual ao valor enviado anteriormente acrescido de um.

Juntamente com o primeiro valor enviado para o controlador de vídeo é gerado o endereço de memória a partir de onde deverão ser escritos os dados gerados. O módulo de vídeo, por sua vez, deve repassar para o sinal *data_in* (sinal ligado à memória DDR) os valores recebidos.

Após o armazenamento da primeira palavra recebida o módulo de vídeo indica, através do sinal *ack_tx*, que o módulo ligado a este (seja o gerador de sinais ou o *wrapper* do controlador de vídeo) deve disponibilizar uma palavra por ciclo de relógio. O módulo de vídeo obedece este padrão devido ao processo de escrita do controlador de memória DDR utilizado no projeto descrito na seção 4.4.2. A Figura 51 mostra um exemplo de escrita.

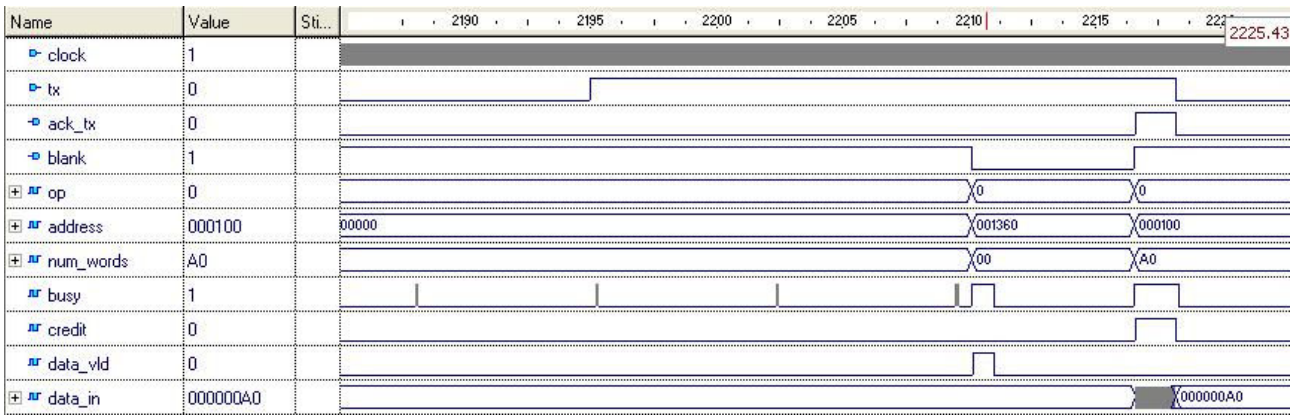


Figura 51 – Formas de onda mostrando um exemplo de escrita de uma mensagem no módulo de vídeo.

O processo de escrita na memória mostrado na Figura 51 ocorre da seguinte forma:

1. O sinal *tx* fica em nível lógico alto quando há dados para escrever na memória.
2. Após o sinal *tx* subir, a escrita dos dados é postergada até a próxima borda de subida de *blank*.
3. O controlador de vídeo atribui nível lógico alto ao sinal *ack_tx* indicando que a escrita na memória começou. A partir deste momento o sinal *data_in* deve ter uma palavra nova a cada pulso de *clock*.
4. O processo de escrita encerra na borda de decida do sinal *ack_tx*.

Após a escrita da primeira palavra na memória de vídeo o controlador de vídeo envia uma palavra por ciclo de *clock*, como mencionado anteriormente. A Figura 52 mostra em detalhe o processo de escrita dos dados através do sinal *data_in*.

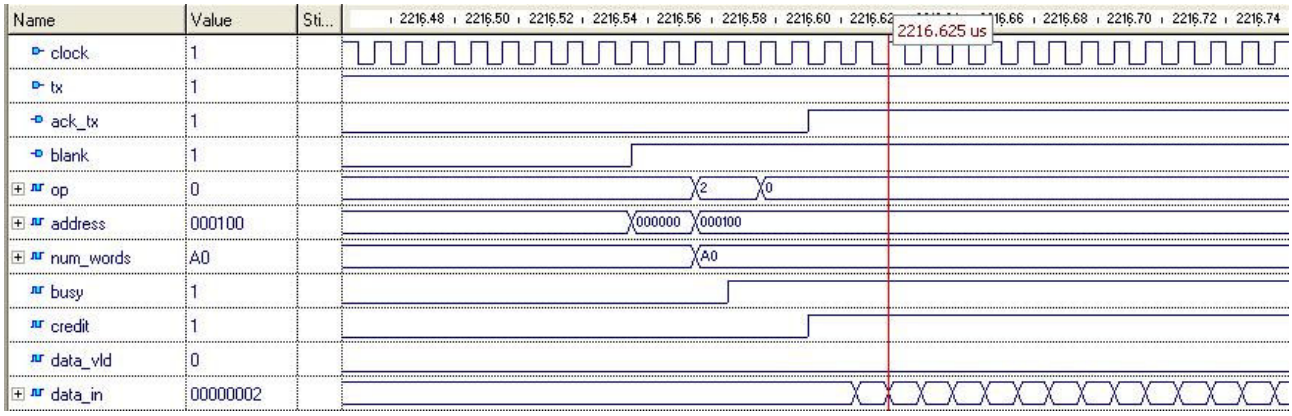


Figura 52 - Formas de onda mostrando o início de escrita de pacotes na memória de vídeo.

7.1.2 Validação do módulo de teclado e wrapper

O módulo de teclado foi validado através da análise dos pacotes enviados para a NoC. Estes pacotes foram gerados conforme o protocolo de comunicação exposto na Seção 6.1.1.

Para verificar a exatidão das mensagens geradas pelo módulo de teclado foram analisados os pacotes que fazem parte do endereçamento (ver Seção 6.1.1) e os pacotes que carregam o valor da tecla pressionada de uma mensagem de requisição de dados destinado ao módulo de teclado. No exemplo mostrado a seguir a foi recebido uma mensagem da tarefa 1 alocada no processador conectado a porta 1 (em formato hexadecimal). A mensagem de resposta gerada é mostrada na Figura 53. Esta mensagem é destinada à tarefa 1 (ver pacotes 7 e 8) endereçada pelo processador conectado à porta 1 (ver primeiro pacote da mensagem).

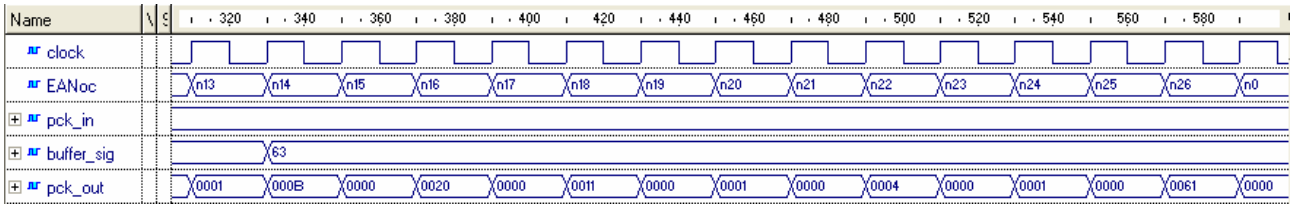


Figura 53 – Mensagem contendo o caractere “a”.

7.2 Validação do Sistema

Para validar o sistema foi executado um teste no qual é feita a entrada de um comando pelo teclado e é analisada a resposta da pelo sistema. O comando digitado é recebido pela tarefa principal e imprime na tela o resultado após seu processamento.

Primeiramente a tarefa portada para a plataforma desenvolvida faz a inicialização das estruturas de dados necessárias para a execução do jogo. Ao terminar o processo de inicialização o sistema requisita que o usuário entre com um comando. Neste momento é digitado o comando “show board” (comando que requisita a impressão do tabuleiro atual) que é transmitido caractere a caractere à tarefa principal conforme descrito na Seção 4.5.

O comando é recebido e então imprime a tela mostrada Figura 54 exibindo o estado inicial do tabuleiro de jogo.

```
Waiting for input...
Entre com o dado:
Parsing input..
white  KQkq

r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
```

Figura 54 – Tela que mostra a execução do comando “show board”.

8 Conclusões e Trabalhos Futuros

Com o desenvolvimento deste trabalho foi percebida a complexidade do desenvolvimento de um projeto deste porte. O desenvolvimento de um console multiprocessado dedicado a jogos computacionais exigem a integração de uma grande quantidade de componentes completamente distintos em seu funcionamento. Como exemplo pode-se citar o módulo de vídeo que faz uso de uma memória DDR dedicada ao armazenamento de dados e a interface VGA. Este módulo deve sincronizar períodos de escrita e leitura da memória fornecendo dados à interface VGA para geração de imagens sem perder o sincronismo que o protocolo define.

O porte do código do jogo GNU Chess também apresentou um grande desafio devido à falta de bibliotecas e funções básicas para a plataforma alvo, visto também que esta encontra-se em fase de desenvolvimento e testes iniciais não contando ainda com uma base sólida para receber aplicativos de grande porte, que é o caso deste.

O desenvolvimento de um jogo que explora as vantagens de uma plataforma multiprocessada torna-se complexo visto que, além de questões particulares da implementação de jogos, devem ser criadas soluções para garantir a integridade dos dados entre os processadores envolvidos na execução do jogo. Também deve ser dada uma especial atenção na comunicação que faz a troca de dados entre os processadores aumentando o tamanho de um programa com finalidade idêntica, mas destinado a ser executado em uma plataforma monoprocessada.

A implementação e teste deste projeto em uma placa de prototipação ficam como sugestão para trabalho futuros, visto que a implementação deste trabalho foi validada apenas através de simulação. Outra sugestão para trabalhos futuros é a paralelização do código do GNU Chess sobre os processadores que compõem a arquitetura alvo, seguindo as propostas realizadas neste trabalho.

Também é possível fazer modificações no módulo de vídeo, visando um melhor aproveitamento de tempo no processo geração de imagens. Com a configuração de vídeo adotada (ver Seção 4.4.1) existe uma janela de tempo de 25.6 us para escrita na memória de vídeo, mas menos de 7% deste tempo, aproximadamente 1,77 μs, é efetivamente utilizado. A Figura 55 mostra as formas de onda com os tempos de *blank* em nível lógico alto (janela de escrita) e o tempo efetivo de escrita em memória (sinal *busy* do controlador de memória DDR em 1).

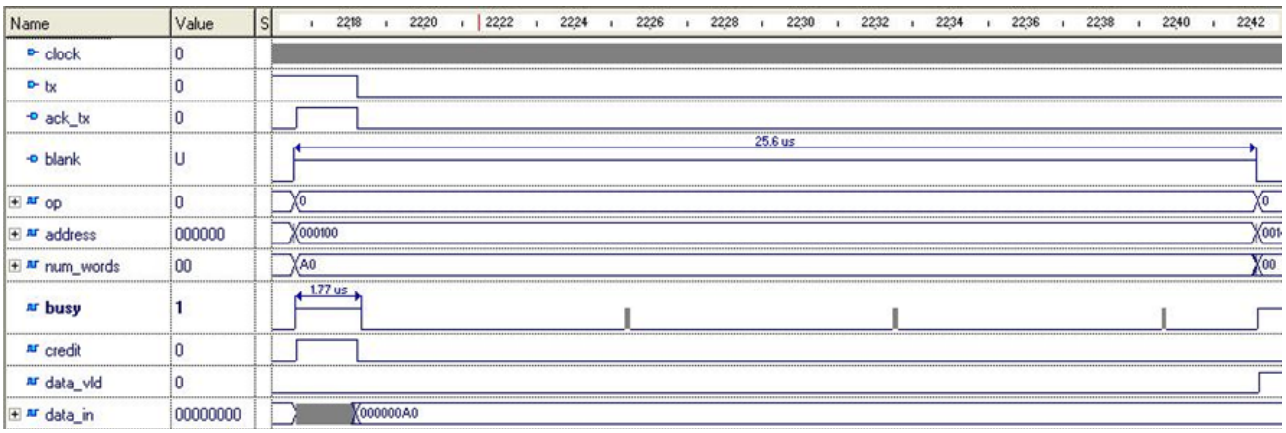


Figura 55 – Tempo de escrita de uma linha gráfica na memória de vídeo.

De forma análoga ao processo de escrita, o processo de leitura da memória de vídeo pode ser melhor aproveitado. O sinal *blank* permanece em nível lógico baixo por 6,44 us. Na implementação atual o módulo de vídeo faz a leitura de uma linha gráfica da memória de vídeo enquanto o *blank* está em nível lógico baixo. A leitura de uma linha gráfica da memória leva aproximadamente 0,86 us (aproveitamento de 13,35% da janela de leitura), tempo em que o sinal *busy*, do controlador de memória DDR, permanece em nível lógico alto. A Figura 56 apresenta o tempo de leitura de uma linha gráfica da memória de vídeo.

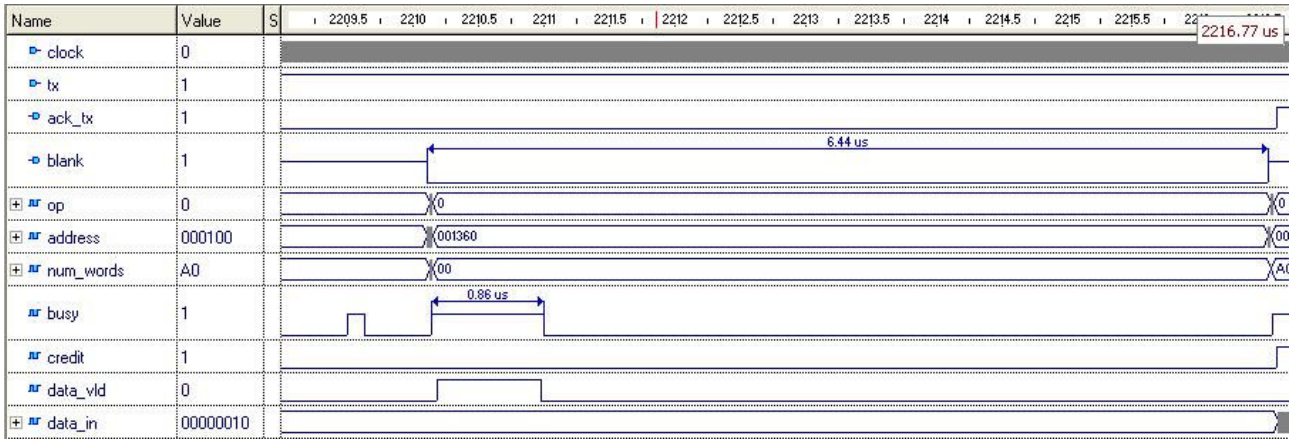


Figura 56 – Tempo de leitura de uma linha gráfica.

9 Referências bibliográficas

[AMO02] Amorin, C. A. "*A Máquina e Seus Limites: Uma Investigação sobre o Xadrez Computacional*". Dissertação de Mestrado, Universidade Federal da Bahia, Salvador, 2002, 143p.

[ART05] Arteris SA. "*A Comparison of Network-on-Chip and Busses*". White paper capturado em http://www.arteris.com/whitepapers.html, Março 2006, 11p.

[BEN02] Benini, L.; De Micheli, G. "*Networks on chips: a new SoC paradigm*". IEEE Computer, 35(1), Janeiro 2002, pp. 70-78.

[BET03] Bethke, E. "*Game development and production*". Wordware Publishing, Plano, Texas, 2003, 414p.

[CAR05] Carara, E. A. "*MPSoC-H –Implementação e Avaliação de Sistema MPSoC Utilizando a Rede Hermes*". Proposta de Trabalho Individual de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2005.

[CAR06] Carara, E. A. "*Controlador da memória DDR SDRAM da plataforma ML401*", Documento Interno, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2006.

[CRA97] Crawford, C. "*The Art of Computer Game Design*". Washington State University, disponível eletonicamente; capturado em http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html, 2006.

[DAL01] Dally, W.J.; Towles, B. "*Route packets, not wires: on-chip interconnection networks*". In: Design Automation Conference, 2001, pp. 684-689.

[EPD06] Extended Position Description. Capturado em: http://chess.verhelst.org/1994/03/12/epd/, Novembro 2006.

[FAN05] Fanet, A. "*Advantages of network-on-chip over traditional architectures*". Capturado em http://www.engineerlive.com/european-electronics-engineer/electronics-design/13335/ advantages-of-networkonchips-over-traditional-architectures.thtml, Março 2006.

[FRI06] Frictional Games. "*Penumbra*" Capturado em: http://penumbragame.com/, Junho 2006.

[GNU06] GNU. "*GNU Chess*". Capturado em http://www.gnu.org/software/chess/, Novembro 2006.

[KAM06] Kamil, A. "*Minimax*". Notas de Aula Opcionais, Curso CS61B - Data Structures, Edição 2003, University of California, Berkeley, CA, capturado em: http://www.cs.berkeley.edu/~kamil /sp03/minimax.pdf, Novembro, 2006.

[KOR05] Korogi, G. K.; Prauchner, J. L.; Fraga, L. M. "*Desenvolvimento de Jogos: Projeto RPGEDU*". Comunicação Privada, Grupo de Realidade Virtual, PUCRS, Julho 2005.

[IBM99] IBM. *"The CoreConnect Bus Architecture"*. Capturado em: http://www-306.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture, Março 2006.

[IME05] IMEC. "*MPSoC Activity*". Capturado em: http://www.imec.be/design/mpsoc/, Março 2006.

[JER05] Jerraya, A. A.; Wolf, W. "*Multiprocessor System-on-Chip*". San Francisco, California : Morgan Kaufmann, 2005, 581 p.

[JER05a] Jerraya, A. A.; Tenhunen, H.; Wolf, W. "*Multiprocessor Systems-On-Chips*", IEEE Computer, 38(7), Julho 2005, pp. 36-40.

[MAT04] Matos, S. B. "*H-PARR: Uma Heurística para Previsão de Ataque ao Roque na Ala do Rei no Jogo de Xadrez*". Monografia, Faculdade de Ciência da Computação, Faculdade Ruy Barbosa, 2004, 57 p.

[MEL03] Mello, A. V.; Möller, L. H. "*Arquitetura Multiprocessada em SoCs: Estudo de Diferentes Topologias de Conexão*". Trabalho de Conclusão, Faculdade de Informática, PUCRS, 2003, 120 p.

[MIC06] Microsoft. "*Xbox*" Capturado em: www.xbox.com/en-US/hardware/xbox360/default.htm, Junho 2006.

[MOO06] Moore, Samuel K. "*Cell's nine processors make it a supercomputer on a chip*". IEEE Spectrum, Janeiro 2006, pp. 20-23.

[MOR04] Moraes, F.; et al. "Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip". Integration the VLSI Journal, 38(1), Outubro. 2004, pp. 69-93.

[OPE06] Opencores. Fórum online.Capturado em: http://www.opencores.org/, Junho 2006.

[PAT96] Patterson, D.; Hennessy, J. L. "*Computer Architecture: A Quantitative Approach*". San Francisco, California : Morgan Kaufmann, 1996, 2nd ed., 760 p.

[PGN06] Portable Game Notation. Página online. Capturado em: http://www.very-best.de/pgn-spec.htm, Novembro, 2006.

[POP06] Pop, R. "*Specialization and Evaluation of Network on Chip Architectures for Multi-media Applications*". Capturado em: http://hem.hj.se/~poru/, Março 2006.

[ROL00] Rollings, A.; Morris, D. "*Game Architecture and Design*". The Coriolis Group, Scottsdale, Arizona, 2000, 742p.

[SOC06] Soccol, C.; Zucolotto, G. "*Implementação de Jogos em Plataformas Multiprocessadas Usando Redes Intra-chip*". Proposta de Trabalho de Conclusão I, Faculdade de Ciência da Computação, PUCRS, 2006, 20 p.

[SIL06] Silva, P. Capturado em : http://turinmachine.weblog.com.pt/arquivo/cat_o_xadrez_dos_ computadores.html, Maio 2006.

[STA96] Stakem, P. H. "*A Practitioner´s Guide to RISC Microprocessor Architecture"*, John Wiley & Sons Inc, 1996, 400 p.

[SUN95] Sun Microsistems Inc.. "*Disney's "Toy Story" Uses More Than 100 Sun Workstations to Render Images for First All-Computer-Based Movie*". Press Release, capturado em: http://www.sun.com/smi/Press/sunflash/1995-11/sunflash.951130.3411.xml, Junho de 2006.

[SWE99] Sweetman, D. "*See MIPS Run*". Morgan Kaufmann, 1999, 488 p.

[WIK06] Wikipedia. Enciclopédia online. Capturado em: www.wikipedia.org, Junho 2006.

[WOS05] Woszezenki, C. R. "*Alocação de Tarefas e Comunicação Entre Tarefas em MPSoCs*". Plano de Estudo e Pesquisa, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2005, 28 p.