Batalha Naval

Projeto - MC613

Felipe Pessina e Rafael Figueiredo Prudencio

4 de Junho de 2018

Conteúdo

1 Diagrama de Blocos

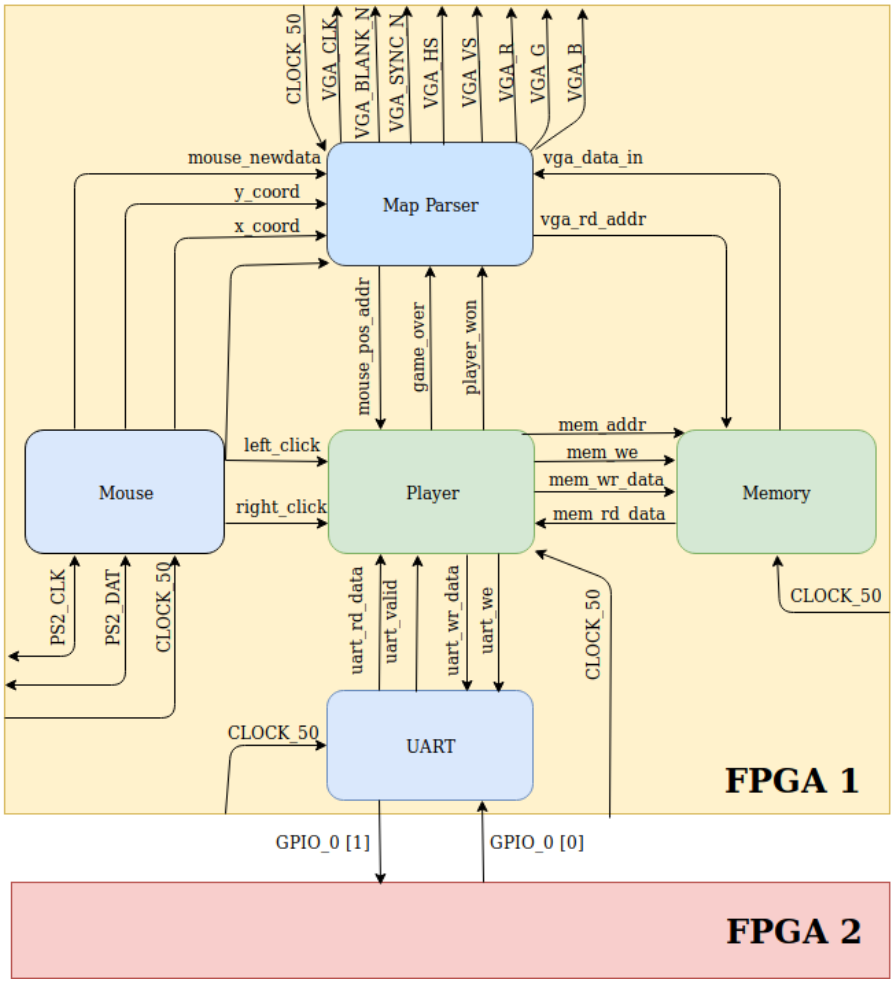

Figura 1: Diagrama de blocos de alto nível para uma implementação do batalha naval. Os sinais com nomes em letras maiúsculas são entradas e saídas da placa. O bloco FPGA2 representa a placa para o segundo jogador com um design análogo à FPGA1. Os blocos verdes são entidades singulares e os blocos azuis representam a entidade de alto nível usada para comunicar-se com os dispositivos de IO. Cada bloco azul tem um diagrama de blocos detalhando as entidades que o compõem.

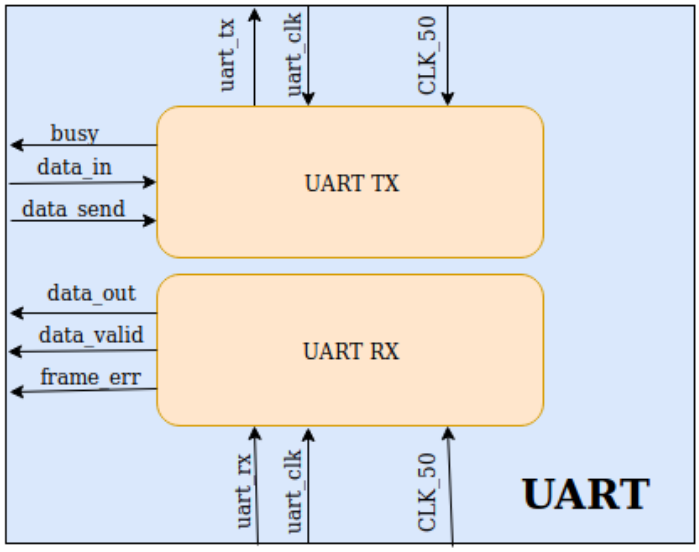

Figura 2: Diagrama de blocos das entidades usadas pela entidade uart para realizar a comunicação entre as duas FPGAs.

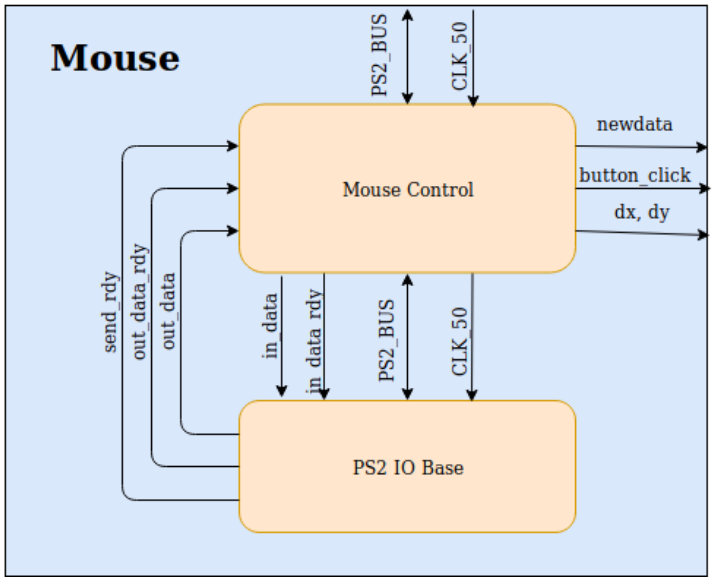

Figura 3: Diagrama de blocos das entidades usadas pela entidade ps2_mouse para comunicar-se com o mouse.

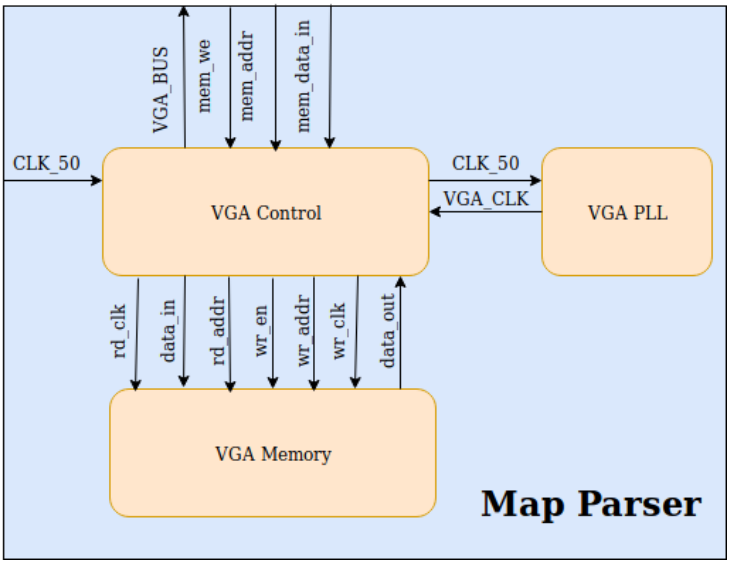

Figura 4: Diagrama de blocos das entidades usadas pela entidade map_parser para desenhar o estado do jogo no display VGA.

2 Descrição Funcional

2.1 MIF

Os arquivos MIF utilizados no projeto não pertencem ao diagrama de blocos mas devem ser mencionados, pois são uma componente essencial para o funcionamento correto do jogo. Ao todo, foram gerados três arquivos MIF: vga_mem.mif, initial_map_p1.mif e initial_map_p2.mif. O arquivo vga_mem.mif é usado na hora de inicializar a memória do display VGA na entidade vga_con. Através de um script em python, desenhou-se o título do jogo "Battleship". Os demais MIFs foram usados para inicializar a memória principal de cada jogador, um para cada placa. Através de um script em python, gera-se os dois MIFs a partir de dois arquivos de texto, um para cada jogador, com uma matriz 10x10 de caracteres onde 'X' indica um barco e '-' inidica água. Cada mapa de 100 posições é representado na memória através de um vetor de 100 posições. O mapa do jogador está nos endereços 0 a 99 e o mapa do oponente está nas posições 128 a 227. Além dos mapas, os MIFs incluem os desenhos das palavras "Vitória"e "Derrota"que serão lidos da memória ao término do jogo.

2.2 Memória

O bloco Memory da Figura 1 é de uma memória de duas portas com largura de dados de 8 bits e largura de endreços de 15 bits utilizada para manter o estado dos mapas dos dois jogadores.

Foi necessário usar uma memória de duas portas pois haviam duas entidades que precisavam fazer uma leitura simultânea da memória: player e map_parser. Os sinais vga_data_in e vga_rd_addr dão acesso de leitura à entidade map_parser, que precisa ler constantemente os vetores que representam os mapas dos jogadores para manter o display atualizado. Os sinais mem_addr, mem_we, mem_wr_data e mem_rd_addr são usados para realizar a leitura e escrita na memória por parte do jogador, que muda o estado do mapa do inimigo sempre que realiza uma jogada e tem o seu mapa alterado sempre que o oponente realiza uma jogada. Os vetores de 100 posições que representam os mapas de cada jogador indicam o estado do jogo através de 4 flags: barco, água, barco_atingido e água_atingida.

2.3 I/O

2.3.1 UART

As duas FPGAs comunicam-se através de uma interface UART, como indicado na Figura 1. Para realizar a comunicação foi necessário obter um cabo ATA de HD de 40 pinos fêmea para 40 pinos fêmea [1] e conectá-lo aos pinos GPIO_0 das duas placas. O código para a interface UART foi obtido online [2] e o diagrama de blocos é dado pela Figura 2. Apesar do repositório incluir código para usar bits de paridade na transferência de dados, não achou-se necessário incluí-los no protocolo de comunicação devido à simplicidade do projeto. Dos 36 pinos para transferência de dados, foram utilizados apenas dois, um para a saída TX da FPGA1 e entrada RX da FPGA2 e outro para a entrada RX da FPGA1 e saída TX da FPGA2.

A única entidade que comunica-se com a interface é a do jogador, que envia e recebe dados da outra placa através dos sinais uart_wr_data, uart_we, uart_rd_addr e uart_valid. A abstração da entidade UART permite uma comunicação quase transparente com a outra placa, já que a transferência de dados acontece analogamente a uma escrita ou leitura na memória, exceto pelo sinal uart_valid, que indica quando o dado pode ser lido seguramente do sinal uart_rd_data. Os dois sinais para transferência de dados transmitem e leem um byte por vez. A interface UART é responsável por transferir a posição do mapa do oponente que foi atingida pelo jogador e ler a posição que o oponente atingiu do mapa do jogador.

2.3.2 Mouse

Cada placa FPGA terá um mouse como dispositivo de entrada. O bloco Mouse na Figura 1 representa a entidade ps2_mouse que, por sua vez, está representada na Figura 3 e utiliza as entidades mouse_ctrl e ps2_iobase dadas no laboratório 7 para obter os dados de entrada do mouse. A entidade ps2_mouse é uma pequena modificação da entidade mouse_test, onde criou-se sinais de saída para as informações do mouse relevantes às demais entidades.

O bloco Mouse comunica-se com as demais entidades através de cinco sinais de saída. Para a entidade player, apenas interessam os sinais indicando se

houveram cliques com os botões direito e esquerdo do mouse. O botão esquerdo é o que o jogador deve utilizar para selecionar onde no mapa deseja atirar e o botão direito foi transferido caso se optasse por implementar uma funcionalidade de jogar novamente. Como essa funcionalidade não teria sentido dado que o mapa do jogador é carregado estaticamente através de um arquivo MIF em tempo de compilação, o sinal acabou não sendo utilizado. Os sinais de saída para a entidade map_parser são as coordenadas x e y relativas à posição inicial do mouse, um sinal que indica quando há novos dados e o sinal para um clique com o botão esquerdo. Esses sinais foram utilizados pelo bloco Map Parser para desenhar o cursor do mouse e determinar qual posição no mapa foi clicada para então enviá-la à entidade player.

2.3.3 VGA

O bloco Map Parser na Figura 1 é responsável pela saída no display VGA. Uma descrição mais detalhada do bloco Map Parser é dada pela Figura 4, onde foram utilizadas as mesmas entidades do laboratório 8: vga_con e vga_pll junto com suas dependências. A entidade map_parser utiliza so sinais de entrada do mouse para desenhar o cursor e enviar a posição no mapa onde foi feito o clique para a entidade player. Os sinais vga_data_in e vga_rd_addr são utilizados para realizar uma leitura da memória principal. É a partir dessa leitura que a entidade map_parser mantém atualizado no display os mapas de ambos os jogadores. Os sinais game_over e player_won são duas entradas que o player envia indicando que o jogo acabou e qual jogador é o vencedor. Esses sinais são usados para desenhar a tela de vitória ou derrota para os dois jogadores após o término do jogo. O sinal de saída mouse_pos_addr indica à entidade player a posição atual do cursor no mapa do oponente.

2.4 Jogador

O bloco Player na Figura 1 representa a entidade player e é responsável pelo tratamento das jogadas. A entidade player implementa uma máquina de estados com os estados wait_turn, wait_read, read_hit, wait_click, val_click, upd_map e end_game.

Os estados wait_turn, wait_read e read_hit são usados para esperar uma jogada do oponente e atualizar a memória do jogador apropriadamente. O estado wait_turn usa o sinal de entrada uart_valid para determinar se pode prosseguir. O estado wait_read e read_hit usam os sinais mem_addr, mem_we, mem_wr_data e mem_rd_data para ler a posição atingida e atualizá-la na memória principal. Enquanto o oponente acerta barcos, o jogador espera por sua vez, alternando entre esses estados.

Os estados wait_click, val_click e upd_map usam o sinal de entrada left_click e mouse_pos_addr para saber quando houve um clique e qual a posição no vetor da memória principal que ocorreu o clique. A partir desses sinais, determina-se se o clique é válido, sendo um click inválido um que ocorreu em uma posição do vetor que já foi descoberta. No estado upd_map, já foi feita a validação do

click e então atualiza-se a memória com o tiro do jogador. Alterna-se entre esses estados enquanto o jogador acertar barcos e muda-se para o estado wait_turn no caso de um acerto em água.

Finalmente, o término do jogo é dado pelo acerto em todos os barcos do mapa. O estado end_game pode ocorrer tanto quando destrói-se todos os barcos do oponente como quando todos os barcos do jogador são destruídos. Durante o estado end_game, envia-se o sinal game_over para indicar o término do jogo e dependendo da condição do término, envia-se no sinal player_won o vencedor do jogo para a entidade map_parser. O jogador que começa a partida deve começar no estado wait_click, pois é sua vez, enquanto o outro começa no estado wait_turn, pois deve esperar a primeira jogada do oponente.

3 Referências

[1] "Lindy connection perfection." `https://www.lindy.co.uk/cables-adapters-c1/i-o-c125/0-6m-2-5-ide-hard-disk-drive-cable-p3707`. Acessado em 2018-06-02.

[2] J. Cabal, "Uart for fpga." `https://github.com/jakubcabal/uart-for-fpga`, Aug 2016. Acessado em 2018-06-02.