**O QUE DEVERÁ TER NO RELATÓRIO**

|  |  |
| --- | --- |
| **Acordado com PC e Padrão IEEE** | **Pedido nas especificações e pensado por Lucas** |
| **Resumo (PARECE OK)** | **. O que consta no artigo, como um todo (SEI)**  **. básico da Apresentação do problema (o que foi construído e correlação com o pedido) (NÃO SEI TODO O CONSTRUÍDO, RESTO SEI)** |
| **Introdução (PARECE OK)** | **. Apresentação detalhada do problema (do que foi pedido) (SEI)** |
| **Fundamento teórico (QUASE COMPLETO, MAS PARECE FALTAR PRÓPRIAS PALAVRAS PARA DEMONSTRAÇÃO DO QUE AQUELE CONHECIMENTO É DE FATO IMPORTANTE PARA FAZER O JOGO)** | **. Aulas de Sistemas Lógicos e Eletrônica Digital (aprendizado de Máquina de estados e Verilog)**  **. Conceito e descrição básica de FPGA**  **. Conceito e descrição básica de Verilog, livro usado para ampliar habilidades para a linguagem**  **. Software EDA Quartus Prime e ModelSim**  **. VGA e descrição básica (pesquisas da net que tem nas atas dia 10 e 17) (PARTE AVANÇADA VAI NO DESENVOLVIMENTO)** |
| **Desenvolvimento (SEI +-), olhar** | **. Descrição dos módulos usados para a lógica (Unidade 1) e VGA (Unidade 2)**  **-> partes constituintes e correlação**  **-> função**  **-> estudos realizados**  **-> propostas dos mapas (ata dia 3)**  **-> trechos muito relevantes de códigos**  **. Processo de simulações e evolução na construção do jogo**  **. Testbenches** |
| **Resultados** | **. Simulações específicas (muito importantes) e previsão do funcionamento na placa**  **. Funcionamento final (na placa), apontando acertos e possíveis melhorias** |
| **Conclusão** | **Reflexão sobre acertos e possíveis melhorias:**  **-> acertos e aprendizado**  **-> abrangência de implementação**  **-> problemas encontrados (dificuldades)**  **-> possíveis melhorias** |
| **Referências (NÃO ENTRA NAS 10 PÁGS)** | **Referências: site e livros (com indicação de notas [x] utilizadas em todo o artigo)** |
| **Anexos (NÃO ENTRA NAS 10 PÁGS)** | **. FSMs relevantes**  **. prints do Quartus com diagrama de blocos da entidade geral**  **. Imagem da placa rodando o jogo** |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**- 4.: Colar descrição e adaptar, sobre cada sinal dos módulos (todos já foram listados); comentar função geral; ver se naquele formato ou tabela**

- FUNÇÕES DE RAM E ROM (tá usando ROM??)

- **FORMATAÇÃO, TÓPICOS, FIGURAS, PREENCHER CARACTERES ASCII e DESENHO DOS BARCOS**

- **REFERÊNCIAS**

- **ANEXOS: DIAGRAMAS E BDF GERAL**

**0. RESUMO**

Este relatório descreve a solução proposta para o jogo Batalha Naval, solicitada pelos professores Paulo César Farias e Wagner Oliveira para a disciplina Laboratório Integrado I. Ele constitui num tabuleiro quadrado com dois modos de jogo, embarcações de cada jogador e placar atualizado. Sua montagem fez uso dos softwares EDA Quartus Prime[1] e ModelSim[2], da Altera, e da descrição do hardware FPGA[3] em Verilog HDL[4]. Também consta aqui a implementação da interface de vídeo VGA para comunicação com o usuário final e os resultados de simulação do jogo, testes na Placa DE2-115 [5] e exibição em monitor de vídeo. Os resultados alcançados são fruto de estudos, discussões e reuniões entre os membros da equipe, cuja comprovação foi feita por atas confeccionadas semanalmente, contendo metas, ideias, fatos e questões de cada sessão PBL (Aprendizado baseado em problemas). Referências utilizadas estão destacadas e apresentadas em lista no final deste artigo, juntamente com diagramas/esquemas/imagens importantes às etapas da solução, apresentadas como anexos.

PALAVRAS-CHAVE: Verilog, Batalha Naval, FPGA, VGA.

**1. INTRODUÇÃO**

A lógica do jogo Batalha Naval e interface de vídeo VGA foram separadas em uma unidade cada, com seus respectivos prazos de entrega. Para a implementação em placa FPGA, utilizou-se o software EDA Quartus Prime para descrição do hardware em linguagem Verilog, simulação e construção dos testbenches com o ModelSim, e envio para a placa.

A interface do jogo exigiu informações relevantes para a compreensão do mesmo, as quais são:

- Tabuleiro quadrado de 64 posições;

- Posicionamento horizontal ou vertical, numa quantidade limitada, das seguintes embarcações: (a) Porta-aviões (b) Encouraçado (c) Hidro-avião (d) Cruzador (e) Submarino

a) b) c) d) e)

- Dois modos de operação: Player1 vs. Player2, Player1 vs. CPU

- Placar eletrônico iniciado em 0, além de indicação de acerto, derrubada de embarcação inteira (incrementa pontuação correspondente) e tiro na água (decrementa 1 ponto).

- Mensagem de indicação do vencedor, quando este derruba todas embarcações do adversário.

A estratégia adotada pela equipe foi dividir a implementação entre seus membros durante as sessões PBL realizadas nas segundas-feiras junto ao professor Paulo Cesar Farias, além de reuniões online. Máquinas de estado são parte muito significativa da implementação em código, pois era necessário coordenar entradas, saídas e sinais internos dentro dos módulos e, após isso, interligar todos os módulos responsáveis pela lógica e interface gráfica.

**2. FUNDAMENTO TEÓRICO**

Tomando o conhecimento base de Máquina de Estados e Verilog, vistos respectivamente nas disciplinas Sistemas Lógicos (ENGC26) e Eletrônica Digital (ENGC40), foi possível construir os diagramas necessários e realizar parte da implementação em código. Parte significativa do conhecimento utilizado foi obtido somente na disciplina de laboratório, com as aulas ministradas pelo professor Wagner, além de material externo como livros e apostilas, indicadas nas referências. Segue descrição básica dos recursos utilizados durante a construção da solução:

**2.1 FPGA (Field-programmable gates array):** Categoria de Circuito Integrado de Aplicação Específica (ASIC) cujas portas (lógicas) são reprogramadas diretamente pelo usuário. Como exemplo de FPGA, há a DE2-115, da Altera, exibida em anexo.

**2.2 Verilog HDL (Verilog Hardware Description Language):** Linguagem para especificação direta pelo usuário dos recursos utilizados do FPGA. Ela é atualmente a mais utilizada dentre todas as linguagens de descrição e foi criada no início dos anos 90. Através de software adequado, é possível descarregar o código Verilog na placa, indo para matrizes de portas lógicas combinacionais e sequenciais que se ajustam para atender à descrição indicada pelo programador. As unidades funcionais são chamadas módulos, cujo detalhamento do jogo está no escopo deste artigo.

**2.3 Softwares EDA Quartus Prime e ModelSim:** Ambos são da empresa Altera, destaque mundial na produção e comercialização de placas, Circuitos integrados (CI’s) e softwares afins. O Quartus pode ser usado para descrição e implementação de hardware em diagrama de blocos ou linguagens como Verilog, VHDL e System Verilog. Já o ModelSim serve para simular o circuito digital construído, sendo possível observar as saídas em forma de onda (waveform) para entradas especificadas pelo usuário; com ele também é possível criar testbenches para validação do código.

**2.4 Testbench:** Genericamente, é uma técnica utilizada para avaliação do desempenho/resposta de um modelo ou programa. No uso do Verilog, arquivos de testbench são produzidos na mesma linguagem e possuem a mesma extensão “.v”; a diferença é que arquivos de descrição comuns devem ter equivalente em circuito lógico, ou seja, devem ser sintetizáveis pelo software utilizado, enquanto que os testbenches não precisam, pois serão somente para simulação. Atrasos no tempo (delays) podem ser colocados, por exemplo, para respeitar o atraso comum de uma porta lógica, da ordem de 10ns.

**2.5 VGA (Video Graphics Array):** Padrão de vídeo utilizado desde a época dos monitores de tubo, na qual a imagem é formada pela emissáo de életrons sob uma superfície coberta por material fosforecente, o que gera um ponto na tela. O conector possui 15 pinos. 2 deles definem sinais de parâmetros de sincronização vertical e horizontal (VSYNC e HSYNC), enquanto que outros 3 controlam a cor de cada pixel (menor unidade de uma imagem) através de uma composição entre vermelho, verde e azul (RGB). O ajuste da imagem no monitor é feito pelo VSYNC e HSYNC, construindo cada pixel a partir de um pulso de clock cuja frequência é tabelada seguindo padrão que considera resolução e taxa de atualização.

FIG. 1[.]

De posse dos conhecimentos acima e com o estudo de técnicas como manipulação de memórias, foi possível dar início à estrutura funcional do jogo, cuja construção é explicada no próximo item.

**3. PROJETO DO MÓDULO**

**3.1. ETAPAS RELEVANTES PARA A CONSTRUÇÃO DO JOGO**

**3.1.1 Exibição de imagem em monitor VGA:** O padrão VGA funciona basicamente com a impressão ordenada dos pixels que compõe uma imagem. Para exibir o mapa do jogo de forma adequada, foi necessário conhecer a fundo esta tecnologia, cuja correlação com o trabalho é demonstrada abaixo.

Partindo da ideia de um tabuleiro quadrado 8x8, a equipe dividiu a tela entre mapa, pontuação e indicação da vez de cada jogador. No mapa, explorou-se seus espaços como posições de água ou embarcações. Da resolução adotada (640x480) e taxa de atualização (60 Hz) foram buscados parâmetros de temporização para sincronização vertical e horizontal, que constam na tabela:

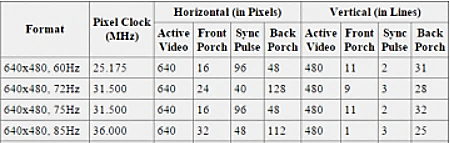


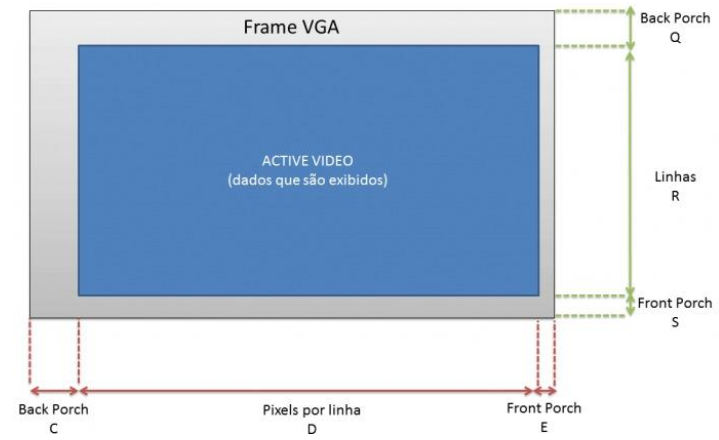
FIG..

Depreende-se que os valores escolhidos para o projeto são os da primeira linha.

- 3.1.1.1 O Frame VGA:

O ajuste do conteúdo a ser exibido por uma tela VGA segue um modelo de “margens” bem definido. Os valores padrão desses atributos visam a melhor impressão da imagem, sem perigo de distorções e falhas de temporização. Seus valores são definidos tanto para horizontal quanto para vertical. A imagem no VGA é formada instantaneamente devido à escrita em altíssima velocidade da cor de cada pixel, da esquerda para direita numa linha e de cima para baixo em cada coluna.

A imagem a seguir ilustra a importância de cada parâmetro:



- Active Video (Região Ativa ou RA): faixa de exibição de conteúdo.

- Back Porch: largura da margem anterior à contagem dos pixels da região ativa

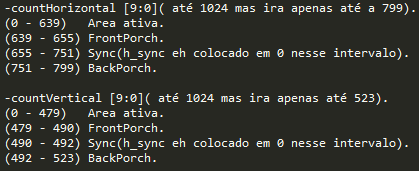
- Front Porch: largura da margem que é alcançada ao final da contagem dos pixels da RA.

- Pixels por linha: Quantidade de pixels em cada linha da RA, 640, no projeto.

- Linhas: Quantidade de linhas da região ativa, sendo o segundo valor da resolução. No nosso caso, 480.

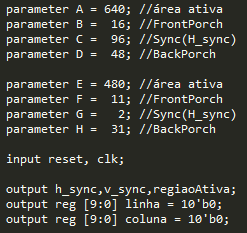
OBS: É importante ressaltar que, o tamanho total do Frame VGA na horizontal é dado por C+D+E e na vertical, Q+R+S. Como a contagem dos pixels ocorre além da RA, os registradores que guardam as “coordenadas” do pixel trabalhado contar além de 640.

Os prints a seguir mostram como se definiu os parâmetros



**É definida a quantidade de pixels para cada região do Frame VGA, tanto na vertical como na horizontal. Os registradores countHorizontal e countVertical guardam os valores nas duas dimensões do pixel que é colorido naquele momento. Eles são de 10 bits (até 1023) para comportar todo o tamanho do Frame VGA.**

Os parâmetros HSYNC e VSYNC, ambos ativos em nível lógico baixo, indicam respectivamente, quando foi percorrida toda a largura e altura do Frame VGA. Quando HSYNC atinge 0, o valor de countHorizontal é zerado. Analogamente, countVertical é zerado no momento em VSYNC vai para 0.



**Os valores de cada atributo são passados como parâmetro, isto é, sinal interno do sistema. As saídas são os valores dos registradores “linha” e “coluna” (coordenadas do pixel), inicialmente definidas em 0, incrementadas adequadamente em cada pulso de clock e reiniciadas em cada reset.**

3.1.1.2 Circuito PLL e o Clock da VGA

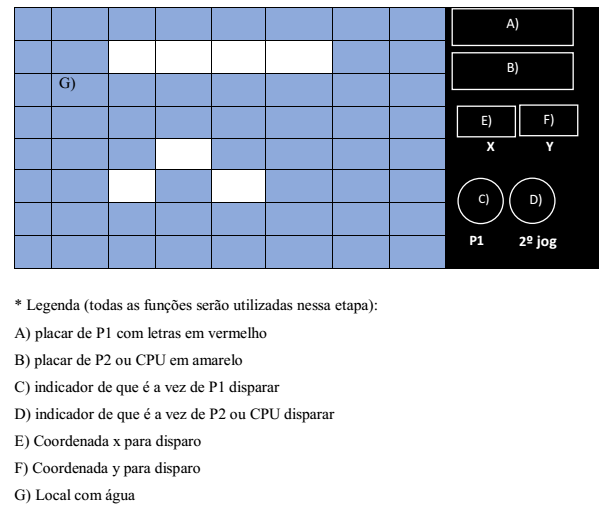
De acordo com a Tabela de Temporização (fig...) era necessário clock de 25,175 MHz para os sinais da VGA. Como a placa DE2-115 oferece clock nativo de 50MHz, foi necessário dividir a frequência pela metade. Para isso, foi utilizado o circuito PLL[.] (traduzido do inglês como “Malha de Captura de Fase”) oferecido pela mesma, sendo sua saída a frequência do sinal de entrada multiplicado por 0,5.

ESQUEMA, como na fig. 7 do arquivo TCCSTS..

Este recurso foi implementado automaticamente com o uso do software Quartus Prime através de descrição estrutural do circuito, sem a necessidade de codificação pelos membros. A altíssima frequência do clock explica o efeito da formação imediata de uma imagem na tela. Apesar de cada pixel ser sequencialmente colorido pela composição RGB, a visão humana não consegue detectar esse curto intervalo de tempo.

3.1.1.3 Construção do mapa

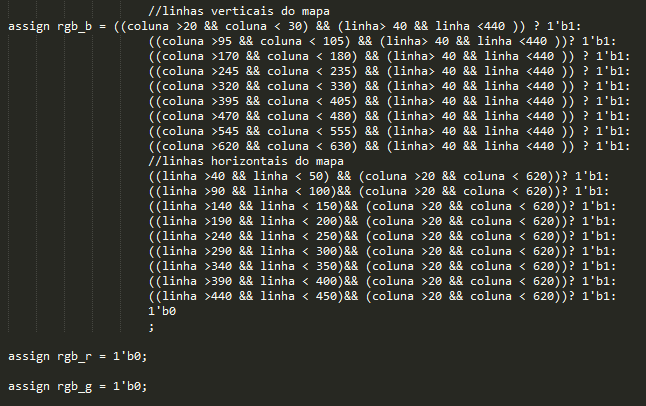
Conforme acordo entre o grupo na segunda sessão PBL da 2ª unidade, a interface do jogo deveria conter três telas: modo de jogo (Tela 1), execução (Tela 2), mensagem do vencedor (Tela 3). A proposta da tela de maior destaque, a de execução do Batalha Naval, é ilustrada a seguir.



Após período de estudo sobre a plotagem de figuras na tela, deu-se início à construção do mapa da Tela 2 e impressão dos caracteres alfanuméricos necessários, indicado nos itens consecutivos.

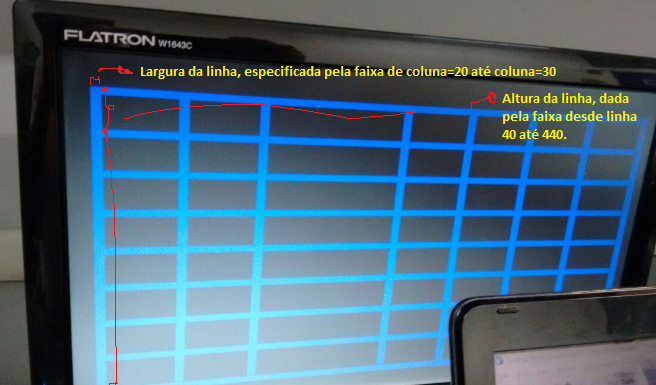
3.1.1.3.1 Linhas delimitadoras de cada posição do tabuleiro

Plotar figuras básicas na tela não foi tarefa difícil, apesar de trabalhoso. Essa tarefa usa técnica bastante “primitiva”, uma vez que a cor de todos os pixels precisa ser especificada através de codificação direta. Para montar as linhas verticais e horizontais que delimitam cada posição, elas tratadas como retângulos, sendo suas dimensões em largura e altura, dados respectivamente pelos valores de “coluna” e “linha”, como pode ser visto abaixo.



**FIG... Nas faixas de pixels especificadas acima, os valores para saída digital “rgb\_b” (azul) são 1, enquanto rgb\_r (vermelho) e rgb\_g (verde) são 0, definindo neles somente a cor azul.**

**- ??? Se saída VGA dos sinais RGB é analógica, pq aqui só se especifica os valores 0 e 1? Onde está a relação desse código com a composição RGB em que cada cor pode ser reg. de 8 bits (de 0 até 255)**



**Resultado em tela do trecho de código da fig... . Dentre todas as colunas, a de maior largura foi posteriormente corrigida, conforme avanço dos testes.**

O esquema geral da Tela 2 (Execução do jogo) segue como anexo a este relatório. Nele é possível visualizar também as dimensões em pixels para a grade e as embarcações, otimizados com utilização de equações para garantir o aspecto simétrico do mapa.

3.1.1.3.2. Desenho das embarcações

3.1.1.3.3 Impressão de caracteres ASCII na tela

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4 DESCRIÇÃO DOS MÓDULOS**

**4.1 MÓDULOS DA PARTE LÓGICA (1ª UNIDADE)**

**OBS: Alguns sinais de entrada são comuns a vários módulos, sendo que cada sinal assume a mesma função em todos. Segue descrição dos mesmos:**

- clk: sinal de clock nativo da placa. Como todos os módulos são descritos através de máquinas de estado, é preciso um sinal que sincronize os eventos/sinas do sistema. Esse é o papel do clock em todos os módulos.

- enable: Habilita o funcionamento de todos os módulos nos quais está presente.

- ready: Indica a conclusão da tarefa/função do respectivo módulo e serve como gatilho para a execução de outra tarefa.

- Enter: sinal de push-button (ativo em 0) da FPGA que representa botão de confirmação pelo Player.

- Select: outro push-button da FPGA (também ativo em 0), que aumenta o valor da coordenada em questão, sendo X ou Y.

- mode: modo de jogo escolhido por switch na placa. Seu valor determina a execução ou não de tarefas específicas, por exemplo, a aleatoriedade para montagem do mapa da CPU. Valor 0 é P1Xcpu; 1 é P1XP2.

SINAIS ESPECÍFICOS

4.1.1. COLISOR (REVER PARA FICAR DE ACORDO COM OS .BDF) :

- Verifica o acerto de tiro na embarcação inimiga

- Exclui da memória caso a coordenada seja encontrada

4.1.1.1. Entradas

- x[3...0]: Vetor de 4 bits que armazena correspondente em binário para a coordenada de disparo direção x, que pode estar entre 1 e 8.

- y[3...0]: Análogo para a coordenada y.

- clk

- enable

- jogador: Sinal de um bit que representa o jogador que atira: caso 0 (Player 1), caso 1 (Player2/CPU)

- memóriaP1[63..0]: vetor de coordenadas lido da memória do Player 1.

- memóriaP2[63..0]: análogo para o Player 2.

4.1.1.2 Saídas

- Hit: indica se parte da embarcação do oponente foi derrubada: caso 0, não houve acerto; caso 1: houve acerto.

- Wrep1: Habilita o módulo Colisor a escrever na memória do Player1.

- Wrep2: Análogo para a memória do Player2.

- Tiro [7..0]: vetor de saída com a coordenada do disparo.

- Addr[4..0]: endereço da memória que o módulo utiliza para localizar cada barco e ler suas posições.

- Clear[7..0]: Zera, na memória, o valor da posição recém-atingida que continha parte de embarcação.

4.1.2 EXECUTANDO JOGO:

- Responsável por intermediar as dinâmicas durante a excução do jogo - ataque, tiro

- A entrada acertou\_tiro vem da Maquina Colisor

- A mudança dos estados verificando\_acertop1 e verificando\_acertop2 depende da borda de descida do enter,

do acertou\_tiro ALTO e da borda de subida do clock.

4.1.2.1 Entradas

- Enable

- Reset:

Enter:

- Select

Clk

Mode

Posição\_rnd[3..0]: Posicao aleatoria vinda do gerador aleatorio de coordenadas X e Y. Este sinal é útil caso o jogo esteja no modo P1xCPU.

Acertou\_tiro: indica se tiro disparado pelo Player em questão acertou oponente. Se sim, seu valor é 1, caso contrário, 0.

Qtd\_P1[3..0]: quantidade restante de peças na memória do Player1.

Qtd\_P2[3..0]: analógo para o segundo jogador, seja ele Player2 ou CPU.

4.1.2.2 Saídas

Ready

Coord\_tiroX[3..0]: coordenada X do tiro

Coord\_tiroY[3..0]: coordenada Y do tiro

4.1.3 POSICIONANDO PEÇAS

4.1.3.1 Entradas

- Enable

- Reset:

Enter:

- Select

Clk

Mode

- Conflito

4.1.3.2 Saídas

- Ready

- Valida

- Tipo[2..0]

- Jogador

- X1[3..0]

- Y1[3..0]

- direção

- orientação [2..0]

4.1.4. POSICIONANDO PECAS\_RANDOM

-4.1.4.1 Entradas

- Enable

- Reset:

- Clk

- Direcao\_rnd

- Posicao\_rnd[3..0]

- Orientacao\_rnd[2..0]

- Conflito

- 4.1.4.2 Saídas

- Ready

- Valida

- direção

- tipo[2..0]

- orientação[2..0]

- x1[3..0]

- y1[3..0]

- jogador

4.1.5 RANDOM

- 4.1.5.1 Entradas

- clock

- reset

- 4.1.5.2 Saídas

- rnd\_direcao

- rnd\_posicao[3..0]

- rnd\_orientacao[2..0]

4.1.6 VALIDADOR: - Responsavel por fazer a construção e validação das peças

- Verificar se a posição de inserção é válida - colisões de borda e outras peças

- 4.1.6.1 Entradas

- Enable

- Clk

- direção

- orientação[2..0]

- x1[3..0]

- y1[3..0]

- jogador

- vetor\_leitura[63..0]

- tipo[2..0]

- 4.1.6.2 Saídas

- ready

- conflitoMemoria\_out

- conflitoBorda\_out

- wrep1

- wrep2

- vetor[63..0]

- read\_addr[4..0]

- write\_addr[4..0]

- conflito

4.1.7 MEMÓRIA\_A e MEMÓRIA\_B:

- 4.1.7.1 Entradas

- address[4..0]

- clk

- data[63..0]

- wren

- 4.1.7.2 Saída

- q[63..0]

**4.2 MÓDULOS RESPONSÁVEIS PELA INTERFACE VGA (2ª UNIDADE)**

4.2.1 VGASync

- 4.2.1.1 Entradas

- reset: volta para posição inicial

- clk: clock gerado pelo PLL

- 4.2.1.2 Saídas

- h\_sync: sincronizador horizontal usado no VGA para exibição das linhas

- v\_sync: sincronizador vertical usado no VGA para do frame na tela

- linha[9..0]: especifica a linha trabalhada naquele momento

- coluna[9..0]:

- regiaoAtiva: sinaliza se linha e coluna em questão da resolução escolhida (640x480)

??ENTRADA: um valor de linha e outro de coluna

SAÍDA: um valor para R, para G e para B.

- Aprendizado: exibição de formas/figuras em VGA

4.2.2 VGA\_pll\_top

- 4.2.2.1 Entradas

- 4.2.2.2 Saídas

4.2.3 PLL\_clk: a cada pulso de clock da entrada, faz divisão da frequência do clock de entrada por 2, isto é, utilizou-se sua propriedade de multiplicação por 0,5.

-4.2.3.1 Entrada: inclk0

- 4.2.3.2 Saída: c0

4.2.4 VGA\_MapaGame: Ajuda a compor o mapa, delimitando, através de quadrado de contorno azul, cada posição.

- 4.2.4.1 Entradas

- 4.2.4.2 Saídas

4.2.5 VGA\_renderiza\_jogador: Busca os dados das embarcações na memória e dispoe nos módulos que possuem as embarcacoes com as informações de cada posição de embarcação os dados de linha e coluna do vga\_sing

- 4.2.5.1 Entradas

- 4.2.5.2 Saídas

4.2.6: VGA\_Submarino, VGA\_Cruzador, VGA\_Encouraçado,, VGA\_Hidroaviao, VGA\_PortaAvioes

4.2.6.1 Entradas:

- clk,

- readEnabled,

- areaAtiva,

- linha,

- coluna,

- posicoesEmbarcacao, // Para o submarino, as posicoes X e Y. Varia de 3-10 bits, 4 primeiros para X e 4 ultimos para Y, totalizando 8 bits

4.2.6.2 Saídas

- rgb\_r,

- rgb\_g,

- rgb\_b

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**4.3 Processo de simulações e evolução na construção do jogo**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**4.4 OS TESTBENCHES**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

5. PROBLEMAS ENCONTRADOS E POSSÍVEIS MELHORIAS: Dentre os problemas encontrados durante a implementação, todas as dificuldades técnicas foram resolvidas com pesquisas em referências e contato com os professores orientadores. A tabela a seguir cita problemas em tarefas específicas, bem como melhorias que poderiam, com mais tempo, ser implementadas.

|  |  |  |  |
| --- | --- | --- | --- |
| **DIFICULDADE/**  **PROBLEMA** | **ABRANGÊNCIA DE IMPLEMENTAÇÃO** | **DESCRIÇÃO DO PROBLEMA** | **SOLUÇÃO (JUNTO AO APRENDIZADO ADQUIRIDO)** |
| 1. Integração  Verilog-VGA |  | Necessidade de fazer os gráficos exibidos no monitor terem total correspondência à descrição do hardware especificada em Verilog. A exibição do andamento do jogo, com atualização do placar e afundamento de uma embarcação, constituiu grande desafio no ajuste de parâmetros de sincronização e manipulação dos barcos exibidos. |  |
| 2. Operar na  memória RAM |  | Necessidade de acessar a RAM várias vezes durante:  - a exibição do mapa,  - atualização de pontuação,  - atualização de posições no vetor do barco quando este era parcialmente derrubado,  - no posicionamento de nova embarcação, eram acessadas as posições de outras peças na memória, para impedir sobreposição. | Para isso, foi estudado e escolhida a opção de construir um controlador de acesso à RAM. |
| 3. Exibição de objetos no VGA |  | Exibição de vários componentes do mapa, simultaneamente. |  |
| 4. Escassez de referências |  | Poquíssimas referências obre uso e implementação de técnicas necessárias, como circuito PLL, cuja simulação no ModelSim não é suportada. |  |
| 5. Posicionamento dos barcos na tela |  | Cálculo dos pixels ocupados na horizontal e vertical por cada barco, a depender do posicionamento escolhido pelo jogador. |  |
|  | | | |
| **POSSÍVEIS MELHORIAS** | **ABRANGÊNCIA DE IMPLEMENTAÇÃO** | **DESCRIÇÃO DA MELHORIA** | |
| 1. Uso de sprites |  | Utilização de sprites[.] para exibir imagens na tela. Ele é um objeto componente da interface e pode ser, por exemplo uma embarcação. Quando vários sprites combinam-se adequadamente, forma-se o mapa. | |

6. REFERÊNCIAS

[1] https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html

[2] https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html

[3] <https://pt.wikipedia.org/wiki/Field-programmable_gate_array>

[4] https://pt.wikipedia.org/wiki/Verilog

[5]

[6]

- <https://www.embarcados.com.br/controlador-vga-parte-1/> - VER FORMATO NO IEEE

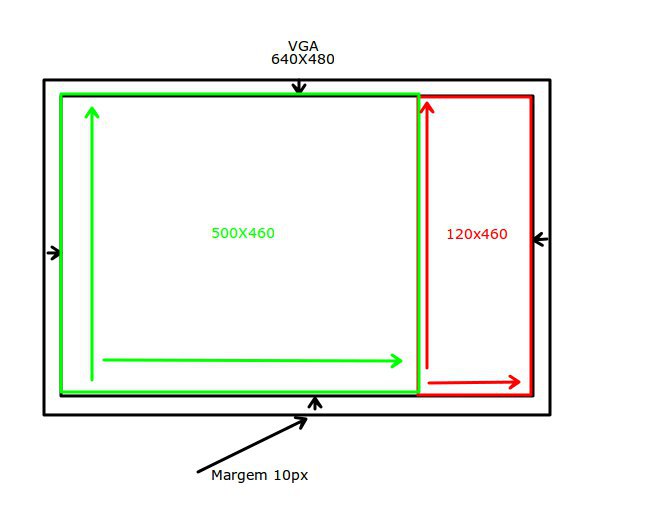
- https://www.embarcados.com.br/controlador-vga-parte-2/

- https://www.embarcados.com.br/caracteres-ascii-com-controlador-vga-parte-3/

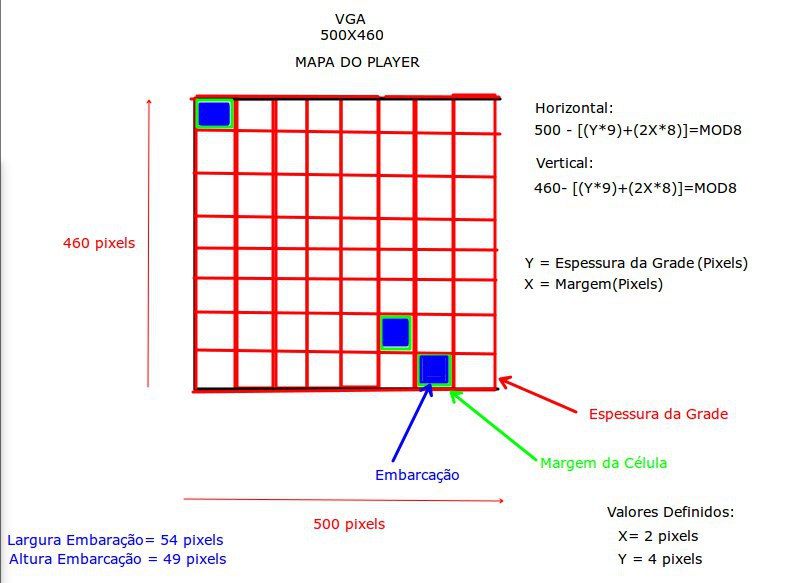
- <https://embeddedmicro.com/tutorials/mojo/writing-test-benches>

- <https://pt.wikipedia.org/wiki/Verilog#In.C3.ADcio>

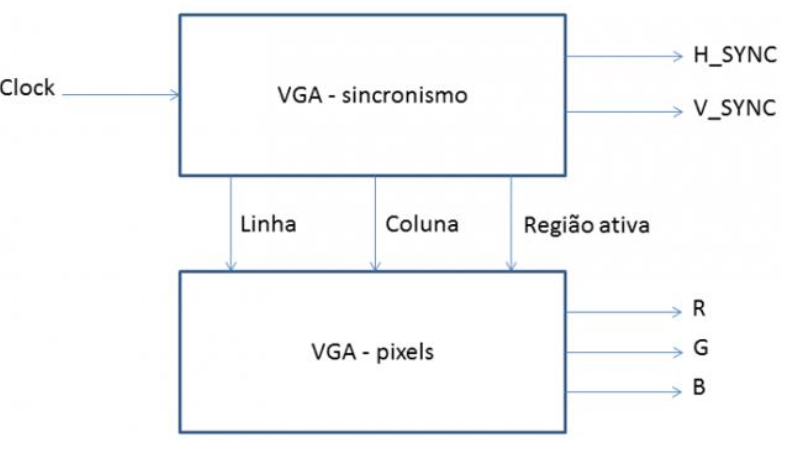
ANEXO I..



**Ilustração do uso da região ativa (640x480) do Frame VGA. O retângulo com contorno verde destina-se ao mapa, enquanto que o vermelho, às informações de placar, coordenadas de disparo e vez do jogador. Cada margem de 10 pixels é usada como medida cautelar contra o uso do limite da RA.**



**Ilustração do tabuleiro 8x8 utilizado. Os valores de X e Y para a dimensões de cada linha da grade foi encontrada através de equações congruenciais. Evitou-se assim o risco de sobrar alguma região do tabuleiro sem ser utilizada.**



**Modelo geral da controladora VGA, ilustrando a entrada de Clock com frequência adequada devido à operação do PLL. As saídas do bloco de sincronismo são usadas durante a varreduras dos pixels em toda a Região Ativa, para que sejam pintados com a composição RGB pela escolha do programador.**