Padrão VGA

“Screenshot_20241008_061619_Drive.jpg” could not be found.

hsync = 0 : Linha finalizada
vsync = 0 : frame finalizado

Metodologia

Arquitetura Geral

Processador GP

O Nios II foi usado para implememtar o processador de proposito geral

Nios II

- RISC
- 32 Bits
- Arquitetura Harvard
- Desenvolvido pela Altera
- Executa o código C dos jogos

O Nios II armazena nas FIFOs todas as instruc¸oes que ˜
devem ser executadas pelo Processador Grafico. O m ´ odulo ´
Gerador de Pulso e respons ´ avel por gerar um ´ unico pulso de ´
escrita em sincroniza com o sinal wrclk, por consequencia, os ˆ
dados de instruc¸ao presentes nos barramentos ˜ dataA e dataB
serao armazenados nas FIFOs uma ˜ unica vez. Cada FIFO ´
possui inicialmente a capacidade de armazenar 16 palavras
de 32-bits. Quando o sinal wrfull esta em n ´ ´ıvel logico alto ´
significa que as FIFOs alcanc¸aram sua capacidade maxima. ´
Desta forma, o circuito interno de protec¸ao das FIFOs ˜ e´
ativado automaticamente para evitar poss´ıveis Overflows.

Duas FIFOs

Intermediários para comunicação do processador e a GPU

Uma PLL

Phase Locked Loop

Gera as frequências de clock necessárias para o funcionamento da GPU

Processador Gráfico

- Gerenciar o processo de renderização
- Executar instruções para mover e controlar sprites
- Modificar layout do background
- Renderizar quadrados e triângulos

Saídas

- hsync
- vsync
- Bits RGB

Arquitetura da GPU

Unidade de Controle

- Ler, decodificar e executar instruções recebidas

Banco de Registradores

32 Registradores: 1 para background e 31 para sprites

Armazena temporariamente as informações associadas a cada spirte:

- Coordenadas
- Offset da memória
- Bit de ativação

Módulo de Desenho

Ele desenha os pixels.

Controlador do VGA

- Gera sinais de sincronização(vsync e hsync)
- Fornece as coordenadas x e y do processo de varredura do monitor
- 60 fps

Memória de Sprite

- Armazena bitmap de cada sprite
- 12.800 palavras de 9 bits
- 3 bits para cada componente RGB
- Bitmap de 20x20, ocupando 400 posiçõess de memória
- Total de 32 sprites diferentes

Memória do Background

- 4.800 palavras de 9 bits
- A inicialização das memórias é realizada durantd o processo de síntese

Co-Processador

- Responsável por gerenciar a construção de polígonos convexos do tipo Quadrado e triângulo
- São renderizados em conjunto com os sprites e background

Memória de Instrução

Armazena dados referentes a polígonos: tamanho, cor, coordenadas e forma

- 16 palavras de 16 bits

Pipeline

- 100 MHz
- 5 vias de instrução(cada estágio pode processar até 5 instruções ao mesmo tempo)
- O pipeline processa 15 polígonos no total, no intervalo de varredura horinzontal do VGA
- Cada via consiste de uma unidade de cálculo e um banco de registradores
- Um comparador é implementado para tratar concorrência
- O ultimo polígono a ser desenhado tem prioridade sobre os outros

Memória de Pixels

Utilizada para permitir o processamento de polígonos durante a renderização do frame

- 640 palavras de 9 Bits

Dessa maneira o Co-Processador e o controlador da vga trabalham em sincronia

Contador

Gera as coordenadas dos polígonos
Gera o endereço de escrita para a memória de pixels

Gerador RGB

Prioooridade: Sprite, polígono e background

Instruções da GPU

Escrita no banco de Registradores(WBR)

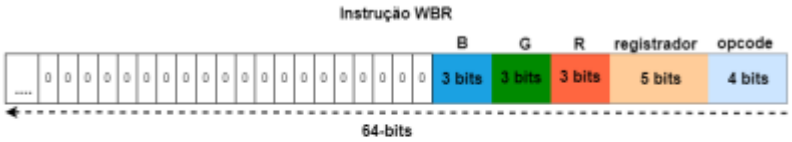


Figura 8: Uso da Instruções WBR para modificação da cor base do *background*.

Figura 9: Uso da Instruções WBR para configurar um *sprite*.

- Opcode : 0000
- Offset : Localização do sprite na memória
- SP : Habilitar ou desabilitar o sprite

(Fig. 10). O campo **opcode** é semelhante à instrução anterior, no entanto, seu valor é configurado em 0001. O campo **endereço de memória** especifica qual local da memória será alterado. Os campos **R**, **G** e **B** definem as novas componentes RGB para o local desejado.

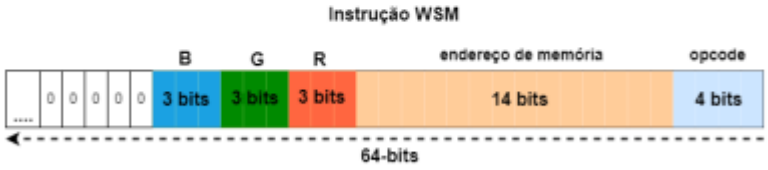
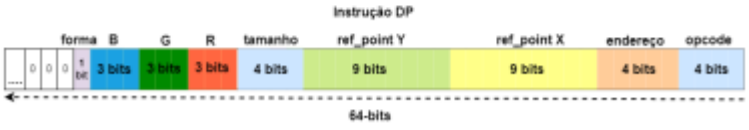


Figura 10: Representação da Instrução WSM.

Escrita na Memória de Background (WBM): Essa instrução armazena ou modifica o conteúdo presente na Memória de *Background*. Sua função é configurar valores RGB para o preenchimento de áreas do *background*. Seus campos são semelhantes ao da instrução WSM (Fig. 10), a única diferença está no campo **endereço de memória** com tamanho de 12 bits. O valor do **opcode** é configurado como 0010. O *background* é divido em pequenos blocos de 8x8 *pixels* e cada endereço de memória corresponde a um bloco. Sendo a resolução de 640x480 *pixels*, temos uma divisão de 80x60 blocos. Isso permite que o *background* seja configurado de formas diferentes de acordo ao preenchimento da memória (Fig. 11). Se um endereço for preenchido com o valor 0b111111110 = 510, o Módulo de Desenho entenderá que o bloco correspondente está desabilitado, assim ocupando os *pixels* da área com a cor base do *background*, um polígono ou *sprite*, caso suas coordenadas coincidam com o bloco.

Tabela II: Dimensões dos Polígonos

Campo **tamanho**	**Dimensão (Base e altura)**
0b0000	*polígono desabilitado*
0b0001	20x20
0b0010	30x30
0b0011	40x40
0b0100	50x50
0b0101	60x60
0b0110	70x70
0b0111	80x80
0b1000	90x90
0b1001	100x100
0b1010	110x110
0b1011	120x120
0b1100	130x130
0b1101	140x140
0b1110	150x150
0b1111	160x160



'igura 12: Uso da instrução DP para definição de um polígono.

- Opcode : 0011

Comunicação

Acessar GPIOs atraves do mapeamento de memória em C.
As intruções devem ser quebradas em partes e colocadas nos barramentos dataA e dataB,

então o sinal start deve ser ativado para que a instrução seja escrita nos FIFOs

API

```
#define LEFT     0
#define RIGHT    4
#define UP       2
#define DOWN     6
#define UPPER_RIGHT 1
#define UPPER_LEFT  3
#define BOTTOM_LEFT 5
#define BOTTOM_RIGHT 7
typedef struct{
        int coord_x, coord_y;
        int direction, offset, data_register;
        int step_x, step_y;
        int ativo,  collision;
} Sprite;
typedef struct{
        int coord_x, coord_y, offset;
        int data_register, ativo;
} Sprite_Fixed;
```

Listing 1: Definição de Constantes e Estruturas Compostas.

```
int set_sprite( int registrador, int x, int y, int
     offset, int activation_bit);
```

```
int set_background_block( int column, int line,
    int R, int G, int B);

int  set_background_color(int R, int G, int B);

void increase_coordinate(Sprite *sp, int mirror);

int  collision(Sprite *sp1, Sprite *sp2);
```

Listing 2: Funções Auxiliares.

Mais informações no TCC