Operações Gráficas na VGA com CPUlator

Curso: CIC0130 - Introdução aos Sistemas Embarcados

Autor: Gabriel da Conceição Miranda

Matrícula: 190106921

1. Introdução

Este relatório descreve a implementação de um programa em linguagem C para a placa DE1-SoC, executado no ambiente de simulação CPUlator. O objetivo principal do trabalho foi desenvolver uma aplicação interativa capaz de renderizar formas geométricas 2D em um display VGA, utilizando a interface JTAG UART para o diálogo com o usuário.

O sistema implementado oferece um menu textual que permite ao usuário selecionar uma cor e comandar o desenho de primitivas gráficas, como linhas, círculos, retângulos vazados e retângulos preenchidos. O projeto aborda conceitos fundamentais de sistemas embarcados, incluindo a programação de periféricos por meio de memória mapeada (memory-mapped I/O), o tratamento de dados de hardware em baixo nível e a aplicação de algoritmos clássicos de computação gráfica.

---

2. Arquitetura e Desenvolvimento

Esta seção detalha a arquitetura do software, a manipulação dos periféricos (VGA e JTAG UART), a lógica de interação com o usuário e os algoritmos gráficos que compõem o núcleo do programa.

2.1 Formato e Manipulação de Dados dos Periféricos

A comunicação com o hardware é realizada através de dois periféricos principais, cada um com um formato de dados específico.

- **JTAG UART (Entrada/Saída de Texto)** O periférico JTAG UART, mapeado no endereço `0xFF201000`, é a ponte de comunicação com o usuário. Os dados lidos deste endereço são palavras de 32 bits com a seguinte estrutura:
    - **Bit 15:** Flag de "Dado Válido". Quando este bit está em 1, significa que um novo caractere foi recebido e está pronto para leitura.
    - **Bits 7:0:** O código ASCII do caractere recebido.
- O tratamento desses dados no código é feito pela função `read_char()`. Ela implementa um laço de *polling* que lê continuamente o endereço da UART até que o bit 15 seja 1. A verificação é feita com a máscara de bits `0x8000`: `while ((data & 0x8000) == 0);`. Uma vez que um dado válido é detectado, o caractere é

extraído dos 8 bits menos significativos com a máscara `0xFF`: `return data & 0xFF;`.

- **Display VGA (Saída de Vídeo)** O display VGA é controlado através de um *framebuffer* na memória, com endereço base em `0xC8000000`. A tela visível tem resolução de **320x240 pixels**. Cada pixel é representado por uma palavra de 16 bits (**`uint16_t`**) no formato de cor **RGB 5-6-5**:
    - **5 bits** para Vermelho (Red)
    - **6 bits** para Verde (Green)
    - **5 bits** para Azul (Blue)
- As cores são pré-definidas no código usando constantes hexadecimais que representam esse formato, como `RED 0xF800` (11111 000000 00000) e `GREEN 0x07E0` (00000 111111 00000).

  Para escrever um pixel, a função `set_pix()` calcula sua posição exata na memória. Embora a largura visível seja de 320 pixels, a memória é organizada em linhas de 512 pixels (`SCREEN_WIDTH`). A fórmula `ppix[lin * SCREEN_WIDTH + col]` corretamente calcula o *offset* do pixel, garantindo que ele seja desenhado na posição visual correta.

2.2 Interação com o Usuário e Tratamento de Entradas

O programa foi projetado para ser robusto na interação com o usuário.

- **Fluxo de Interação:** O laço principal em `main()` primeiro exibe um menu de opções com `print_menu()`. A entrada do usuário é lida como uma string pela função `read_line()`, que oferece uma experiência de terminal amigável com eco dos caracteres digitados e suporte a *backspace*. A entrada é então convertida para maiúsculas com `to_upper()` para tornar a seleção de comandos e cores insensível a maiúsculas/minúsculas. Com base na string de comando, o programa solicita os parâmetros numéricos necessários (coordenadas e raio), que são extraídos da string de entrada usando `sscanf()`.

Tratamento de Entradas Inválidas (Clipping): Uma característica importante da implementação é a sua resiliência a coordenadas inválidas. Se o usuário fornecer coordenadas que estão fora dos limites da tela visível (ex: `lin < 0` ou `col >= 320`), o programa **não gera um erro nem trava**.

Este comportamento é gerenciado de forma elegante pela função `set_pix(int lin, int col)`. No seu início, há uma verificação crucial:

C
```
if (lin < 0 || lin >= VISIBLE_HEIGHT || col < 0 || col >= VISIBLE_WIDTH) return;
```

- Essa condição atua como um guarda (*guard clause*). Se a coordenada `(lin, col)` estiver fora da área visível, a função retorna imediatamente, sem tentar escrever na memória. O resultado prático é que qualquer forma geométrica que exceda os limites da tela é automaticamente **recortada** (*clipped*). Por exemplo, ao desenhar um círculo cujo centro está na borda da tela, apenas o arco do círculo que se encontra dentro da área visível será renderizado, como se a borda da tela estivesse cortando a forma. Isso torna o software robusto e previsível.

2.3 Análise das Funções e Algoritmos Gráficos

As funcionalidades de desenho são encapsuladas em funções modulares que utilizam algoritmos clássicos de computação gráfica.

- **`draw_line(int x0, int y0, int x1, int y1)`** Esta função implementa o **Algoritmo de Bresenham para Linhas**. É um método incremental eficiente que usa apenas aritmética de inteiros. As variáveis `dx` e `dy` armazenam as distâncias absolutas, enquanto `sx` e `sy` determinam a direção do incremento (1 ou -1). A variável `err` (erro) é inicializada e atualizada a cada passo para decidir se o próximo pixel deve ser o vizinho na direção principal de movimento ou na diagonal, minimizando o desvio da linha ideal.

- **`draw_circle(int lin, int col, int r)`** Para círculos, foi implementado o **Algoritmo de Bresenham para Círculos** (também conhecido como *Midpoint Circle Algorithm*). Este algoritmo é extremamente eficiente, pois explora a simetria óctupla de um círculo. Para cada pixel `(x, y)` calculado em um octante, a função desenha simultaneamente os pixels correspondentes nos outros sete octantes. Assim como na versão para linhas, uma variável de erro (`err`) é usada para tomar decisões incrementais, escolhendo o próximo pixel que mais se aproxima da circunferência ideal.

- **`draw_rect(int y0, int x0, int y1, int x1)`** A renderização de um retângulo vazado é feita de forma composta. Em vez de um novo algoritmo, a função simplesmente reutiliza a função `draw_line()` quatro vezes para desenhar as quatro arestas que conectam os vértices do retângulo.

- **`draw_tile(int y0, int x0, int y1, int x1)` e `fill_screen()`** O preenchimento de áreas é realizado por um método de varredura (*rasterization*). A função `draw_tile()` utiliza dois laços `for` aninhados para iterar sobre cada linha e coluna dentro da caixa delimitadora definida pelos dois pontos. Para cada pixel na área, a função `set_pix()` é chamada. A função `fill_screen()` opera de forma idêntica, mas com limites fixos de `(0, 0)` a `(319, 239)` para cobrir toda a tela visível.

---

3. Testes e Resultados

3. Testes e Resultados

A validação do sistema foi realizada em duas etapas: um teste inicial focado na interface de comunicação e, em seguida, testes sistemáticos para cada funcionalidade gráfica.

3.1 Teste Inicial de Comunicação (JTAG UART)

Antes de desenvolver a lógica gráfica, foi crucial validar o mecanismo de comunicação com o usuário. Para isso, um programa simplificado foi implementado para testar a captura de texto do terminal, o eco de caracteres e o tratamento de `backspace`. O teste foi bem-sucedido, confirmando o entendimento da interface e permitindo prosseguir com confiança para a implementação do menu interativo principal.

3.2 Validação das Funcionalidades Gráficas

Após a validação da comunicação, a aplicação completa foi testada. Cada opção do menu foi executada para verificar o comportamento visual no display VGA. Os testes confirmaram que todos os algoritmos operavam corretamente, incluindo o recorte de formas nas bordas da tela.

Abaixo estão **5 exemplos de sequências de comandos** usados gerar a imagem:

Lista de Comandos para Geração da Imagem

Teste 1: Preenchimento de Fundo (Cor Neutra)

1. **Comando:** 1 ou `COLOR`
2. **Cor:** `GRAY`
3. **Comando:** 6 ou `FUNDO`
    - *Este teste valida a função de preenchimento total da tela.*

Teste 2: Círculo Sobre o Fundo

1. **Comando:** 1 ou `COLOR`
2. **Cor:** `PURPLE`
3. **Comando:** 3 ou `CIRC`
4. **Coordenadas:** `120 160 100` (Centro em lin=120, col=160; raio=100)
    - *Valida o desenho de círculo sobre uma base já existente.*

Teste 3: Retângulo Preenchido (Tile) Ciano

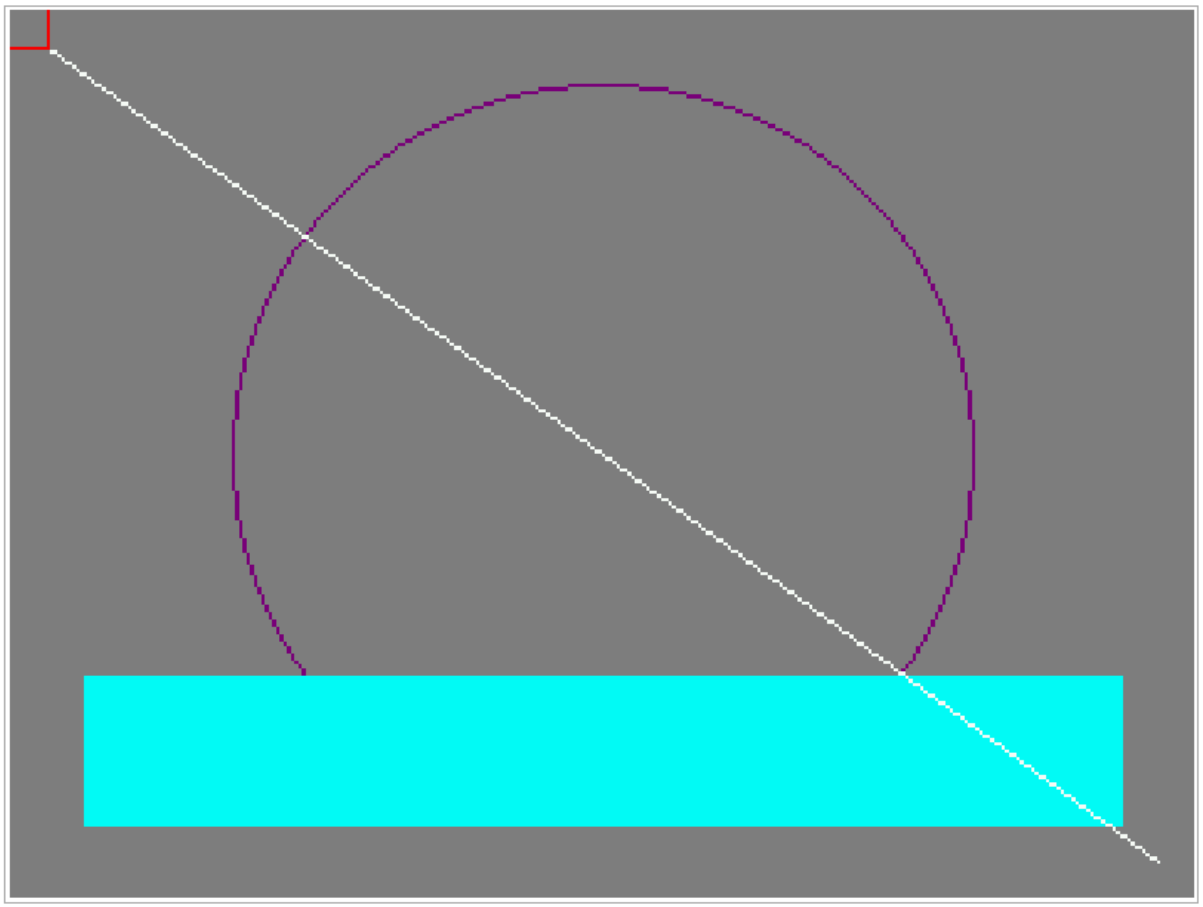
1. **Comando:** 1 ou `COLOR`
2. **Cor:** `CYAN`
3. **Comando:** 5 ou `TILE`
4. **Coordenadas:** `180 20 220 300`
    - *Valida o preenchimento de uma área retangular específica.*

Teste 4: Linha Branca para Contraste

1. **Comando:** 1 ou `COLOR`
2. **Cor:** `WHITE`
3. **Comando:** 2 ou `LINE`
4. **Coordenadas:** `10 10 230 310`
    - *Valida o algoritmo de linha e o contraste de cores.*

Teste 5: Retângulo Vazado (Rect) Vermelho

1. **Comando:** 1 ou `COLOR`
2. **Cor:** `RED`
3. **Comando:** `4 ou RECT`
4. **Coordenadas:** `-50 -50 10 10`
    - *Valida o desenho de um retângulo específico.*



---

4. Conclusão

O trabalho foi concluído com sucesso, atingindo todos os objetivos propostos no roteiro. Foi desenvolvida uma aplicação funcional e robusta, capaz de interpretar comandos do usuário e renderizar primitivas gráficas no display VGA simulado.

A implementação demonstrou o domínio sobre conceitos essenciais de sistemas embarcados, como a manipulação direta de periféricos via memória mapeada, a importância do formato de dados de hardware e a aplicação de algoritmos eficientes para tarefas gráficas. A estratégia de *clipping* implementada na função `set_pix()` provou ser uma solução simples e eficaz para garantir a robustez do programa contra entradas inválidas, um aspecto crucial em software embarcado.

Como possíveis melhorias futuras, o projeto poderia ser expandido para incluir o desenho de polígonos, a implementação de transformações geométricas (escala, rotação) ou a renderização de fontes de texto diretamente na tela VGA.

---

5. Apêndice - Código Fonte

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <stdint.h>

#define IO_JTAG_UART (*(volatile unsigned int *)0xFF201000)

#define VGA_BASE 0xC8000000
#define SCREEN_WIDTH 512
#define VISIBLE_WIDTH 320
#define VISIBLE_HEIGHT 240

#define BLACK   0x0000
#define RED     0xF800
#define GREEN   0x07E0
#define BLUE    0x001F
#define GRAY    0x8410
#define WHITE   0xFFFF
#define YELLOW  0xFFE0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define ORANGE  0xFC00
#define PURPLE  0x780F
#define BROWN   0xA145
#define PINK    0xF81F
#define LIME    0x07F0
#define NAVY    0x000F
```

```
#define TEAL    0x0410

uint16_t *ppix = (uint16_t *)VGA_BASE;
uint16_t current_color = WHITE;

char read_char() {
    unsigned int data;
    do {
        data = IO_JTAG_UART;
    } while ((data & 0x8000) == 0);
    return data & 0xFF;
}

void read_line(char *buffer, int max_len) {
    int i = 0;
    while (i < max_len - 1) {
        char c = read_char();
        if (c == '\r' || c == '\n') break;
        if ((c == 8 || c == 127)) {
            if (i > 0) {
                i--;
                printf("\b \b");
                fflush(stdout);
            }
        } else if (c >= 32 && c <= 126) {
            buffer[i++] = c;
            putchar(c);
            fflush(stdout);
        }
    }
    buffer[i] = '\0';
    putchar('\n');
}

void to_upper(char *str) {
    while (*str) {
        *str = toupper(*str);
        str++;
    }
}

void set_pix(int lin, int col) {
    if (lin < 0 || lin >= VISIBLE_HEIGHT || col < 0 || col >= VISIBLE_WIDTH) return;
    ppix[lin * SCREEN_WIDTH + col] = current_color;
}

void draw_line(int x0, int y0, int x1, int y1) {
    int dx = abs(x1 - x0), sx = x0 < x1 ? 1 : -1;
```

```
    int dy = -abs(y1 - y0), sy = y0 < y1 ? 1 : -1;
    int err = dx + dy, e2;
    while (1) {
        set_pix(y0, x0);
        if (x0 == x1 && y0 == y1) break;
        e2 = 2 * err;
        if (e2 >= dy) { err += dy; x0 += sx; }
        if (e2 <= dx) { err += dx; y0 += sy; }
    }
}

void draw_circle(int lin, int col, int r) {
    int x = -r, y = 0, err = 2 - 2 * r;
    do {
        set_pix(lin - y, col + x);
        set_pix(lin - x, col - y);
        set_pix(lin + y, col - x);
        set_pix(lin + x, col + y);
        int e2 = err;
        if (e2 <= y) err += ++y * 2 + 1;
        if (e2 > x || err > y) err += ++x * 2 + 1;
    } while (x < 0);
}

void draw_rect(int y0, int x0, int y1, int x1) {
    draw_line(x0, y0, x1, y0);
    draw_line(x1, y0, x1, y1);
    draw_line(x1, y1, x0, y1);
    draw_line(x0, y1, x0, y0);
}

void draw_tile(int y0, int x0, int y1, int x1) {
    int ymin = y0 < y1 ? y0 : y1;
    int ymax = y0 > y1 ? y0 : y1;
    int xmin = x0 < x1 ? x0 : x1;
    int xmax = x0 > x1 ? x0 : x1;

    for (int y = ymin; y <= ymax; y++) {
        for (int x = xmin; x <= xmax; x++) {
            set_pix(y, x);
        }
    }
}

void fill_screen() {
    for (int y = 0; y < VISIBLE_HEIGHT; y++) {
        for (int x = 0; x < VISIBLE_WIDTH; x++) {
            set_pix(y, x);
```

```
        }
    }
}

void print_menu() {
    printf("\nMenu de opcoes:\n");
    printf("1) COLOR - Define a cor atual\n");
    printf("2) LINE  - Desenha uma linha\n");
    printf("3) CIRC  - Desenha um circulo\n");
    printf("4) RECT  - Desenha um retangulo vazado\n");
    printf("5) TILE  - Desenha um retangulo cheio\n");
    printf("6) FUNDO - Preenche a tela\n");
}

void set_color(char *color_name) {
    if (strcmp(color_name, "BLACK") == 0) current_color = BLACK;
    else if (strcmp(color_name, "RED") == 0) current_color = RED;
    else if (strcmp(color_name, "GREEN") == 0) current_color = GREEN;
    else if (strcmp(color_name, "BLUE") == 0) current_color = BLUE;
    else if (strcmp(color_name, "GRAY") == 0) current_color = GRAY;
    else if (strcmp(color_name, "WHITE") == 0) current_color = WHITE;
    else if (strcmp(color_name, "YELLOW") == 0) current_color = YELLOW;
    else if (strcmp(color_name, "CYAN") == 0) current_color = CYAN;
    else if (strcmp(color_name, "MAGENTA") == 0) current_color = MAGENTA;
    else if (strcmp(color_name, "ORANGE") == 0) current_color = ORANGE;
    else if (strcmp(color_name, "PURPLE") == 0) current_color = PURPLE;
    else if (strcmp(color_name, "BROWN") == 0) current_color = BROWN;
    else if (strcmp(color_name, "PINK") == 0) current_color = PINK;
    else if (strcmp(color_name, "LIME") == 0) current_color = LIME;
    else if (strcmp(color_name, "NAVY") == 0) current_color = NAVY;
    else if (strcmp(color_name, "TEAL") == 0) current_color = TEAL;
    else {
        printf("Entrada invalida\n");
        return;
    }
    printf("Cor definida como %s\n", color_name);
}

int main() {
    char input[100];
    printf("Sistema de desenho VGA - DE1-SoC\n");

    while (1) {
        print_menu();
        printf("> ");
        read_line(input, 100);
        to_upper(input);
```

```
    if (strcmp(input, "1") == 0 || strcmp(input, "COLOR") == 0) {
        printf("Formato (COLOR): <cor>\n");
        printf("Cores: BLACK RED GREEN BLUE GRAY WHITE ");
        printf("YELLOW CYAN MAGENTA ORANGE PURPLE BROWN PINK LIME NAVY
TEAL\n> ");
        read_line(input, 100);
        to_upper(input);
        set_color(input);
    }
    else if (strcmp(input, "2") == 0 || strcmp(input, "LINE") == 0) {
        int y0, x0, y1, x1;
        printf("Formato (LINE): lin0 col0 lin1 col1\n> ");
        read_line(input, 100);
        if (sscanf(input, "%d %d %d %d", &y0, &x0, &y1, &x1) == 4) {
            draw_line(x0, y0, x1, y1);
        } else {
            printf("Entrada invalida\n");
        }
    }
    else if (strcmp(input, "3") == 0 || strcmp(input, "CIRC") == 0) {
        int lin, col, r;
        printf("Formato (CIRC): lin col raio\n> ");
        read_line(input, 100);
        if (sscanf(input, "%d %d %d", &lin, &col, &r) == 3) {
            draw_circle(lin, col, r);
        } else {
            printf("Entrada invalida\n");
        }
    }
    else if (strcmp(input, "4") == 0 || strcmp(input, "RECT") == 0) {
        int y0, x0, y1, x1;
        printf("Formato (RECT): lin0 col0 lin1 col1\n> ");
        read_line(input, 100);
        if (sscanf(input, "%d %d %d %d", &y0, &x0, &y1, &x1) == 4) {
            draw_rect(y0, x0, y1, x1);
        } else {
            printf("Entrada invalida\n");
        }
    }
    else if (strcmp(input, "5") == 0 || strcmp(input, "TILE") == 0) {
        int y0, x0, y1, x1;
        printf("Formato (TILE): lin0 col0 lin1 col1\n> ");
        read_line(input, 100);
        if (sscanf(input, "%d %d %d %d", &y0, &x0, &y1, &x1) == 4) {
            draw_tile(y0, x0, y1, x1);
        } else {
            printf("Entrada invalida\n");
        }
```

```
        }
        else if (strcmp(input, "6") == 0 || strcmp(input, "FUNDO") == 0) {
            fill_screen();
            printf("Tela preenchida com a cor atual\n");
        }
        else {
            printf("Entrada invalida\n");
        }
    }
    return 0;
}
```