Laboratório 01 - Display

Equipe
Alfons Carlos Cesar Heiermann de Andrade
Mateus Filipe de Ornelas Rampim
Guilherme Corrêa Koller

Iniciamos através das funções exemplo disponibilizadas pelo professor a impressão de um texto teste para verificar o funcionamento. A maior dificuldade enfrentada pelo grupo foi a etapa de gravação do código na placa devido à falta de prática com o novo equipamento, após solucionarmos os estes problemas o laboratório fluiu como o esperado.

```
static void draw_example_text(void){
    st7789_fill_screen_dma(C_BLACK);
    st7789_draw_text_5x7(10, 10, "ST7789 Demo", C_YELL, 2, 0, 0);
    st7789_draw_text_5x7(10, 40, "UTFPR - SIS.EMBARCADOS", C_YELL, 2, 0, 0);
    st7789_draw_text_5x7(10, 80, "MENU DEMO", C_CYAN, 2, 0, 0);
}
```

A partir da impressão do texto foi possível iniciar o primeiro requisito do laboratório, a impressão do tempo corrente em segundos. No código abaixo verificamos o tempo atual através da função `millis` para passar como parâmetro para a função *drawheader*, que verifica se o tempo de troca de cor do quadrado (1 segundo) foi atingido ao subtrair do último tempo armazenado. Caso esta condição seja satisfeita, a função realiza a limpeza do cabeçalho, imprime o uptime no display e atualiza o quadrado colorido. Como o display não aceita em sua instrução de *drawtext5x7* o valor em uint32, convertemos o valor utilizando um buffer e a função `snprintf`.

```
#define HEADER_HEIGHT 32
uint32_t last_tick_ms = 0;
uint32_t last_tick_s = 0;
uint32_t color_tick_ms = 1000;

static void clean_header(void)
{
    st7789_fill_rect_dma(0, 0, LCD_W, HEADER_HEIGHT, C_BLACK);
}

static void draw_header(uint32_t uptime_ms)
{
    if (uptime_ms - last_tick_ms >= color_tick_ms)
    {
        clean_header();
        uint32_t seconds = uptime_ms / 1000u;
        char value_to_string[24];
        snprintf(value_to_string, sizeof(value_to_string), "Uptime: %lu s",
           seconds);
        st7789_draw_text_5x7(0, 0, value_to_string, C_WHITE, 2, 0, 0);
    }
}
```

Para completar o requisito do cabeçalho, implementamos a impressão de um quadrado colorido no canto oposto ao uptime. A cada atualização do uptime, a cor do quadrado é alterada, iterando sobre um vetor de cores. O quadrado é sempre desenhado na mesma posição, apenas mudando sua cor.

```
static void draw_header_square(uint8_t i)
{
    uint16_t x = (uint16_t)(LCD_W - SQUARE_SIZE - PADDING);
    uint16_t y = (uint16_t)PADDING;
    uint16_t w = (uint16_t)SQUARE_SIZE;
    uint16_t h = (uint16_t)SQUARE_SIZE;
    st7789_fill_rect_dma(x, y, w, h, colors[i]);
}
```

Outro requisito do laboratório foi a implementação de uma animação de uma bola que se move horizontalmente e quica nas bordas do display. A velocidade da bola pode ser ajustada via comandos seriais, e a cada 200ms é enviado um log para o terminal serial com a posição atual da bola e o tempo de execução do sistema.

```
volatile uint32_t ball_delay = 20;

void increase_ball_speed(void)
{
    if (ball_delay > 2)
        ball_delay -= 2;
}

void decrease_ball_speed(void)
{
    if (ball_delay < 100)
        ball_delay += 2;
}

void animate_bouncing_circle(void)
{
    static int x = 50, y = 160, r = 20, dx = 4;
    static uint32_t last_update = 0;
    static uint32_t last_log = 0;
    uint32_t now = millis();
    if (now - last_update < ball_delay)
        return;
    last_update = now;
}
```

A função `animate_bouncing_circle` controla a posição da bola, apaga a posição anterior, atualiza a posição e desenha a nova posição. O intervalo entre atualizações é controlado pela variável `ball_delay`, que pode ser ajustada pelo usuário via comandos seriais. O log serial é enviado a cada 200ms, informando o tempo de execução e a posição X da bola, facilitando o acompanhamento do comportamento do sistema em tempo real.

Implementou-se uma animação com posição x, raio r e velocidade dx. A cada `ball_delay` ms a rotina apaga a posição anterior (cor de fundo) e desenha a nova posição (cor verde). As colisões com as bordas esquerda/direita invertem o sinal de dx, garantindo o quique. O cabeçalho não é afetado.

```
volatile uint32_t ball_delay = 20; // ms
```

```
static void animate_bouncing_circle(void) {
    static int x = 50, y = 160, r = 20, dx = 4;
    static uint32_t last_update = 0;
    static uint32_t last_log = 0;

    uint32_t now = millis();
    if (now - last_update < ball_delay) return;
    last_update = now;

    int old_x = x;

    st7789_fill_circle(old_x, y, r, C_BLACK);

    x += dx;
    if (x + r >= LCD_W)  { x = LCD_W - r; dx = -dx; }
    else if (x - r <= 0) { x = r;         dx = -dx; }

    st7789_fill_circle(x, y, r, C_GREEN);

    if (now - last_log >= 200) {
        printf("[LOG] Uptime: %lu ms | Ball X: %d\r\n", now, x);
        last_log = now;
    }
}
```

O log é emitido na mesma rotina de animação usando um marcador temporal (`last_log`). O conteúdo reporta o *uptime* real (em ms, via `millis()`) e a posição x da bolinha. O período é controlado por comparação de diferenças de tempo, sem `delay`.

A velocidade é controlada pela variação do período de atualização (`ball_delay`). As teclas **1** e **2** recebidas via UART ajustam o atraso em passos de 2 ms dentro do intervalo $[2, 100]$ ms.

```
static inline int  uart_rx_ready(void){ return (USART1->SR & USART_SR_RXNE) !=
    0; }
static inline char uart_getc(void)    { return (char)USART1->DR; }

static void increase_ball_speed(void){
    if (ball_delay > 2) ball_delay -= 2;
}

static void decrease_ball_speed(void){
    if (ball_delay < 100) ball_delay += 2;
}

for(;;){
    draw_header(millis());
    animate_bouncing_circle();

    if (uart_rx_ready()){
        char c = uart_getc();
        switch(c){
            case '1': increase_ball_speed(); break;
            case '2': decrease_ball_speed(); break;
            default: break;
        }
    }
}
```