Aula 5 - Ponteiro para função

Revisando a aula 4

- Implementada a SPI por IT
- Buscado melhorar o gerenciamento do buffer, para evitar conflitos e sobreescrições

Solução de problemas da aula 4...

- O HAL_Delay fazia o código não funcionar. Isso ocorre pq ele usa timer SYStick, que é do próprio núcleo.Esse timer é utilizado por outros periféricos também, como a interrupção da SPI. O que ocorria então era uma falha de prioridade, pois o HAL_Delay e a SPI queriam usar o mesmo timer ao mesmo tempo, e ambos tinham a mesma prioridade.
    - Pra solucionar isso, é preciso ir no .ioc e mudar a prioridade de um dos dois no system core aba NVIC
    - SPI2 Interrupt ou Tim base: System tick timer.
- Expandido para todas as funções a checagem de Buffer Busy no início da função

Máquina de estados

A máquina de estados permite que o processador faça outras coisas ao meso tempo em que controla o display.

Os estados são as imagens, a função de set_XY em (0,0), a função clear e um delay;

Timer 11

O tim é usado para o processador olhar pro display de tempos em tempos, permitindo ele fazer outras coisas enquanto o display roda.

{84*10^6}/200 = 1000 * 420

> PSC = 419
>
> ARR = 999

Problema do Const

Eu usei a LCD_write recebendo o SharedBuffer_t, com os dados sendo um ponteiro. Porém ocorria um erro com o LCD_clear, pois o programa não rodava quando a variável TelaLimpa era const. Mas sem o const, ela era alterada no programa, o que gerava erros de escrita no display.

Então eu voltei pra escrita original da função,que recebia o buffer e o seu tamanho (e o modo) na LCD_write. Creio que não tanta diferença assim em processamento, pois ao fazer o clear, eu envio o endereço do TelaLimpa para um ponteiro, então da mesma forma não há necessidade de copiar os dados de um buffer pra outro.

Biblioteca do arm - arquivo .a

A biblioteca possui as funções e variáveis usadas nela. Aqui está declarado quais são, quais são seus tipos e quanto de memória ela usa no total.

O .o é o arquivo compilado para a linguagem de máquina. O .o possui todas as funções da biblioteca.

Já o .a possui apenas as funções que estão sendo usadas no programa e suas dependencias.

Compilador x Linker

O **compilador** traduz o código fonte para assembler, linguagem de máquina.

O **Linker** vai definir os endereços de memória das variáveis e funções que estão declaradas no código para o dispositivo onde o projeto será usado.

Ponteiro para funções

Assim como tu pode apontar o endereço de uma variável, tu também pode apontar o endereço de uma função!

```
// Sintaxe
tipo_da_funcao (*nome_do_ponteiro)(tipo arg1, tipo arg2);

// exemplo:
int (*pWrite)(uint8_t *dado, uint16_t tam);
```

Defines com relação ao display

```
/* Tipo de dado enviado ao transmissão */
#define LCD_DATA        1
#define LCD_COMMAND     0
```

Códigos

bd40214

IOC

Configurar o Tim11

configurar as NVIC

main

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "spi.h"
#include "tim.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "LCD.h"
#include "imagem.h"


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DISP_DELAY 2000
#define DTELA 400; //2s
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */
```

```c
/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

uint16_t delayTela;
uint8_t me_display;
uint8_t tela;

uint16_t a;

LCD_HandleTypeDef hlcd; // handle do display
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick.
*/
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI2_Init();
  MX_TIM11_Init();
  /* USER CODE BEGIN 2 */
```

```
  HAL_TIM_Base_Start_IT(&htim11);

  //
==========================================================================
====
  // --- Configurações do hardware do display ---
    hlcd.hspi = &hspi2;

    hlcd.CS_Port = NK_CS_GPIO_Port;
    hlcd.CS_Pin = NK_CS_Pin;

    hlcd.DC_Port = NK_DC_GPIO_Port;
    hlcd.DC_Pin = NK_DC_Pin;

    hlcd.RS_Port = NK_RS_GPIO_Port;
    hlcd.RS_Pin = NK_RS_Pin;

    // manda a estrutura de dados com as configurações do display pra
inicialização
    LCD5110_init(&hlcd);

    // aula 4
    while(LCD5110_set_XY(0, 0)!=HAL_OK){};
    while(LCD5110_write_str("Hello World")!=HAL_OK){};
    while(LCD5110_set_XY(0, 2)!=HAL_OK){};
    while(LCD5110_write_str("Estou Vivo")!=HAL_OK){};
    //HAL_Delay(DISP_DELAY);


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */


    tela = 1;
    me_display = 0;
    delayTela = DTELA;

  while (1)
  {


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 84;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{

    if(htim->Instance == TIM11)
    {
          switch(me_display)
          {
            case 0:
                delayTela--;

                  if (delayTela == 0)
                  {
                      delayTela=DTELA;
                      me_display++;
                  }
            break;

            case 1:
                if (LCD5110_clear()==HAL_OK)
```

```
                {
                    me_display++;
                }
            break;

            case 2:
                if (LCD5110_set_XY(0,0)==HAL_OK)
                {
                    me_display += tela;
                }
            break;

            case 3:
                if (LCD5110_write_str("gremio")==HAL_OK)
                {
                    me_display = 0;
                    tela++;
                }
            break;

            case 4:
                if (LCD5110_write(liber_bmp2, 504)==HAL_OK)
                {
                    me_display = 0;
                    tela++;
                }
            break;

            case 5:
                if (LCD5110_write_str("boca juniors na bombonera")==HAL_OK)
                {
                    me_display = 0;
                    tela = 1;
                }
            break;
          }
    }

}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
```

```
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

LCD.c

```
//
===========================================================================
============================
// === Includes ===

#include "stm32f4xx.h"
#include "stm32f4xx_hal_gpio.h"
#include "stm32f4xx_hal_spi.h"

#include "LCD.h"
#include "font.h"
#include "grafic.h"

//
===========================================================================
============================
// === Defines ===

#define LARGFONTE   6
#define TAMTELA     504
#define TAMBUF      504

//
===========================================================================
============================
// === Tipos ===

// estados do buffer
typedef enum {B_FREE=0, B_BUSY} BufStatus_t;

/*
 * Estrutura de dados para compartilhar o buffer
 * Realiza o gerenciamento do buffer
```

```
 */
typedef struct{
    uint8_t dado[TAMBUF];
    BufStatus_t status;
    uint16_t ocupacao;
} SharedBuffer_t;

//
============================================================================
============================
// === Variaveis ===

// manipulador do display
static LCD_HandleTypeDef *lcd;
// declarado como static para garantir que seus dados não serão alterados

// declaração da variável tipo Buffer compartilhado
static SharedBuffer_t buf;

const uint8_t TelaLimpa[TAMBUF]={0};




//Define the LCD Operation function
void LCD5110_LCD_write_byte(unsigned char dat,unsigned char mode);



//
============================================================================
============================================
// === Funções ===

void LCD5110_init(LCD_HandleTypeDef *hlcd5110)
{
    // phspi irá apontar pro endereço de dados da SPI escolhida
    lcd = hlcd5110;

    //inicialização do buf
    buf.ocupacao=0;
    buf.status=B_FREE;

    // inicialização dos pinos de CS e de DS
    HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 1);
    HAL_GPIO_WritePin(lcd->DC_Port, lcd->DC_Pin, 1);

    // reset do display
    HAL_GPIO_WritePin(lcd->RS_Port, lcd->RS_Pin, 0);
    HAL_Delay(100);
    HAL_GPIO_WritePin(lcd->RS_Port, lcd->RS_Pin, 1);

    LCD5110_LCD_write_byte(0x21,0);
    LCD5110_LCD_write_byte(0xc6,0); //ajusta o contraste do display
    LCD5110_LCD_write_byte(0x06,0);
    LCD5110_LCD_write_byte(0x13,0);
    LCD5110_LCD_write_byte(0x20,0);
    LCD5110_clear();
```

```
    // espera transmitir tudo para depois continuar
    while(__HAL_SPI_GET_FLAG(lcd->hspi, SPI_FLAG_BSY)){};
    LCD5110_LCD_write_byte(0x0c,0);
}


// ainda necessária para fazer a configuração inicial
void LCD5110_LCD_write_byte(unsigned char dat,unsigned char mode)
{
    // define se é dado ou comando
    HAL_GPIO_WritePin(lcd->DC_Port, lcd->DC_Pin, mode);

    // habilita o chip select
    HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 0);

    // Transmissão dos dados feitos pela SPI do HAL
    HAL_SPI_Transmit(lcd->hspi, &dat, 1, 200);

    // desabilita o chip select
    HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 1);
}


// ------------------------------------------------------------------------------
-----------------------------------------------
// ---- Função para enviar um bloco de dados ao display ----
/**
 * @brief Realiza a transmissão de dados por blocos
 *
 * @param data dados já formatados da informação que se quer escrever
 * @param tam tamanho dos dados
 * @param mode modo: (0)comando, (1)escrita
 */
HAL_StatusTypeDef LCD5110_write(uint8_t *data, uint16_t tam)
{
    HAL_StatusTypeDef status;

    // testa se a SPI está livre
    if(__HAL_SPI_GET_FLAG(lcd->hspi, SPI_FLAG_BSY))
        return HAL_BUSY;

    // define se é dado ou comando
    HAL_GPIO_WritePin(lcd->DC_Port, lcd->DC_Pin, 1);

    // habilita o chip select
    HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 0);

    // Agora será transmitido um bloco de tamanho tam de dados
    status = HAL_SPI_Transmit_IT(lcd->hspi, data, tam);

    return status;
}


//------------------------------------------------------------------------------
-------
// --- função de transmissão no modo não bloqueante
HAL_StatusTypeDef LCD_write_IT(uint8_t *data, uint16_t tam, uint8_t mode)
```

```
{
    HAL_StatusTypeDef status;

    // testa se a SPI está livre
    if(__HAL_SPI_GET_FLAG(lcd->hspi, SPI_FLAG_BSY))
        return HAL_BUSY;

    // define se é dado ou comando
    HAL_GPIO_WritePin(lcd->DC_Port, lcd->DC_Pin, mode);

    // habilita o chip select
    HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 0);

    // Agora o bloco será transmitido por interrupção
    status = HAL_SPI_Transmit_IT(lcd->hspi, data, tam);

    return status;

    // Só pode ser levantado depois do final da transmissão
    //HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 1);
}

void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
    if(hspi->Instance == lcd->hspi->Instance)
    {
        // levanta o chip select, pois a transmissão acabou
        HAL_GPIO_WritePin(lcd->CS_Port, lcd->CS_Pin, 1);
        // libera o buffer
        buf.status = B_FREE;
    }
}

// ------------------------------------------------------------------------------
-----------------------------------------------
// ---- Desenhar uma string no display ----
/*
 * Rotina de desenhar uma string no display é feita com 3 funções;
 *
 *      # void LCD5110_write_str(char *s)
 *          - recebe a string a ser escrita
 *          - chama a draw string para construir a string
 *          - chama a LCD_write para enviar os dados ao display
 *
 *      # uint16_t LCD_draw_string(char *s)
 *          - recebe a mesma string da anterior
 *          - chama a draw char para montar os dados dos caracteres
 *          - organiza os dados no Buffer
 *
 *      # void LCD_draw_char(char letra, uint8_t *buf)
 *          - constroi os dados de cada caractere
 *
 */

void LCD_draw_char(char letra, uint8_t *buf)
{
    uint8_t i;
    letra -= ' ';
```

```
    // pega o byte da biblioteca fonte6_8 e salva no Buffer dado
    for(i=0;i<LARGFONTE;i++)
    {
        *buf = font6_8[(uint8_t)letra][i];
        buf++;
    }
}

uint16_t LCD_draw_string(char *s)
{
    uint8_t *c;
    uint16_t tamanho=0;


    // aponta para o vetor dos dados do caractere
    c=buf.dado;

    // passa pela string pedida até encontrar o seu final
    while(*s!='\0')
    {
        // monta caractere por caractere no Buffer, apontado por c
        LCD_draw_char(*s, c);
        s++;
        c+=LARGFONTE;
        tamanho+=LARGFONTE; // define quantos caracteres foram passados
    }

    return tamanho;
}

HAL_StatusTypeDef LCD5110_write_str(char *s)
{
    HAL_StatusTypeDef status;

    // testa o estado do buffer
    if(buf.status==B_BUSY)
        return HAL_BUSY; //retorna que o buffer está ocupado

    // atualiza o status do buffer
    buf.status=B_BUSY;

    // preenche o Buffer com os dados de cada caractere e retorna o tamanho
final
    buf.ocupacao=LCD_draw_string(s);

    // manda os dados da string à função de escrever no display
    status = LCD_write_IT(buf.dado, buf.ocupacao, 1);

    // atualiza o status do buffer (para modo pooling)
    //buf.status=B_FREE;

    return status;
}

// --- fim da rotina de desenhar uma string ---
// ------------------------------------------------------------------------------
-----------------------------------------------
```

```c
HAL_StatusTypeDef LCD5110_clear()
{
    HAL_StatusTypeDef status;

    // testa o estado do buffer
    if(buf.status==B_BUSY)
        return HAL_BUSY; //retorna que o buffer está ocupado

    buf.status = HAL_BUSY;
    // limpa o display escrevendo 0 em todos os pixels
    status = LCD_write_IT(TelaLimpa, TAMTELA, 1);

    return status;
}


// ------------------------------------------------------------------------------
-----------------------------------------------
// ---- Função para setar a posição XY no display ----

HAL_StatusTypeDef LCD5110_set_XY(uint8_t x, uint8_t y)
{
    HAL_StatusTypeDef status;

    // testa o estado do buffer
    if(buf.status==B_BUSY)
        return HAL_BUSY; //retorna que o buffer está ocupado

    // atualiza o status
    buf.status = B_BUSY;

    // acerta o endereço em função da largura da fonte
    x *= LARGFONTE;

    // garante que x será de 7bits e que y será de 3bits
    x &= 0x7f;
    y &= 0x07;


    buf.dado[0] = 0x40|y;
    buf.dado[1] = 0x80|x;
    buf.ocupacao = 2;

    // envia o comando de setar a posição ao display
    status = LCD_write_IT(buf.dado, buf.ocupacao, 0);

    return status;
}
```

LCD.h

```
typedef struct
{
    SPI_HandleTypeDef *hspi;        // Handler pra SPI usada

    GPIO_TypeDef    *CS_Port;       // Porta do Chip select
    uint16_t        CS_Pin;         // Pino do CS

    GPIO_TypeDef    *DC_Port;       // Porta do Dado ou comando
    uint16_t        DC_Pin;         // Pino do DC

    GPIO_TypeDef    *RS_Port;       // Porta do Reset
    uint16_t        RS_Pin;         // Pino do DC

}LCD_HandleTypeDef;

//---

void LCD5110_init(LCD_HandleTypeDef *hlcd5110);

HAL_StatusTypeDef LCD5110_write(uint8_t *data, uint16_t tam);

//---

void LCD_draw_char(char c, uint8_t *dat);

uint16_t LCD_draw_string(char *s);

HAL_StatusTypeDef LCD5110_write_str(char *s);

//---

HAL_StatusTypeDef LCD5110_clear(void);

HAL_StatusTypeDef LCD5110_set_XY(uint8_t x, uint8_t y);
```