УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Проектирование вычислительных систем»

Лабораторная работа №4

Вариант 15

Студент

Крюков А. Ю.

Патутин В. М.

P34101

Преподаватель

Пинкевич В. Ю.

Санкт-Петербург, 2022 г.

Цель работы

1. Изучить устройство и принципы работы дисплейного модуля в стенде SDK-1.1M.
2. Получить навыки программирования внешних дисплейных устройств.

Задание лабораторной работы

Разработать программу, которая включает драйвер OLED-дисплея. Для организации обмена данными с дисплеем можно использовать те же три способа, что и в предыдущей работе. Поскольку дисплей управляется по той же шине I2C, что и клавиатура, необходимо избежать конфликтов доступа к шине. При работе с I2C по опросу они не возникнут, а при работе по прерыванию следует планировать вычисления так, чтобы транзакции не пересекались по времени.

Если транзакции I2C генерируются по прерыванию от таймера, следует встроить транзакции обновления видеопамяти дисплея в расписание опроса клавиатуры. При этом следует учесть, что отправка буфера данных в контроллер дисплея происходит значительно дольше, чем любая транзакция опроса клавиатуры. Например, после 8 вызовов обработчика прерываний для опроса клавиатуры можно инициировать отправку буфера и несколько вызовов обработчика отправлять никаких транзакций по I2C, давая время на завершение отправки буфера. Начальную инициализацию дисплея, как и клавиатуры, удобнее делать в режиме опроса, пока не включены прерывания от таймера.

Вариант задания

Адаптировать программу-кодовый замок для использования с дисплеем и клавиатурой SDK-1.1M. Кнопки клавиатуры использовать для:

– ввода пароля (цифры 0 – 9);

– перехода в режим изменения пароля/сохранения нового пароля.

Отображение статуса выполняется на дисплее. В обычном режиме на экране выведено текстовое приглашение для ввода пароля. После ввода производится проверка его правильности, и на экран выводится соответствующее сообщение. При переходе в режим изменения пароля выводится текстовое приглашение вида «введите новый пароль». Длина пароля – от 8 до 12 цифр. Если пользователь пытается сохранить пароль неправильной длины, выводится сообщение об ошибке.

Исходный код

```
/* USER CODE BEGIN Header */
/

**********************************************************************
**********
  * @file           : main.c
  * @brief          : Main program body

**********************************************************************
**********
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 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 "i2c.h"
#include "tim.h"
```

```
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "kb.h"
#include "sdk_uart.h"
#include "pca9538.h"
#include "oled.h"
#include "fonts.h"
#include "buzzer.h"
/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
queue input_queue;
queue output_queue;
```

```
char output_buffer[256];

uint8_t input_byte;
/* USER CODE END PV */

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

void oled_Reset( void );
/* USER CODE END PFP */

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

uint8_t KeyboardIndex(uint8_t a){
  uint8_t res=255;
  switch(a){
  case 99:
    res = 1;
    break;
  case 83:
    res = 2;
    break;
  case 51:
    res = 3;
    break;
  case 98:
    res = 4;
    break;
  case 82:
```

```
    res = 5;
    break;
  case 50:
    res = 6;
    break;
  case 97:
    res = 7;
    break;
  case 81:
    res = 8;
    break;
  case 49:
    res = 9;
    break;
  case 96:
    res = 11;
    break;
  case 80:
    res = 0;
    break;
  case 48:
    res = 12;
    break;
  }
  return res;
}
void print_string(char* string) {
  oled_Reset();
  oled_SetCursor(0, 0);


  char third[512];
    snprintf(third, sizeof third, "%s", string);
```

```
  oled_WriteString(string, Font_7x10, White);
  oled_UpdateScreen();
}

music init_melody() {
  music melody = create_music(20);
  add_note(&melody, ut);
  add_note(&melody, mi);
  add_note(&melody, sol);
  add_note(&melody, ut);
  add_note(&melody, mi);
  add_note(&melody, sol);
  add_note(&melody, ut);
  add_note(&melody, fa);
  add_note(&melody, la);
  add_note(&melody, ut);
  add_note(&melody, fa);
  add_note(&melody, la);
  add_note(&melody, ut);
  add_note(&melody, re);
  add_note(&melody, mi);
  add_note(&melody, fa);
  add_note(&melody, sol);
  add_note(&melody, la);
  add_note(&melody, si);
  add_note(&melody, ut_2);
  return melody;
}

void print_game_started() {
  print_string("Game is starting!\n\r");
}
```

```
void print_game_finished() {
  sprintf(output_buffer, "Game finished! Your score is:
%"PRIu32"!\n\r", get_game_score());
  print_string(output_buffer);
}
key char_to_key(uint8_t c){
  switch(c){
  case 1:
    return KEY_1;
  case 2:
    return KEY_2;
  case 3:
    return KEY_3;
  case 4:
    return KEY_4;
  case 5:
    return KEY_5;
  case 6:
    return KEY_6;
  case 7:
    return KEY_7;
  case 8:
    return KEY_8;
  case 9:
    return KEY_9;
  case 10:
    return KEY_A;
  case 11:
    return KEY_PLUS;
  case 12:
    return KEY_ENTER;
  default:
    return NO_KEY;
```

```
  }
}

void HAL_TIM_PeriodElapsedCallback

(TIM_HandleTypeDef *htim) {
  if (htim->Instance == TIM6) {
    on_game_timeout();
  }
}

/* USER CODE END 0 */

/
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
//  uint32_t zelda_melody[] = {
//      N_AS4, 0, 0, N_AS4, N_AS4, N_AS4, N_AS4, N_AS4, 0,
N_GS4, N_AS4, 0, 0, N_AS4, N_AS4, N_AS4, N_AS4, N_AS4, 0,
N_GS4, N_AS4, 0, 0, N_AS4, N_AS4, N_AS4, N_AS4, N_AS4,
N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3,
N_AS4, N_F3, N_F3, 0, N_AS4, N_AS4, N_C5, N_D5, N_DS5,
N_F5, 0, N_F5, N_F5, N_FS5, N_GS5, N_AS5, 0, N_AS5, N_AS5,
N_AS5, N_GS5, N_FS5, N_GS5, 0, N_FS5, N_F5, N_F5, N_DS5,
N_DS5, N_F5, N_FS5, N_F5, N_DS5, N_CS5, N_CS5, N_DS5,
N_F5, N_DS5, N_CS5, N_C5, N_C5, N_D5, N_E5, N_G5, N_F5,
N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3,
N_AS4, N_F3, N_F3, 0, N_AS4, N_AS4, N_C5, N_D5, N_DS5,
N_F5, 0, N_F5, N_F5, N_FS5, N_GS5, N_AS5, 0, N_CS6, N_C6,
```

```
N_A5, 0, N_F5, N_FS5, 0, N_AS5, N_A5, N_F5, 0, N_F5, N_FS5, 0,
N_AS5, N_A5, N_F5, 0, N_D5, N_DS5, 0, N_FS5, N_F5, N_CS5, 0,
N_AS4, N_C5, N_C5, N_D5, N_E5, 0, N_G5, N_F5, N_F3, N_F3,
N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_AS4,
N_F3, N_F3, 0, N_AS4, N_AS4, N_C5, N_D5, N_DS5, N_F5, 0,
N_F5, N_F5, N_FS5, N_GS5, N_AS5, 0, N_AS5, N_AS5, N_AS5,
N_GS5, N_FS5, N_GS5, 0, N_FS5, N_F5, N_F5, N_DS5, N_DS5,
N_F5, N_FS5, N_F5, N_DS5, N_CS5, N_CS5, N_DS5, N_F5,
N_DS5, N_CS5, N_C5, N_C5, N_D5, N_E5, N_G5, N_F5, N_F3,
N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3,
N_AS4, N_F3, N_F3, 0, N_AS4, N_AS4, N_C5, N_D5, N_DS5,
N_F5, 0, N_F5, N_F5, N_FS5, N_GS5, N_AS5, 0, N_CS6, N_C6,
N_A5, 0, N_F5, N_FS5, 0, N_AS5, N_A5, N_F5, 0, N_F5, N_FS5, 0,
N_AS5, N_A5, N_F5, 0, N_D5, N_DS5, 0, N_FS5, N_F5, N_CS5, 0,
N_AS4, N_C5, N_C5, N_D5, N_E5, 0, N_G5, N_F5, N_F3, N_F3,
N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3, N_F3
//  };
//  uint32_t zelda_delays[] = {
//      2, 8, 8, 8, 8, 8, 8, 6, 16, 16, 4, 8, 8, 8, 8, 8, 8, 6, 16, 16, 4, 8, 8, 8,
8, 8, 8, 8, 16, 16, 8, 16, 16, 8, 16, 16, 8, 8, 4, 4, 6, 16, 16, 16, 16, 16,
16, 2, 8, 8, 8 ,8 ,8, 2, 8, 8, 8 ,8 ,8, 8, 6, 16, 16, 2, 4, 8, 16, 16, 2, 8, 8,
8, 16, 16, 2, 8, 8, 8, 16, 16, 2, 4, 8, 16, 16, 8, 16, 16, 8, 16, 16, 8, 8, 4,
4, 6, 16, 16, 16, 16, 16, 16, 2, 8, 8, 8 ,8 ,8, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4,
4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 8, 16, 16, 4, 4, 4, 8, 16, 16,
8, 16, 16, 8, 16, 16, 8, 8, 4, 4, 6, 16, 16, 16, 16, 16, 16, 2, 8, 8, 8 ,8 ,8,
2, 8, 8, 8 ,8 ,8, 8, 6, 16, 16, 2, 4, 8, 16, 16, 2, 8, 8, 8, 16, 16, 2, 8, 8, 8,
16, 16, 2, 4, 8, 16, 16, 8, 16, 16, 8, 16, 16, 8, 8, 4, 4, 6, 16, 16, 16, 16,
16, 16, 2, 8, 8, 8 ,8 ,8, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4,
4, 4, 2, 4, 4, 4, 4, 4, 4, 8, 16, 16, 4, 4, 4, 8, 16, 16, 8, 16, 16, 8, 16, 16,
8, 8
//  };

  /* 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_I2C1_Init();
  MX_USART6_UART_Init();
  MX_TIM2_Init();
  MX_TIM1_Init();
  MX_TIM4_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  oled_Init();
  Buzzer_Init();
  initialize_io(&input_queue, &output_queue);
```

```
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
  HAL_TIM_Base_Start_IT(&htim6);

  input_queue = create_queue(256);
  output_queue = create_queue(256);

  music melody = init_melody();
  set_game_melody(&melody);
  /* USER CODE END 2 */

/* Infinite loop */
  /* USER CODE BEGIN WHILE */
//  uint64_t ass = 0;
//  uint8_t mode = 0;
//  uint8_t lastkey = 255;
  while (1)
  {
    uint8_t index = KeyboardIndex(Check_Row());
    if (index != 255){
//      char r[10];
//      itoa(index, r, 10);
          on_key_press(char_to_key(index));
//      print_string(r);
    }

    /* 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_REGULATO
R_VOLTAGE_SCALE1);

  / Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

//uint8_t KeyboardIndex(uint8_t a){
//  uint8_t res=255;
//  switch(a){
//  case 99:
//    res = 1;
//    break;
//  case 83:
//    res = 2;
//    break;
```

```
//  case 51:
//    res = 3;
//    break;
//  case 98:
//    res = 4;
//    break;
//  case 82:
//    res = 5;
//    break;
//  case 50:
//    res = 6;
//    break;
//  case 97:
//    res = 7;
//    break;
//  case 81:
//    res = 8;
//    break;
//  case 49:
//    res = 9;
//    break;
//  case 96:
//    res = 11;
//    break;
//  case 80:
//    res = 0;
//    break;
//  case 48:
//    res = 12;
//    break;
//  }
//  return res;
////}
```

```
//void KB_Test( uint64_t* ass, uint8_t* mode, uint8_t* lastkey) {
//  UART_Transmit( (uint8_t*)"KB test start\n" );
//  uint8_t R = 0, C = 0, L = 0, Row[4] = {0xF7, 0x7B, 0x3D, 0x1E},
Key, OldKey, OLED_Keys[12] =
{0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30};
//  oled_Reset();
//  oled_SetCursor(0, 0);
//
//    snprintf(third, sizeof third, "%s %i", "UberTimer:", *ass);
//  oled_WriteString(third, Font_7x10, White);
//  oled_UpdateScreen();
//  char p[17];
//  char k[17];
//
//  char third[512];
//  uint8_t index = KeyboardIndex(Check_Row());
//
//  if(index!=255 && *lastkey!=index){
//
//    if(*mode == 0){
//      if(index==11){
//        *ass = 0;
//      }
//      else if(index<10){
//        *ass = *ass*10+index;
//      }else if(index == 12){
//        *mode=1;
//      }
//    }
//    else if(*mode == 1){
//      if(index==11){
//        *ass = 0;
//        *mode = 0;
```

```
//      }
//      if(index==12){
//        *mode = 0;
//      }
//    }
//  }
//  *lastkey = index;
//  itoa(*ass, p, 10);
//
//    snprintf(third, sizeof third, "%s %i", "UberTimer:", *ass);
//  oled_WriteString(third, Font_7x10, White);
//  oled_UpdateScreen();
//}

void oled_Reset( void ) {
  oled_Fill(Black);
  oled_SetCursor(0, 0);
}
/* 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 */

  /* 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,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Вывод

В данной лабораторной работе мы научились работать с драйвером OLED-дисплея, а также повторили информацию из прошлой лабораторной работы, связанную с шиной I2C.