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\_REGULATOR\_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.