УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Проектирование вычислительных систем»

**Лабораторная работа №2**

Вариант 15

Студент

*Крюков А. Ю.*

*Патутин В. М.*

*P34101*

Преподаватель

*Пинкевич В. Ю.*

Санкт-Петербург, 2022 г.

Цель работы

1. Изучить протокол передачи данных по интерфейсу UART.

2. Получить базовые знания об организации системы прерываний в микроконтроллерах на примере микроконтроллера STM32.

3. Изучить устройство и принципы работы контроллера интерфейса UART, получить навыки организации обмена данными по UART в режимах опроса и прерываний.

Задание лабораторной работы

Разработать и реализовать два варианта драйверов UART для стенда SDK-1.1M: с использованием и без использования прерываний. Драйверы, использующие прерывания, должны обеспечивать работу в «неблокирующем» режиме (возврат из функции происходит сразу же, без ожидания окончания приема/отправки), а также буферизацию данных для исключения случайной потери данных. В драйвере, не использующем прерывания, функция приема данных также должна быть «неблокирующей». Прерывания от соответствующего блока UART должны быть запрещены при использовании режима «без прерываний».

Написать с использованием разработанных драйверов программу, которая выполняет определенную вариантом задачу. Для всех вариантов должно быть реализовано два режима работы программы: с использованием и без использования прерываний. Каждый принимаемый стендом символ должен отсылаться обратно, чтобы он был выведен в консоли (так называемое «эхо»). Каждое новое сообщение от стенда должно выводиться с новой строки. Если вариант предусматривает работу с командами, то на каждую команду должен 67 выводиться ответ, определенный в задании или «ОК», если ответ не требуется. Если введена команда, которая не поддерживается, должно быть выведено сообщение об этом.

Вариант задания

Доработать программу кодового замка. Теперь ввод кода должен происходить не с помощью кнопки стенда, а по UART. После ввода единственно верной последовательности из не более чем восьми латинских букв без учета регистра и цифр должен загореться зеленый светодиод, обозначающий «открытие» замка. Светодиод горит некоторое время, потом гаснет, и система вновь переходит в «режим ввода». Каждый неправильно введенный элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием желтого. После трех неправильных вводов начинает мигать красный светодиод, и через некоторое время система вновь возвращается в «режим ввода». Если код не введен до конца за некоторое ограниченное время, происходит сброс в «начало».

Должно быть предусмотрено изменение отпирающей последовательности, что производится следующей последовательностью действий:

– ввод символа «+»;

– ввод новой последовательности, который завершается либо по нажатию enter, либо по достижению восьми значений;

– стенд отправляет сообщение произвольного содержания, спрашивая, сделать ли последовательность активной, и запрашивает подтверждение, которое должно быть сделано вводом символа y;

– после ввода y вверденная последовательность устанавливается как активная.

Включение/отключение прерываний должно осуществляться нажатием кнопки на стенде и сопровождаться отправкой в последовательный порт сообщения произвольного содержания, сообщающего, какой режим включен (с прерываниями или без прерываний).

Исходный код

/\* 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** "usart.h"

**#include** "gpio.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

**#include** <stdint.h>

**#include** <string.h>

**#include** <ctype.h>

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

**typedef** **struct**

{

uint8\_t length;

**char** value[8];

uint8\_t is\_correct;

uint8\_t current\_pos;

uint8\_t number\_of\_mistakes;

uint32\_t input\_time\_start;

uint32\_t input\_time\_limit;

} Password;

**typedef** **enum**

{

*INPUT\_CHANGE\_PASS\_COMMAND* = 0,

*INPUT\_LOWERCASE\_ALPHA* = 1,

*INPUT\_UPPERCASE\_ALPHA* = 2,

*INPUT\_ENTER* = 3,

*INPUT\_UNKNOWN* = 4

} INPUT\_TYPE;

**typedef** **enum**

{

*PASSWORD\_CHANGE\_NONE* = 0,

*PASSWORD\_CHANGE\_INPUT* = 1,

*PASSWORD\_CHANGE\_CONFIRM* = 2

} PASSWORD\_CHANGE\_STATE;

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

**#define** MAX\_PASSWORD\_LEN 8

**#define** DEFAULT\_PASSWORD "password"

**#define** DEFAULT\_PASSWORD\_LEN strlen(DEFAULT\_PASSWORD)

**#define** ERR\_PASS\_EMPTY "\r\nPassword must have at least 1 character\r\n"

**#define** MSG\_CONFIRM\_PASS "\r\nApply new password (y/n)? [n] "

**#define** MSG\_PASS\_CHANGED "\r\nPassword has been changed\r\n"

**#define** MSG\_PASS\_UNCHANGED "\r\nPassword has not been changed\r\n"

**#define** MSG\_ENTER\_NEW\_PASS "\r\nNew password>"

**#define** MSG\_INTERRUPT\_MODE\_ENABLED "\r\nInterrupt mode - on\r\n"

**#define** MSG\_INTERRUPT\_MODE\_DISABLED "\r\nInterrupt mode - off\r\n"

**#define** COM\_CHANGE\_PASS '+'

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

**#define** MIN\_SEC\_TO\_MS(minutes, seconds) ((minutes) \* 60 \* 1000 + (seconds) \* 1000)

/\* USER CODE END PM \*/

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

/\* USER CODE BEGIN PV \*/

**char** init\_password\_value[8];

uint8\_t init\_password\_len;

**char** new\_password\_value[8];

uint8\_t new\_password\_len;

**const** uint32\_t DEFAULT\_PASSWORD\_INPUT\_TIME\_LIMIT = MIN\_SEC\_TO\_MS(1, 0);

uint8\_t char\_readed = 0;

uint8\_t char\_written = 0;

uint8\_t interruption\_mode\_enabled = 0;

PASSWORD\_CHANGE\_STATE password\_change\_mode = 0;

/\* 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 \*/

**void** **blink\_yellow**()

{

**for** (uint8\_t i = 0; i < 10; i++)

{

HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_14);

HAL\_Delay(50);

}

}

**void** **blink\_red**()

{

**for** (uint8\_t i = 0; i < 10; i++)

{

HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_15);

HAL\_Delay(50);

}

}

**void** **light\_red**()

{

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_15, *GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_14, *GPIO\_PIN\_RESET*);

HAL\_Delay(MIN\_SEC\_TO\_MS(0, 2));

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_15, *GPIO\_PIN\_RESET*);

}

**void** **light\_green**()

{

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_13, *GPIO\_PIN\_SET*);

HAL\_Delay(MIN\_SEC\_TO\_MS(0, 2));

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_13, *GPIO\_PIN\_RESET*);

}

**void** **send\_msg**(**const** **char** \*msg, uint8\_t msg\_len)

{

**if** (msg\_len == 0) // When we send the messages which we define in the top, go this branch.

msg\_len = **strlen**(msg);

**if** (interruption\_mode\_enabled){

char\_written = 0;

HAL\_UART\_Transmit\_IT(&huart6, msg, msg\_len);

**while** (!char\_written); // If the char is not written, then keep waiting here

}

**else**

HAL\_UART\_Transmit(&huart6, msg, msg\_len, MIN\_SEC\_TO\_MS(0, 1));

}

uint8\_t **is\_password\_input\_time\_expired**(Password \*password)

{

**return** HAL\_GetTick() > (password->input\_time\_start + password->input\_time\_limit);

}

**char** **read\_input**(Password \*password)

{

**static** uint8\_t btn\_pressed = 0;

char\_readed = 0;

**char** c;

**while** (!char\_readed)

{

**if** (interruption\_mode\_enabled)

{

HAL\_UART\_Receive\_IT(&huart6, &c, 1); // For interrupt mode

}

**else**

{

HAL\_StatusTypeDef status = HAL\_UART\_Receive(&huart6, &c, 1, MIN\_SEC\_TO\_MS(0, 1)); // Receive 1 char.

**if** (status == *HAL\_OK*) // If we get it successfully, then set signal.

char\_readed = 1;

}

// The button is pressed and the pressing has not yet been fixed (reset-low level)

**if** (HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_15) == *GPIO\_PIN\_RESET* && !btn\_pressed)

{

btn\_pressed = 1;

interruption\_mode\_enabled = !interruption\_mode\_enabled;

**if** (interruption\_mode\_enabled)

send\_msg(MSG\_INTERRUPT\_MODE\_ENABLED, 0);

**else**

send\_msg(MSG\_INTERRUPT\_MODE\_DISABLED, 0);

}

// pressing was fixed and the button was released

**else** **if** (HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_15) == *GPIO\_PIN\_SET* && btn\_pressed)

btn\_pressed = 0;

// User takes too long to enter password -> reset progress

**if** (!password\_change\_mode && is\_password\_input\_time\_expired(password))

init\_password(password);

}

**return** c;

}

**void** **HAL\_UART\_RxCpltCallback**(UART\_HandleTypeDef \*huart)

{

char\_readed = 1;

}

**void** **HAL\_UART\_TxCpltCallback**(UART\_HandleTypeDef \*huart)

{

char\_written = 1;

}

INPUT\_TYPE **input\_type**(**char** val)

{

**if** (val == COM\_CHANGE\_PASS)

**return** *INPUT\_CHANGE\_PASS\_COMMAND*;

**if** (val == '\r')

**return** *INPUT\_ENTER*;

**if** (val >= 'a' && val <= 'z')

**return** *INPUT\_LOWERCASE\_ALPHA*;

**if** (val >= 'A' && val <= 'Z')

**return** *INPUT\_UPPERCASE\_ALPHA*;

**return** *INPUT\_UNKNOWN*;

}

**void** **init\_password**(Password \*password)

{

**memcpy**(password->value, init\_password\_value, init\_password\_len);

password->length = init\_password\_len;

password->current\_pos = 0;

password->is\_correct = 0;

password->number\_of\_mistakes = 0;

password->input\_time\_start = HAL\_GetTick();

password->input\_time\_limit = DEFAULT\_PASSWORD\_INPUT\_TIME\_LIMIT;

}

**void** **reset\_number\_of\_mistakes**(Password \*password)

{

password->number\_of\_mistakes = 0;

}

**void** **process\_password**(Password \*password, **char** val)

{

uint32\_t pass = password->value[password->current\_pos] == val;

**if** (pass)

password->current\_pos++; // If correct, then set the position to next.

**else**

password->current\_pos = 0; // If anyone is incorrect, then reset.

**if** (password->current\_pos == password->length) // If check the final one, and it's correct, then unlock

password->is\_correct = 1;

}

**void** **finish\_password\_change**(Password \*password, **char** val)

{

**if** (val == 0) // Call this function in the first time. Or we get the too long password, we will be there. And let the user try again.

{

password\_change\_mode = *PASSWORD\_CHANGE\_CONFIRM*;

send\_msg(MSG\_CONFIRM\_PASS, 0);

**return**;

}

**if** (val == 'y') // If we get 'y', then change the password.

{

password\_change\_mode = *PASSWORD\_CHANGE\_NONE*;

**if** (new\_password\_len < 1) // If there is no new pwd, then send error message.

{

send\_msg(ERR\_PASS\_EMPTY, 0);

**return**;

}

**memcpy**(init\_password\_value, new\_password\_value, MAX\_PASSWORD\_LEN);

init\_password\_len = new\_password\_len;

init\_password(password);

send\_msg(MSG\_PASS\_CHANGED, 0); // Send message for changing pwd successfully

}

**else** // If input any other things, then don't change pwd.

{

password\_change\_mode = *PASSWORD\_CHANGE\_NONE*;

send\_msg(MSG\_PASS\_UNCHANGED, 0);

}

}

**void** **process\_password\_change**(Password \*password, **char** val)

{

**if** (new\_password\_len <= MAX\_PASSWORD\_LEN) // If the length is ok, then reset.

new\_password\_value[new\_password\_len++] = val;

**if** (new\_password\_len > MAX\_PASSWORD\_LEN) // If not, then let user retry.

finish\_password\_change(password, 0);

}

**void** **check\_password**(Password \*password)

{

**if** (password->is\_correct) // If it's all right, then unlock, reset the states and light green.

{

light\_green();

init\_password(password);

**return**;

}

**if** (password->current\_pos == 0) // If it's incorrect.

{

password->number\_of\_mistakes += 1; // count mistakes.

**if** (password->number\_of\_mistakes == 3) // If fail 3 times, then light red and reset mistake counter.

{

light\_red();

reset\_number\_of\_mistakes(password);

}

**else** // Fail less than 3 times, then just blink.

blink\_red();

}

**else** // If correct but not over, then blink yellow.

blink\_yellow();

}

/\* 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\_USART6\_UART\_Init();

/\* USER CODE BEGIN 2 \*/

Password password = {}; // Init pwd.

**memcpy**(init\_password\_value, DEFAULT\_PASSWORD, DEFAULT\_PASSWORD\_LEN); // set the 'password' as the default password.

init\_password\_len = DEFAULT\_PASSWORD\_LEN;

init\_password(&password);

interruption\_mode\_enabled = 0;

password\_change\_mode = *PASSWORD\_CHANGE\_NONE*;

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

**char** val = read\_input(&password); // Read the input

send\_msg(&val, 1); // Show what do we read.

**switch** (input\_type(val)) // Determine the type of input

{

**case** *INPUT\_CHANGE\_PASS\_COMMAND*: // If we input '+' to change pwd.

**if** (password\_change\_mode == *PASSWORD\_CHANGE\_NONE*)

{

password\_change\_mode = *PASSWORD\_CHANGE\_INPUT*;

**memset**(new\_password\_value, 0, MAX\_PASSWORD\_LEN);

new\_password\_len = 0;

send\_msg(MSG\_ENTER\_NEW\_PASS, 0);

}

**break**;

**case** *INPUT\_UPPERCASE\_ALPHA*: // If we get uppercase, then change to lowercase. No break, so we will go lowercase again.

val -= ('A' - 'a');

**case** *INPUT\_LOWERCASE\_ALPHA*: // If we get lowercase, then Check which mode is on now.

**switch** (password\_change\_mode)

{

**case** *PASSWORD\_CHANGE\_NONE*: // If we don't input '+', then check the password.

process\_password(&password, val);

check\_password(&password);

**break**;

**case** *PASSWORD\_CHANGE\_INPUT*: // If we do, then change the password.

process\_password\_change(&password, val);

**break**;

**case** *PASSWORD\_CHANGE\_CONFIRM*: // If it's the final position, then finish the change.

finish\_password\_change(&password, val);

**break**;

}

**break**;

**case** *INPUT\_ENTER*: // If we press Enter, then finish the changing of password.

**if** (password\_change\_mode == *PASSWORD\_CHANGE\_INPUT*)

finish\_password\_change(&password, 0);

**break**;

**case** *INPUT\_UNKNOWN*: // If input unknown thing, then just don't return anything.

**break**;

}

}

/\* 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\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

**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\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

**if** (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != *HAL\_OK*)

{

Error\_Handler();

}

}

/\* USER CODE BEGIN 4 \*/

/\* 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) \*/

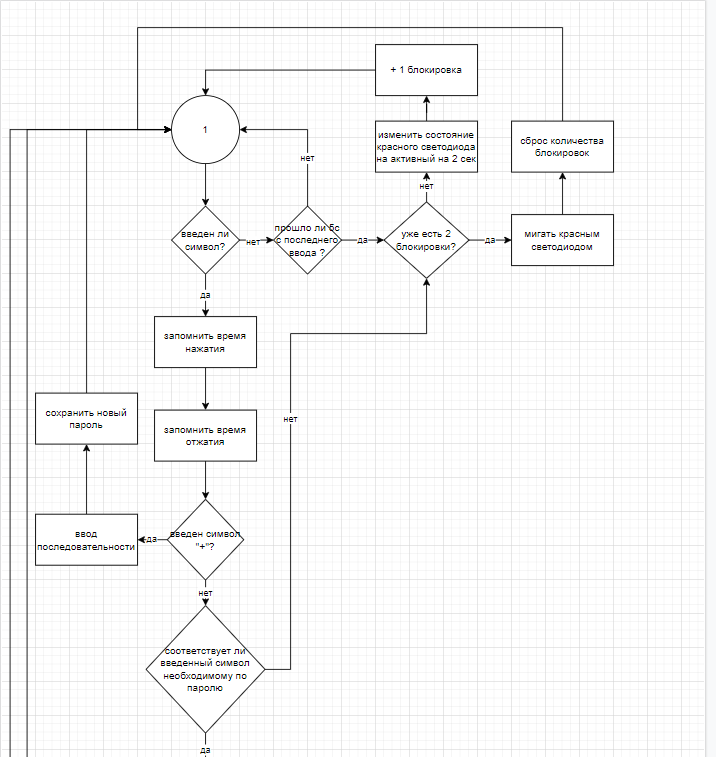
/\* USER CODE END 6 \*/

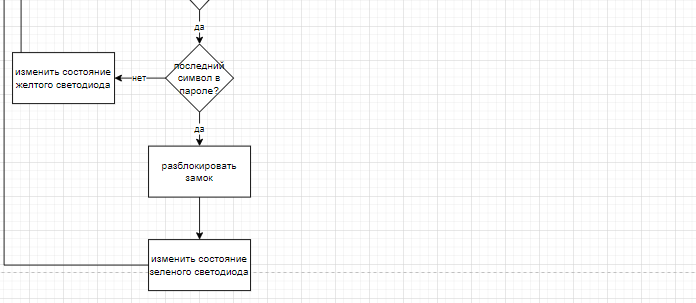
}

**#endif** /\* USE\_FULL\_ASSERT \*/

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

Блок схема





Вывод

В этой лабораторной работе мы изучили протокол передачи данных по интерфейсу UART, получили базовые знания об организации системы прерываний в микроконтроллерах на примере микроконтроллера STM32, а также изучили устройство и принципы работы контроллера интерфейса UART, получить навыки организации обмена данными по UART в режимах опроса и прерываний. В результате мы разработали программу, которая работает по принципу замка.