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;
```

```c
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 */
```

```c
/* 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.
```

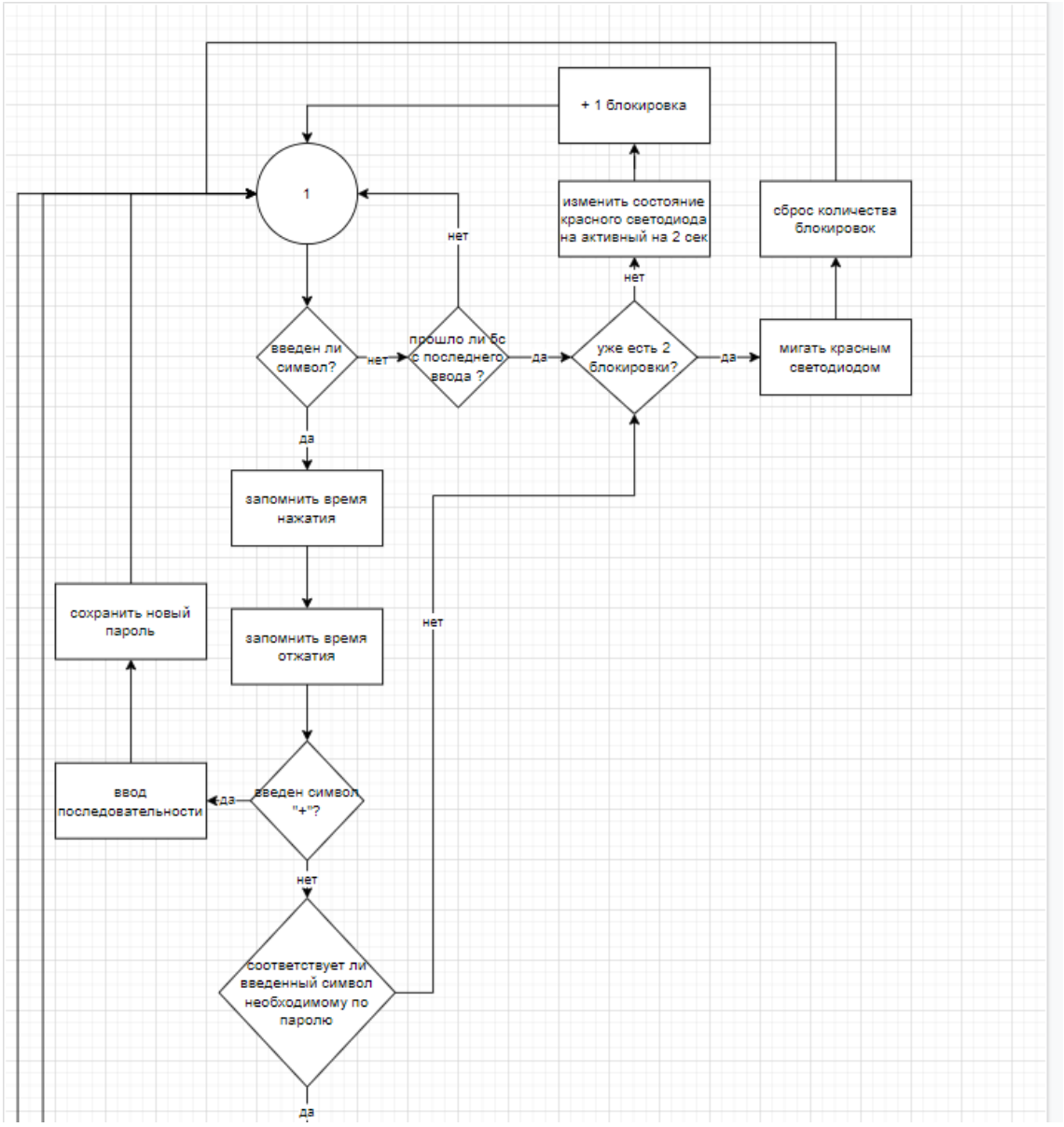
```
    {
      send_msg(ERR_PASS_EMPTY, 0);
      return;
    }

    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
  */
```

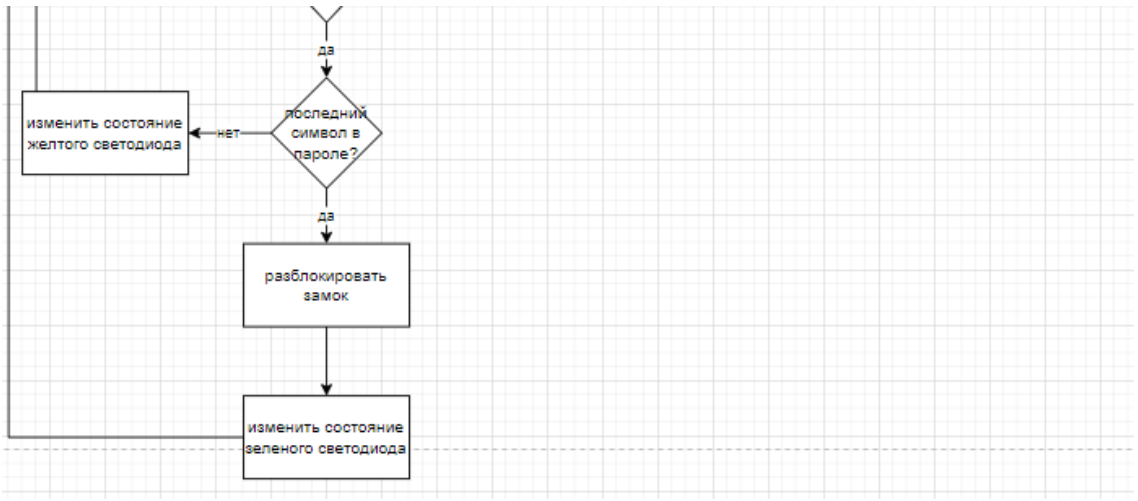
```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* 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


+ 1 блокировка
1
изменить состояние красного светодиода на активный на 2 сек
сброс количества блокировок
нет
нет
введен ли символ?
нет
прошло ли 5с с последнего ввода ?
да
уже есть 2 блокировки?
да
мигать красным светодиодом
да
запомнить время нажатия
сохранить новый пароль
запомнить время отжатия
нет
ввод последовательности
да
введен символ "+"?
нет
соответствует ли введенный символ необходимому по паролю
да


да
изменить состояние желтого светодиода
нет
последний символ в пароле?
да
разблокировать замок
изменить состояние зеленого светодиода

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.