ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Профессор департамента программной инженерии факультета компьютерных наук, Профессор.

_____ И. Р. Агамирзян
«__» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия»

_____ В. В. Шилов
«__» _____ 2020 г.

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС УПРАВЛЕНИЯ ИДЕНТИФИКАЦИОННЫМИ ДАННЫМИ
Текст программы

ЛИСТ УТВЕРЖДЕНИЯ
RU.17701729.01.01-01 12 01-1-ЛУ

Исполнитель студент группы БПИ173
_____ / Дубина Д. О. /
«__» _____ 2020 г.

Подп. И дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. И дата	
Инв. № подл.	

Москва 2020

УТВЕРЖДЕНО
RU.17701729.01.01-01 12 01-1-ЛУ

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС УПРАВЛЕНИЯ ИДЕНТИФИКАЦИОННЫМИ ДАННЫМИ

Текст программы
RU.17701729.01.01-01 12 01-1

Листов 50

Подп. И дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. И дата	
Инв. № подл.	

Москва 2020

Содержание

1. Основные термины и определения

I2C - последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов. Использует две двунаправленные линии связи (SDA и SCL), применяется для соединения низкоскоростных периферийных компонентов с процессорами и микроконтроллерами (например, на материнских платах, во встраиваемых системах, в мобильных телефонах).

Прерывание - сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), которая реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

Регистр - последовательное или параллельное логическое устройство, используемое для хранения n-разрядных двоичных чисел и выполнения преобразований над ними.

SPI - последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии. SPI также иногда называют четырёхпроводным интерфейсом.

EEPROM- электрически стираемое перепрограммируемое ПЗУ (ЭСППЗУ), один из видов энергонезависимой памяти (таких, как PROM и EPROM). Память такого типа может стираться и заполняться данными до миллиона раз.

FLASH- разновидность полупроводниковой технологии электрически перепрограммируемой памяти (EEPROM). Это же слово используется в электронной схемотехнике для обозначения технологически законченных решений постоянных запоминающих устройств в виде микросхем на базе этой полупроводниковой технологии. В быту это словосочетание закрепилось за широким классом твердотельных устройств хранения информации.

2. Текст программы комплекса

2.1. Файл main.h

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.h
 * @brief          : Header for main.c file.
 *                   This file contains the common defines of the application.
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2020 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 */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f2xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ssd1306.h"
#include "fonts.h"
```

```
#include "stm32f2xx_it.h"
/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */
#define usbBuferSize 128
#define usbBlockSize 64

//Переменные кнопок
_Bool leftButtonStatus;
_Bool rightButtonStatus;
_Bool bothButtonStatus;

//шаги инициации
uint8_t* initStatus;
uint8_t* initStatusStep1;
uint8_t* initStatusStep2;
uint8_t* restoreStatusStep1;
uint8_t* restoreStatusStep2;

//шаги установки пароля
uint8_t* setPasswordStep1;
uint8_t* setPasswordStep2;

//шаги установки защиты
uint8_t* ProtectType;
uint8_t* setProtectTypeStep1;
uint8_t* setProtectTypeStep2;
uint8_t* passwordInputStatus;

//буферы
uint8_t dataReciveBufer[usbBuferSize];
int16_t bufer[20];
char bufer2[64];
```

```
//Данные
uint8_t DataCount;

//переменные команд
uint8_t* restoreStatus;
uint8_t* setPasswordStatus;
uint8_t* settingsStatus;
uint8_t* ResetComand;
uint8_t* chpassComand;
uint8_t* cProtectComand;

//переменные разрешений
uint8_t* exportEnable;
uint8_t* addDeviceEnable;
uint8_t* M5PCIDdefaultIsGetted;
uint8_t* isInit;
uint8_t* dataTransferEnable;

//переменные меню
uint8_t DataInfoMenu;
uint8_t* devpreinitmenu;
uint8_t* datasettingsStatus;
uint8_t* menuStatus;
uint8_t* settingsMenuStatus;
uint8_t* dataControlMenuStatus;

//Секретные ключи
int16_t privateKey[20];
int16_t publicKey[20];
char* passFrase[12];

//генератор рандоманых чисел
uint16_t RNGNumbers[12];

//хранение данных о надежных ПК
uint8_t M5PPCIDCount;
uint8_t PCIDOne[24];
uint8_t PCIDTwo[24];
```

```
uint8_t PCIDThre[24];
uint8_t PCIDFour[24];
uint8_t PCIDFive[24];
uint8_t PCIDbuf[24];
uint8_t PCIDSix[24];
uint8_t PCIDSeven[24];

typedef struct {
        char *login[16];
        char *password[16];
        char *url[16];
        char *number[16];
}accauntBlock;

typedef struct {
        int16_t pointer;
        accauntBlock *blocks;
}menuAB;

menuAB menu;
int8_t pointer;
int8_t Unlocked;
int8_t updownpointer;
uint8_t password[6];
uint8_t imputpassword[6];

uint8_t test[24];

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */
```

```
void leftButtonActions(void);
void rightButtonActions(void);
void bothButtonActions(void);
void generatePassFrase(void);
void changePasswordData(void);
void generateRandomNumbers(uint16_t blocknumber,uint16_t filter);

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

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

2.2. Файл main.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2020 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 "usb_device.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "data.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 ---------------------------------------------------------*/
CRC_HandleTypeDef hcrc;

CRYP_HandleTypeDef hcryp;
__ALIGN_BEGIN static const uint32_t pKeyCRYP[6] __ALIGN_END = { 0x00000000,
               0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000 };

HASH_HandleTypeDef hhash;
```

```
I2C_HandleTypeDef hi2c1;

RNG_HandleTypeDef hrng;

TIM_HandleTypeDef htim6;

/* USER CODE BEGIN PV */

char ghf[64];
uint16_t dictionarySeze = 500;
char *wordsForPassFrase[] = { "aiken", "durga", "essen", "evers", "haiti",
                "horus", "issus", "kamet", "klimt", "laius", "locke", "lorre", "lowry",
                "mamet", "marti", "medea", "niger", "oates", "potos", "quito", "senor",
                "turin", "ushas", "wells", "aboil", "acari", "acoma", "actin", "adage",
                "adeem", "adfix", "adion", "afore", "agasp", "aglet", "alans", "albin",
                "algic", "alody", "amide", "ample", "ancon", "anker", "annat", "annie",
                "anous", "aoife", "aotes", "argil", "aries", "arkab", "arneb", "artal",
                "arvel", "arzan", "astay", "atter", "avoid", "awash", "axion", "axoid",
                "ayond", "bache", "bahay", "bajra", "balai", "baloo", "bando", "barbe",
                "bayal", "beamy", "becut", "bedad", "beget", "belga", "bemad", "benne",
                "betta", "bezel", "birle", "blast", "blimp", "bloat", "bocal", "bodge",
                "bonny", "boose", "boral", "bortz", "bosom", "brass", "bring", "broll",
                "buddy", "bully", "bushy", "butch", "cajun", "canna", "canoe", "carol",
                "carse", "carya", "cased", "casse", "catti", "celom", "chaya", "cheve",
                "chips", "choel", "claim", "clary", "claut", "clava", "cleek", "cloot",
                "clove", "clump", "coapt", "cobia", "cobus", "cogue", "colla", "comma",
                "copsy", "corps", "covet", "crash", "cress", "creta", "crete", "croci",
                "daffy", "dafla", "darer", "deign", "denda", "dewey", "diner", "dinus",
                "disco", "dixit", "dizzy", "domal", "douar", "dover", "dreng", "dropt",
                "drown", "drunk", "dural", "dusun", "easer", "echis", "elmer", "elops",
                "elute", "elves", "embed", "emcee", "emmer", "envoy", "erian", "erick",
                "erose", "erupt", "every", "exdie", "fanon", "fanti", "fanwe", "fatal",
                "favus", "fedia", "feint", "fesse", "fiard", "finer", "fiver", "flame",
                "flare", "flary", "fleam", "fleet", "flesh", "flong", "foaly", "fogle",
                "forth", "fosse", "found", "freed", "freit", "fresh", "fritt", "frizz",
                "fubsy", "futon", "gaine", "ganch", "gatch", "genin", "genus", "gipon",
                "gippy", "given", "glaik", "gland", "glazy", "gledy", "gloom", "goban",
                "golee", "gorra", "gourd", "gouty", "grail", "grebo", "gripy", "gugal",
```

```
"gypsy", "habit", "halse", "harpa", "herne", "hevea", "hocky", "howso",
"humph", "ictic", "iddat", "idose", "illth", "imber", "infer", "inlaw",
"innet", "input", "irfan", "irone", "itchy", "jaman", "jamie", "jenny",
"jural", "kafiz", "kanji", "kapai", "kappe", "keleh", "kench", "khaya",
"khoja", "kissy", "klosh", "known", "kodro", "kokio", "krems", "lacer",
"lacet", "lairy", "lammy", "larch", "large", "lever", "ligas", "lived",
"lobed", "loner", "lotta", "louey", "lowth", "lucan", "luigi", "lyard",
"maggy", "mahdi", "maidy", "mamma", "manei", "mapau", "masty", "mayan",
"mease", "merak", "merop", "metal", "metol", "miaul", "mikie", "minty",
"misty", "moity", "mossy", "mourn", "moyen", "muffy", "namda", "nanes",
"nanga", "nasch", "nasty", "navar", "nayar", "nazir", "nigre", "niqab",
"niter", "norie", "nunni", "nuque", "nyxis", "oasal", "oasis", "ohmic",
"onymy", "otary", "oxbow", "oxlip", "pacer", "padre", "padus", "palar",
"palpi", "parra", "parse", "parts", "pasmo", "patly", "peasy", "peaty",
"pedal", "peggy", "pekan", "penta", "pesky", "phase", "pinko", "pinky",
"pinny", "plaga", "plaid", "plica", "plyer", "pokom", "pommy", "poria",
"prase", "pudic", "puppy", "quart", "quoit", "raggy", "raker", "raman",
"raphe", "rapic", "rebid", "rebus", "refan", "renet", "repew", "resay",
"rewed", "richt", "rinse", "rohob", "rondo", "royal", "runed", "ryder",
"sabra", "salma", "samen", "sanai", "sandy", "savor", "schwa", "sclaw",
"scope", "scout", "scrim", "segno", "senci", "septi", "seral", "sereh",
"serum", "seven", "shahi", "shiko", "shire", "shive", "shoya", "sided",
"sidth", "sigeh", "simar", "sinew", "sirih", "skank", "skill", "slent",
"slive", "snafu", "snake", "sneap", "spale", "spang", "spece", "sprig",
"squab", "steri", "stilt", "stoff", "stong", "stosh", "strag", "stree",
"strow", "stunk", "sturt", "suant", "suety", "surfy", "swile", "swoop",
"tahil", "taich", "taler", "tangi", "tanti", "tanzy", "taraf", "techy",
"tellt", "tenty", "terce", "terse", "tetum", "thatn", "thawy", "thymy",
"tilde", "titar", "tizzy", "toity", "toned", "tongs", "torah", "torma",
"trill", "trixy", "trope", "truck", "tryst", "tufty", "tumor", "turco",
"uinal", "unhad", "unhid", "unket", "upend", "urare", "ureic", "utick",
"vagas", "valve", "vealy", "vepse", "vibex", "vicar", "virtu", "volet",
"volva", "vuggy", "wabby", "walth", "waltz", "wamus", "wawah", "weaky",
"wendy", "whalp", "wheen", "while", "wined", "wings", "wisse", "words",
"wrack", "wrive", "xylon", "yummy", "zanze", "ziega", "zonta" };

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
```

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CRYP_Init(void);
static void MX_HASH_Init(void);
static void MX_I2C1_Init(void);
static void MX_RNG_Init(void);
static void MX_CRC_Init(void);
static void MX_TIM6_Init(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_CRYP_Init();
MX_HASH_Init();
MX_I2C1_Init();
MX_RNG_Init();
MX_CRC_Init();
MX_USB_DEVICE_Init();
MX_TIM6_Init();
/* USER CODE BEGIN 2 */
ssd1306_Init();
ssd1306_Fill(Black);
uint8_t check = 0;
uint8_t lokalcheck = 1;

initConstants();

ssd1306_UpdateScreen();

HAL_Delay(500);
if (isInit == 0) {
	deviceIsntInit();
}

else {
	if (ProtectType == 0) {
		Unlocked = 1;
		menuStatus = 1;
		initMenu();
	} else if (ProtectType == 1) {

		setPasswordStep2 = 1;
		passwordInputStatus = 1;
		setPasswordProcess2();
	} else if (ProtectType == 2) {
```

```
                ssd1306_Fill(Black);
                ssd1306_SetCursor(2, 2);
                ssd1306_WriteString("Conect device to", Font_7x10, White);
                ssd1306_SetCursor(2, 12);
                ssd1306_WriteString("your safe PC", Font_7x10, White);
                ssd1306_UpdateScreen();
                //

        } else if (ProtectType == 3) {

                ssd1306_Fill(Black);
                ssd1306_SetCursor(2, 2);
                ssd1306_WriteString("Conect device to", Font_7x10, White);
                ssd1306_SetCursor(2, 12);
                ssd1306_WriteString("your safe PC", Font_7x10, White);
                ssd1306_UpdateScreen();
                //

        }

}

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1) {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */

        HAL_Delay(300);
        if (dataReciveBufer[0] != 0) {
                //начать инициацию

                if (dataReciveBufer[0] == 'P') {

                        if (M5PCIDdefaultIsGetted == 0) {
                                for (uint8_t i = 0; i < 24; i++) {
```

```
                PCIDbuf[i] = dataReciveBufer[i + 2];
        }
        for (uint8_t i = 0; i < 24; i++) {
                PCIDOne[i] = PCIDbuf[i];
        }
        M5PCIDdefaultIsGetted = 1;
        M5PPCIDCount = 1;

        uploadPCIDmas();
        uploadPCIDcount();
        uploadIsfirstPC();
        CDC_Transmit_FS("OK", 2);

        initStatus = 1;
        initChoseProcess();
} else if (addDeviceEnable == 1) {
        for (uint8_t i = 0; i < 24; i++) {
                PCIDbuf[i] = dataReciveBufer[i + 2];
        }
        downloadPCIDmas();
        downloadPCIDcount();
        lokalcheck = 1;
        check = 0;
        for (uint8_t i = 0; i < 24; i++) {
                if (PCIDbuf[i] != PCIDOne[i]) {
                        check = 1;
                }
        }
        if (check == 0)
                lokalcheck = 0;
        check = 0;
        for (uint8_t i = 0; i < 24; i++) {
                if (PCIDbuf[i] != PCIDSeven[i]) {
                        check = 1;
                }
        }
        if (check == 0)
                lokalcheck = 0;
        check = 0;
```

```
for (uint8_t i = 0; i < 24; i++) {
	if (PCIDbuf[i] != PCIDSix[i]) {
		check = 1;
	}
}
if (check == 0)
	lokalcheck = 0;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
	if (PCIDbuf[i] != PCIDFour[i]) {
		check = 1;
	}
}
if (check == 0)
	lokalcheck = 0;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
	if (PCIDbuf[i] != PCIDFive[i]) {
		check = 1;
	}
}
if (check == 0)
	lokalcheck = 0;

if (lokalcheck == 1) {
	if (M5PPCIDCount == 1) {
		M5PPCIDCount++;
		//PCIDSeven=PCIDbuf;
		for (uint8_t i = 0; i < 24; i++) {
			PCIDSeven[i] = PCIDbuf[i];
		}
		uploadPCIDmas();
		uploadPCIDcount();
		CDC_Transmit_FS("OK", 2);
	} else if (M5PPCIDCount == 2) {
		M5PPCIDCount++;
		//PCIDSix=PCIDbuf;
		for (uint8_t i = 0; i < 24; i++) {
			PCIDSix[i] = PCIDbuf[i];
```

```
                }
                uploadPCIDmas();
                uploadPCIDcount();
                CDC_Transmit_FS("OK", 2);
            } else if (M5PPCIDCount == 3) {
                M5PPCIDCount++;
                //PCIDFour=PCIDbuf;
                for (uint8_t i = 0; i < 24; i++) {
                    PCIDFour[i] = PCIDbuf[i];
                }
                uploadPCIDmas();
                uploadPCIDcount();
                CDC_Transmit_FS("OK", 2);
            } else if (M5PPCIDCount == 4) {
                M5PPCIDCount++;
                //PCIDFive=PCIDbuf;
                for (uint8_t i = 0; i < 24; i++) {
                    PCIDFive[i] = PCIDbuf[i];
                }
                uploadPCIDmas();
                uploadPCIDcount();
                CDC_Transmit_FS("OK", 2);
            } else if (M5PPCIDCount == 5) {
                CDC_Transmit_FS("Your cant add new device", 26);
            }
            addDeviceEnable = 0;
            uploadaddDeviceEnable();
        } else {
            CDC_Transmit_FS("The device already added", 26);
            addDeviceEnable = 0;
            uploadaddDeviceEnable();
        }
    }

    else {
        for (uint32_t i = 0; i < 24; i++) {
            PCIDbuf[i] = dataReciveBufer[i + 2];
        }
        downloadPCIDmas();
```

```
downloadPCIDcount();
lokalcheck = 1;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
        if (PCIDbuf[i] != PCIDOne[i]) {
                check = 1;
        }
}
if (check == 0)
        lokalcheck = 0;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
        if (PCIDbuf[i] != PCIDSeven[i]) {
                check = 1;
        }
}
if (check == 0)
        lokalcheck = 0;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
        if (PCIDbuf[i] != PCIDSix[i]) {
                check = 1;
        }
}
if (check == 0)
        lokalcheck = 0;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
        if (PCIDbuf[i] != PCIDFour[i]) {
                check = 1;
        }
}
if (check == 0)
        lokalcheck = 0;
check = 0;
for (uint8_t i = 0; i < 24; i++) {
        if (PCIDbuf[i] != PCIDFive[i]) {
                check = 1;
        }
```

```
                }
                if (check == 0)
                        lokalcheck = 0;

                if (lokalcheck == 0) {
                        CDC_Transmit_FS("OK", 2);
                        if (ProtectType == 2) {
                                Unlocked = 1;
                                menuStatus = 1;
                                initMenu();

                        } else if (ProtectType == 3) {
                                setPasswordStep2 = 1;
                                passwordInputStatus = 1;
                                setPasswordProcess2();
                        }
                        //HAL_Delay(1000);
                        //CDC_Transmit_FS("NO", 2);
                } else {
                        //CDC_Transmit_FS("Your devise is unsuported", 26);
                        CDC_Transmit_FS("NO", 2);
                }
        }

        //CDC_Transmit_FS("OK", 2);

}

if (Unlocked == 1) {
        //добавить идент. данные
        if (dataReciveBufer[0] == 'N') {
                char str[64] = "";
                uint8_t pasgen[37] = "";
                uint8_t domen[16];
                for (uint32_t i = 0; i < 16; i++) {
                        str[i] = dataReciveBufer[i + 18];
                        domen[i] = dataReciveBufer[i + 18];
                }
                downloadPrivate();
```

```
//uint8_t pasgen[37];
for (int i = 0; i < 20; i++) {
	pasgen[i] = privateKey[i];
}
for (int i = 20; i < 36; i++) {
	pasgen[i] = dataReciveBufer[i - 2];
}
pasgen[16] = (uint8_t) privateKey
		& (uint8_t) domen + (uint8_t) 1;
HAL_HASH_Init(&hhash);
HAL_HASH_SHA1_Start(&hhash, &pasgen, 36, &bufer,
		HAL_MAX_DELAY);
HASH_Finish(&hhash, &bufer, HAL_MAX_DELAY);
char pass[16];
for (int i = 0; i < 16; i++) {
	pass[i] = bufer[i];
	bufer[i] = 0;
}

for (uint32_t i = 16; i < 32; i++) {
	str[i] = pass[i - 16];
}
for (uint32_t i = 32; i < 48; i++) {
	str[i] = dataReciveBufer[i - 30];
}
char iter[16] = "1               ";
for (uint32_t i = 48; i < 64; i++) {
	str[i] = iter[i - 48];
}
//str[0]=DataCount;
writeToEeprom(0x1000 + DataCount * 64, str, 64);
DataCount++;
uploadDataCount();
accauntBlock blocksbuf[DataCount + 1];
for (uint16_t i = 0; i < DataCount; i++) {

	readFromEeprom(0x1000 + 64 * i, bufer2, 64);
	stringToStruct(&bufer2, &blocksbuf[i]);
}
```

```
        menu.blocks = blocksbuf;
        CDC_Transmit_FS(str, 64);

        if (menuStatus) {
                updateScreen();
        }

}
//импорт
if (dataReciveBufer[0] == 'I') {
        if (dataTransferEnable == 1) {
                HAL_Delay(10);
                CDC_Transmit_FS("begin(", 6);
                for (uint16_t i = 0; i < DataCount; i++) {
                        char buf[16] = "abc";
                        readFromEeprom(0x1000 + 64 * i, buf, 16);
                        HAL_Delay(10);
                        CDC_Transmit_FS(buf, 16);
                        HAL_Delay(10);
                        readFromEeprom(0x1000 + 64 * i + 32, buf, 32);
                        HAL_Delay(10);
                        CDC_Transmit_FS(buf, 32);
                        HAL_Delay(10);
                }
                HAL_Delay(10);
                CDC_Transmit_FS(")end", 4);
                HAL_Delay(10);
                CDC_Transmit_FS("OK", 2);
        } else {
                CDC_Transmit_FS("You need Export Mode", 21);
        }
}
//сброс
if (dataReciveBufer[0] == 'C') {
        //clearDevice();
        clearDevice();
        deviceIsntInit();
        CDC_Transmit_FS("OK", 2);
}
```

```
                        //добавить безопасный ПК
                        if (dataReciveBufer[0] == 'A') {
                                addDeviceEnable = 1;
                                uploadaddDeviceEnable();
                                // downloadaddDeviceEnable(); uploadaddDeviceEnable();
                                CDC_Transmit_FS("OK", 2);
                        }
                }
                for (uint32_t i = 0; i < usbBuferSize; i++) {
                        dataReciveBufer[i] = 0;
                }
        } else {
                //      ssd1306_SetCursor(2,2);
                //      ssd1306_WriteString("0", Font_7x10, White);
                //      ssd1306_UpdateScreen();
        }

    }

    /* USER CODE END 3 */
}

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

    /** Initializes the CPU, AHB and APB busses clocks
     */
    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 = 12;
    RCC_OscInitStruct.PLL.PLLN = 192;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
```

```
	RCC_OscInitStruct.PLL.PLLQ = 8;
	if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
		Error_Handler();
	}
	/** Initializes the CPU, AHB and APB busses clocks
	 */
	RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
			| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
	RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
	RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
	RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
	RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

	if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK) {
		Error_Handler();
	}
}

/**
 * @brief CRC Initialization Function
 * @param None
 * @retval None
 */
static void MX_CRC_Init(void) {

	/* USER CODE BEGIN CRC_Init 0 */

	/* USER CODE END CRC_Init 0 */

	/* USER CODE BEGIN CRC_Init 1 */

	/* USER CODE END CRC_Init 1 */
	hcrc.Instance = CRC;
	if (HAL_CRC_Init(&hcrc) != HAL_OK) {
		Error_Handler();
	}
	/* USER CODE BEGIN CRC_Init 2 */

	/* USER CODE END CRC_Init 2 */
```

```
}

/**
 * @brief CRYP Initialization Function
 * @param None
 * @retval None
 */
static void MX_CRYP_Init(void) {

        /* USER CODE BEGIN CRYP_Init 0 */
        hcryp.Init.pKey = "test";
        /* USER CODE END CRYP_Init 0 */

        /* USER CODE BEGIN CRYP_Init 1 */

        /* USER CODE END CRYP_Init 1 */
        hcryp.Instance = CRYP;
        hcryp.Init.DataType = CRYP_DATATYPE_32B;
        hcryp.Init.pKey = (uint32_t*) pKeyCRYP;
        hcryp.Init.Algorithm = CRYP_TDES_ECB;
        hcryp.Init.DataWidthUnit = CRYP_DATAWIDTHUNIT_WORD;
        if (HAL_CRYP_Init(&hcryp) != HAL_OK) {
               Error_Handler();
        }
        /* USER CODE BEGIN CRYP_Init 2 */

        /* USER CODE END CRYP_Init 2 */

}

/**
 * @brief HASH Initialization Function
 * @param None
 * @retval None
 */
static void MX_HASH_Init(void) {

        /* USER CODE BEGIN HASH_Init 0 */
```

```
	/* USER CODE END HASH_Init 0 */

	/* USER CODE BEGIN HASH_Init 1 */

	/* USER CODE END HASH_Init 1 */
	hhash.Init.DataType = HASH_DATATYPE_8B;
	if (HAL_HASH_Init(&hhash) != HAL_OK) {
		Error_Handler();
	}
	/* USER CODE BEGIN HASH_Init 2 */

	/* USER CODE END HASH_Init 2 */

}

/**
 * @brief I2C1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_I2C1_Init(void) {

	/* USER CODE BEGIN I2C1_Init 0 */

	/* USER CODE END I2C1_Init 0 */

	/* USER CODE BEGIN I2C1_Init 1 */

	/* USER CODE END I2C1_Init 1 */
	hi2c1.Instance = I2C1;
	hi2c1.Init.ClockSpeed = 100000;
	hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
	hi2c1.Init.OwnAddress1 = 0;
	hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
	hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
	hi2c1.Init.OwnAddress2 = 0;
	hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
	hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
```

```
        if (HAL_I2C_Init(&hi2c1) != HAL_OK) {
                Error_Handler();
        }
        /* USER CODE BEGIN I2C1_Init 2 */

        /* USER CODE END I2C1_Init 2 */

}

/**
 * @brief RNG Initialization Function
 * @param None
 * @retval None
 */
static void MX_RNG_Init(void) {

        /* USER CODE BEGIN RNG_Init 0 */

        /* USER CODE END RNG_Init 0 */

        /* USER CODE BEGIN RNG_Init 1 */

        /* USER CODE END RNG_Init 1 */
        hrng.Instance = RNG;
        if (HAL_RNG_Init(&hrng) != HAL_OK) {
                Error_Handler();
        }
        /* USER CODE BEGIN RNG_Init 2 */

        /* USER CODE END RNG_Init 2 */

}

/**
 * @brief TIM6 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM6_Init(void) {
```

```
	/* USER CODE BEGIN TIM6_Init 0 */

	/* USER CODE END TIM6_Init 0 */

	TIM_MasterConfigTypeDef sMasterConfig = { 0 };

	/* USER CODE BEGIN TIM6_Init 1 */

	/* USER CODE END TIM6_Init 1 */
	htim6.Instance = TIM6;
	htim6.Init.Prescaler = 24000;
	htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
	htim6.Init.Period = 10;
	htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
	if (HAL_TIM_Base_Init(&htim6) != HAL_OK) {
		Error_Handler();
	}
	sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
	sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
	if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig)
			!= HAL_OK) {
		Error_Handler();
	}
	/* USER CODE BEGIN TIM6_Init 2 */

	/* USER CODE END TIM6_Init 2 */

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void) {
	GPIO_InitTypeDef GPIO_InitStruct = { 0 };

	/* GPIO Ports Clock Enable */
```

```
	__HAL_RCC_GPIOH_CLK_ENABLE();
	__HAL_RCC_GPIOC_CLK_ENABLE();
	__HAL_RCC_GPIOA_CLK_ENABLE();
	__HAL_RCC_GPIOB_CLK_ENABLE();

	/*Configure GPIO pins : PC8 PC9 */
	GPIO_InitStruct.Pin = GPIO_PIN_8 | GPIO_PIN_9;
	GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
	GPIO_InitStruct.Pull = GPIO_NOPULL;
	HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

	/* EXTI interrupt init*/
	HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
	HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

}

/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
	if (GPIO_Pin == GPIO_PIN_8 && !leftButtonStatus) {
		HAL_TIM_Base_Stop(&htim6);
		tim6_counter = 0;
		leftButtonStatus = 1;
		HAL_TIM_Base_Start(&htim6);
		HAL_TIM_Base_Start_IT(&htim6);
	}
	if (GPIO_Pin == GPIO_PIN_9 && !rightButtonStatus) {
		HAL_TIM_Base_Stop(&htim6);
		tim6_counter = 0;
		rightButtonStatus = 1;
		HAL_TIM_Base_Start(&htim6);
		HAL_TIM_Base_Start_IT(&htim6);
	}
	if (rightButtonStatus && leftButtonStatus) {
		bothButtonStatus = 1;
	}
}

void changePasswordData() {
```

```
//CDC_Transmit_FS(menu.blocks[menu.pointer].login, 16);
//CDC_Transmit_FS(menu.blocks[menu.pointer].url, 16);
//CDC_Transmit_FS(menu.blocks[menu.pointer].number, 16);
//char buf[37];
//char buf[64];
char hashbuf[20];
char prepass[37];
char str[64];
for (uint8_t l = 0; l < 64; l++) {
	str[l] = 0;
}
readFromEeprom(0x1000 + 64 * (menu.pointer), &str, 64);
downloadPrivate();
//CDC_Transmit_FS(str, 64);
for (uint8_t l = 0; l < 20; l++) {
	prepass[l] = privateKey[l];
}
for (uint8_t l = 0; l < 16; l++) {
	prepass[20 + l] = str[l];
}
uint8_t num = (uint8_t) str[48];
num++;
str[48] = num;
prepass[36] = num;
HAL_HASH_Init(&hhash);
HAL_HASH_SHA1_Start(&hhash, prepass, 37, hashbuf, HAL_MAX_DELAY);
HASH_Finish(&hhash, hashbuf, HAL_MAX_DELAY);
HAL_HASH_Init(&hhash);
for (uint8_t l = 0; l < 16; l++) {
	str[l + 16] = hashbuf[l + 3];
}
CDC_Transmit_FS(hashbuf, 20);
writeToEeprom(0x1000 + 64 * (menu.pointer), str, 64);

accauntBlock blocks[DataCount];
for (uint16_t i = 0; i < DataCount; i++) {
	char buf[64] = "";
	readFromEeprom(0x1000 + 64 * i, buf, 64);
```

```
                stringToStruct(&buf, &blocks[i]);
        }
        menu.blocks = blocks;

}

void leftButtonActions() {
        //ssd1306_SetCursor(10, 0);
        //ssd1306_WriteString("L", Font_7x10, White);
        //ssd1306_UpdateScreen();
        if (menuStatus) {
                menu.pointer--;
                updateScreen();
        } else if (initStatus) {
                initProcess1();
        } else if (initStatusStep1) {
                //initProcess1Next();
        } else if (initStatusStep2) {
                initStatus = 1;
                initStatusStep2 = 0;
                initProcess1();
        } else if (setPasswordStep1) {
                setPasswordProcess1Down();
        } else if (setPasswordStep2) {
                setPasswordProcess1Down();
        } else if (restoreStatusStep1) {
                initStatus = 1;
                restoreStatusStep1 = 0;
                initChoseProcess();
        } else if (setProtectTypeStep1) {
                setProtectTypeProcess1Down();
        } else if (settingsMenuStatus) {
                settingsMenuDown();
        } else if (dataControlMenuStatus) {
                dataControlMenuDown();
        } else if (ResetComand) {
                settingsMenuStatus = 1;
                ResetComand = 0;
                settingsMenu();
```

```
        } else if (dataTransferEnable) {
                dataTransferEnable = 0;
                settingsMenuStatus = 1;

                settingsMenu();
        } else if (DataInfoMenu) {
                DataInfoMenu = 0;
                dataControlMenuStatus = 1;
                dataControlMenu();
        }
}

void rightButtonActions() {
//      ssd1306_SetCursor(24, 0);
//      ssd1306_WriteString("R", Font_7x10, White);
//      ssd1306_UpdateScreen();
        if (menuStatus) {
                menu.pointer++;
                updateScreen();
        } else if (initStatus) {
                restoreProcess1();
        } else if (initStatusStep1) {
                //initProcess1Next();
        } else if (initStatusStep2) {
                initProcess2Next();
        } else if (setPasswordStep1) {
                setPasswordProcess1Up();
        } else if (setPasswordStep2) {
                setPasswordProcess1Up();
        } else if (restoreStatusStep1) {
                initStatus = 1;
                restoreStatusStep1 = 0;
                initChoseProcess();
        } else if (setProtectTypeStep1) {
                setProtectTypeProcess1Up();
        } else if (settingsMenuStatus) {
                settingsMenuUp();
        } else if (dataControlMenuStatus) {
                dataControlMenuUp();
```

```
        } else if (ResetComand) {
                settingsMenuStatus = 0;
                ResetComand = 0;
                clearDevice();
                deviceIsntInit();
        } else if (dataTransferEnable) {
                dataTransferEnable = 0;
                settingsMenuStatus = 1;

                settingsMenu();
        } else if (DataInfoMenu) {
                DataInfoMenu = 0;
                dataControlMenuStatus = 1;
                dataControlMenu();
        }
        //endDataPointer++;
}

void bothButtonActions() {
        //ssd1306_SetCursor(17, 0);
        //ssd1306_WriteString("B", Font_7x10, White);
        //ssd1306_UpdateScreen();
        //initProcess1();
        if (menuStatus) {
                chooseMainMenu();
        } else if (initStatus) {
                //initProcess1();
        } else if (initStatusStep1) {
                initProcess1Next();
        } else if (setPasswordStep1) {
                setPasswordProcess1Next();
        } else if (setPasswordStep2) {
                setPasswordProcess2Next();
        } else if (restoreStatusStep1) {
                initStatus = 1;
                restoreStatusStep1 = 0;
                initChoseProcess();
        } else if (setProtectTypeStep1) {
                setProtectTypeProcess1Next();
```

```
        } else if (settingsMenuStatus) {
                settingsMenuSelect();
        } else if (dataControlMenuStatus) {
                dataControlMenuSelect();
        } else if (ResetComand) {

        } else if (dataTransferEnable) {
                dataTransferEnable = 0;
                settingsMenuStatus = 1;

                settingsMenu();
        } else if (DataInfoMenu) {
                DataInfoMenu = 0;
                dataControlMenuStatus = 1;
                dataControlMenu();
        }
}

void generatePassFrase() {
        uint16_t num;
        uint16_t bfstr[60];
        generateRandomNumbers(500, 0xfff);
        for (int i = 0; i < 12; i++) {
                passFrase[i] = wordsForPassFrase[RNGNumbers[i]];
        }
        for (int i = 0; i < 12; i++) {
                for (int j = 0; j < 5; j++) {
                        bfstr[i * 5 + j] = passFrase[i][j];
                }
        }
        HAL_HASH_Init(&hhash);
        HAL_HASH_SHA1_Start(&hhash, &bfstr, 60, &privateKey, HAL_MAX_DELAY);
        HASH_Finish(&hhash, &privateKey, HAL_MAX_DELAY);
        HAL_HASH_DeInit(&hhash);
        HAL_HASH_Init(&hhash);
        HAL_HASH_SHA1_Start(&hhash, &privateKey, 60, &publicKey, HAL_MAX_DELAY);
        HASH_Finish(&hhash, &publicKey, HAL_MAX_DELAY);
        HAL_HASH_DeInit(&hhash);
```

```
	uploadPrivate();
	uploadadPublic();
}

void generateRandomNumbers(uint16_t blocknumber, uint16_t filter) {
	uint16_t j = 0;
	for (uint16_t i = 0; i < 12; i++) {
		RNGNumbers[i] = HAL_RNG_GetRandomNumber(&hrng) & filter;
		//isOk=0;
		while ((RNGNumbers[i] >= blocknumber)) {
			RNGNumbers[i] = HAL_RNG_GetRandomNumber(&hrng) & filter;
		}

	}
	for (uint16_t i = 0; i < 12; i++) {
		j = 0;
		while (j < i) {
			if (RNGNumbers[i] == RNGNumbers[j]
					|| (RNGNumbers[i] >= blocknumber)) {
				RNGNumbers[i] = HAL_RNG_GetRandomNumber(&hrng) & filter;
				j = 0;
			} else
				j++;
		}
	}
}

/* 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 */

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

2.3. Файл data.h

```
#include "main.h"
#define usbBlockSize 64
void writeToEeprom(uint16_t memoryAddres, uint8_t *data, uint16_t dataLength);
void readFromEeprom(uint16_t memoryAddres, uint8_t *data, uint16_t dataLength);
```

2.4. Файл data.c

```
#include "data.h"

devAddr = (0x50 << 1);
startaddressfordata = 0x1000;
```

```
void deviceIsntInit() {
	if (M5PCIDdefaultIsGetted) {
		if (isInit == 0) {
			initStatus = 1;
			initChoseProcess();
		} else if (ProtectType == 2) {
			ssd1306_Fill(Black);
			ssd1306_SetCursor(2, 2);
			ssd1306_WriteString("Conect device to", Font_7x10, White);
			ssd1306_SetCursor(2, 12);
			ssd1306_WriteString("your safe PC", Font_7x10, White);
			ssd1306_UpdateScreen();
		} else if (ProtectType == 3) {
			ssd1306_Fill(Black);
			ssd1306_SetCursor(2, 2);
			ssd1306_WriteString("Conect device to", Font_7x10, White);
			ssd1306_SetCursor(2, 12);
			ssd1306_WriteString("your safe PC", Font_7x10, White);
			ssd1306_UpdateScreen();
		}

	} else {
		ssd1306_Fill(Black);
		ssd1306_SetCursor(2, 2);
		ssd1306_WriteString("Start init from", Font_7x10, White);
		ssd1306_SetCursor(2, 12);
		ssd1306_WriteString("your safe PC", Font_7x10, White);
		ssd1306_UpdateScreen();
	}

}

void uploadIsInit() {
	writeToEeprom(0x0000, &isInit, 1);
}

void downloadIsInit() {
	readFromEeprom(0x0000, &isInit, 1);
```

```
//      ssd1306_SetCursor(2, 2);
//      ssd1306_WriteStringUint(isInit, Font_7x10, White);
//      ssd1306_UpdateScreen();
}

void uploadSecureOpt() {
        writeToEeprom(0x0000 + 1, &ProtectType, 1);
}

void downloadSecureOpt() {
        readFromEeprom(0x0000 + 1, &ProtectType, 1);
//      ssd1306_SetCursor(12, 2);
//      ssd1306_WriteStringUint(ProtectType, Font_7x10, White);
//      ssd1306_UpdateScreen();
}

void uploadPassword() {
        writeToEeprom(0x0000 + 2, &password, 6);
}

void downloadPassword() {
        readFromEeprom(0x0000 + 2, &password, 6);
//      ssd1306_SetCursor(12, 50);
//      ssd1306_WriteStringUint(password[0], Font_7x10, White);
        //ssd1306_WriteStringUint(password[1], Font_7x10, White);
        //ssd1306_WriteStringUint(password[2], Font_7x10, White);
        //ssd1306_WriteStringUint(password[3], Font_7x10, White);
        //ssd1306_WriteStringUint(password[4], Font_7x10, White);
        //ssd1306_UpdateScreen();
}

void uploadPCIDcount() {
        writeToEeprom(0x0200, &M5PPCIDCount, 2);
}

void downloadPCIDcount() {
        readFromEeprom(0x0200, &M5PPCIDCount, 2);
        //ssd1306_SetCursor(22, 2);
//      ssd1306_WriteStringUint(M5PPCIDCount, Font_7x10, White);
```

```
        //ssd1306_UpdateScreen();
}

void uploadDataCount() {
        writeToEeprom(0x0000 + 9, &DataCount, 1);
}

void downloadDataCount() {
        readFromEeprom(0x0000 + 9, &DataCount, 1);
        //ssd1306_SetCursor(32, 2);
        //ssd1306_WriteStringUint(DataCount, Font_7x10, White);
        //ssd1306_UpdateScreen();
}
void uploadIsfirstPC() {
        writeToEeprom(0x0000 + 10, &M5PCIDdefaultIsGetted, 1);
}

void downloadIsfirstPCt() {
        readFromEeprom(0x0000 + 10, &M5PCIDdefaultIsGetted, 1);
//      ssd1306_SetCursor(2, 2);
//      ssd1306_WriteStringUint(isInit, Font_7x10, White);
//      ssd1306_UpdateScreen();
}

//addDeviceEnable

void uploadaddDeviceEnable() {
        writeToEeprom(0x0000 + 11, &addDeviceEnable, 1);
}

void downloadaddDeviceEnable() {
        readFromEeprom(0x0000 + 11, &addDeviceEnable, 1);
}

void uploadPrivate() {
        writeToEeprom(0x0000 + 12, &privateKey, 20);
}

void downloadPrivate() {
```

```
        readFromEeprom(0x0000 + 12, &privateKey, 20);
}

void uploadadPublic() {
        writeToEeprom(0x0000 + 32, &publicKey, 20);
}

void downloadPublic() {
        readFromEeprom(0x0000 + 32, &publicKey, 20);
}

void uploadPCIDmas() {

        writeToEeprom(0x0300 + 100, &PCIDFour, 24);
        writeToEeprom(0x0300 + 150, &PCIDFive, 24);
        writeToEeprom(0x0300 + 200, &PCIDOne, 24);

        writeToEeprom(0x0300 + 400, &PCIDSeven, 24);
        writeToEeprom(0x0300 + 450, &PCIDSix, 24);

        //writeToEeprom(0x0000,&isInit ,1);
}

void downloadPCIDmas() {

//      for(uint8_t i=0;i<M5PPCIDCount;i++)
//      {
        //readFromEeprom(0x0300,&PCIDOne ,24);
        //readFromEeprom(0x0300+24,&PCIDSeven ,24);
        //readFromEeprom(0x0300+48,&PCIDSix ,24);
        //readFromEeprom(0x0300+72,&PCIDFour ,24);
        //readFromEeprom(0x0300+96,&PCIDFive ,24);
//
        //readFromEeprom(0x0300+260,&PCIDSeven ,24);
        //readFromEeprom(0x0300+310,&PCIDSix ,24);
        readFromEeprom(0x0300 + 100, &PCIDFour, 24);
        readFromEeprom(0x0300 + 150, &PCIDFive, 24);
        readFromEeprom(0x0300 + 200, &PCIDOne, 24);
```

```
        readFromEeprom(0x0300 + 400, &PCIDSeven, 24);
        readFromEeprom(0x0300 + 450, &PCIDSix, 24);
        //}

}

uint8_t lt[] = {

0B00000000, 0B00000000, 0B00000001, 0B00000011, 0B00000111, 0B00001111,
               0B00011111, 0B00111111, 0B00111111, 0B00011111, 0B00001111, 0B00000111,
               0B00000011, 0B00000001, 0B00000000, 0B00000000 };

uint8_t rt[] = {

0B00000000, 0B00000000, 0B10000000, 0B11000000, 0B11100000, 0B11110000,
               0B11111000, 0B11111100, 0B11111100, 0B11111000, 0B11110000, 0B11100000,
               0B11000000, 0B10000000, 0B00000000, 0B00000000 };

uint8_t cancel[] = {

0B10000001, 0B01000010, 0B00100100, 0B00011000, 0B00011000, 0B00100100,
               0B01000010, 0B10000001 };

uint8_t ok[] = {

0B00000000, 0B00000001, 0B00000010, 0B00000100, 0B10001000, 0B01010000,
               0B00100000, 0B00000000 };

uint8_t line[] = {

0B00000000, 0B11111111, 0B00000000, 0B00000000, 0B00000000, 0B00000000,
               0B00000000, 0B00000000 };

uint8_t linebold[] = {

0B00000000, 0B11111111, 0B11111111, 0B00000000, 0B00000000, 0B00000000,
               0B00000000, 0B00000000 };

uint8_t upwd[] = {
```

```
0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000,
                0B00000001, 0B10000000, 0B00000011, 0B11000000, 0B00000111, 0B11100000,
                0B00001111, 0B11110000, 0B00011111, 0B11111000

};

uint8_t downwd[] = { 0B00011111, 0B11111000, 0B00001111, 0B11110000, 0B00000111,
                0B11100000, 0B00000011, 0B11000000, 0B00000001, 0B10000000, 0B00000000,
                0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000

};

uint8_t newDev[] = {

0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000,
                0B00000000, 0B00000000, 0B00001111, 0B10000000, 0B00000001, 0B11110000,
                0B01111111, 0B11111111, 0B11111111, 0B11111110, 0B01000000, 0B00000000,
                0B00000000, 0B00000010, 0B01000000, 0B00000000, 0B00000000, 0B00000010,
                0B01000000, 0B00000001, 0B10000000, 0B00000010, 0B01000000, 0B00000001,
                0B10000000, 0B00000010, 0B01000000, 0B00000111, 0B11100000, 0B00000010,
                0B01000000, 0B00000111, 0B11100000, 0B00000010, 0B01000000, 0B00000001,
                0B10000000, 0B00000010, 0B01000000, 0B00000001, 0B10000000, 0B00000010,
                0B01000000, 0B00000000, 0B00000000, 0B00000010, 0B01000000, 0B00000000,
                0B00000000, 0B00000010, 0B01111111, 0B11111111, 0B11111111, 0B11111110,
                0B00000000, 0B00000000, 0B00000000, 0B00000000 };

uint8_t restoreDev[] = {

0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000, 0B00000000,
                0B00000000, 0B00000000, 0B00001111, 0B10000000, 0B00000001, 0B11110000,
                0B01111111, 0B11111111, 0B11111111, 0B11111110, 0B01000000, 0B00000001,
                0B00000000, 0B00000010, 0B01000000, 0B00000011, 0B00000000, 0B00000010,
                0B01000000, 0B00000111, 0B10000000, 0B00000010, 0B01000000, 0B00000011,
                0B01000000, 0B00000010, 0B01000000, 0B00000001, 0B00100000, 0B00000010,
                0B01000000, 0B00001000, 0B00100000, 0B00000010, 0B01000000, 0B00000100,
                0B01000000, 0B00000010, 0B01000000, 0B00000011, 0B10000000, 0B00000010,
                0B01000000, 0B00000000, 0B00000000, 0B00000010, 0B01000000, 0B00000000,
                0B00000000, 0B00000010, 0B01111111, 0B11111111, 0B11111111, 0B11111110,
```

```
                0B00000000, 0B00000000, 0B00000000, 0B00000000 };

uint8_t gear[] = { 0B00000000, 0B00000001, 0B10000000, 0B00000000, 0B00000000,
                0B00000011, 0B11000000, 0B00000000, 0B00000000, 0B00000111, 0B11100000,
                0B00000000, 0B00011110, 0B00001111, 0B11110000, 0B01111000, 0B00011111,
                0B10011110, 0B01111001, 0B11111000, 0B00011111, 0B11111100, 0B00111111,
                0B11111000, 0B00011111, 0B11100000, 0B00000111, 0B11111000, 0B00001111,
                0B00000000, 0B00000000, 0B11111000, 0B00001110, 0B00000111, 0B11100000,
                0B01110000, 0B00000110, 0B00001110, 0B01110000, 0B01100000, 0B00000110,
                0B00011000, 0B00011000, 0B01100000, 0B00001100, 0B00110000, 0B00001100,
                0B00110000, 0B00011100, 0B01100001, 0B10000110, 0B00111000, 0B00111100,
                0B11000011, 0B11000011, 0B00111100, 0B01111001, 0B10000111, 0B11100001,
                0B10011110, 0B11110001, 0B10001110, 0B01110001, 0B10011111, 0B11110001,
                0B10001110, 0B01110001, 0B10011111, 0B01111001, 0B10000111, 0B11100001,
                0B10011110, 0B00111100, 0B11000011, 0B11000011, 0B00111100, 0B00011100,
                0B01100001, 0B10000110, 0B00111000, 0B00001100, 0B00110000, 0B00001100,
                0B00110000, 0B00000110, 0B00011000, 0B00011000, 0B01100000, 0B00000110,
                0B00001110, 0B01110000, 0B01100000, 0B00001110, 0B00000111, 0B11100000,
                0B01110000, 0B00001111, 0B00000000, 0B00000000, 0B11111000, 0B00011111,
                0B11100000, 0B00000111, 0B11111000, 0B00011111, 0B11111100, 0B00111111,
                0B11111000, 0B00011111, 0B10011110, 0B01111001, 0B11111000, 0B00011110,
                0B00001111, 0B11110000, 0B01111000, 0B00000000, 0B00000111, 0B11100000,
                0B00000000, 0B00000000, 0B00000011, 0B11000000, 0B00000000, 0B00000000,
                0B00000001, 0B10000000, 0B00000000

};
/*
 oled.OLED_Write_To_Bufer(x,y,w,h,zero);
 else if (data==0x31)
 oled.OLED_Write_To_Bufer(x,y,w,h,one);
 else if (data==0x32)
 oled.OLED_Write_To_Bufer(x,y,w,h,two);
 else if (data==0x33)
 oled.OLED_Write_To_Bufer(x,y,w,h,three);
 else if (data==0x34)
 oled.OLED_Write_To_Bufer(x,y,w,h,four);
 else if (data==0x35)
 oled.OLED_Write_To_Bufer(x,y,w,h,five);
 else if (data==0x36)
```

```
 oled.OLED_Write_To_Bufer(x,y,w,h,six);
 else if (data==0x37)
 oled.OLED_Write_To_Bufer(x,y,w,h,seven);
 else if (data==0x38)
 oled.OLED_Write_To_Bufer(x,y,w,h,eight);
 else if (data==0x39)
 oled.OLED_Write_To_Bufer(x,y,w,h,nine);

 */

void initConstants() {
        dataTransferEnable = 0;
        isInit = 0;
        ProtectType = 0;
        settingsMenuStatus = 0;
        dataControlMenuStatus = 0;
        DataCount = 0;
        initStatus = 0;
        initStatusStep1 = 0;
        initStatusStep2 = 0;
        restoreStatusStep1 = 0;
        restoreStatusStep2 = 0;
        menuStatus = 0;
        restoreStatus = 0;
        setPasswordStatus = 0;
        settingsStatus = 0;
        datasettingsStatus = 0;
        setPasswordStep1 = 0;
        setPasswordStep2 = 0;

        setProtectTypeStep1 = 0;
        setProtectTypeStep2 = 0;
        passwordInputStatus = 0;

        leftButtonStatus = 0;
        rightButtonStatus = 0;
        bothButtonStatus = 0;

        M5PCIDdefaultIsGetted = 0;
```

```
        downloadIsInit();
        downloadSecureOpt();
        downloadPassword();
        downloadPCIDcount();
        downloadDataCount();
        //DataCount = DataCount;
        downloadaddDeviceEnable();
        downloadPCIDcount();
        downloadIsfirstPCt();
        downloadPublic();
        downloadPrivate();
        downloadPCIDmas();
        //generatePassFrase();

}

void initChoseProcess() {
        generatePassFrase();
        ssd1306_Fill(Black);
        ssd1306_Write_To_Bufer(16, 24, 32, 16, newDev);
        ssd1306_Write_To_Bufer(80, 24, 32, 16, restoreDev);
        ssd1306_UpdateScreen();
}

void initProcess1() {
        initStatus = 0;
        initStatusStep1 = 1;
        pointer = -1;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(2, 5);
        ssd1306_WriteString("Write all 12 words in a safe place", Font_7x10, White);
        ssd1306_SetCursor(2, 15);
        ssd1306_WriteString("into a safe place.", Font_7x10, White);
        ssd1306_SetCursor(2, 25);
        ssd1306_WriteString("Press two buttons if you spelled the word", Font_7x10,
                        White);
        ssd1306_SetCursor(2, 35);
        ssd1306_WriteString("if you wrote word", Font_7x10, White);
```

```
        ssd1306_SetCursor(2, 45);
        ssd1306_WriteString("or to continue.", Font_7x10, White);
        ssd1306_UpdateScreen();
        //
        //char* passFrase[12];

}

void initProcess1Next() {
        //Font_16x26  Font_11x18
        pointer++;
        if (pointer < 12) {

                ssd1306_Fill(Black);
                ssd1306_SetCursor(40, 10);
                char *text = "Word ";
                //strcat(text, (char)(pointer+1));
                ssd1306_WriteString(text, Font_7x10, White);
                ssd1306_WriteStringUint((pointer + 1), Font_7x10, White);
                ssd1306_SetCursor(25, 25);
                ssd1306_WriteString(passFrase[pointer], Font_16x26, White);
                ssd1306_UpdateScreen();
                //char* passFrase[12];
        } else {
                //fCheckStatus=1;
                //initStatus=0;
                initStatusStep1 = 0;
                initStatusStep2 = 1;
                initProcess2();
        }

}

void initProcess2() {

        pointer = -1;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(15, 10);
        ssd1306_WriteString("Check 4 random", Font_7x10, White);
```

```
        ssd1306_SetCursor(45, 20);
        ssd1306_WriteString("words..", Font_7x10, White);
        ssd1306_Write_To_Bufer(2, 54, 8, 8, cancel);
        ssd1306_Write_To_Bufer(120, 54, 8, 8, ok);
        ssd1306_UpdateScreen();
        generateRandomNumbers(12, 0xf);

}

void initProcess2Next() {

        pointer++;
        if (pointer < 4) {
                //uint16_t num;
                //num=HAL_RNG_GetRandomNumber(&hrng)&0x0000000F;
                ssd1306_Fill(Black);
                ssd1306_SetCursor(40, 10);
                char *text = "Word ";
                //strcat(text, (char)(pointer+1));
                ssd1306_WriteString(text, Font_7x10, White);
                ssd1306_WriteStringUint(RNGNumbers[pointer] + 1, Font_7x10, White);
                ssd1306_SetCursor(25, 25);
                ssd1306_WriteString(passFrase[RNGNumbers[pointer]], Font_16x26, White);
                ssd1306_Write_To_Bufer(2, 54, 8, 8, cancel);
                ssd1306_Write_To_Bufer(120, 54, 8, 8, ok);
                ssd1306_UpdateScreen();
        } else {
                //fCheckStatus=1;
                //initStatus=0;
                initStatusStep2 = 0;
                setPasswordProcess1();
        }

}

void generateExtraData() {
        isInit = 1;
        uploadIsInit();
        uploadSecureOpt();
```

```
        uploadPassword();
        uploadPCIDcount();
        DataCount = 0;
        //DataCount = DataCount;
        uploadDataCount();

        //uploadPCIDmas();

        downloadIsInit();
        downloadSecureOpt();
        downloadPassword();
        downloadPCIDcount();
        downloadDataCount();

        //downloadPCIDmas();
        //HAL_Delay(10000);
}

void clearDevice() {
        isInit = 0;
        ProtectType = 0;
        settingsMenuStatus = 0;
        dataControlMenuStatus = 0;
        DataCount = 0;
        initStatus = 0;
        initStatusStep1 = 0;
        initStatusStep2 = 0;
        restoreStatusStep1 = 0;
        restoreStatusStep2 = 0;
        menuStatus = 0;
        restoreStatus = 0;
        setPasswordStatus = 0;
        settingsStatus = 0;
        datasettingsStatus = 0;
        setPasswordStep1 = 0;
        setPasswordStep2 = 0;
        setProtectTypeStep1 = 0;
        setProtectTypeStep2 = 0;
        passwordInputStatus = 0;
```

```
	leftButtonStatus = 0;
	rightButtonStatus = 0;
	bothButtonStatus = 0;
	M5PCIDdefaultIsGetted = 0;
	uploadIsInit();
	uploadSecureOpt();
	uploadPassword();
	uploadPCIDcount();
	uploadDataCount();
	//uploadPCIDmas();

	uint8_t nul = 0;
	for (int i = 0; i < 1000; i++) {
		writeToEeprom(i, &nul, 1);
	}
	for (int i = 0; i < 1000; i++) {
		writeToEeprom(0x0300 + i, &nul, 1);
	}
	for (int i = 0; i < 1000; i++) {
		writeToEeprom(0x1000 + i, &nul, 1);
	}

}

void restoreProcess1() {
	initStatus = 0;
	restoreStatusStep1 = 1;
	pointer = -1;

	ssd1306_Fill(Black);
	ssd1306_SetCursor(10, 10);
	ssd1306_WriteString("Restore mode,", Font_7x10, White);
	ssd1306_SetCursor(10, 20);
	ssd1306_WriteString("init your device", Font_7x10, White);
	ssd1306_SetCursor(10, 30);
	ssd1306_WriteString("by using PC app", Font_7x10, White);
	ssd1306_SetCursor(10, 50);
	ssd1306_WriteString("Press any key", Font_7x10, White);
	ssd1306_UpdateScreen();
```

```
}

void setPasswordProcess1() {
	pointer = 0;
	updownpointer = 0;
	setPasswordStep1 = 1;
	ssd1306_Fill(Black);
	ssd1306_SetCursor(7, 10);
	ssd1306_WriteString("Create password", Font_7x10, White);
	ssd1306_SetCursor(27, 20);
	ssd1306_WriteString("for device", Font_7x10, White);
	ssd1306_Write_To_Bufer(28, 50, 8, 8, line);
	ssd1306_Write_To_Bufer(44, 50, 8, 8, line);
	ssd1306_Write_To_Bufer(60, 50, 8, 8, line);
	ssd1306_Write_To_Bufer(76, 50, 8, 8, line);
	ssd1306_Write_To_Bufer(92, 50, 8, 8, line);
	ssd1306_SetCursor(28 + pointer * 16, 40);
	ssd1306_WriteStringUint(updownpointer, Font_7x10, White);
	ssd1306_Write_To_Bufer(28, 50, 8, 8, linebold);
	ssd1306_UpdateScreen();

}

void setPasswordProcess1Next() {
	password[pointer] = updownpointer;
	if (pointer < 4) {
		ssd1306_Write_To_Bufer(28, 50, 8, 8, line);
		ssd1306_Write_To_Bufer(44, 50, 8, 8, line);
		ssd1306_Write_To_Bufer(60, 50, 8, 8, line);
		ssd1306_Write_To_Bufer(76, 50, 8, 8, line);
		ssd1306_Write_To_Bufer(92, 50, 8, 8, line);
		pointer++;
		ssd1306_SetCursor(28 + pointer * 16, 40);
		ssd1306_WriteStringUint(updownpointer, Font_7x10, White);
		ssd1306_Write_To_Bufer(28 + pointer * 16, 50, 8, 8, linebold);
		ssd1306_UpdateScreen();
		//uint16_t num;
		//num=HAL_RNG_GetRandomNumber(&hrng)&0x0000000F;
```

```
                //ssd1306_Fill(Black);
                //setPasswordProcess1();
                //ssd1306_SetCursor(28+pointer*16,40);
                //char* text="Word ";
                //strcat(text, (char)(pointer+1));
                //ssd1306_WriteString(text, Font_7x10, White);

                //ssd1306_UpdateScreen();
                //char* passFrase[12];
        } else {
                //fCheckStatus=1;
                //initStatus=0;
                //generateandcheckPassword();
                setPasswordStep2 = 1;
                setPasswordStep1 = 0;
                setPasswordProcess2();
        }

}

void setPasswordProcess1Up() {
        updownpointer++;
        if (updownpointer > 9) {
                updownpointer = 0;
        }

        ssd1306_SetCursor(28 + (pointer) * 16, 40);
        ssd1306_WriteStringUint(updownpointer, Font_7x10, White);
        ssd1306_UpdateScreen();
}

void setPasswordProcess1Down() {
        updownpointer--;
        if (updownpointer < 0) {
                updownpointer = 9;
        }
        ssd1306_SetCursor(28 + (pointer) * 16, 40);
        ssd1306_WriteStringUint(updownpointer, Font_7x10, White);
        ssd1306_UpdateScreen();
```

```
}

void setPasswordProcess2() {
        pointer = 0;
        updownpointer = 0;
        setPasswordStep2 = 1;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(20, 10);
        ssd1306_WriteString("Write password", Font_7x10, White);
        //ssd1306_SetCursor(50, 20);
        //ssd1306_WriteString("again", Font_7x10, White);
        ssd1306_Write_To_Bufer(28, 50, 8, 8, line);
        ssd1306_Write_To_Bufer(44, 50, 8, 8, line);
        ssd1306_Write_To_Bufer(60, 50, 8, 8, line);
        ssd1306_Write_To_Bufer(76, 50, 8, 8, line);
        ssd1306_Write_To_Bufer(92, 50, 8, 8, line);
        ssd1306_SetCursor(28 + pointer * 16, 40);
        ssd1306_WriteStringUint(updownpointer, Font_7x10, White);
        ssd1306_Write_To_Bufer(28 + pointer * 16, 50, 8, 8, linebold);
        ssd1306_UpdateScreen();

}

void setPasswordProcess2Next() {
        imputpassword[pointer] = updownpointer;
        if (pointer < 4) {

                ssd1306_Write_To_Bufer(28, 50, 8, 8, line);
                ssd1306_Write_To_Bufer(44, 50, 8, 8, line);
                ssd1306_Write_To_Bufer(60, 50, 8, 8, line);
                ssd1306_Write_To_Bufer(76, 50, 8, 8, line);
                ssd1306_Write_To_Bufer(92, 50, 8, 8, line);

                pointer++;
                ssd1306_SetCursor(28 + pointer * 16, 40);
                ssd1306_WriteStringUint(updownpointer, Font_7x10, White);
                ssd1306_Write_To_Bufer(28 + pointer * 16, 50, 8, 8, linebold);

                ssd1306_UpdateScreen();
```

```
        //uint16_t num;
        //num=HAL_RNG_GetRandomNumber(&hrng)&0x0000000F;
        //ssd1306_Fill(Black);
        //setPasswordProcess1();
        //ssd1306_SetCursor(28+pointer*16,40);
        //char* text="Word ";
        //strcat(text, (char)(pointer+1));
        //ssd1306_WriteString(text, Font_7x10, White);

        //ssd1306_UpdateScreen();
        //char* passFrase[12];
} else {
        /*
         ssd1306_SetCursor(2,50);
         ssd1306_WriteStringUint(password[0], Font_7x10, White);
         ssd1306_WriteStringUint(password[1], Font_7x10, White);
         ssd1306_WriteStringUint(password[2], Font_7x10, White);
         ssd1306_WriteStringUint(password[3], Font_7x10, White);
         ssd1306_WriteStringUint(password[4], Font_7x10, White);
         ssd1306_WriteStringUint(imputpassword[0], Font_7x10, White);
         ssd1306_WriteStringUint(imputpassword[1], Font_7x10, White);
         ssd1306_WriteStringUint(imputpassword[2], Font_7x10, White);
         ssd1306_WriteStringUint(imputpassword[3], Font_7x10, White);
         ssd1306_WriteStringUint(imputpassword[4], Font_7x10, White);
         ssd1306_UpdateScreen();*/
        if (passwordInputStatus == 0) {
                if (imputpassword[0] == password[0]
                                && imputpassword[1] == password[1]
                                && imputpassword[2] == password[2]
                                && imputpassword[3] == password[3]
                                && imputpassword[4] == password[4]) {

                        setPasswordStep2 = 0;
                        setProtectTypeProcess1();

                } else {
                        setPasswordStep1 = 1;
                        setPasswordStep2 = 0;
                        password[0] = 0;
```

```
                password[1] = 0;
                password[2] = 0;
                password[3] = 0;
                password[4] = 0;
                imputpassword[0] = 0;
                imputpassword[1] = 0;
                imputpassword[2] = 0;
                imputpassword[3] = 0;
                imputpassword[4] = 0;
                setPasswordProcess1();
        }
} else if (chpassComand == 1) {

        if (imputpassword[0] == password[0]
                        && imputpassword[1] == password[1]
                        && imputpassword[2] == password[2]
                        && imputpassword[3] == password[3]
                        && imputpassword[4] == password[4]) {
                uploadPassword();
                setPasswordStep2 = 0;
                settingsMenuStatus = 1;
                chpassComand = 0;
                settingsMenu();
        } else {
                setPasswordStep2 = 0;
                setPasswordProcess1();
        }
} else {
        if (imputpassword[0] == password[0]
                        && imputpassword[1] == password[1]
                        && imputpassword[2] == password[2]
                        && imputpassword[3] == password[3]
                        && imputpassword[4] == password[4]) {
                setPasswordStep2 = 0;
                Unlocked = 1;
                menuStatus = 1;
                initMenu();
        } else {
                setPasswordProcess2();
```

```
                        }
                }
        }
}

void setProtectTypeProcess1() {
        setProtectTypeStep1 = 1;
        pointer = ProtectType;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(2, 10);
        ssd1306_WriteString("Choose secure mode", Font_7x10, White);
        ssd1306_Write_To_Bufer(2, 50, 8, 16, lt);
        ssd1306_Write_To_Bufer(120, 50, 8, 16, rt);
        setProtectTypeProcessDefault();
}

void setProtectTypeProcess1Next() {
        if (cProtectComand == 1) {
                ProtectType = pointer;
                uploadSecureOpt();
                setProtectTypeStep1 = 0;
                settingsMenuStatus = 1;
                cProtectComand = 0;
                settingsMenu();
        } else {
                ProtectType = pointer;
                generateExtraData();
                menuStatus = 1;
                Unlocked = 1;
                setProtectTypeStep1 = 0;
                initMenu();
        }

}

void setProtectTypeProcessDefault() {
        switch (pointer) {
        case 0:
                ssd1306_SetCursor(10, 35);
```

```
                ssd1306_WriteString("      none      ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 1:
                ssd1306_SetCursor(10, 35);
                ssd1306_WriteString("    password    ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 2:
                ssd1306_SetCursor(10, 35);
                ssd1306_WriteString("      PC ID     ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 3:
                ssd1306_SetCursor(10, 35);
                ssd1306_WriteString("PC ID + password", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;

        }
}

void setProtectTypeProcess1Up() {
        pointer++;
        if (pointer > 3) {
                pointer = 0;

        }
        setProtectTypeProcessDefault();
}
void setProtectTypeProcess1Down() {
        pointer--;
        if (pointer < 0) {
                pointer = 3;
        }
        setProtectTypeProcessDefault();
}

void sendAllData() {
```

```
	CDC_Transmit_FS("begin(", 6);
	for (uint16_t i = 0; i < DataCount; i++) {
		char buf[16] = "abc";
		readFromEeprom(startaddressfordata + 64 * i, buf, 16);
		HAL_Delay(10);
		CDC_Transmit_FS(buf, 16);
		HAL_Delay(10);
		readFromEeprom(startaddressfordata + 64 * i + 32, buf, 32);
		HAL_Delay(10);
		CDC_Transmit_FS(buf, 32);
		HAL_Delay(10);
	}
	HAL_Delay(10);
	CDC_Transmit_FS(")end", 4);
}

void addDataBlock(uint8_t *data) {
	DataCount++;
	menu.pointer = DataCount - 1;
	accauntBlock buf[DataCount + 1];
	for (uint16_t i = 0; i < DataCount - 1; i++) {
		buf[i] = menu.blocks[i];
	}
	writeToEeprom(startaddressfordata + 64 * (DataCount - 1), data, 64);
	char buf2[64] = "";
	readFromEeprom(startaddressfordata + 64 * (DataCount - 1), buf2, 64);
	stringToStruct(&buf2, &buf[DataCount - 1]);
	menu.blocks = buf;
	updateScreen();
}

void initMenu() {
	menu.pointer = -1;
	accauntBlock blocks[DataCount + 1];
	for (uint16_t i = 0; i < DataCount; i++) {
		char buf[64] = "";
		readFromEeprom(startaddressfordata + 64 * i, buf, 64);
		stringToStruct(&buf, &blocks[i]);
	}
```

```
	menu.blocks = blocks;
	updateScreen();
}
//settingsMenu;menu.pointer
void updateScreen() {
	if (menu.pointer < -1) {
		menu.pointer++;
	} else if (menu.pointer > DataCount - 1) {
		menu.pointer--;
	} else if (menu.pointer != -1) {
		visualizeStruct(&menu.blocks[menu.pointer]);
	} else {
		ssd1306_Fill(Black);
		ssd1306_SetCursor(40, 20);
		ssd1306_Write_To_Bufer(48, 20, 32, 32, gear);

	}
	if (DataCount != 0) {
		if (menu.pointer > -1)
			ssd1306_Write_To_Bufer(2, 50, 8, 16, lt);
		if (menu.pointer < DataCount - 1)
			ssd1306_Write_To_Bufer(120, 50, 8, 16, rt);
	}

	ssd1306_UpdateScreen();
}

void chooseMainMenu() {
	if (menu.pointer == -1) {
		menuStatus = 0;
		settingsMenu();
	} else {
		menuStatus = 0;
		dataControlMenu(&menu.blocks[menu.pointer]);
	}
}

void settingsMenuDefault() {
	switch (pointer) {
```

```
        case 0:
                ssd1306_SetCursor(7, 30);
                ssd1306_WriteString(" Change password ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 1:
                ssd1306_SetCursor(7, 30);
                ssd1306_WriteString("   Reset device  ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 2:
                ssd1306_SetCursor(7, 30);
                ssd1306_WriteString("   Export mode   ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 3:
                ssd1306_SetCursor(7, 30);
                ssd1306_WriteString("Change Protection", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;
        case 4:
                ssd1306_SetCursor(7, 30);
                ssd1306_WriteString("      Back       ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;

        }
}

void settingsMenu() {
        settingsMenuStatus = 1;
        pointer = 0;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(35, 5);
        ssd1306_WriteString("Settings", Font_7x10, White);
        ssd1306_Write_To_Bufer(2, 50, 8, 16, lt);
        ssd1306_Write_To_Bufer(120, 50, 8, 16, rt);
        settingsMenuDefault();
}
```

```
void settingsMenuUp() {
	pointer++;
	if (pointer > 4) {
		pointer = 0;

	}
	settingsMenuDefault();

}

void settingsMenuDown() {
	pointer--;
	if (pointer < 0) {
		pointer = 4;

	}
	settingsMenuDefault();

}

void settingsMenuSelect() {
	switch (pointer) {
	case 0:
		passwordChangeMenu();
		break;
	case 1:
		resetDeviceMenu();
		break;
	case 2:
		exportModeMenu();
		break;
	case 3:
		changeProtectionMenu();
		break;
	case 4:
		settingsMenuStatus = 0;
		menuStatus = 1;
		updateScreen();
```

```
                break;

        }

}

void resetDeviceMenu() {
        settingsMenuStatus = 0;
        ResetComand = 1;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(2, 0);
        ssd1306_WriteString("Are you shure to ", Font_7x10, White);
        ssd1306_SetCursor(2, 10);
        ssd1306_WriteString("reset device?", Font_7x10, White);
        ssd1306_Write_To_Bufer(2, 54, 8, 8, cancel);
        ssd1306_Write_To_Bufer(120, 54, 8, 8, ok);
        ssd1306_UpdateScreen();

}

void changeProtectionMenu() {
        setProtectTypeStep1 = 1;
        cProtectComand = 1;
        ssd1306_Fill(Black);
        setProtectTypeProcess1();
        ssd1306_UpdateScreen();

}

void passwordChangeMenu() {
        passwordInputStatus = 1;
        settingsMenuStatus = 0;
        setPasswordStep1 = 1;
        chpassComand = 1;
        ssd1306_Fill(Black);
        setPasswordProcess1();
        ssd1306_UpdateScreen();

}
```

```
void exportModeMenu() {
	settingsMenuStatus = 0;
	dataTransferEnable = 1;
	ssd1306_Fill(Black);
	ssd1306_SetCursor(2, 5);
	ssd1306_WriteString("Export mode", Font_7x10, White);

	ssd1306_SetCursor(2, 40);
	ssd1306_WriteString("Press any key to return", Font_7x10, White);
	ssd1306_UpdateScreen();		//exportEnable
}

void dataControlMenuDefault() {
	switch (updownpointer) {
	case 0:
		ssd1306_SetCursor(10, 30);
		ssd1306_WriteString("   Login input    ", Font_7x10, White);
		ssd1306_UpdateScreen();
		break;
	case 1:
		ssd1306_SetCursor(10, 30);
		ssd1306_WriteString("  Password input  ", Font_7x10, White);
		ssd1306_UpdateScreen();
		break;
	case 2:
		ssd1306_SetCursor(10, 30);
		ssd1306_WriteString("     Delete       ", Font_7x10, White);
		ssd1306_UpdateScreen();
		break;
	case 3:
		ssd1306_SetCursor(10, 30);
		ssd1306_WriteString("      Info        ", Font_7x10, White);
		ssd1306_UpdateScreen();
		break;
	case 4:
		ssd1306_SetCursor(10, 30);
		ssd1306_WriteString(" Change password  ", Font_7x10, White);
		ssd1306_UpdateScreen();
```

```
                break;
        case 5:
                ssd1306_SetCursor(10, 30);
                ssd1306_WriteString("      Back        ", Font_7x10, White);
                ssd1306_UpdateScreen();
                break;

        }

}

void dataControlMenu() {
        /*
         ssd1306_Fill(Black);
         ssd1306_SetCursor(40,20);
         ssd1306_WriteString(inn->url, Font_7x10, White);
         ssd1306_SetCursor(40,30);
         ssd1306_WriteString(inn->login, Font_7x10, White);
         ssd1306_SetCursor(40,40);
         ssd1306_WriteString(inn->password, Font_7x10, White);
         ssd1306_UpdateScreen();
         */
        menuStatus = 0;
        dataControlMenuStatus = 1;
        updownpointer = 0;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(40, 3);
        ssd1306_WriteString(menu.blocks[menu.pointer].url, Font_7x10, White);
        ssd1306_Write_To_Bufer(2, 50, 8, 16, lt);
        ssd1306_Write_To_Bufer(120, 50, 8, 16, rt);
        dataControlMenuDefault();
}

void dataControlMenuDown() {
        updownpointer--;
        if (updownpointer < 0) {
                updownpointer = 5;

        }
```

```
	dataControlMenuDefault();
}

void dataControlMenuUp() {
	updownpointer++;
	if (updownpointer > 5) {
		updownpointer = 0;

	}
	dataControlMenuDefault();
}

void dataControlMenuSelect() {
	switch (updownpointer) {
	case 0:

		bufer[0] = 'l';
		bufer[1] = 'o';
		bufer[2] = 'g';
		bufer[3] = ':';
		memcpy(&bufer[4], menu.blocks[menu.pointer].login, 16);
		CDC_Transmit_FS(bufer, 20);
		break;
	case 1:

		bufer[0] = 'p';
		bufer[1] = 'a';
		bufer[2] = 's';
		bufer[3] = ':';
		memcpy(&bufer[4], menu.blocks[menu.pointer].password, 16);
		CDC_Transmit_FS(bufer, 20);
		break;
	case 2:
		menuStatus = 1;
		dataControlMenuStatus = 0;
		deleteData();

		break;
	case 3:
```

```
            //инфа
            //dataControlMenuStatus = 0;
            showDataInfo();
            break;
        case 4:
            //смена пароля
            changePasswordData();
            break;
        case 5:
            dataControlMenuStatus = 0;
            menuStatus = 1;
            updateScreen();
            break;
        }

}

void deleteData() {
        DataCount--;

        //menu.pointer = DataCount - 1;
        accauntBlock buf[DataCount];

        for (uint16_t i = menu.pointer + 1; i < DataCount + 1; i++) {
            char buf[64] = "abc";
            readFromEeprom(startaddressfordata + 64 * (i), buf, 64);
            writeToEeprom(startaddressfordata + 64 * (i - 1), buf, 64);
        }

        menu.pointer--;
        accauntBlock blocks[DataCount + 1];
        for (uint16_t i = 0; i < DataCount; i++) {
            char buf[64] = "";
            readFromEeprom(startaddressfordata + 64 * i, buf, 64);
            stringToStruct(&buf, &blocks[i]);
        }
        menu.blocks = blocks;
        menuStatus = 1;
        uploadDataCount();
```

```
        dataControlMenuStatus = 0;
        updateScreen();

}

void showDataInfo() {
        dataControlMenuStatus = 0;
        DataInfoMenu = 1;
        ssd1306_Fill(Black);
        ssd1306_SetCursor(30, 20);
        //ssd1306_WriteString("link: ", Font_7x10, White);
        ssd1306_WriteString((&menu.blocks[menu.pointer])->url, Font_7x10, White);
        //ssd1306_SetCursor(2, 120);
        //ssd1306_WriteString("        ", Font_7x10, White);
        ssd1306_SetCursor(30, 30);
        //ssd1306_WriteString("link: ", Font_7x10, White);
        ssd1306_WriteString((&menu.blocks[menu.pointer])->login, Font_7x10, White);
        ssd1306_SetCursor(30, 40);
        ssd1306_WriteString("Iteration: ", Font_7x10, White);
        ssd1306_WriteString(menu.blocks[menu.pointer].number, Font_7x10, White);
        ssd1306_UpdateScreen();
}

void stringToStruct(char *inn, accauntBlock *out) {
        memcpy(out->url, &inn[0], sizeof(out->url));
        memcpy(out->password, &inn[16], sizeof(out->password));
        memcpy(out->login, &inn[32], sizeof(out->login));
        memcpy(out->number, &inn[48], sizeof(out->number));
}

void structToString(accauntBlock *inn, char *out) {
        memcpy(&out[0], inn->url, sizeof(inn->url));
        memcpy(&out[16], inn->password, sizeof(inn->password));
        memcpy(&out[32], inn->login, sizeof(inn->login));
        memcpy(&out[48], inn->number, sizeof(inn->number));
}

void visualizeStruct(accauntBlock *inn) {
        ssd1306_Fill(Black);
```

```
        ssd1306_SetCursor(40, 30);
        ssd1306_WriteString(inn->url, Font_7x10, White);
        //ssd1306_SetCursor(40, 30);
        //ssd1306_WriteString(inn->login, Font_7x10, White);
        ssd1306_UpdateScreen();
        //HAL_Delay(100);
}

void writeToEeprom(uint16_t memoryAddres, uint8_t *data, uint16_t dataLength) {
        HAL_StatusTypeDef status;
        HAL_I2C_Mem_Write(&hi2c1, devAddr, memoryAddres, I2C_MEMADD_SIZE_16BIT,
                        (uint8_t*) data, dataLength, HAL_MAX_DELAY);
        status = HAL_I2C_IsDeviceReady(&hi2c1, devAddr, 1, HAL_MAX_DELAY);
        while (status != HAL_OK) {
                status = HAL_I2C_IsDeviceReady(&hi2c1, devAddr, 1, HAL_MAX_DELAY);
        }
        //ssd1306_SetCursor(0,0);
        //ssd1306_WriteString(data, Font_7x10, White);
        // HAL_Delay(100);
}

void readFromEeprom(uint16_t memoryAddres, uint8_t *data, uint16_t dataLength) {
        HAL_StatusTypeDef status;
        HAL_I2C_Mem_Read(&hi2c1, devAddr, memoryAddres, I2C_MEMADD_SIZE_16BIT,
                        (uint8_t*) data, dataLength, HAL_MAX_DELAY);
        status = HAL_I2C_IsDeviceReady(&hi2c1, devAddr, 1, HAL_MAX_DELAY);
        while (status != HAL_OK) {
                status = HAL_I2C_IsDeviceReady(&hi2c1, devAddr, 1, HAL_MAX_DELAY);
        }
        //ssd1306_SetCursor(0,20);
        //ssd1306_WriteString(data, Font_7x10, White);
        // HAL_Delay(100);

}
```

2.5. Файл ssd1306.h

```
#include "stm32f2xx_hal.h"
#include "fonts.h"



#ifndef ssd1306
#define ssd1306


#define SSD1306_I2C_PORT             hi2c1

#define SSD1306_I2C_ADDR        (0x3C << 1)

#define SSD1306_WIDTH           130

#define SSD1306_HEIGHT          64

typedef enum {
        Black = 0x00,
        White = 0x01
} SSD1306_COLOR;


typedef struct {
        uint16_t CurrentX;
        uint16_t CurrentY;
        uint8_t Inverted;
        uint8_t Initialized;
} SSD1306_t;


extern I2C_HandleTypeDef SSD1306_I2C_PORT;


uint8_t ssd1306_Init(void);
void ssd1306_Fill(SSD1306_COLOR color);
void ssd1306_UpdateScreen(void);
void ssd1306_DrawPixel(uint8_t x, uint8_t y, SSD1306_COLOR color);
char ssd1306_WriteChar(char ch, FontDef Font, SSD1306_COLOR color);
char ssd1306_WriteString(char *str, FontDef Font, SSD1306_COLOR color);
void ssd1306_SetCursor(uint8_t x, uint8_t y);

#endif
```

2.6. Файл ssd1306.c

```
#include"ssd1306.h"


static uint8_t SSD1306_Buffer[SSD1306_WIDTH * SSD1306_HEIGHT / 8];


static SSD1306_t SSD1306;


static void ssd1306_WriteCommand(uint8_t command) {
        HAL_I2C_Mem_Write(&SSD1306_I2C_PORT, SSD1306_I2C_ADDR, 0x00, 1, &command, 1,
                        10);
}


uint8_t ssd1306_Init(void) {
```

```
	HAL_Delay(100);


	ssd1306_WriteCommand(0xAE);
	ssd1306_WriteCommand(0x20);
	ssd1306_WriteCommand(0x10);
	ssd1306_WriteCommand(0xB0);
	ssd1306_WriteCommand(0xC8);
	ssd1306_WriteCommand(0x00);
	ssd1306_WriteCommand(0x10);
	ssd1306_WriteCommand(0x40);
	ssd1306_WriteCommand(0x81);
	ssd1306_WriteCommand(0xFF);
	ssd1306_WriteCommand(0xA1);
	ssd1306_WriteCommand(0xA6);
	ssd1306_WriteCommand(0xA8);
	ssd1306_WriteCommand(0x3F);
	ssd1306_WriteCommand(0xA4);
	ssd1306_WriteCommand(0xD3);
	ssd1306_WriteCommand(0x00);
	ssd1306_WriteCommand(0xD5);
	ssd1306_WriteCommand(0xF0);
	ssd1306_WriteCommand(0xD9);
	ssd1306_WriteCommand(0x22);
	ssd1306_WriteCommand(0xDA);
	ssd1306_WriteCommand(0x12);
	ssd1306_WriteCommand(0xDB);
	ssd1306_WriteCommand(0x20);
	ssd1306_WriteCommand(0x8D);
	ssd1306_WriteCommand(0x14);
	ssd1306_WriteCommand(0xAF);


	ssd1306_Fill(Black);


	ssd1306_UpdateScreen();


	SSD1306.CurrentX = 0;
	SSD1306.CurrentY = 0;

	SSD1306.Initialized = 1;

	return 1;
}


void ssd1306_Fill(SSD1306_COLOR color) {

	uint32_t i;

	for (i = 0; i < sizeof(SSD1306_Buffer); i++) {
		SSD1306_Buffer[i] = (color == Black) ? 0x00 : 0xFF;
	}
}

void ssd1306_UpdateScreen(void) {
	uint8_t i;

	for (i = 0; i < 8; i++) {
		ssd1306_WriteCommand(0xB0 + i);
		ssd1306_WriteCommand(0x00);
		ssd1306_WriteCommand(0x10);

		HAL_I2C_Mem_Write(&SSD1306_I2C_PORT, SSD1306_I2C_ADDR, 0x40, 1,
				&SSD1306_Buffer[SSD1306_WIDTH * i], SSD1306_WIDTH, 100);
	}
}

void ssd1306_DrawPixel(uint8_t x, uint8_t y, SSD1306_COLOR color) {
	if (x >= SSD1306_WIDTH || y >= SSD1306_HEIGHT) {

		return;
```

```
	}


	if (SSD1306.Inverted) {
		color = (SSD1306_COLOR) !color;
	}


	if (color == White) {
		SSD1306_Buffer[x + (y / 8) * SSD1306_WIDTH] |= 1 << (y % 8);
	} else {
		SSD1306_Buffer[x + (y / 8) * SSD1306_WIDTH] &= ~(1 << (y % 8));
	}
}


char ssd1306_WriteChar(char ch, FontDef Font, SSD1306_COLOR color) {
	uint32_t i, b, j;


	if (SSD1306_WIDTH <= (SSD1306.CurrentX + Font.FontWidth) ||
	SSD1306_HEIGHT <= (SSD1306.CurrentY + Font.FontHeight)) {

		return 0;
	}

	for (i = 0; i < Font.FontHeight; i++) {
		b = Font.data[(ch - 32) * Font.FontHeight + i];
		for (j = 0; j < Font.FontWidth; j++) {
			if ((b << j) & 0x8000) {
				ssd1306_DrawPixel(SSD1306.CurrentX + j, (SSD1306.CurrentY + i),
					(SSD1306_COLOR) color);
			} else {
				ssd1306_DrawPixel(SSD1306.CurrentX + j, (SSD1306.CurrentY + i),
					(SSD1306_COLOR) !color);
			}
		}
	}


	SSD1306.CurrentX += Font.FontWidth;


	return ch;
}

char ssd1306_WriteString(char *str, FontDef Font, SSD1306_COLOR color) {

	while (*str) {
		if (ssd1306_WriteChar(*str, Font, color) != *str) {

			return *str;
		}


		str++;
	}


	return *str;
}

void ssd1306_SetCursor(uint8_t x, uint8_t y) {
	SSD1306.CurrentX = x;
	SSD1306.CurrentY = y;
}

void ssd1306_Clear_Bufer_part(int x, int y, int width, int height) {
	for (int j = 0; j < height; j++) {
		for (int i = 0; i < width; i++) {

			ssd1306_DrawPixel(x + i, y + j, Black);
		}
	}
}
```

```
void ssd1306_Write_To_Bufer(int x, int y, int width, int height,
		const uint8_t *img) {
	for (int j = 0; j < height; j++) {
		for (int i = 0; i < width; i++) {
			if (((img[j * width / 8 + (i / 8)] >> (7 - i % 8)) & 0b00000001)
					== 1)
				ssd1306_DrawPixel(x + i, y + j, White);
			else
				ssd1306_DrawPixel(x + i, y + j, Black);
		}
	}
}

void ssd1306_WriteStringUint(uint16_t inn, FontDef Font, SSD1306_COLOR color) {
	if (inn == 0) {
		ssd1306_WriteString("0", Font, color);

	} else {
		uint16_t count = 0;
		uint16_t dev = 1;
		uint16_t num = 0;
		while (inn / dev != 0) {
			count++;
			dev *= 10;
		}
		dev = dev / 10;
		for (uint16_t i = 0; i < count; i++) {
			num = inn / dev;
			inn = inn % dev;
			dev = dev / 10;
			switch (num) {
			case 1:
				ssd1306_WriteString("1", Font, color);
				break;
			case 2:
				ssd1306_WriteString("2", Font, color);
				break;
			case 3:
				ssd1306_WriteString("3", Font, color);
				break;
			case 4:
				ssd1306_WriteString("4", Font, color);
				break;
			case 5:
				ssd1306_WriteString("5", Font, color);
				break;
			case 6:
				ssd1306_WriteString("6", Font, color);
				break;
			case 7:
				ssd1306_WriteString("7", Font, color);
				break;
			case 8:
				ssd1306_WriteString("8", Font, color);
				break;
			case 9:
				ssd1306_WriteString("9", Font, color);
				break;
			case 0:
				ssd1306_WriteString("0", Font, color);
				break;
			}

		}
	}
}
```

3. Текст программы WiN10 приложения

3.1. Файл MainWindow.xaml.cs

```
using RJCP.IO.Ports;
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Management;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace IDMCompanion
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private bool usbThreadWorking;
        private string usbbufer;
        private string naeedTotransfer;
```

```
private Thread usbThread;
private Thread usbThreadAn;
private Thread datatransfer;
private SerialPortStream serialPort;
private int counter = 0;
private string login = "";
private string password = "";
bool naeedTotransfercheck;

public MainWindow()
{
    InitializeComponent();
    usbbufer = ""; naeedTotransfer = "";
    usbThread = new Thread(new ThreadStart(usbThreadFunk));
    usbThreadAn = new Thread(new ThreadStart(usbThreadAnalys));
    datatransfer = new Thread(new ThreadStart(traansferdata));
    usbThread.Start();
    usbThreadAn.Start();
    datatransfer.Start();
}


//подключение устройства
/*
private void Button_Click(object sender, RoutedEventArgs e)
{
    usbThreadWorking = !usbThreadWorking;
    if (usbThreadWorking)
    {
        usbbufer = "";
        usbThread = new Thread(new ThreadStart(usbThreadFunk));
        usbThreadAn = new Thread(new ThreadStart(usbThreadAnalys));
        datatransfer = new Thread(new ThreadStart(traansferdata));
        usbThread.Start();
        usbThreadAn.Start();
        datatransfer.Start();
    }
    else
    {
```

```
        usbThread.Abort();
        usbThreadAn.Abort();
        datatransfer.Abort();
        serialPort.Close();
        dataBox.Text += ("Disconnected" + "\n");
        usbbufer = "";

    }

}*/

[STAThread]
private void usbThreadFunk()
{
    while (true)
    {
        if (SerialPortStream.GetPortNames().Length > 0)
        {
            foreach (string name in SerialPortStream.GetPortNames())
            {
                try
                {
                    using (serialPort = new SerialPortStream(name))
                    {
                        serialPort.OpenDirect();
                        //string data = "P " + GetMD5OFIDs();
                        //string data = "P 12345678901234567890123456";
                        string data = "P 09876543210987654321123456";
                        //string data = "P 0987654321dddd6543211234";
                        //string data = "P 1187654321ddgd6543211234";
                        //data = data.Insert(5, "s");/////
                        serialPort.Write(data);
                        this.Dispatcher.Invoke(() => dataBox.Text = dataBox.Text += "Wait...\n");
                        //System.Threading.Thread.Sleep(100);
                        int i = 0;
                        char a, b;
                        a = (char)serialPort.ReadChar();
                        b = (char)serialPort.ReadChar();
```

```
                                if (a == 'O' && b == 'K') this.Dispatcher.Invoke(() => dataBox.Text =
dataBox.Text += "Connected\n");
                                else if (a == 'N' && b == 'O')
                                {
                                    this.Dispatcher.Invoke(() => dataBox.Text += ("Your PC is unsaved or you
didn't connect the device" + "\n"));
                                }
                                else
                                {
                                   // this.Dispatcher.Invoke(() => dataBox.Text += ("Something  wrong..." +
"\n"));
                                }
                                usbThreadWorking = true;
                                while (serialPort.IsOpen)
                                {
                                    char ch = (char)serialPort.ReadChar();
                                    this.Dispatcher.Invoke(() => usbbufer += (ch));
                                    this.Dispatcher.Invoke(() => dataBox.Text += (ch));
                                    this.Dispatcher.Invoke(() => dataBox.ScrollToEnd());
                                    //this.Dispatcher.Invoke(() => labelcount.Content= usbbufer.Length);

                                }
                            }

                        }
                        catch { }
                    }
                    this.Dispatcher.Invoke(() => dataBox.Text = dataBox.Text += "No device found\n");
                    System.Threading.Thread.Sleep(500);


                }
                //this.Dispatcher.Invoke(() => dataBox.Text = dataBox.Text += "Can not connect to
device...\n");
                usbThreadWorking = false;
                this.Dispatcher.Invoke(() => usbbufer = "");
            }
        }
```

```
private void traansferdata()
{
    while (true)
    {
        if (naeedTotransfercheck)
        {
            int a = naeedTotransfer.IndexOf("begin(");
            int b = naeedTotransfer.IndexOf(")end");
            naeedTotransfer = naeedTotransfer.Substring(a + 6, b - a - 6);
            int num = 0;
            string nacurFileme = "";
            string curFile = "save" + num.ToString() + ".txt";
            while (File.Exists(curFile))
            {
                num++;
                curFile = "save" + num.ToString() + ".txt";
            }
            using (FileStream fstream = new FileStream(curFile, FileMode.OpenOrCreate))
            {
                byte[] array = System.Text.Encoding.Default.GetBytes(naeedTotransfer);
                for (int i = 0; i < naeedTotransfer.Length; i++)
                {
                    if (i % 48 == 0 && i != 0) fstream.WriteByte((byte)'\n');
                    fstream.WriteByte(array[i]);
                }
                this.Dispatcher.Invoke(() => dataBox.Text = dataBox.Text += "\nWrited\n");
            }
            naeedTotransfercheck = false;
            naeedTotransfer = "";

        }

    }

}

private void usbThreadAnalys()
{
    bool log = false;
```

```
            bool pas = false;
            while (true) {
                while (usbThreadWorking)
                {
                    if (usbbufer.Length > 0)
                    {
                        if (usbbufer.Length > 64)
                        {
                            usbbufer = usbbufer.Substring(usbbufer.Length/3*2);
                        }
                            System.Threading.Thread.Sleep(1000);
                        //this.Dispatcher.Invoke(() => dataBox.Text = dataBox.Text += "\nRaeding\n");
                        log = false;
                        pas = false;
                        try
                        {
                            if (usbbufer.IndexOf("l") >= 0)
                            {
                                //this.Dispatcher.Invoke(() => dataBox.Text += "!" +
usbbufer.Substring(usbbufer.IndexOf("l") + 3, 16) + "!");
                                if (usbbufer.IndexOf("l") >= 0 && usbbufer.IndexOf("o") >= 0 &&
usbbufer.IndexOf("g") >= 0)
                                {
                                    if (usbbufer[usbbufer.IndexOf("l") + 2] == 'o' &&
usbbufer[usbbufer.IndexOf("l") + 4] == 'g' && usbbufer[usbbufer.IndexOf("l") + 6] == ':')
                                    {


                                        this.Dispatcher.Invoke(() => { if (usbbufer.Length > 0) login =
usbbufer.Substring(usbbufer.IndexOf("l") + 7); });
                                        log = true;
                                        usbbufer = "";
                                    }
                                }


                            }
                            if (usbbufer.IndexOf("p") >= 0)
                            {
                                if (usbbufer.IndexOf("p") >= 0 && usbbufer.IndexOf("a") >= 0 &&
usbbufer.IndexOf("s") >= 0)
                                {
```

```
                                    if (usbbufer[usbbufer.IndexOf("p") + 2] == 'a' &&
usbbufer[usbbufer.IndexOf("p") + 4] == 's' && usbbufer[usbbufer.IndexOf("p") + 6] == ':')

                                    {

                                        this.Dispatcher.Invoke(() => { if (usbbufer.Length > 0) password =
usbbufer.Substring(usbbufer.IndexOf("p") + 7); });

                                        pas = true;

                                        usbbufer = "";

                                    }

                                }

                            }

                            if (usbbufer.IndexOf("b") >= 0)

                            {

                                System.Threading.Thread.Sleep(400);

                                if (usbbufer.IndexOf("b") >= 0 && usbbufer.IndexOf("e") >= 0 &&
usbbufer.IndexOf("g") >= 0 && usbbufer.IndexOf("i") >= 0 && usbbufer.IndexOf("n") >= 0 &&
usbbufer.IndexOf("(") >= 0)

                                {

                                    while (usbbufer.IndexOf(")") < 0 && usbbufer.IndexOf("e") < 0 &&
usbbufer.IndexOf("n") < 0 && usbbufer.IndexOf("d") < 0) { this.Dispatcher.Invoke(() => dataBox.Text =
dataBox.Text += "."); }

                                    this.Dispatcher.Invoke(() => dataBox.Text = dataBox.Text += " Data
recived...\n");

                                    naeedTotransfer = usbbufer;

                                    usbbufer = "";

                                    naeedTotransfercheck = true;

                                }



                            }

                        }

                        catch { usbbufer = ""; }

                        //System.Threading.Thread.Sleep(100);

                        if (log)

                        {

                            SendKeys.SendWait(login.Split(' ')[0]);

                            SendKeys.SendWait("{ENTER}");

                        }

                        else//System.Threading.Thread.Sleep(100);

                        if (pas)

                        {

                            byte[] bytes = Encoding.ASCII.GetBytes(password);

                            StringBuilder hex = new StringBuilder(bytes.Length * 2);

                            foreach (byte b in bytes)
```

```
                        hex.AppendFormat("{0:x2}", b);
                    password = hex.ToString();
                    SendKeys.SendWait(password);
                    SendKeys.SendWait("{ENTER}");


                }
                System.Threading.Thread.Sleep(100);
            }
        }
    }

}



    //отправка данных
    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
        if (usbThreadWorking)
        {
            //comandBox
            //inndataBox
            string comandBoxstr = comandBox.Text;
            string inndataBoxstr = inndataBox.Text;
            string data = "";
            if (comandBoxstr.Length>16 || inndataBoxstr.Length > 16)
            {
                dataBox.Text += ("Too long domen or login" + "\n");
            }
            else
            {

                while (comandBoxstr.Length < 16) { comandBoxstr += " "; }
                while (inndataBoxstr.Length < 16) { inndataBoxstr += " "; }
                data = "N " + inndataBoxstr + comandBoxstr ;
                while (data.Length < 64) { data += " "; }
            }
            serialPort.Write(data);
```

```
        dataBox.Text += ("\nОтправлено:" + data+ " \n Получено:");
    }
    else
    {

        dataBox.Text += ("Disconnected, cant send" + "\n");
    }
    dataBox.ScrollToEnd();

}

//начать инициацию
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    if (usbThreadWorking)
    {
        string data = "P " + GetMD5OFIDs();
        serialPort.Write(data);
        dataBox.Text += ("\nОтправлено: " + "Init comand " + " \n Получено:");
    }
    else
    {

        dataBox.Text += ("Disconnected, cant send" + "\n");
    }
    dataBox.ScrollToEnd();
}

//импорт
private void Button_Click_3(object sender, RoutedEventArgs e)
{
    if (usbThreadWorking)
    {
        string data = "I";
        serialPort.Write(data);
        dataBox.Text += ("\nОтправлено: " + "Get all data command" + " \n Получено:");
    }
    else
    {
```

```
        dataBox.Text += ("Disconnected, cant send" + "\n");
    }
    dataBox.ScrollToEnd();
}

//сброс устройства
private void Button_Click_4(object sender, RoutedEventArgs e)
{
    if (usbThreadWorking)
    {
        string data = "C";
        serialPort.Write(data);
        dataBox.Text += ("\nОтправлено: " + "Clear comand " + " \n Получено:");
    }
    else
    {
        dataBox.Text += ("Disconnected, cant send" + "\n");
    }
    dataBox.ScrollToEnd();
}

//Добавить новый надежный ПК
private void Button_Click_5(object sender, RoutedEventArgs e)
{
    if (usbThreadWorking)
    {
        string data = "A";
        serialPort.Write(data);
        dataBox.Text += ("\nОтправлено: " + "Add safe PC comand " + " \n Получено:");
    }
    else
    {
        dataBox.Text += ("Disconnected, cant send" + "\n");
    }
    dataBox.ScrollToEnd();
}
```

```
        private void Button_Click_7(object sender, RoutedEventArgs e)
        {
            dataBox.Text = "";
            dataBox.ScrollToEnd();
        }

         public string GetMD5OFIDs()
            {
                Dictionary<string, string> ids =
               new Dictionary<string, string>();
                ManagementObjectSearcher searcher;
                //процессор
                searcher = new ManagementObjectSearcher("root\\CIMV2",
                       "SELECT * FROM Win32_Processor");
                foreach (ManagementObject queryObj in searcher.Get())
                    ids.Add("ProcessorId", queryObj["ProcessorId"].ToString());
                //мать
                searcher = new ManagementObjectSearcher("root\\CIMV2",
                       "SELECT * FROM CIM_Card");
                foreach (ManagementObject queryObj in searcher.Get())
                    ids.Add("CardID", queryObj["SerialNumber"].ToString());
                //UUID
                searcher = new ManagementObjectSearcher("root\\CIMV2",
                       "SELECT UUID FROM Win32_ComputerSystemProduct");
                foreach (ManagementObject queryObj in searcher.Get())
                    ids.Add("UUID", queryObj["UUID"].ToString());
                string outp = "";
                foreach (var x in ids)
                    outp += x.Value;
                var md5 = MD5.Create();
                var hash = md5.ComputeHash(Encoding.UTF8.GetBytes(outp));
                return Convert.ToBase64String(hash);
            }
    }
}
```

Лист регистрации изменений

Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Входящий № сопроводительного докум. и дата	Подпись	Дата
	Измененных	Замененных	Новых	Анулированных					

Изм.	Лист	№ докум.	Подп.	Дата
Инв. № подл.	Подп. И дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.01.01-01 12 01-1				