Střední průmyslová škola Třebíč

Maturitní práce

ELEKTRONICKÝ ZÁMEK S RFID

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITB4

Školní rok: 2024/2025 Michal Paulas

SPŠT
Střední průmyslová škola Třebíč
Manželů Curieových 734, 674 01 Třebíč

Zadání práce

Obor studia: **18-20-M/01 Informační technologie**

Celé jméno studenta: **Michal Paulas**

Třída: **ITB4** Školní rok: **2024/2025**

Číslo tématu: **7**

Název tématu: **Elektronický zámek s RFID**

Rozsah práce: **15–25 stránek textu**

Specifické úkoly, které tato práce řeší:

Navrhněte a zrealizujte elektricky ovládaný dveřní zámek pomocí čipu RFID a klávesnice. Zámek umožní vstup buď po načtení autorizovaného čipu nebo po zadání PIN na klávesnici. Po úspěšné autorizaci se sepne relé ovládající zámek a zazní zvukový signál. Navrhněte způsob autorizace čipů (max. 10 čipů). ID čipu a PIN bude uložen tak, aby se neztratil vypnutím napájení. Pro realizaci prostředí Atmel Studio a použijte školní stavebnici.

Termín odevzdání: **28. března 2025, 23.00**

Vedoucí projektu: **Ing. Ladislav Havlát**

Oponent: **Ing. Jana Veselá**

Schválil: **Ing. Petra Hrbáčková, ředitelka školy**

ABSTRAKT

Tato práce se zabývá návrhem a realizací elektronického zámku s RFID čtečkou, LCD displejem a klávesnicí. Cílem bylo vytvořit systém, který umožní přístup buď pomocí RFID čipu, nebo zadáním správného PIN kódu. Systém využívá mikrokontroléru ATmega644A, který zajišťuje komunikaci s jednotlivými komponentami, jako je RFID čtečka, klávesnice, LCD displej a také možnost uložení dat do jeho EEPROM. Práce popisuje hardware a software, použitý kód a principy fungování celého zařízení. Výsledkem je funkční prototyp, který umožňuje řízení přístupu a ukládání autorizovaných čipů uživatelů.

KLÍČOVÁ SLOVA

RFID, elektronický zámek, ATmega644A, klávesnice, EEPROM, mikrokontrolér, Ledky, Displej

PODĚKOVÁNÍ

Děkuji Ing. Ladislavu Havlátovi za cenné připomínky a rady, které mi poskytl při vypracování maturitní práce.

V Třebíči dne podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a.

V Třebíči dne

podpis autora

Obsah

Úvod

Tento projekt se zaměřuje na návrh a vývoj elektronického dveřního zámku, který využívá technologii RFID spolu s rozhraním klávesnice pro lepší uživatelské pohodlí. Systém umožňuje přístup dvěma způsoby: buď naskenováním autorizovaného RFID čipu, nebo zadáním správného PIN kódu na klávesnici. Jakmile je uživatel úspěšně ověřen, aktivují se ledky, které indikují že byl přístup povolen.

Jádro projektu je postaveno na mikrokontroléru ATmega644A, který řídí činnost různých komponent včetně RFID čtečky, klávesnice, LCD displeje a Ledek. Kromě toho se EEPROM používá k ukládání důležitých dat, jako jsou ID čipů RFID a PIN kód, což zajišťuje uchování dat i v případě výpadku napájení. Vývojovým prostředím použitým pro tento projekt je Atmel Studio se všemi hardwarovými komponenty poskytnutými školou.

Tato práce podrobně popisuje návrh a integraci hardwarových i softwarových prvků a řeší problémy, které se vyskytly během vývoje.

1 Použité vývojové prostředí a nástroje

1.1 Atmel Studio

Atmel Studio je integrované vývojové prostředí (IDE) určené pro vývoj a ladění aplikací na mikrokontrolérech AVR a SAM. Poskytuje většinou bezproblémové a uživatelsky přívětivé prostředí pro psaní, vytváření a ladění kódu v C/C++ nebo v assembleru. Atmel Studio také umožňuje import náčrtů Arduino jako projekty C++, což usnadňuje přechod z prototypování k samotné výrobě a programování reálného produktu. [1]

1.2 SenderAVR

SenderAVR je program od SPŠT který se využívá pro nahrávání zkompilovaného programu na samotný mikroprocesor. Využívá k tomu USB, to stačí z mikroprocesoru zapojit do počítače, nastavit v programu správný COM port a pak už jen nahrát samotný program.

1.3 GitHub

GitHub je široce používaná platforma pro správu verzí, spolupráci a správu kódu, postavená na systému správy verzí Git. Poskytuje cloudové prostředí, kde mohou vývojáři ukládat, sdílet a spravovat své projekty, ať už pracují samostatně nebo v týmech. GitHub zjednodušuje vývoj softwaru tím, že zajišťuje strukturovaný pracovní postup a bezproblémovou spolupráci. Platforma podporuje veřejná i soukromá úložiště, takže je vhodná pro open-source projekty i pro soukromý vývoj.[2][3]

GitHub jsem hojně využíval pro verzování, sdílení a ukládání všech souborů a dokumentace souvisejících s projektem. Byl to základní nástroj pro organizaci mé práce, sledování změn v průběhu času a zajištění toho, že všechny projektové zdroje jsou bezpečně uloženy a dostupné odkudkoli. Tímto jsem se nemusel bát jakékoliv ztráty mých dat a samotné práce.[2][3]

2 Hardware

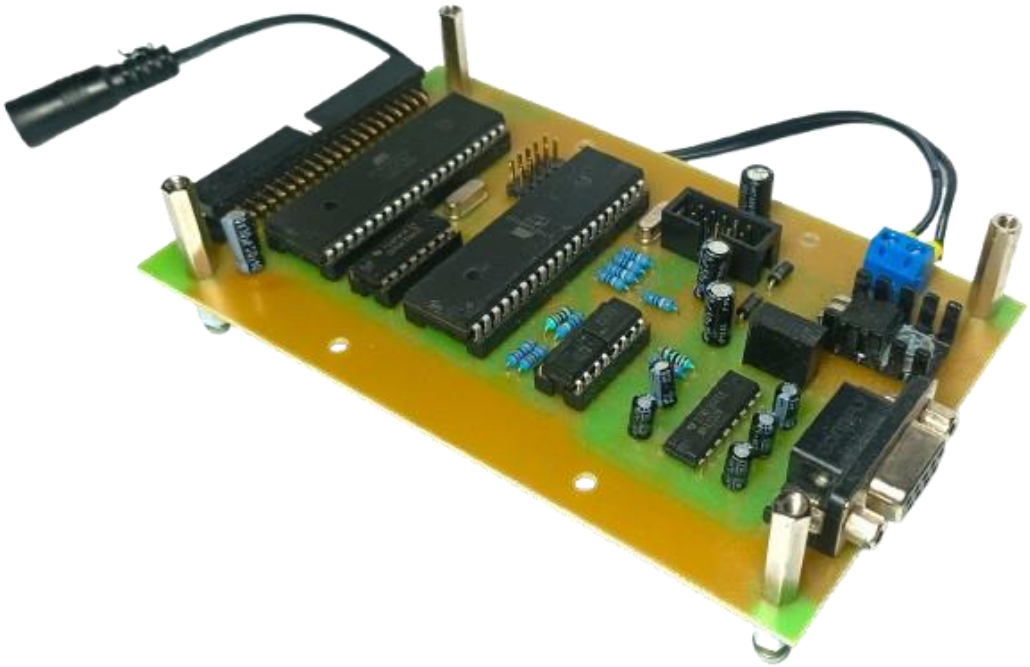
Veškeré HW prostředky byly poskytnuty školou. Během prvotního testování jsem narazil na problém s mikroprocesorem ATmega644A, na kterém byl zprvu pouze jeden nefunkční pin na PORTC, ale postupem času na něj bylo obtížnější až nakonec nemožné nahrávat jakýkoliv program. Nakonec mi byl mikroprocesor dodán funkční a já mohl bez problému pokračovat ve vývoji RFID Zámku.

HW poskytnutý školou dohromady funguje jako taková stavebnice. Každá z komponent je přidělána na PCB destičce a na ní deseti pinový konektor. Ten je u všech modulů stejný. Tím se pak pomocí rozbočovače, co jde přímo z mikroprocesoru, dají zapojit moduly na potřebné porty. Porty se jmenují PORTA, PORTB, PORTC a PORTD a v projektu jsem využil porty všechny. Na PORTA je připojený LCD display, na PORTB jsou připojené Ledky, na PORTC je připojená klávesnice a na PORTD samotná RFID čtečka.

2.1 ATmega644A

ATmega644A je 8bitový CMOS mikrokontrolér založený na architektuře AVR® RISC, určený pro vysoce výkonné a nízkoenergetické aplikace. V tomto případě je k mikroprocesoru připojený externí krystal pro přesnější časování mikroprocesoru. Použitý krystal kmitá ve frekvenci 11059200 Hz.

V projektu byl mikroprocesor použit jako mozek celého projektu. Využíval jsem na něm funkce jako I2C pro komunikaci s LCD displejem, USART ke komunikaci s RFID čtečkou a čítač/časovač pro umožnění programu mít funkci „timesUP()“ která přeruší různé akce po určitém čase. [4]



Obrázek 1 Hlavní modul s ATmega644 A

2.1.1 Klíčové vlastnosti:

Paměť a úložiště:

- 64KB programové paměti Flash
- 2KB EEPROM
- 4KB SRAM

Zpracování a výkon:

- Pokročilá architektura RISC se 131 výkonnými instrukcemi
- Propustnost až 20 MIPS při 20 MHz
- 32×8 univerzálních pracovních registrů

Periferní rozhraní:

- Dva 8bitové časovače/čítače a jeden 16bitový časovač/čítač
- Šest PWM kanálů
- 8kanálový 10bitový ADC s volitelným zesílením
- Dvě rozhraní USART, SPI a I2C
- Podpora JTAG pro ladění a programování

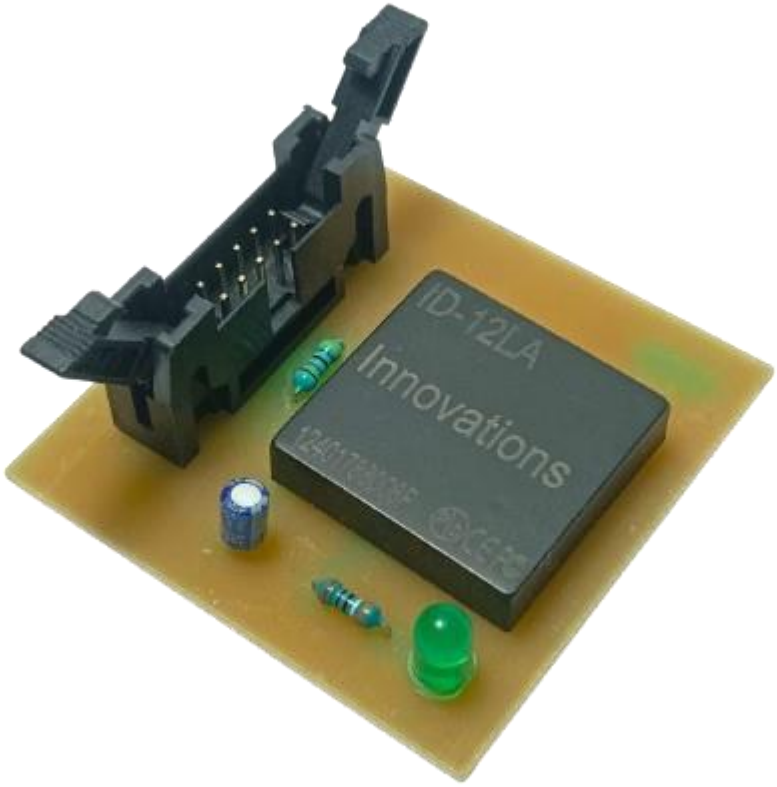
Energetická účinnost:

- Pracuje při napětí mezi 2,7V a 5,5V
- Má několik režimů úspory energie jako power-down a standby režimu [4]

2.2 RFID čtečka a čipy

Jako RFID čtečku byl modul již zmíněné stavebnice, na tomto modulu se nachází čtečka ID-12LA od společností SparkFun Electronics. Jedná se o snadno použitelnou čtečku, která umožnuje přijímat a přenášet na master zařízení ID naskenovaného čipu. Čip komunikuje skrze rozhraní USART, tudíž má výstupy RX a TX.

Jako čipy pro RFID jsem použil školní čip, jelikož pracuje na frekvenci, jakou RFID čtečka dokázala přečíst. Dále se také dal použít průkaz ISIC, jelikož disponuje tímto typem čtečky.



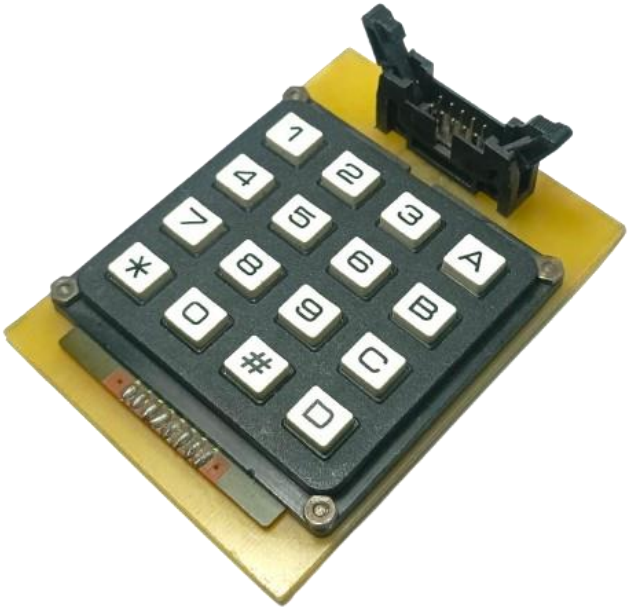
Obrázek 2 Modul s RFID čtečkou

2.2.1 Specifikace čtečky

- Napájecí napětí: 2,8 V – 5 V
- Nosná frekvence: 125 kHz
- Rozsah čtení: až 120 mm
- Standard: EM4001 ISO RFID IC
- Komunikace: sériové rozhraní TTL a RS232 – 9600bps
- Vestavěná anténa
- Rozteč pinů: 2 mm
- Rozměry: 25 x 26 x 7 mm [5]

2.3 Klávesnice

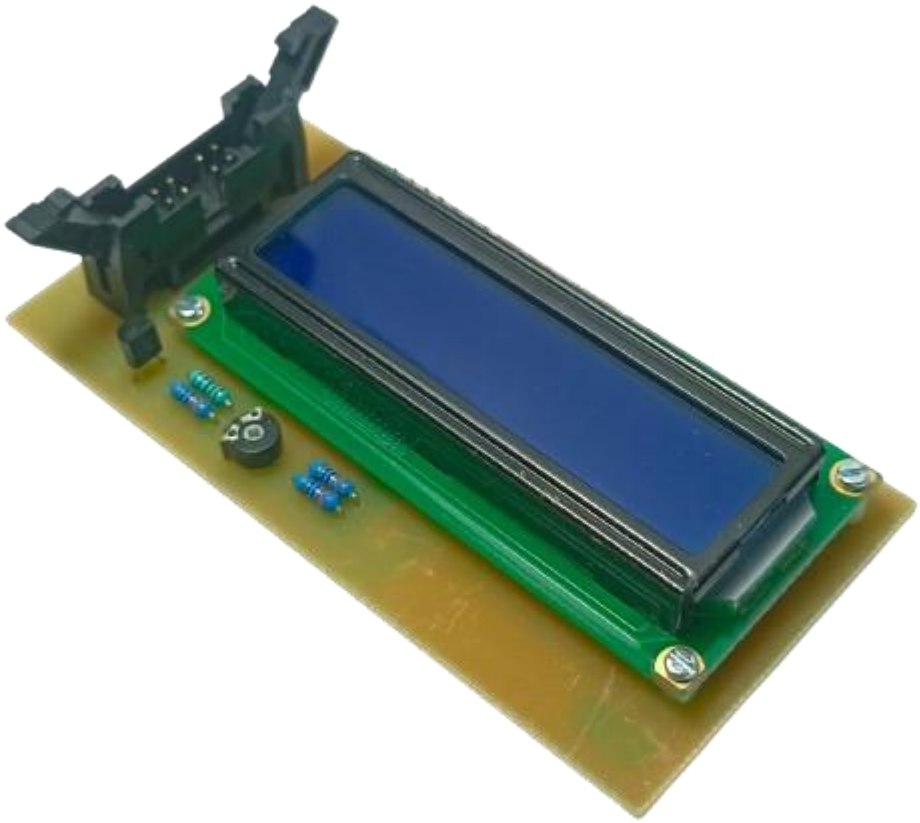
Pro klávesnici byla použita 4x4 klávesnice. Ta funguje na principu čtyř a čtyř vodičů položených na sebe pod úhlem devadesátí stupňů a tím, jak se dva vodiče spojí, tak to mikroprocesor zaregistruje. Na klávesnici se nachází celkem 16 tlačítek, tlačítka na sobě mají buďto čísla, písmena a znaky. Čísla od 0 až 9, písmena od A do D a dva znaky * a #.



Obrázek 3 Modul klávesnice 4x4

2.4 LCD display

Jako display byl použit standardní LCD display 16x2 s dvěma řádky a šestnácti poli pro znaky. Display disponuje modrým podsvícením. Na modulu je také obvod, který umožnuje regulovat intenzitu podsvícení pomocí potenciometru. Modul komunikuje skrze I2C komunikaci, ta je u modulu dána na piny SDA – B7, SCL – B6 a PWR – B0.



Obrázek 4 Modul s LCD Displejem

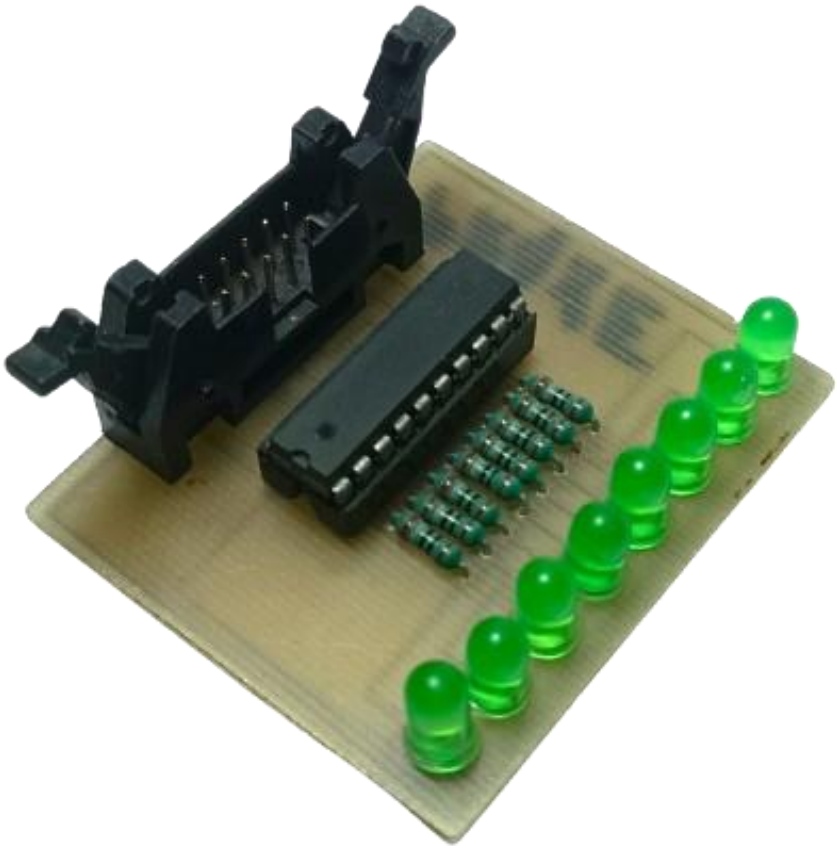
2.4.1 Specifikace LCD displeje

- Rozměry modulu: 80 x 35 x 11 mm
- Velikost samotné informační plochy: 64,5 x 14 mm
- LCD displej: s modrým podsvícením
- Široký pozorovací úhel a vysoký kontrast

- Průmyslový standard: HD44780
- Napájení: 5 V DC
- Pod světlení: A, K je na 5 V, stabilizátor je již na DPS displeje [6]

2.5 LED světla

Pro indikaci, zda je zámek otevřený či zavřený byl použit modul s LEDkami. Ten disponuje osmi LEDkami, každá na 5 V a budič sběrnice M74HCT245B1. Ten využívá pouze tří pinů z portu, na který se připojí a dokáže z toho různě rozsvěcovat ledky na modulu.



Obrázek 5 Modul s LEDkami

2.5.1 Specifikace M74HCT245B1

- Budič
- Má 8 výstupních kanálů
- V pouzdře DIP20
- Montáž THT [7]

3 Software

Program pro RFID čtečku je psaný v programovacím jazyce C++. Psaný byl ve vývojovém prostředí Atmel Studio, které bylo nastavené pro práci s mikroprocesorem ATmega644A. V programu jsem využil Přerušení, jak od časovače, tak od sériové komunikace, která byla využita pro komunikaci s RFID čtečkou. Dále byla využita EEPROM paměť pro ukládání čipů, pinu a stavu v jakém se čtečka nachází. Také byl využit Watchdog Reset Flagu pro restartování mikroprocesoru, to bylo využito ve chvíli kdy uživatel restartoval pin.

3.1 Popis kódu

3.1.1 Definice

Na začátku programu se nachází definice. Nastavuji zde frekvenci mikroprocesoru, který má externí krystal. Pro upřesnění je to frekvence 11059200 Hz. Dále jsou zde definovány ibit SDA, SCL a PWR (Napájecí) pin. Tyto piny využívá použitý LCD display pro komunikaci skrze I2C a napájení. Nastavují se na PORTA, a to přesně na jeho piny 7, 6 a 0. Na další definici nastavujeme „kbd_port“ na PORTC. To je proměnná z knihovny, která je zde využita modulem klávesnice.

```
1   #define F_CPU 11059200
2   #define sda ibit (PORTA, 7)
3   #define scl ibit (PORTA, 6)
4   #define pwr ibit (PORTA, 0)
5   #define kbd_port PORTC
```

Obrázek 6 Ukázka definicí v kódu

3.1.2 Include

Používají se zde dva typy include. První, který se pozná použitím těchto závorek: <>, využívá knihovny, které jsou přístupné od samotného Atmel Studia. Poté druhý typ, který používá uvozovky. Tento se využívá pro využití knihoven, které jsou uloženy na disku v počítači. Do těchto uvozovek se pak napíše cesta, kde je knihovna uložena.

Jako první se zaměříme na include, které jsou přístupné od samotného Atmel Studia. Automaticky je zde přidaný include: #include <avr/io.h>, toto je základní knihovna, která se zde používá pro práci se samotným mikroprocesorem, jeho registry atd... Dále je zde použita knihovna: #include <util/delay.h>. Tato knihovna je zde využita pro funkci: „_delay_ms()“. Tato funkce způsobí že se na požadovaný čas v milisekundách mikroprocesor zastaví. Poté je zde využita knihovna: #include <avr/interrupt.h>. Jelikož je v kódu použito přerušení od čítače/časovače a od sériové komunikace, tak zde tato knihovna musí být přidána pro zajištění funkčnosti přerušení. Další include je: #include <string.h>. Tato knihovna je zde použita pro práci s textovými řetězci. A nakonec je tu include: #include <avr/wdt.h>, ten přidá knihovnu, který nám umožní využívat WatchDogTimer pro reset mikroprocesoru.

Další tu jsou include, které využívají externí soubory jako knihovny. Všechny zde použité knihovny jsou dodané od školy, autorem většiny těchto knihoven je Ladislav Havlát. Aby byl kód funkční na každém zařízení s Atmel Studiem bude potřeba upravit cestu k těmto souborům. Prvním z include je: #include "G:\\MIR\\INCLUDE\\lcd_i2c_u.h". Tato knihovna přidá funkce pro práci s LCD displejem, který komunikuje pomocí I2C. Další include: #include "G:\\MIR\\INCLUDE\\i2c_u.h" umožní samotnou komunikaci skrze I2C. Poslední include: #include "G:\\MIR\\INCLUDE\\kbd_u.h" umožňuje správnou práci s připojenou klávesnicí.

```
13  #include <avr/io.h>
14  #include <util/delay.h>
15  #include <avr/interrupt.h>
16  #include <string.h>
17  #include <avr/wdt.h>
18
19  #include "../../INCLUDE/lcd_i2c_u.h"
20  #include "../../INCLUDE/i2c_u.h"
21  #include "../../INCLUDE/kbd_u.h"
```

Obrázek 7 Ukázka include z kódu

3.1.3 Definice vlastní

Pro ulehčení následné modularity kódu jsme veškeré důležité hodnoty, které jsou limitované zadáním také vložil do definice. Pro příklad by tento mikroprocesor dokázal uložit přes 30 různých RFID čipů.

- MAX_CHIPS – Maximální počet čipů, které mohou být přidány.
- STATUS_START_ADDR – Adresa, na které se začíná ukládat status každého čipu, zatím se využívá pouze že je slot zabraný nebo volny, ale pokud by se na zámku dále pokračovalo, bylo by možné implementovat funkce které by mohli nějaký čip pozastavit ve funkčnosti a nemazat, nebo cokoliv jiného.
- DATA_START_ADDR – Adresa, na které začíná ukládání samotných čipů do EEPROM paměti.
- FIRST_STARTUP – Hodnota, co nabývá buďto 0 nebo 255 indikující, jestli je zámek zapnut poprvé. Pokud ano, tak zámek požádá o zadání pinu.
- PIN_START_ADDR – Adresa, kam se uloží pin, jeho délka je 6 znaků a jeden ukončovací znak.
- CHIP_LENGTH – Určuje délku samotného čipu, u většiny je 16, ale pro jistotu se to stále dá pozměnit.
- PIN_LENGTH – Délka pinu, samotný pin je dlouhý na 6 a je zde přidaný ukončovací znak, takže je celková délka 7.
- TIME_TO_EXIT_MS – Čas v milisekundách který určí jak dlouho má uživatel pro dokončení akce. Po tomto čase se ukáže že byla akce přerušena a zámek se vrátí do základního stavu.

```
23  #define MAX_CHIPS 10
24  #define STATUS_START_ADDR 10
25  #define DATA_START_ADDR 20
26  #define FIRST_STARTUP 0
27  #define PIN_START_ADDR 1
28  #define CHIP_LENGTH 16
29  #define PIN_LENGTH 7
30  #define TIME_TO_EXIT_MS 10000
```

Obrázek 8 Ukázka vlastních definicí

3.1.4 Přerušení od sériové komunikace

Toto přerušení se zavolá kdykoliv přijde nový bajt. Kód v přerušení slouží k uložení veškerých dat do proměnné: „data_recived“, jedná se o pole proměnných typu char, což jsou znaky. Jako první se v kódu ptáme na registr UDR0, jestli má hodnotu 0x02, což značí že se začal číst čip. Pokud ano, tak se proměnná „start_OK“ typu bool nastaví na true. Dále je zde podmínka, která se ptá, jestli je již zmiňovaná proměnná „start_OK“ nastavena na true. Pokud to platí tak postupně další přečtené bajty ukládáme do pole „data_recived“. V této podmínce je také ještě podmínka, která hlídá, jestli už byl čip kompletně přečtený a pokud ano tak nastaví proměnnou „data_OK“ na false a vynuluje celočíselnou proměnou i, která počítala počet kolik bajtů už bylo přečteno.

```
41  ISR(USART0_RX_vect)
42  {
43      if(UDR0 == 0x02)
44      {
45          start_OK = true;
46      }
47
48      if (start_OK)
49      {
50          data_received[i] = UDR0;
51          i++;
52
53          if(i > 13)
54          {
55              data_OK = true;
56              i = 0;
57              start_OK = false;
58          }
59      }
60  }
```

Obrázek 9 Ukázka přerušení od sériové komunikace

3.1.5 Přerušení od čítače/časovače

Vyvolá se pokaždé když má čítač/časovač stejnou hodnotu jako registr OCR1A. Je to nastaveno, aby se to stalo každých 10ms. Pokaždé co se přerušení vyvolá tak se do celočíselné proměnné „idle_time“ přičte 10. To slouží k počítání času kdy nastane vypršení již dříve nastaveného času v milisekundách.

```
62  ISR(TIMER1_COMPA_vect) {
63      idle_time += 10;
64
65  }
```

Obrázek 10 Ukázka přerušení od čítače/časovače

3.1.6 Funkce na čtení a zápis do EEPROM

Tyto funkce byly celé převzaty z dokumentace mikroprocesoru ATmega644 A. V oběma funkcích prvně počkáme, dokud se nedokončí předchozí zápis, to zjistíme tím, že je v registru EECR namaskovaná logická 1 na místě EEPE. Dokud je to pravda, tak je program zaseklý v cyklu. V samotné funkci pro čtení poté uložíme do registru EEAR adresu, ze které chceme číst, ta je celočíselná, nezáporná proměnná. Poté pro začátek čtení z EEPROM namaskujeme v registru EECR logickou 1 na místo EERE. Nakonec vrátíme proměnnou typu unsigned char, v niž je hodnota registru EEDR, ve kterém je osmi bitová hodnota.

Funkce pro zápis po vyčkání na dokončení předchozího zápisu uloží adresu na kterou chceme psát do registru EEAR. Dále uloží data která chceme zapsat do registru EEDR, ukládá se zde proměnná typu unsigned char. Dále namaskuje logickou 1 do registru EECR na místo EEMPE. Nakonec namaskuje logickou 1 na místo EEPE což započne zápis do EEPROM.

```
71  unsigned char EEPROM_read(unsigned int uiAddress)
72  {
73      while(EECR & (1<<EEPE))
74      ;
75      /* Set up address register */
76      EEAR = uiAddress;
77      /* Start eeprom read by writing EERE */
78      EECR |= (1<<EERE);
79      /* Return data from Data Register */
80      return EEDR;
81  }
82
83  void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
84  {
85      /* Wait for completion of previous write */
86      while(EECR & (1<<EEPE))
87      ;
88      /* Set up address and Data Registers */
89      EEAR = uiAddress;
90      EEDR = ucData;
91      /* Write logical one to EEMPE */
92      EECR |= (1<<EEMPE);
93      /* Start eeprom write by setting EEPE */
94      EECR |= (1<<EEPE);
95  }
```

Obrázek 11 Ukázka funkcí pro čtení a zápis do EEPROM

3.1.7 Funkce pro zapnutí vypršení času

Logika pro celkový systém vypršení času spočívá v proměnné „idle_time“, ta je typu unsigned int. Jak bylo již zmíněno v části o přerušení od čítače/časovače, přidáváme k ní každých 10ms číslo 10. V této funkci její hodnotu nastavíme na nulu a bude tím tak počítat čas od zavolání této funkce. Používá se kdykoliv je potřeba použít funkci pro vypršení času, ta je popsaná v následující části.

```
97  void startTimer(){
98
99      idle_time = 0;
100
101 }
```

Obrázek 12 Ukázka funkce pro zapnutí vypršení času

3.1.8 Funkce pro zjištění vypršení času

Tato funkce vrací proměnnou typu bool. Pokud se již uplynulý čas do zapnutí vypršení času rovná, nebo je větší než ve vlastních definicích TIME_TO_EXIT_MS tak vrátí true.

```
103 bool timesUP(){
104     return (idle_time >= TIME_TO_EXIT_MS);
105 }
```

Obrázek 13 Ukázka funkce pro zjištění vypršení času

3.1.9 Funkce pro zobrazení základního textu na displej

Aby bylo v hlavní části jednoduší přepisovat displej tak vznikla tato funkce. Kdykoliv se zámek zapne, když už má uživatel nastavený pin tak se na displeji musí zobrazit instrukce pro uživatele co má vlastně dělat a tato funkce to zařídí. Používá funkce z knihovny pro LCD displej které zapíší na display požadovanou posloupnost znaků.

```
111 void showMenuText(){
112
113     text_row1("Enter PIN");
114     text_row2("Enclose CHIP");
115
116 }
```

Obrázek 14 Ukázka funkce pro zobrazení základního textu na displej

3.1.10 Funkce pro smazání displeje

Pro jednoduší mazání displeje. Používá stejné funkce z knihovny pro LCD displej, ale místo textu do něj vkládá prázdno.

```
114 void clearDisplay() {
115 
116     text_row1("");
117     text_row2("");
118 
119 }
```

Obrázek 15 Ukázka funkce pro smazání displeje

3.1.11 Funkce pro uložení pinu do paměti EEPROM

Na vstupu je konstantní pole znaků což je pole typu char. Konstantní znamená že se ve funkci daná proměnná nemůže upravovat. Tím zajistíme že by se v budoucnu nemělo stát, aby se nijak neupravoval pin, co se musí uložit. Ve funkci se nachází cyklus for, který bude probíhat, dokud bude celočíselná proměnná i menší než již dříve zapsaná definice PIN_LENGTH. Pokaždé co se projede cyklem se proměnná i zvětší o jedničku a tím se pak také samotný kód v cyklu řídí. Ten využívá této proměnné pro zapsání celého pole postupně do EEPROM paměti pomocí funkce: „EEPROM_write()“.

```
121 void savePin(const char pin[6]) {
122 
123     for (int i = 0; i < PIN_LENGTH; i++) {
124         EEPROM_write(PIN_START_ADDR + i, pin[i]);
125 
126     }
127 
128 }
```

Obrázek 16 Ukázka kódu pro uložení pinu do EEPROM paměti

3.1.12 Funkce pro první zapnutí zámku

Na začátku funkce se vytvoří proměnná typu pole char o velikosti 7 pro uložení budoucího pinu co se zadá. Poté nezáporná celočíselná osmibitová proměnná „index“ která bude určovat na jaké místo v poli se budou zadané znaky zapisovat, a nakonec proměnnou typu char „key“ do které se vždy uloží jaký znak byl zadán.

Zavolá se funkce „clearDisplay()“ pro smazání obsahu displeje a poté se zapíše na vrchní řádek displeje výzva pro zadání pinu.

Následně je cyklus, který se ukončí až poté co uživatel zadá a potvrdí pin. V cyklu zavoláme „kb_on_timer1()“, je to funkce potřebná pro správné fungování klávesnice. Následuje podmínka, ve které se ptáme, jestli byla stisknuta klávesa. Pokud ano tak se uloží tento znak do proměnné „key“.

V další podmínce se poté ptáme, o jaký se jedná znak a podle toho pak program reaguje. Když se jedná o číslo a zároveň ještě nebyl zapsán požadovaný počet znaků, tak se uloží na pozici určenou proměnnou „index“ do pole. Proměnná index se poté zvětší o jedno a vypíše se celý obsah pole na displej. Pokud už se jedná o poslední znak, co se zadává, tak se na konec do pole vloží ukončovací znak.

Při další podmínce se ptáme, jestli není znak *, což smaže jeden zadaný znak z pole, zmenší proměnnou „index“ o jedno a opět vypíše na displej.

Nakonec se pak v podmínce ptáme, jestli je zadaný znak D a zároveň je už zadaný dostatečný počet znaků. V tomto případě pak uložíme zadaný pin do pole „pristupovyPin“ což bude následně použito pro odemčení zámku pomocí pinu. Dále se uloží pomocí funkce „savePin“ pin. Nakonec už se pouze do EEPROM paměti na adresu FIRST_STARTUP, která byla na začátku kódu definována, uloží že byl zámek už jednou zapnut, tudíž má uložený pin a vypíše se zpráva pro uživatele na displej, že byl pin uložen.

```
129 void FirstStartUp() {
130     char code[7] = {0};
131     uint8_t index = 0;
132     char key;
133 
134     clearDisplay();
135 
136     text_row1("Enter PIN:");
137 
138     while (1) {
139 
140         kb_on_timer1();
141 
142         if (KB_CMD1 & 0x80) {
143 
144             key = (char)(KB_CMD1 & 0x7F);
145             KB_CMD1 = 0;
146 
147             if (key >= '0' && key <= '9' && index < 6) {
148 
149                 code[index] = key;
150                 code[index + 1] = '\0';
151                 index++;
152                 text_row2(code);
153 
154                 } else if (key == '*' && index > 0) {
155 
156                 index--;
157                 code[index] = '\0';
158                 text_row2(code);
159 
160                 } else if (key == 'D' && index == 6) {
161 
162                 for (int i = 0; i < 7; i++) {
163                     pristupovyPin[i] = code[i];
164                 }
165 
166                 savePin(code);
167                 EEPROM_write(FIRST_STARTUP, 0x00);
168 
169                 text_row1("Pin saved!");
170                 _delay_ms(1000);
171 
172                 break;
173             }
174         }
175     }
176 }
```

Obrázek 17 Ukázka funkce FirstStartUp

3.1.13 Funkce pro porovnání pinu

Jedná se o funkci vracející bool hodnotu a na vstupu má textový řetězec ve kterém je pin co uživatel zadal pro přístup do zámku. Uvnitř je podmínka, která pomocí funkce „strcmp“ z knihovny string.h porovnává dva textové řetězce, jestli jsou stejné. Pokud ano tak funkce vrátí true, naopak false.

```
180  bool ComparePin(const char* zadanyPin) {
181
182      if (strcmp(zadanyPin, pristupovyPin) == 0) return true;
183
184      return false;
185
186  }
```

Obrázek 18 Ukázka funkce pro porovnání pinů

3.1.14 Funkce pro prvotní nastavení veškerých funkcí

Nastavují se zde veškeré registry, porty, inicializace komponent atd... Jako první se nastavují registry pro nastavení přerušení od příjmu na serial komunikaci. To je důležité pro funkčnost komunikace s RFID čtečkou. Také se nastaví registry pro čítač/časovač, aby každých 10 milisekund nastalo přerušení, že se hodnota v registru OCR1A rovná hodnotě čítače/časovače.

Následně se nastavují porty A a B na 255 což znamená logická 1, DDRA na logickou nulu a DDRB na logickou 1. To znamená že je DDRA nastavený pro vstup a DDRB pro výstup. Tímto se zajistí správná funkčnost ledek a LCD displeje.

Poté jsou inicializace veškerých komponent. Inicializuje se klávesnice a i2c komunikace pomocí funkcí „kb_init()“ a „i2c_init()“. Tyto funkce jsou z vložených knihoven pro i2c komunikaci a klávesnici. Nastaví se zde také pro displej PWR pin. Po tomto všem se zavolá funkce „sei()“ která globálně povolí přerušení, takže veškeré nastavení registrů na přerušení bude funkční.

Nakonec se v podmínce zavolá funkce „i2c_open_wr()“ která vezme hardwarovou adresu LCD displeje a zkusí s ním komunikovat. Pokud je to úspěšné tak se display vyčistí a problikne.

```
188 void SetRegistersInit(){
189
190     UCSR0A = 0;
191     UCSR0B |= (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0); //Povolení přerušení od přijmu
192     UCSR0C |= (1<<UPM01)|(1<<UCSZ01)|(1<<UCSZ00);
193     UBRR0 = 71; //9600
194
195     TCCR1B = (1 << WGM12);                // CTC mód
196     TCCR1B |= (1 << CS12) | (1 << CS10);  // Předdělička 1024
197     OCR1A = 108;                          // Hodnota pro 10 ms interval
198     TIMSK1 = (1 << OCIE1A);               // Povolení přerušení na compare match
199
200     DDRA = 0b11111111;
201     PORTA= 0b10000000;
202
203     DDRB = 255;
204     PORTB = 255;
205
206     kb_init();
207     i2c_init();
208     set_bit(prt(pwr));
209
210     sei();
211
212     if (i2c_open_wr(HWA,0))
213     {
214         lcd_clr();
215         lcd_on();
216         cur_on();
217         cur_blink_off();
218         _delay_ms(1);
219         i2c_stop();
220     }
221
222 }
```

Obrázek 19 Ukázka funkce pro prvotní nastavení

3.1.15 Funkce pro otevření zámku

Jelikož se pouze jedná o ukázku zámku tak zde není reálný zámek, ale pouze ledky, které ukazují, jestli je zámek otevřen nebo ne. Funkce prvně využije funkci „clearDisplay()“ ta displej smaže. Poté nastaví PORTB na logickou nulu, takže se ledky rozsvítí. Dále se na displej vypíše zpráva pro uživatele, že byl zámek otevřen. Program počká jednu vteřinu a poté ledky znovu zhasne nastavením logické 1 na PORTB. Nakonec se zavolá funkce „showMenuText()“ a program se vrátí na hlavni obrazovku.

```
224 void OpenLock(){
225
226     clearDisplay();
227     PORTB = 0;
228     text_row1("Access allowed!");
229     _delay_ms(1000);
230     PORTB = 255;
231     showMenuText();
232
233 }
```

Obrázek 20 Ukázka funkce pro otevření zámku

3.1.16 Funkce pro zamítnutí přístupu

Jelikož je zámek normálně vždy zamčený tak se ve funkci pouze vyčistí displej funkcí „clearDisplay()“, vypíše se na displej zpráva, že byl přístup zamítnut, program počká vteřinu a vrátí se na hlavní obrazovku.

```
235 void AccessDenied(){
236
237     clearDisplay();
238     text_row1("Access denied!");
239     _delay_ms(1000);
240     showMenuText();
241 }
```

Obrázek 21 Ukázka funkce pro zamítnutí přístupu

3.1.17 Funkce pro nalezení volného místa v paměti

Ve funkci je pouze jeden cyklus, který projede všechny možné pozice pro uložení čipu. Cyklus jede tak dlouho dokud je proměnná typu „int“ menší než dříve nadefinovaná hodnota MAX_CHIPS. V cyklu se poté nachází podmínka, ve které se ptáme, jestli přečtená hodnota z EEPROM z definované adresy STATUS_START_ADDR a k ní přidaná dříve zmírněná proměnná typu „int“ je rovna 255 (0xFF je zapsáno hexadecimálně). To znamená že je místo volné pro zapsání čipu a funkce vrátí hodnotu v dříve zmíněné proměnné „int“. Pokud nebylo žádné volné místo nalezeno tak se navrátí hodnota -1.

```
244 int findFreeSlot() {
245
246     for (int i = 0; i < MAX_CHIPS; i++) {
247
248         if (EEPROM_read(STATUS_START_ADDR + i) == 0xFF) { // 255(0xFF) volno, 0(0x00) obsazeno
249             return i;
250
251         }
252     }
253
254     return -1; // Když není žádné volné místo
255
256 }
```

Obrázek 22 Ukázka funkce pro nalezení volného místa v paměti

3.1.18 Funkce pro uložení čipu

Funkce má dva parametry. Pole char, ve kterém je uložený RFID čip, který je potřeba uložit a proměnou slot typu „int“ která říká do jakého slotu se má uložit RFID čip. Jako první přepíšeme adresu statusu daného slotu na 0, což bude indikovat že byl slot zabrán. Poté se vypočítá z proměnné slot adresa, na kterou máme začít ukládat RFID čip. Samotný čip se poté uloží následujícím cyklem, který z pomocí vypočítané adresy postupně zapíše všech šestnáct char z pole. Pro ukázku se na konci smaže displej a na chvíli se na něj vypíše celé pole obsahující RFID čip.

```
258 void saveChip(char chip[16], int slot) {
259
260     EEPROM_write(STATUS_START_ADDR + slot, 0x00);
261
262     int baseAddress = DATA_START_ADDR + slot * CHIP_LENGTH;
263
264     for (int i = 0; i < CHIP_LENGTH; i++) {
265         EEPROM_write(baseAddress + i, chip[i]);
266
267     }
268
269
270     clearDisplay();
271
272     text_row2(chip);
273     _delay_ms(500);
274
275 }
```

Obrázek 23 Ukázka funkce pro uložení čipu

3.1.19 Funkce pro přečtení čipu pomocí indexu

Funkce má opět dva parametry, proměnnou slot typu int a pole char, ale to zde slouží jako buffer pro čip, který se přečte. Dále funkce vrací proměnnou int, která bude indikovat co se ve funkci stalo. Jako první se podmínkou zjistí, zda náhodou není zadaný slot mimo hranici. To by se nemělo stát, ale pokud by byla omylem zadaná jiná hodnota, tak aby to nedělalo problémy. Když to nastane tak se vrátí 2.

Dále se podmínkou ptáme, jestli je na daném slotu vůbec nějaký čip uložený, pokud není tak funkce vrátí 1.

Z proměnné slot se dopočítá adresa, ze které budeme v následujícím cyklu číst čip. V cyklu opět projedeme všech šestnáct pozic a postupně je uložíme do buffer. Nakonec ještě na poslední pozici buffer vložíme ukončovací znak a vrátíme 0, což indikuje že vše proběhlo v pořádku.

```
277 int readChip(int slot, char buffer[16]) {
278
279     if (slot < 0 || slot >= MAX_CHIPS) {
280
281         return 2; // Mimo hranici
282
283     }
284
285     if (EEPROM_read(STATUS_START_ADDR + slot) == 0xFF) {
286
287         return 1; // Na slotu není uložen chip
288
289     }
290
291     int baseAddress = DATA_START_ADDR + slot * CHIP_LENGTH;
292
293     for (int i = 0; i < CHIP_LENGTH; i++) {
294
295         buffer[i] = EEPROM_read(baseAddress + i);
296
297     }
298
299     buffer[CHIP_LENGTH - 1] = '\0';
300
301     return 0; // Všechno v pořádku
302
303 }
```

Obrázek 24 Ukázka funkce pro přečtení čipu

3.1.20 Funkce pro zjištění, zda je čip uložený

Funkce vrací int značící, jestli je čip uložený, nepoužívá se zde bool, kdyby byly náhodou do funkce přidány další ověření a vracela by poté další hodnoty. Jako parametr je pole char, ve kterém je uložený RFID čip. V samotné funkci se založí další pole char délkou určenou definicí CHIP_LENGTH. Následuje cyklus, který projede veškeré sloty kde mohou být uloženy čipy. V cyklu se nachází podmínka, která volá funkci „readChip“ která vrátí 0 pouze když daný čip nalezla a do buffer uloží přečtený čip. Zároveň ale musí funkce z knihovny „string.h“ „strcmp“ vrátit nulu, to se stane, když se buffer rovná čipu, který jsme zadali. Pokud vše sedí tak se vrátí 1. Pokud ne tak se vrátí 0.

```
305 int isChipStored(char chip[16]) {
306
307     char buffer[CHIP_LENGTH];
308
309     for (int i = 0; i < MAX_CHIPS; i++) {
310
311         if (readChip(i, buffer) == 0 && strcmp(buffer, chip) == 0) {
312             return 1; // Čip nalezen
313
314         }
315
316     }
317
318     return 0; // Čip nebyl nalezen
319
320 }
```

Obrázek 25 Ukázka funkce pro zjištění, zda je čip uložen

3.1.21 Funkce pro přidání čipu

Jako u každé funkce, kde je potřeba ukončit ji za nějaký časový interval, tak jako první pomocí funkce „startTimer()“ zapneme stopování času. Do celočíselné proměnné se poté uloží hodnota z funkce „findFreeSlot()“, ta určí, kde je volné místo v EEPROM. Pokud nebyl žádný volný slot nalezen, tak funkce vrátí 0, což značí že se nepodařilo uložit čip z důvodu plné paměti. Dále se zobrazí na displej text sdělující uživateli, že má přiložit čip pro přidání. Následně funkce zajede do nekonečného cyklu, ten je zde tvořen pomocí „while(1)“ tudíž dokud pravda. V něm se poté ptáme, jestli byl přiložen čip. Pokud ano tak se zjistí, jestli náhodou není čip už uložen pomocí funkce „isChipStored()“, pokud ano tak se funkce ukončí s návratovou hodnotou 3, to

znamená že čip nemohl být uložen z důvodu přiložení již uloženého čipu. Když se funkce neukončila, tak vše zatím proběhlo správně a uloží se čip pomocí funkce „saveChip()“ do EEPROM a vrátí se 2, což znamená že vše proběhlo v pořádku. V samotném cyklu se také ptáme na funkci „timesUP()“ pro ukončení po určené době. Pokud toto nastane, tak se funkce ukončí s návratovou hodnotou 1, což značí že vypršel čas. V cyklu se také nachází krátká pauza „_delay_ms()“ nastavena na 50 milisekund, aby se funkce v cyklu stíhaly provádět.

```
322 int AddChip() {
323
324     startTimer();
325
326     int freeSLot = findFreeSlot();
327
328     if(freeSLot == -1){
329         return 0; // Překročen maximální počet čipů
330
331     }
332
333     text_row1("Enclose CHIP");
334     text_row2("for ADD");
335
336     while (1) {
337
338         if (data_OK) {
339
340             char chip[16];
341
342             for (int i = 0; i < 16; i++) {
343                 chip[i] = (char)data_prijata[i + 2];
344             }
345
346             data_OK = false;
347
348             if (isChipStored(chip)) {
349
350                 return 3; // Indikace, že čip existuje
351
352             }
353
354             saveChip(chip, freeSLot);
355
356             return 2; // Vše ok
357
358         }
359
360         if(timesUP()) {
361             return 1; // Čas vypršel
362         }
363
364         _delay_ms(50);
365
366     }
367 }
```

Obrázek 26 Ukázka funkce pro přidání čipu

3.1.22 Funkce pro odstranění čipu

Jako v předchozí funkci se jako první zapne časovač pro předběžné ukončení funkce po vypršení času. Vypíše se na displej zpráva pro uživatele, aby přiložil čipu pro odstranění z EEPROM. Dále následuje opět nekonečný cyklus „while(1)“. V něm se opět čte čip. Avšak se zde také nachází cyklus, který projede veškeré čipy v paměti. To se zjistí pomocí funkce „readChip()“ kam vkládáme id od 0 do maximálního počtu čipů. Pokud se nachází shoda tak se čip odstraní tím, že se na status adresu čipu 255, což značí že je na tomto místě již volno. Pokud toto nastane, tak funkce vrátí 0, to znamená že všechno proběhlo v pořádku. Pokud nebyl žádný shodný čip nalezen, tak funkce vrátí 1. Na konci cyklu se opět ptáme na funkci „timesUP()“, pokud již čas vypršel tak se vrátí 2. Opět je zde také krátká pauza 50 milisekund pro dokončení všechny funkcí v cyklu.

```
int RemoveChip() {

    startTimer();

    text_row1("Enclose CHIP");
    text_row2("for REMOVE");

    while (1) {

        if (data_OK) {

            char chip[16];

            for (int i = 0; i < 16; i++) {
                chip[i] = (char)data_prijata[i + 2];

            }

            data_OK = false;

            for (int i = 0; i < MAX_CHIPS; i++) {

                char buffer[16];

                if (readChip(i, buffer) == 0 && strcmp(buffer, chip) == 0) {

                    EEPROM_write(STATUS_START_ADDR + i, 0xFF);

                    return 0; // Čip byl úspěšně odstraněn
                }
            }

            return 1; // Čip nebyl nalezen
        }

        if (timesUP()) {
            return 2; // Čas vypršel

        }
        _delay_ms(50);

    }
}
```

Obrázek 27 Ukázka funkce pro odstranění čipu

3.1.23 Funkce pro zjištění, zda byl zámek otevřen pomocí RFID čipu

Jednoduchá funkce, která se ptá, jestli byl přiložen čip. Pokud ano tak se čip přečte a pomocí funkce „isChipStored()“ zjistí, jestli je tento čip uložen. Pokud ano tak funkce vrátí true. Pokud ne, tak funkce vrátí false.

```
414 bool CheckForOpenByRFID(bool &open){
415
416     if(data_OK)
417     {
418         char text[16];
419
420         for (int i = 0; i < 16; i++) {
421             text[i] = (char)data_prijata[i + 2];
422         }
423
424         open = isChipStored(text);
425
426         data_OK = false;
427
428         return true;
429     }
430
431     return false;
432
433 }
```

Obrázek 28 Ukázka funkce pro zjištění otevření zámku pomocí RFID čipu

3.1.24 Funkce pro ověření uživatele

Zapne se jako první časovač pro předběžné ukončení po určené době. Nadefinují se proměnné pro pin, index, který bude určovat na jaké místo se daný znak uloží a samotný znak. Poté se vyčistí displej pomocí funkce „clearDisplay()“ a započne nekonečný cyklus. Zde se čtou znaky napsané na klávesnici. Pokud se zadá číslo 0 až 9 tak se zapíše do proměnné pro pin, zvětší se index o 1 a na displej se vypíše všechny znaky uložené v proměnné pro pin. Pokud uživatel klikne na znak „*“ tak se zmenší hodnota index o 1 a jeden znak se smaže pomocí uložené prázdného znaku na zmenšeném index. Do této podmínky se ale můžeme dostat jen tehdy, když je index větší jak 0. Poté je zde další podmínka, která když uživatel zadá znak „D“ a už je index roven 6 pin porovná s uloženým pinem pomocí funkce „ComparePin()“. Pokud se piny shodují tak se na displej zobrazí zpráva, že byl uživatel verifikován, nechá se pauza 1 vteřina, aby uživatel stihl přečíst zprávu a funkce se ukončí s návratovou hodnotou 0. Pokud se však piny neshodují, tak se vrátí 1. Dále, pokud uživatel stiskne klávesu „#“ tak funkce vrátí 3, to značí že byla akce ověření uživatele přerušena. Na konci cyklu se také nachází funkce „timesUP()“ pro zjištění vypršení času, pokud ano, vrátí se 2.

```
435 int verifyUser(){
436
437     startTimer();
438
439     char code[7] = {0};
440     uint8_t index = 0;
441     char key;
442
443     clearDisplay();
444
445     text_row1("Enter PIN:");
446
447     while (1) {
448
449         kb_on_timer1();
450
451         if (KB_CMD1 & 0x80) {
452
453             key = (char)(KB_CMD1 & 0x7F);
454             KB_CMD1 = 0;
455
456             if (key >= '0' && key <= '9' && index < 6) {
457
458                 code[index] = key;
459                 code[index + 1] = '\0';
460                 index++;
461                 text_row2(code);
462
463                 } else if (key == '*' && index > 0) {
464
465                 index--;
466                 code[index] = '\0';
467                 text_row2(code);
468
469                 } else if (key == 'D' && index == 6) {
470
471                     if(ComparePin(code)){
472
473                         text_row1("Verified!");
474                         _delay_ms(1000);
475
476                         return 0;
477
478                     } else {
479
480                         return 1; // Špatný pin
481
482                     }
483
484
485
486                     break;
487                 } else if (key == '#') {
488
489                     return 3; // Akce přerušena
490
491                 }
492         }
493
494         if(timesUP()) return 2; // Čas vypršel
495
496     }
497
498 }
```

Obrázek 29 Ukázka funkce pro ověření uživatele

3.1.25 Funkce pro restart

Funkce pro restart mikroprocesoru. Pomocí funkce „wdt_enable()“ se povolí watchdog s timoutem 15 milisekund, to způsobí že se za 15 milisekund procesor restartuje. Aby však nedělal další úkony po tuto dobu, je na konec vložený nekonečný cyklus.

```
500 void restart() {
501     wdt_enable(WDTO_15MS); // Povolit watchdog s timeout 15 ms
502     while(1);              // Počkání dokud watchdog neresetuje MCU
503 }
```

Obrázek 30 Ukázka funkce pro restart

3.1.26 Vstupní bod programu

Ihned na začátku se vymaže vlajka pro watchdog reset, aby se mikroprocesor přestal restartovat. Samotný watchdog se poté vypne funkcí „wdt_disable()“. Poté se zavolá funkce „SetRegistersInit()“ pro inicializaci veškerých potřebných věcí a funkcí. Dále se podmínkou zjistí, zdali se zámek zapíná poprvé, pokud ano, tak se zavolá funkce „FirstStartUp()“. Pokud byl již zámek jednou zapnut, tak se do proměnné pro přístupový pin uloží pin z EEPROM. Po tomto se na displej zobrazí instrukce pro uživatele pomocí funkce „showMenuText()“. Nadefinují se potřebné proměnné a započne nekonečný cyklus. V něm se ptáme, zdali se uživatel snaží otevřít zámek pomocí čipu funkcí „CheckForOpenByRFID()“. Ta přepíše proměnnou „canOpen“ a tím zjistíme, zda je čip autorizován pro otevření zámku. Pokud ano tak se zavolá funkce „OpenLock()“ a otevře se zámek, naopak se zavolá funkce „AccesDenied()“. Následuje kód pro celkové ovládání zámku pomocí klávesnice. Když uživatel zadává čísla, zámek reaguje jako by se zadával přístupový pin. Tomu se poté pomocí „*“ umaže jeden znak. Po stisku „D“ se ověří, jestli je zadaný celý pin, pokud ano, zjistí se, zdali je pin zadaný správně. Pokud ano zámek se otevře, pokud ne, vypíše se zpráva, že byl zadaný špatný pin. Pokud se stiskne klávesa „A“, zavolá se funkce pro „AddChip()“ pro přidání čipu a podle vrácené hodnoty vypíše na displej potřebné informace pro uživatele. To samé se stane při stisku „B“, ale místo přidání se zavolá funkce „RemoveChip()“ pro odstranění čipu. Při stisku klávesy „C“ se smaže uložený pin smazáním adresy na vlastní definici FIRST_STARTUP a zavolá se funkce

„restart()“ která restartuje mikroprocesor. Pro všechny znaky A, B i C se ještě před veškerými funkcemi zavolá funkce pro ověření uživatele „verifyUser()“.

```
505 int main(void)
506 {
507
508     MCUSR &= ~(1<<WDRF); // Vymazání flagu watchdog resetu
509     wdt_disable();       // Vypnutí watchdogu
510
511     SetRegistersInit();
512
513     if(EEPROM_read(FIRST_STARTUP) == 0xFF){
514
515         FirstStartUp();
516
517     } else {
518
519         for (int i = 0; i < 6; i++) {
520             pristupovyPin[i] = EEPROM_read(PIN_START_ADDR + i);
521
522         } pristupovyPin[6] = '\0';
523
524     }
525
526     showMenuText();
527
528     bool canOpen = false;
529     char key;
530     char zadanyPin[7] = {0};
531     uint8_t pinIndex = 0;
532
533     while (1)
534     {
535
536         if(CheckForOpenByRFID(canOpen)){
537
538             if(canOpen) OpenLock();
539             else AccessDenied();
540
541         }
542
543         kb_on_timer1();
544
545         if (KB_CMD1 & 0x80){
546
547             key = (char)(KB_CMD1 & 0x7F);
548             KB_CMD1 = 0;
549
550             if (key >= '0' && key <= '9' && pinIndex < 6) {
551
552                 zadanyPin[pinIndex] = key;
553                 zadanyPin[pinIndex + 1] = '\0';
554                 pinIndex++;
555                 text_row2(zadanyPin);
556
557             } else if (key == '*' && pinIndex > 0) {
558
559                 pinIndex--;
560                 zadanyPin[pinIndex] = '\0';
561                 text_row2(zadanyPin);
562
563             } else if (key == 'D' && pinIndex == 6) {
564
565                 if(ComparePin(zadanyPin)) OpenLock();
566                 else AccessDenied();
567
568                 pinIndex = 0;
```

Obrázek 31 Ukázka vstupního bodu programu 1

```
570             } else if(key == 'A') {
571 
572                 int resultt = verifyUser();
573 
574                 if(resultt == 0){
575 
576                     int result = AddChip();
577                     clearDisplay();
578 
579                     if(result == 0) text_row1("No Space!");
580                     else if(result == 1) text_row1("Timed out!");
581                     else if (result == 2) text_row1("Success!");
582                     else if (result == 3) text_row1("Already Added!");
583 
584                 } else if (resultt == 1) AccessDenied();
585                 else if (resultt == 2) text_row1("Timed out!");
586                 else if (resultt == 3) text_row1("Event cancelled!");
587 
588                 _delay_ms(1000);
589                 showMenuText();
590 
591             } else if (key == 'B') {
592 
593                 int resultt = verifyUser();
594 
595                 if(resultt == 0){
596 
597                     int result = RemoveChip();
598                     clearDisplay();
599 
600                     if (result == 0) text_row1("Chip removed!");
601                     else if (result == 1) text_row1("Not found!");
602                     else if (result == 2) text_row1("Timed out!");
603 
604                 } else if (resultt == 1) AccessDenied();
605                 else if (resultt == 2) text_row1("Timed out!");
606                 else if (resultt == 3) text_row1("Event cancelled!");
607 
608                 _delay_ms(1000);
609                 showMenuText();
610 
611             } else if (key == 'C') { // Reset PINU
612 
613                 int resultt = verifyUser();
614 
615                 if(resultt == 0){
616                     clearDisplay();
617                     EEPROM_write(FIRST_STARTUP, 0xFF);
618                     text_row1("PIN Removed!");
619                     restart();
620 
621                 } else if (resultt == 1) AccessDenied();
622                 else if (resultt == 2) text_row1("Timed out!");
623                 else if (resultt == 3) text_row1("Event cancelled!");
624 
625                 _delay_ms(1000);
626                 showMenuText();
627 
628             }
629         }
630     }
631 }
```

Obrázek 32 Ukázka vstupního bodu programu 2

3.2 Styl uložení do EEPROM

Celkově má EEPROM paměť 512 bytů pro uložení dat. Na prvním místě se ukládá hodnota 0 nebo 255 pro uložení, jestli byl zámek již zapnut a nastaven předtím. Dále na adrese 1 až 7 se ukládá přístupový pin. Dále od adresy 10 až do adresy 19, zde se ukládá status na každém slotu pro uložení čipu. Dále už pouze od adresy 20 se ukládají čipy, vždy 16 bytů jeden čip.

Tabulka 1 Ukázka struktury ukládání do EEPROM

FIRST_STARTUP	PIN	CHIP_STATUS	CHIP_1	CHIP_2	CHIP_N
0	1-7	10-19	20-35	36-51	52-N

3.3 Jak pracovat s RFID zámkem

Po prvním zapnutí zámku vás požádá o zadání přístupového pinu, ten si zadáte, jaký chcete, on se uloží do paměti. Poté už se dá zámek tímto pinem otevřít.

3.3.1 Stisk „A“

Po stisku klávesy „A“ vás zámek požádá o zapsání přístupového pinu pro ověření uživatele. Po úspěšném ověření můžete přiložit čip, který se poté zapíše do paměti a půjde jim poté zámek otevřít. Je zde ale časovač, který zajistí, že se zámek vrátí na hlavní stránku. Pokud by chtěl uživatel akci zrušit, muže buďto počkat pár vteřin, nebo stisknout klávesu „#“ která akci ukončí, ale pouze ověření uživatele, samotné přidání už ne.

3.3.2 Stisk „B“

Po stisku klávesy „B“ zámek opět požádá o zadání přístupového pinu. Poté požádá o přiložení čipu, který chce uživatel odstranit z paměti. Čas může opět vypršet a verifikace uživatele přerušena stiskem klávesy „#“. Pokud se čip nenachází v paměti, zámek o tom vypíše zprávu.

3.3.3 Stisk „C“

Po stisku klávesy „C“ se opět požádá o verifikaci uživatele, která může, jak časově vypršet, tak být zrušena klávesou „#“. Po úspěšném ověření uživatele se zámek restartuje, pin smaže a zámek je jako při prvním zapnutí.

Závěr

Tato práce se zaměřila na návrh a realizaci elektronického dveřního zámku, který využívá technologii RFID a klávesnici jako metody ověřování uživatelů. Hlavním cílem bylo vytvořit systém, který by umožňoval autorizaci přístupu buď pomocí RFID čipu, nebo zadáním PIN kódu na klávesnici. Celý projekt byl realizován na mikrokontroléru ATmega644A, který zajišťoval řízení všech komponent, jako je RFID čtečka, klávesnice, LCD displej a EEPROM pro ukládání přístupových dat.

Během návrhu a implementace se objevilo několik technických výzev. Jedním z prvních problémů byla inicializace mikrokontroléru, kdy se ukázalo, že některé piny na PORTC nefungují správně, což vedlo k nutnosti výměny čipu.

Po dokončení vývoje se podařilo vytvořit funkční prototyp zámku, který umožňuje ukládání a ověřování RFID čipů, jejich mazání a možnost resetu PIN kódu. Systém obsahuje také vizuální indikaci přístupu, což viditelně naznačuje odemčení zámku.

Celkově lze projekt hodnotit jako úspěšný, neboť splňuje všechny základní požadavky kladené na elektronický zámek s více možnostmi ověřování. Výsledný systém by mohl být dále rozšířen například o možnost připojení k síti pro vzdálenou správu uživatelů, rozšíření paměti pro ukládání více čipů nebo přidání dalších bezpečnostních prvků. I přes určité komplikace během tvorby se podařilo vytvořit funkční a použitelný produkt, který demonstruje praktickou aplikaci využití mikrokontroléru v oblasti řízení přístupu.

Seznam použitých zdrojů

[1] *Microchip Studio for AVR® and SAM Devices.* Online. MicroChip. 2022. Dostupné z: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio. [cit. 2025-03-24].

[2] *GitHub.* Online. 2008. Dostupné z: https://github.com/. [cit. 2025-03-24].

[3] *GitHub.* Online. Wikipedia. 2009. Dostupné z: https://en.wikipedia.org/wiki/GitHub. [cit. 2025-03-24].

[4] *MICROCHIP TECHNOLOGY INC.* ATmega644P Datasheet [online]. Chandler: Microchip Technology Inc., [cit. 2025-03-26]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42744-ATmega644P_Datasheet.pdf.

[5] *Čtečka RFID ID-12LA - 125kHz - SparkFun SEN-11827.* Online. Botland. 2019. Dostupné z: https://botland.cz/stazene-produkty/1757-ctecka-rfid-id-12la-125khz-sparkfun-sen-11827-5904422330309.html. [cit. 2025-03-24].

[6] *LCD displej - Modrý, 16x2 znaků.* Online. Dratek.cz. 2010. Dostupné z: https://dratek.cz/arduino/836-display-modry-16x2-znaku.html. [cit. 2025-03-24].

[7] *M74HCT245B1 budič sběrnice.* Online. Kondik.cz. 2017. Dostupné z: https://www.kondik.cz/io-m74hct245b1/?srsltid=AfmBOorwGk_bVB0WJTZDiTPNDUaOFoSeI-31_KFC23gw7RY26j2e_vWh. [cit. 2025-03-24].

Seznam použitých zkratek

- RFID – Radio Frequency Identification (Identifikace na rádiové frekvenci)
- LCD – Liquid Crystal Display (Displej z tekutých krystalů)
- EEPROM – Electrically Erasable Programmable Read-Only Memory (Elektricky mazatelná a programovatelná paměť pouze pro čtení)
- IDE – Integrated Development Environment (Integrované vývojové prostředí)
- AVR – Alf-Egil Bogen & Vegard Wollan RISC (8bitová rodina mikrokontrolérů od Atmelu)
- SAM – Smart ARM Microcontroller (Mikrokontrolér od Atmelu s architekturou ARM)
- COM – Communication (Komunikační port, např. sériový port)
- USB – Universal Serial Bus (Univerzální sériová sběrnice)
- HW – Hardware (Fyzické vybavení počítače nebo elektronického zařízení)
- PCB – Printed Circuit Board (Deska plošných spojů)
- COMS – Complementary Metal-Oxide Semiconductor (Doplňkový polovodičový obvod s oxidy kovů)
- USART – Universal Synchronous/Asynchronous Receiver/Transmitter (Univerzální synchronní/asynchronní přijímač/vysílač)
- I2C – Inter-Integrated Circuit (Sběrnice pro komunikaci mezi čipy)
- RISC – Reduced Instruction Set Computing (Počítač s redukovanou instrukční sadou)
- SRAM – Static Random Access Memory (Statická operační paměť s náhodným přístupem)
- PWM – Pulse Width Modulation (Pulzně šířková modulace)
- ADC – Analog-to-Digital Converter (Analogově-digitální převodník)
- SPI – Serial Peripheral Interface (Sériové periferní rozhraní)
- JTAG – Joint Test Action Group (Standard pro testování a ladění elektronických obvodů)
- RX – Receive (Příjem dat, často v sériové komunikaci)
- TX – Transmit (Odesílání dat, často v sériové komunikaci)
- SDA – Serial Data Line (Datová linka pro I2C sběrnici)
- SCL – Serial Clock Line (Hodinová linka pro I2C sběrnici)
- DPS – Deska plošných spojů (Český ekvivalent PCB)
- THT – Through-Hole Technology (Technologie součástek s vývody pro montáž skrz otvory v PCB)
- UDR0 – USART I/O Data Register (Registr pro odesílání/příjem dat přes USART)
- OCR1A – Output Compare Register 1A (Registr pro porovnání výstupu, často u časovače)
- EECR – EEPROM Control Register (Řídicí registr EEPROM paměti)
- EEPE – EEPROM Program Enable (Bit umožňující zápis do EEPROM)
- EEAR – EEPROM Address Register (Registr pro adresování EEPROM paměti)
- EERE – EEPROM Read Enable (Bit umožňující čtení z EEPROM)

- EEDR – EEPROM Data Register (Registr pro ukládání dat do EEPROM)
- EEMPE – EEPROM Master Program Enable (Master povolení pro zápis do EEPROM)
- DDRA – Data Direction Register A (Registr pro nastavení směru portu A)
- DDRB – Data Direction Register B (Registr pro nastavení směru portu B)
- MCU – Micro Controller Unit (Mikroprocesor)
- MIPS – Microprocessor without Interlocked Pipeline Stages (Mikroprocesor bez vzájemně blokujících se pipeline stupňů)

Seznam obrázků

Seznam tabulek