OBSAH

SEZNAM OBRÁZKŮ

1. POPIS FUNKCE

1.1 POUŽITÍ

Klíčový trezor je určen pro zabezpečení klíčů a jejich evidenci. Použití zařízení je možné na vrátnicích, recepcích, v hotelech a všude tam, kde dochází k vyzvedávání klíčů samotnými uživateli nebo k výdeji pověřenou obsluhou a kde je nebezpečí zneužití klíčů.

1.2 OVLÁDÁNÍ

Do skříně mají přístup pouze vybraní uživatelé. Identifikace se provádí pomocí bezkontaktní karty v místě s anténou, která je na plášti skříně. Po platné identifikaci uživatel otevře dvířka a pozice s klíči, ke kterým má právo přístupu jsou uvolněny a indikovány svítícími LED diodami. Tyto klíče si uživatel může odnést. Jestliže určeného časového intervalu klíče neodebere, uvolněné pozice se opět uzamknou a klíč již není možno odebrat bez další identifikace. Zde tedy dochází k adresnému odběru klíčů uživateli a k časové evidenci jejich výdeje i příjmu.

1.3 KONSTRUKCE SKŘÍNĚ

Klíčový trezor se skládá ze 16 pozic pro klíče.

1.4 UZAVÍRÁNÍ SKŘÍNĚ

Pro zabezpečení klíčů proti zneužití je skříň uzavřena neprůhlednou roletou, která se odsouvá do boku. Roleta je jištěna zámkem, který se ovládá pomocí bezkontaktní identifikace.

1.5 POZICE PRO KLÍČ

Každá pozice je vybavena elektromagnetickým zámkem, do kterého se zasunuje zabezpečovací trn (přívěsek), na němž je zavěšen klíč. Přívěsek má dvě funkce:

- po zasunutí do své pozice v trezoru zajišťuje klíč proti neoprávněnému vytažení
- kontroluje své uložení do správné pozice

Na každou klíčovou pozici je optická signalizace LED, která indikuje různé stavy zařízení,

Přívěsek se zasouvá do otvoru zámku až na doraz. V této poloze již přívěsek nelze vyjmout bez opětovné identifikace. Pokud je přívěsek zasunut správně do klíčového zámku, je systémem prověřeno jeho uložení v odpovídající pozici.

1.6 ELEKTRONIKA

Klíčový trezor je vybaven několika čtecími zařízeními. Prvá čtečka slouží k identifikaci uživatele a uvolňuje mu roletu skříně. Další čtečky jsou umístěny v každé klíčové pozici a slouží k ověření správnosti uložení klíčů.

Data ze všech klíčových pozic zpracovává řídící jednotka, která zabezpečuje také komunikaci s PC. V každém klíčovém trezoru je jedna řídící jednotka.

1.7 PROGRAMOVÉ VYBAVENÍ (SW)

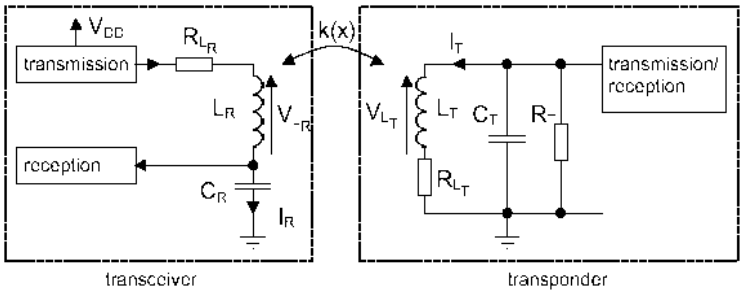
Pomocí SW se provádí konfigurace zavádění transpondérů klíčů a uživatelů, přidělování práv přístupu do trezoru a práv odběru klíčů apod. Celá konfigurace se nahraje do řídící jednotky a systém může pracovat autonomně. Pro snadné editování transpondérů slouží tzv. správcovská čtečka, kterou má správce systému u PC. Pomocí této čtečky se zavádějí do evidence nové klíče.

2. BEZKONTAKTNÍ IDENTIFIKACE

2.1 RFID TECHNOLOGIE

Bezkontaktní identifikační systém - RFID - Radio Frequency Identification Systems se skládá ze tří základních částí: Antény (cívky), vysílače (s dekóderem) a transpondéru (běžně nazývaného RFID Tag), který obsahuje svůj unikátní kód. Představuje pokročilou informační technologii, se kterou se setkáváme v běžném životě při různých aplikacích. Používá se všude tam, kde je nutný přesný sběr dat, který nelze vždy zabezpečit kontaktními systémy nebo systémy s čárovým kódem.

Tento bezkontaktní identifikační systém je založen na principu rádiového přenosu dat mezi čtečkou (vysílačem) a pohybujícím se objektem vybaveným takzvaným transpondérem, což je elektronický obvod, který obsahuje anténu umožňující příjem a vysílání dat, nabíjecí kondenzátor a paměť. Transpondéry nepotřebují napájení z baterie, energii k vysílání dat získávají z přijímaného signálu. Mohou být v různém provedení, tvar je specifický podle charakteru aplikace (např. karty velikosti kreditních karet, skleněné tyčinky, plastové disky, válce atd...). Jsou neopotřebovatelné a prakticky nezničitelné, jejich kopírovatelnost je téměř nemožná. Umístění snímače může být libovolné, třeba i za stěnou nebo v ní. Může být umístěn dokonce i zcela mimo identifikační místo, kde bude namontována pouze anténa. Zde je tedy výhoda oproti optickým i magnetickým identifikačním systémům, přičemž ceny snímačů vycházejí téměř stejné (i o něco levnější), cena karet je vyšší.



Transpondéry jsou vhodné do extrémních podmínek identifikace s nestandardní teplotou, nečistotami, vlhkostí, mlhou apod. Všude tam lze registrovat pohyb označeného předmětu, vyhodnocovat data a rozhodovat o sledovaném procesu. Konkrétní výběr transpondéru závisí na aplikaci.

Co se týče funkce, existují typy určené pouze pro čtení uloženého kódu (R/O transpondéry), stejně jako typy s možností naprogramování kódu o předurčené délce do interní paměti EEPROM (R/W transpondéry). Interní paměť dosahuje u transpondérů kapacity až 2kbit.

R/O transpondéry jsou užívány jako jedinečné a nekopírovatelné, každý takovýto transpondér obsahuje unikátní kód. Tyto prvky jsou široce použitelné ve všech aplikacích zabývajících se velkými databázemi s nezáměnnými položkami.

R/W transpondéry jsou určeny mimo jiné pro ukládání dat, nebo pro uživatelsky definovatelné identifikační kódování. Mohou být programovány, čteny a měněny tisícekrát. Programování se děje rovněž bezkontaktně - elektromagnetickým polem vytvářeným snímačem. Uživatel si tak může sám tvořit kódy ke snadné integraci s jeho počítačovým systémem zpracování dat.

RFID technologie je spolehlivou informační metodou. Aplikací této identifikace lze vyloučit ruční třídění a zápis, manuální kontroly apod. Výsledkem je zvýšení výkonu a rychlosti sběru informací vedoucí k lepšímu rozhodování na základě nejaktuálnějších informací. Tato technologie nabízí pro vývojové pracovníky možnosti různých nových aplikací nebo inovací v mnoha odvětvích trhu. Nabízí se zde vznik netypických aplikací – sledování cest zboží přes rozdělující uzly až k uzavření distribučního cyklu u zákazníka.

2.1.1 Hlavní výhody RFID

- není nutná přímá viditelnost pro čtení a zapisování do RFID tagů
- snížení chybovosti
- zlepšené řízení toku zboží
- vyšší stupeň automatizace
- digitální získávání informací
- rychlost pořízení informace
- mobilita
- možnost mnohačetného čtení
- odolnost a variabilita media

2.1.2 Použitá frekvence

Systémy RFID se provozují na různých vlnových délkách. Volba nejvhodnější frekvence je jedna z nejdůležitějších fází návrhu takovéhoto řešení. Z této volby totiž vyplývá celá řada dalších (nejen fyzických) omezení, jako například dosah čtečky, zákonná omezení, rychlost čtení a zapisování, použitelnost v různém prostředí a další.

Nízká frekvence 125–134 KHz LF Tag	Vysoká frekvence 13.56 MHz HF Tag	Velmi vysoká frekvence 860 – 930 MHz UHF tag	Mikrovlnná frekvence 2.45, 5.8 GHz MW tag
dosah pod 0,5 m malá rychlost čtení vysoké výrobní náklady možnost snímání na kovu a přes kapalinu	dosah do 1 m dostatečná rychlost čtení vysoké výrobní náklady obtížné čtení přes kapalinu	dosah do 3 m velká rychlost čtení nelze číst přes kapalinu, obtížné čtení z kovu	dosah do 10 m možnost čtení při extrémně vysokých rychlostech velká cena RFID tagu

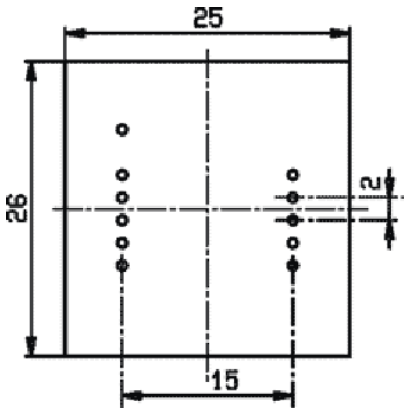
2.2 ČTEČKA RFID AXA 012

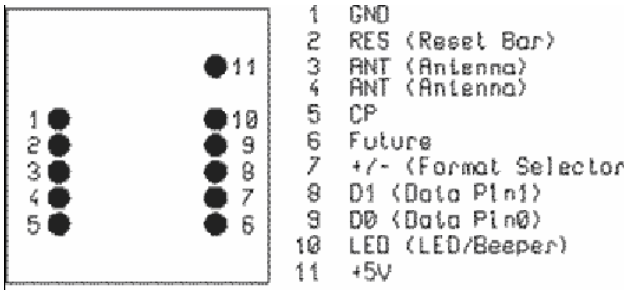
AXA 012 je nenápadná černá krabička připomínající spíše relé, ale obsahující hotovou čtečku RFID 125kHz včetně antény, převodníku na výstup TTL a schopností číst identifikační čipy na vzdálenost až 12 cm. Vzhledem k minimální náročnosti na vnější součástky je určen k téměř okamžitému nasazení do provozu.

RFID čtečka série AXA012 (100% náhrada za ID-12 Picomax) je hybridní obvod pro čtení bezkontaktních identifikátorů na frekvenci 125 kHz. Čtečka ID je navržena pro co nejjednodušší připojení k řídícímu mikroprocesoru nebo počítači. Při vložení bezkontaktního identifikátoru do elektromagnetického pole čtečky AXA 012 je na výstup čtečky vyslán přímo identifikační kód bezkontaktního identifikátoru (tagu). Čtečky AXA mají výstup podporující ASCII standard a Wiegand 26. ASCII formát je dostupný v CMOS úrovních nebo jako invertovaný TTL s protokolem RS232 9600,8,1. [6]

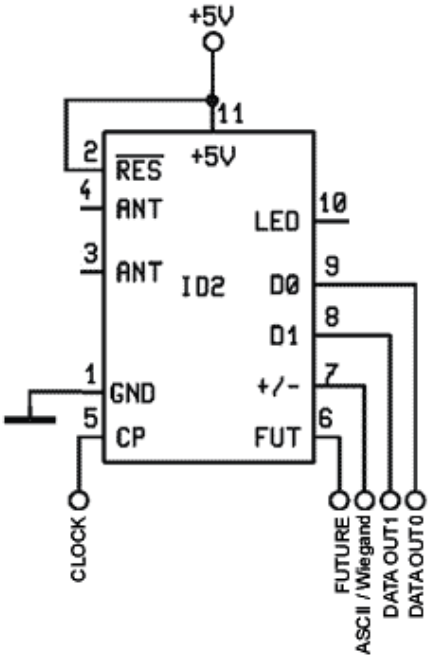
2.2.1 Vlastnosti RFID čtečky AXA-012 :

- Pracovní rozsah: 12 cm
- Rozměry : 26 mm x 25 mm x 7 mm
- Pracovní frekvence : 125 kHz
- Formát komunikujících RFID čipů: EM 4001 nebo kompatibilní (Unique)
- Kódování: Manchester 64-bitů
- Napájení: +5V @ 13mA
- Maximální výstupní proud u I/O pinu: 75 mA
- Rozsah napájecího napětí: +4.5 až 5.5 V





Obrázek 2.1: Rozložení vývodů čtečky AXA 012



Obrázek 2.2: Typické zapojení čtečky AXA 012

Použitím čtečky AXA 012 lze velmi rychle získat plnohodnotné čtecí zařízení pro bezkontaktní identifikaci. Po připojení mikroprocesoru či PC pro vyhodnocení transpondérů a systém, je připraven k okamžitému použití. V případě potřeby lze pro zvýšení dosahu čtečky AXA 012 doplnit externí anténou.[6]

2.3 RFID TRANSPONDÉR EM 4100

Nejjednodušším prvkem pro bezkontaktní identifikace jsou identifikační čipy, například EM 4100 s obchodním názvem UNIQUE. Jde o R/O čip s jedinečným 64-bit kódem vyráběný firmou EM-Microelectronic. Tento čip nese informaci o délce 64 bitů z čehož je 9 bitů hlavička (synchronizační bity), dále 40 datových bitů v Manchester kódu, 14 paritních bitů a 1 stop bit. Informace je vložena do čipu laserem při výrobě a výrobce garantuje, že nevyrobí dva čipy se shodným kódem. Kromě čipu v Manchester kódu lze získat i provedení s PSK kódováním, kde data jsou přenášena na poloviční frekvenci s fázovým posunem. Rezonanční frekvence čipu je 125 kHz a přenosová rychlost se pohybuje kolem 50 kBd.

Čipy Unique se zapouzdřují do různých materiálů s rozmanitými tvary. Takto zapouzdřeným čipů se říká indetifikátor (tag, transpondér). Nejběžnějším transpondérem na našem trhu je bezkontaktní karta o velikosti platební karty dle ISO 7816-1. Karty mohou být alternativně vybaveny i magnetickým proužkem. Použitím tzv. e-unitu, což je identifikační čip s nabondovanou anténou, může zákazník realizovat vlastní mechanické provedení, které odpovídá nejlépe požadavkům jeho aplikace, např. designem, teplotní a chemickou odolností, způsobem upevňování na zboží a pod.. Dosah identifikačního čipu závisí hlavně na konstrukci vysílací antény, jejím tvaru a dále pak na velikosti elektromagnetického pole čtečky a její citlivosti. [6]

2.3.1 Vlastnosti

- 64 bitové číslo naprogramované laserem při výrobě
- Přesnost probíhá 64-bitovým kódováním Manchester
- K čipu je připojen kondenzátor pro resonanční obvod
- Pracovní frekvence 100 - 150 kHz
- Velmi malý čip pro umístění do pouzdra
- Napájení z elektromagnetického pole čtečky

2.3.2 Aplikace

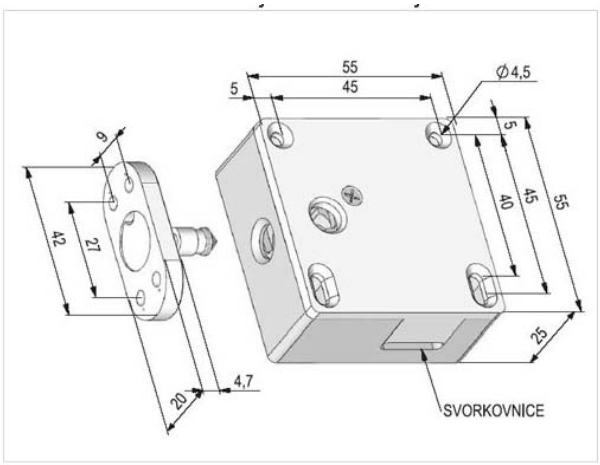
Tyto identifikátory nacházejí nejčastější využití při kontrole docházky (tzv. píchačky), v přístupových systémech, při automatické identifikaci materiálu nebo zboží na skladě, označování zakázek (čistírny, odvoz odpadu, opravny, oběh vratných obalů), ochraně zboží proti krádeží v obchodních domech, označování značkového zboží pro odlišení od padělku (parfémy, koňak, oděvy) a pod.. [6]

3. SKŘÍŇOVÝ ZÁMEK

Uzamykání dvířek je tvořeno elektrickým skříňovým zámkem firmy Bera. Jedná se o typ Skříňkový zámek s pružinou a signalizací SZEPS. Tento zámek se skládá ze dvou částí. Jedna s kolíkem připevněná na pohyblivou část a druhá samotný zámek. Ten v zavřeném stavu drží mechanicky kolík proti vysunutí. Otevření je provedeno elektrickým impulsem, který uvolní kolík a pružinka, integrovaná do zámku, ho vytlačí kousek ven. Tím je naznačeno i otevření zámku. Zavření zámku dojde okamžitě po opětovném zasunutí kolíku do zámku.

Dále pak zámek disponuje i indikací otevření zámku. Což je nezbytné pro naši úlohu řízení.

Schéma zámku včetně rozměrů je na Obrázek 3.1.



Obrázek 3.1: Schéma elektronického zámku

3.1 TECHNICKÉ PARAMETRY:

Rozměry zámku v25 x š55 x d55

Rozteč děr 45 x 45

Vruty 4x4 délka 30 -35, zapuštěný

Deska pro kolík š20 x v42

Rozteč děr 9 x 27

Vruty 4x4 délka 13 – 38, zapuštěný

Napětí 12V DC

Proudové zatížení 340 mA

Pracovní poloha svislá nebo vodorovná

4. ŘÍDICÍ JEDNOTKA

Jako řídicí jednotka byl vybrán mikrokontrolér firmy ATMEL AVR ATMega162.

4.1 ARCHITEKTURA AVR MIKROKONTROLERŮ

Jádro AVR-série se podobá jádru většiny RISC-procesorů, které jsou dostupné na trhu. AVR jádro se skládá ze 32 stejných 8-bitových registrů, které mohou obsahovat jak data tak adresy. Posledních 6 registrů můžeme ve dvojici použít jako ukazatele adresy pro nepřímé adresování paměti dat. Tyto registry označované písmeny X, Y a Z dovolují libovolné ukládací operace (Load/Store) viz.obr.č.1. Programátor má například na výběr, zda ukazatel adresy bude po zpracování určité istrukce inkrementovat nebo před zpracováním této instrukce dekrementovat. Užitečné je pro adresování využít možnosti 6-bitového posunu v ukazateli adresy v dvojitých registrech Y a Z. [3]

AVR architektura má 5 adresovacích módů pro paměť dat:

- přímé adresování
- nepřímé adresování s posunutím (6-bitový posun)
- nepřímé adresování
- nepřímé adresování s dekrementací ukazatele adresy před zpracováním instrukce
- nepřímé adresování s inkrementací ukazatele adresy po zpracovaní instrukce

4.1.1 Sériové rozhraní USART

AVR obsahuje plně duplexní univerzální asynchronní sériový kanál. Jednotka USART zvoleného procesoru má následující vlastnosti:

- dvě nezávislá rozhraní
- rámce o velikostech 5 až 9 bitů s 1 nebo 2 stopbity
- generování sudé a liché parity vyřešeno hardwarově
- detekce přečtení a chyby
- možnost vyvolat přerušení (vysílání ukončeno, vysílací zásobník prázdný, příjem ukončen)
- 4 režimy: asynchronní, asynchronní s dvojnásobnou rychlostí, synchronní master/slave

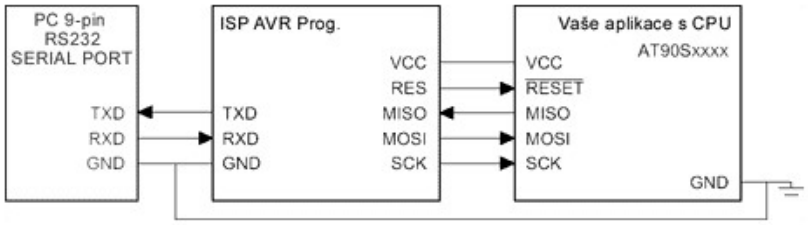
4.1.2 Systém přerušení

AVR má 19 zdrojů požadavků na přerušení. 2 vnější zdroje přerušení INT0, INT1, 7 zdrojů přerušení od časovače/čítače, 4 zdroje přerušení od UASRT, 1 zdroj od AD převodníku, 1 zdroj přerušení od EEPROM, 1 zdroj přerušení od analogového komparátoru a 1 zdroj od dvouvodičové sériové sběrnice. Priorita je pevně daná. [3]

5. PROGRAMÁTOR ŘÍDICÍ JEDNOTKY

Procesory ATMEL AVR a některé x51 kompatibilní procesory lze programovat nejenom klasicky v programátoru, pomocí paralelního programování vnitřní FLASH, ale i po speciálním rozhraní ISP (In System Programming), které používá 6-ti vodičové připojení procesoru k programátoru. ISP obsahuje napájecí napětí, ovládání pinu /RESET procesoru, vodiče pro sériový zápis (MOSI) a čtení (MISO) obsahu interních pamětí FLASH a EEPROM + jeden vodič pro synchronizaci přenosu dat (SCK).

Vzhledem k tomu, že připojení k programátoru zabere pouze tři vývody procesoru využitelné k jiným účelům, lze ve většině případů procesor programovat přímo v aplikační desce bez nutnosti vytahovat jej z patice a zasunovat do patice na desce programátoru. Odtud plyne i název protokolu – In System Programming. Díky definici rozhraní je možné napájet aplikační destičku z desky programátoru nebo naopak. [3]



Obrázek 5.1: Blokové zapojení ISP programátoru

V této úloze byla zvolena konstrukce programátoru vycházejícího z AN910_5 firmy ATMEL - základem je procesor AVR ATTiny2313, který vytváří převodník RS232 / ISP.

Napěťové úrovně RS232 jsou vstupními obvody upraveny na TTL a přivedeny na PD0 (2) vstup procesoru ATTiny2313, který programátor řídí. Výstupní úrovně jsou snímány z pinu PD1 (3) a přes vstupní obvody opět zkonvertovány na napěťové úrovně RS 232. Zde použité zapojení těchto napěťových úprav nedosahuje kvality zapojení s použitím MAX 232, ale v praxi plně dostačuje.

K řídícímu procesoru je připojen krystal 7,3728 MHz a na jeho výstupech PB4 až PB7 jsou už vyvedeny přímo vývody SPI, které se připojuje do programovaného procesoru. Oproti původnímu AN910 je doplněna červená dioda indikující napájecí napětí a dvoubarevná obousměrná dioda připojená na mezi piny PB3 a PB0 procesoru, která by měla sloužit pro indikaci stavů programátoru.

6. SCHÉMA ZAPOJENÍ

Tato část je věnována návrhu celkového zapojení mikrokontroléru, čtečky, konfiguračního počítače a programátoru. Kompletní schéma zapojení je umístěno v příloze 10.2 a případně zapojení ISP programátoru v příloze 10.1.

Na schématu je znázorněn zvolený procesor ATMega162 a k němu jsou připojeny: Čtečka karet AXA 012 na první USART jednotku, převodník úrovní MAX232 na druhou USART jednotku, LED diody na portc pro signalizaci stavu a případných chyb procesoru, na porta je připojen výstup na trezor a v poslední řadě je pak přes konektor AVR ISP připojen programátor.

7. ŘÍDICÍ PROGRAM

Tato část se zabývá popisem jednotlivých částí řídicího programu. Kompletní funkční výpis je znovu vypsán v příloze a je také obsažen na přiloženém CD.

V rámci této práce byl vytvořen program, který zajišťuje identifikaci, vyhodnocení, zpřístupnění trezoru, vydání jednotlivých klíčů a zaznamenání jednotlivých přístupů. V dalších krocích budou popsány jednotlivé části programu, k čemu slouží a jak pracují.

Tato část kódu má za úkol vypočítat BAUD rate rychlost na základě kmitočtu oscilátoru mikrokontroléru. Tyto musí být nastaveny na obou zařízeních, který spolu chtějí komunikovat.

```
#define F_OSC 8000000       // Kmitočet oscilátoru mikrokontroléru
#define BAUD 9600
#define MYUBRR 51           // (F_OSC/(16*BAUD))-1
```

Zde se definuje velikost datového bufferu, jehož princip bude popsán dále přímo u samotného použití. A dále pak je zde definice maximálního počtu karet, přístupových úrovní a délky čísla karet v bytech.

```
#define BUFFER_SIZE     16     // velikost modulo 2: jen 2,4,8,16,32 bajtu
#define BUFFER_MASK     (BUFFER_SIZE-1)
```

```
#define MaxPocetKaret       100
#define NumberOfLevels      3
#define NumberLenght        4
```

V této části kódu jsou obecné definice typů. Mezi nimi jsou například pole uložených karet, pole vyhodnocení, dále pak struktury pro práci s jednotkou USART a v neposlední řadě různá pomocná pole atd.

```
typedef enum{ FALSE=0, TRUE=1} Boolean_t ;
typedef unsigned char pom0 [5];
typedef unsigned char pom1 [5];

typedef unsigned char PoleKaret [NumberOfLevels][MaxPocetKaret][NumberLenght];
typedef unsigned char PocetKaret [NumberOfLevels];
typedef unsigned char PoleVyhodnoceni [NumberOfLevels][MaxPocetKaret][NumberLenght];


typedef struct{ unsigned char UartData0[ BUFFER_SIZE ];
        unsigned char NumberOfBytes0;
        unsigned char UartStatus0;
            unsigned char AccesLevel0 } Uart_t;

typedef struct{ unsigned char UartData1[ BUFFER_SIZE ];
        unsigned char NumberOfBytes1;
        unsigned char UartStatus1;
            unsigned char AccesLevel1 } Uart_t;
```

Zde jsou definice statických proměnných převážně pro práci s jednotkou USART a dále také pomocné proměnné.

```
static unsigned char            RxBuf0[ BUFFER_SIZE ];
static volatile unsigned char     RxProducer0;
static volatile unsigned char     RxConsumer0;
static unsigned char            TxBuf0[ BUFFER_SIZE ];
static volatile unsigned char     TxProducer0;  /
static volatile unsigned char     TxConsumer0;

static unsigned char            RxBuf1[ BUFFER_SIZE ];
static volatile unsigned char   RxProducer1;
static volatile unsigned char   RxConsumer1;
static unsigned char            TxBuf1[ BUFFER_SIZE ];
static volatile unsigned char   TxProducer1;
static volatile unsigned char   TxConsumer1;

//static volatile unsigned char      PocetKaret1;
//static volatile unsigned char      PocetKaret2;
//static volatile unsigned char      PocetKaret3;


volatile unsigned char znak;
volatile unsigned int receive_done;
volatile unsigned int i=1;

volatile unsigned char error=0x00;
```

Zde jsou definice prototypů funkcí. Ty se zavádějí kvůli přehlednosti a následnému snazšímu použití.

```
Boolean_t ReceiveData0( Uart_t0 * Data0 );
Boolean_t TransmitData0( Uart_t0 * Data0 );

Boolean_t ReceiveData1( Uart_t1 * Data1 );
Boolean_t TransmitData1( Uart_t1 * Data1 );

void Delay( void );
```

Zde už začínají jednotlivé funkce. Tato ma za úkol vytvořit zpoždění, které se následně použije v jiné funkci, když bude potřeba na určitý čas pozdržet vykonávaní programu.

```
void Delay( void )
{
 unsigned int i= 0x7FFF;
 while(--i);
}
```

Další funkce má za úkol inicializovat port C. Nastaví chování tohoto portu tak, aby se všechny jeho piny chovali jako výstupní. Následně se na něj pošle inicializační číslo 0xAA, tak aby bylo patrné, že k této inicializaci došlo.

```
void PORTC_Init ()
        {
        DDRC = 0xFF;
        PORTC = 0xAA;
        }
```

Následuje inicializace USART jednotek. Zde se pro obě jednotky nastaví baud rate nastavením příslušných registrů, povolí se příjímání i odesílání, nastaví se formát slova a nakonec dojde k vynulování indexu bufferu.

```
void USART_Init(int ubrr)
{
        // nastavi baud rate
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;

        UBRR1H = (unsigned char)(ubrr>>8);
        UBRR1L = (unsigned char)ubrr;

        // Povoli prijimani  a odesilani
        UCSR0B = 0x98;
        UCSR1B = 0x98;

        // Nastavi format slova na: 8 bit data, 1 stop bit
        UCSR0C = 0x86;
        UCSR1C = 0x86;

        // Vynulovani indexu bufferu
        RxConsumer0    = 0;
        RxProducer0    = 0;
        TxConsumer0    = 0;
        TxProducer0    = 0;
        RxConsumer1    = 0;
        RxProducer1    = 0;
        TxConsumer1    = 0;
        TxProducer1    = 0;
        }
```

Zde už jsou obslužné funkce pro obě jednotky USART. Vzhledem k tomu, že obě funkce jsou téměř identické, budou zde popsány dohromady. Konkrétně zde se zavádí funkce softwarového „bufferu“, jež byl už zmiňován.

V první části bude popsáno, jak probíhá příjímání dat pomocí jednotky USART. Přijdou-li na vstupní pin USART jednotky data a dojde k naplnění vstupního zásobníku, jehož velikost je 1 byte, vyšle tato jednotka požadavek na přerušení. Procesor na to zareaguje tak, že si uloží práci, kterou v tu chvíli dělal a začne zpracovávat přicházející data. Tyto data se v této chvíli uloží do softwarového bufferu, ve kterém budou čekat na další zpracovaní. Po tomto se procesor opět vrátí k předchozí práci. A až tuto práci dodělá, vrátí se zpět ke zpracování dat uložených v softwarovém bufferu.

```
SIGNAL(USART_RXC0_vect)
{
   unsigned char data;

   data = UDR;                          // čtení přijímaných dat
   RxBuf0[RxProducer0] = data;      // uloženi přijatého bytu do bufferu
   RxProducer0++;                       // inkrementovat index bufferu
   RxProducer0 &= BUFFER_MASK0;      // Index omezit popř. na začátek bufferu

         if (RxProducer0 == RxConsumer0 )
         {
            // Chyba! prebeh prijimaciho bufferu
         }
      }
```

```
// Prijimaci funkce UART
// -------------------------------------------------------
Boolean_t ReceiveData0( Uart_t0 * Data0 )
{
  unsigned char i=0;

  // Smycka k predani prijatych dat
  while (( RxConsumer0 != RxProducer0 ) && ( i < BUFFER_SIZE ) )
  {
    Data0->UartData0[i++] = RxBuf0[RxConsumer0++];

    RxConsumer0 &=BUFFER_MASK;     // omezit Consumer a ev. na
zacatek bufferu
  }

  Data0->NumberOfBytes0 = i ;

  if( i ) return TRUE ;
   else return FALSE ;
}
```

Druhá část je věnována vysílací funkcí. Pokud je potřeba vyslat nějaká data, zapíší se tyto do vysílacího bufferu. Toto vyvolá přerušení, jež bylo popsáno v předchozí části, a které ošetří tato vysílací funkce. Nakonec se zkontroluje, jestli byla všechna data odeslaná a program se vrátí ke své předchozí práci.

```
SIGNAL(USART_UDRE0_vect)
{
   // Test, zda vysilani neni hotovo
   if ( TxConsumer0 != TxProducer0 )
   {
    ++TxConsumer0;              // aktualizovat index bufferu
TxConsumer0 &= BUFFER_MASK;   // omezit index a ev. na zacatek bufferu
      UDR0 = TxBuf0[TxConsumer0];    // predani bajtu
    }
   else
   {
      UCSR0B &= ~UDRIE0;          // Zablokovani preruseni od UDRE
   }
}
```

```
// Vysilaci funkce
// -------------------------------------------------------
Boolean_t TransmitData0( Uart_t0 * Data0 )
{
   unsigned char i=0;

   while (( i < Data0->NumberOfBytes0 ) && ( i < BUFFER_SIZE ) )     //
predani prijimanych dat
   {
     TxBuf0[TxProducer0++] = Data0->UartData0[i++] ;

     TxProducer0 &= BUFFER_MASK;   // omezit Producer index a popr. na
zacatek bufferu
   }

   UDR0 = TxBuf0[TxConsumer0];        // spustit vysilani
   ++TxConsumer0;
   TxConsumer0 &= BUFFER_MASK;   // omezit Consumer index

   UCSRB0 |=  UDRIE0 ;
   Data0 -> NumberOfBytes0 = 0;
   return TRUE ;
}
```

Zde je obsluha příjímání a vysílaní pro druhou USART jednotku. Popis viz výše.

```
//  Recieve-Interrupt Handler
// ---------------------------------------------------------

SIGNAL(USART_RXC1_vect)
{
   unsigned char data;

   data = UDR1;                // cteni prijimanych dat
   RxBuf1[RxProducer1] = data;   // ulozeni prijateho bajtu do bufferu
   RxProducer1++;              // nastavit index bufferu
   RxProducer1 &= BUFFER_MASK;     // Index omezit popr. na zacatek bufferu

   if (RxProducer1 == RxConsumer1 )
   {
    error = 0x01         // Chyba! přeběh přijímacího bufferu
   }
}


// Prijimaci funkce UART
// ---------------------------------------------------------
Boolean_t ReceiveData1( Uart_t1 * Data1 )
{
   unsigned char i=0;

   // Smyčka k předáni přijatých dat
   while (( RxConsumer1 != RxProducer1 ) && ( i < BUFFER_SIZE ) )
   {
```

```
            Data1->UartData1[i++] = RxBuf1[RxConsumer1++];

            RxConsumer1 &= BUFFER_MASK;     // omezit Consumer a ev. na
zacatek bufferu
          }

          Data1 -> NumberOfBytes1 = i ;

          if( i ) return TRUE ;
           else return FALSE ;
        }


        //  Transmit-Interrupt Handler
        //  ------------------------------------------------------------

        SIGNAL(USART_UDRE1_vect)
        {
          // Test, zda vysilani neni hotovo
          if ( TxConsumer1 != TxProducer1 )
          {
            ++TxConsumer1;              // aktualizovat index bufferu
            TxConsumer1 &= BUFFER_MASK;    // omezit index a ev. na zacatek
bufferu
            UDR1 = TxBuf1[TxConsumer1];    // predani bajtu
           }
          else
          {
            UCSRB1 &= ~UDRIE1; // Zablokovani preruseni od UDRE
          }
        }
```

```
// Vysilaci funkce
// -------------------------------------------------------
Boolean_t TransmitData1( Uart_t1 * Data1 )
{
  unsigned char i=0;

  while (( i < Data1->NumberOfBytes1 ) && ( i < BUFFER_SIZE ) )     //
predani prijimanych dat
  {
    TxBuf1[TxProducer1++] = Data1->UartData1[i++] ;

    TxProducer1 &= BUFFER_MASK;   // omezit Producer index a popr. na
zacatek bufferu
  }

  UDR1 = TxBuf1[TxConsumer1];       // spustit vysilani
  ++TxConsumer1;
  TxConsumer1 &= BUFFER_MASK;   // omezit Consumer index

  UCSRB1 |=  UDRIE1 ;
  Data1->NumberOfBytes1 = 0;
  return TRUE ;
}
```

Zde už je samotná hlavní funkce, která má za úkol v daný okamžik spouštět funkce, které jsou popsány výše. Tato hlavní funkce se skládá z několika částí a každá tato část bude popsána samostatně.

První část je tzv. inicializační, ve které se nastaví mikrokontrolér a jeho periferie na výchozí hodnoty včetně kalibrace vnitřního oscilátoru mikrokontroléru, tak aby se minimalizovalo množství chyb při přenosech vlivem třeba i malé odchylky frekvence tohoto oscilátoru, ze kterého jsou odvozovány další parametry přenosů. A na konec vynulování všech pomocných proměnných.

```
void main( void )
{
   OSCCAL = 0xA2;            // Kalibrace vnitřního oscilátoru procesoru
   Uart_t UartData;

   PORTC_Init();

   USART_Init ( MYUBRR );     // Baudrate = 9600 při frekvenci krystalu 8MHz
  sei();        // Povoleni všech přerušení => Povolení přerušeni od  USART jednotky
  unsigned char j=0;
   unsigned char k=0;
   unsigned char l=0;

   for( ;; )               // Nekonečná smyčka
   {
```

Tato část funkce obstarává zpracování čísla karty přijatého jak z počítače tak ze čtečky karet. Tato funkce uloží přijaté číslo do jednorozměrného pole, a protože součástí přijatého čísla jsou i další hodnoty, které slouží k identifikaci o začátku a konci celého paketu, proto se tato část dat odfiltruje. Zbylé číslo se pak porovná s databází čísel uložených ve dvourozměrném poli, pokud bylo přijaté ze čtečky a případně se odešle na port počítače. Pokud však číslo bylo přijaté z počítače, dojde k jeho uložení do databáze i s přístupovou úrovní.

```
//-------- Zpracování čísla karty přijaté z PC ------------------------------------

    if( ReceiveData0( &UartData0 ) )
    {
      &UartData0 << 8;

      for (j=0;j=5;j++)
            &UartData0[j] -> pom0[j];

      NumberOfBytes0 == 5;

      if pom0[0]= 1
            {
            if PocetKaret[1] =< MaxPocetKaret
                  {
                  pom0 << 8;
                  NumberOfBytes0 == NumberOfBytes0 - 1;
                  PocetKaret[1]==PocetKaret[1]+1;

                  for (j=0; j=NumberOfBytes0; j++)
                  pom0[j] -> PoleKaret[1][PocetKaret[1]][j];
                  }
            else error=0x03;
```

```
        }

if pom0[0]= 2
        {
        if PocetKaret[2] =< MaxPocetKaret
                {
                pom0 << 8;
                NumberOfBytes0 == NumberOfBytes0 - 1;
                PocetKaret[2]=PocetKaret[2]+1;

                for (j=0; j=NumberOfBytes0; j++)
                pom0[j] -> PoleKaret[2][PocetKaret[2]][j];
                }
        else error=0x03;
        }

if pom0[0]= 3
        {
        if PocetKaret[3] =< MaxPocetKaret
                {
                pom0 << 8;
                NumberOfBytes0 == NumberOfBytes0 - 1;
                PocetKaret[3]=PocetKaret[3]+1;

                for (j=0; j=NumberOfBytes0; j++)
                pom0[j] -> PoleKaret[3][PocetKaret[3]][j];
                }
        else error=0x03;
        }

//TransmitData0( & UartData0 );     // Echo prijateho bajtu
```

```
if( ReceiveData1( &UartData1 ) )
{

&UartData1 << 8;

for (j=0;j=4;j++)
        &UartData1[j] -> pom[j];

NumberOfBytes1 == 4;

for (j=0; j=NumberOfLevels; j++)
        {
for (k=0; k=MaxPocetKaret; k++)
                        {
                        for (l=0; l=NumberLenght; l++)
                                {
                                if pom1[l] = PoleKaret[j][k][l]
                                        {
                                        PoleVyhodnoceni[l]==1;
                                        }
                                else PoleVyhodnoceni[j][k][l]==0;

        }
        repeatuntil
```

A v poslední řadě tu je část funkce, která cyklicky vypisuje stav proměnné error na PORTC, na kterém jsou připojené led diody. Tak aby, pokud dojde k nějaké chybě, bylo zřejmé, k jaké chybové události došlo.

```
PORTC = error;

Delay();
   }
}
```

7.1 APLIKACE PRO PLATFORMU MICROSOFT WINDOWS

Dalším úkolem bylo navrhnout aplikaci pro Microsoft Windows, která by měla za úkol vkládat čísla do databáze mikrokontroléru a nastavovat jejich přístupová práva.

Počítač, na kterém běží tato aplikace, je připojen přes sériový port COM1 k mikrokontroléru. Pokud mikrokontrolér přijme číslo ze čtečky, které není v databázi čísel, ho pošle po sériové lince počítači. Aplikace pak toto číslo zobrazí ve svém okně. Potom se tomuto číslu nastaví přístupová práva a tlačítkem „Send“ se odešle toto číslo včetně nastavení těchto práv zpátky do mikrokontroléru. Ten toto číslo vyhodnotí tak, že oddělí číslo karty od čísla reprezentující přístupová práva a podle těchto práv se uloží do příslušného pole databáze čísel v mikrokontroléru.

Tato aplikace je přiložena na CD na konci této práce.

8. ZÁVĚR

Úkolem této práce bylo navrhnout a realizovat řízení systému pro správu klíčů za použití RFID technologie. Práce zahrnovala vytvoření řídicího programu pro mikrokontrolér firmy ATMEL a vytvoření aplikace pro platformu Microsoft Windows, ve které se budou nastavovat přístupová práva. Dalším dílčím úkolem bylo sestavit ISP programátor se specifikací AVR910_5.

V této práci bylo navrženo zapojení mikrokontroléru, čtečky RFID karet, ISP programátoru a počítače. Schéma zapojení je v příloze na Obrázek 10.2.

Byl sestaven ISP programátor dle pokynů vedoucího, jehož schéma je v příloze Obrázek 10.1: Schéma zapojení programátoru AVR910_5. Jako řídicí jednotka byl zvolen mikrokontrolér ATMega162 firmy ATMEL. Tento mikrokontrolér je přímo spojen se čtečkou RFID karet AXA012 popsanou dříve. Pro připojení počítače k mikrokontroléru se musel navíc použít převodník úrovní MAX232 firmy Maxim. Programátor je připojen přes 10-ti pinový konektor a může být trvale součástí celého systému pro snadnější přístup k mikrokontroléru.

Dále byl vytvořen řídicí program pro mikrokontrolér, který po přijmutí čísla karty ze čtečky, toto číslo porovná s uloženou databází a na port pošle hodnotu, která reprezentuje, jaké klíče mohou být vydány. V případě, že číslo karty v databázi nenajde, pošle toto číslo druhou sériovou jednotkou do počítače.

Posledním bodem zadání bylo vytvoření aplikace pro platformu MS Windows, která bude nastavovat přístupová práva pro jednotlivá čísla karet. Tato aplikace je na CD přiloženého k této práci.

9. POUŽITÁ LITERATURA

[1] Stelzer, G.: Klein, aber leistungsfahig. Elektronik č.22 (1997), s.46 - 55.

[2] Zmrzlý, S.: Mikroprocesorová technika. Skripta VUT FEI Brno, PC-DIR Brno 1996.

[3] Firemní literatura: Katalogové listy firmy ATMEL www.atmel.com. 2007

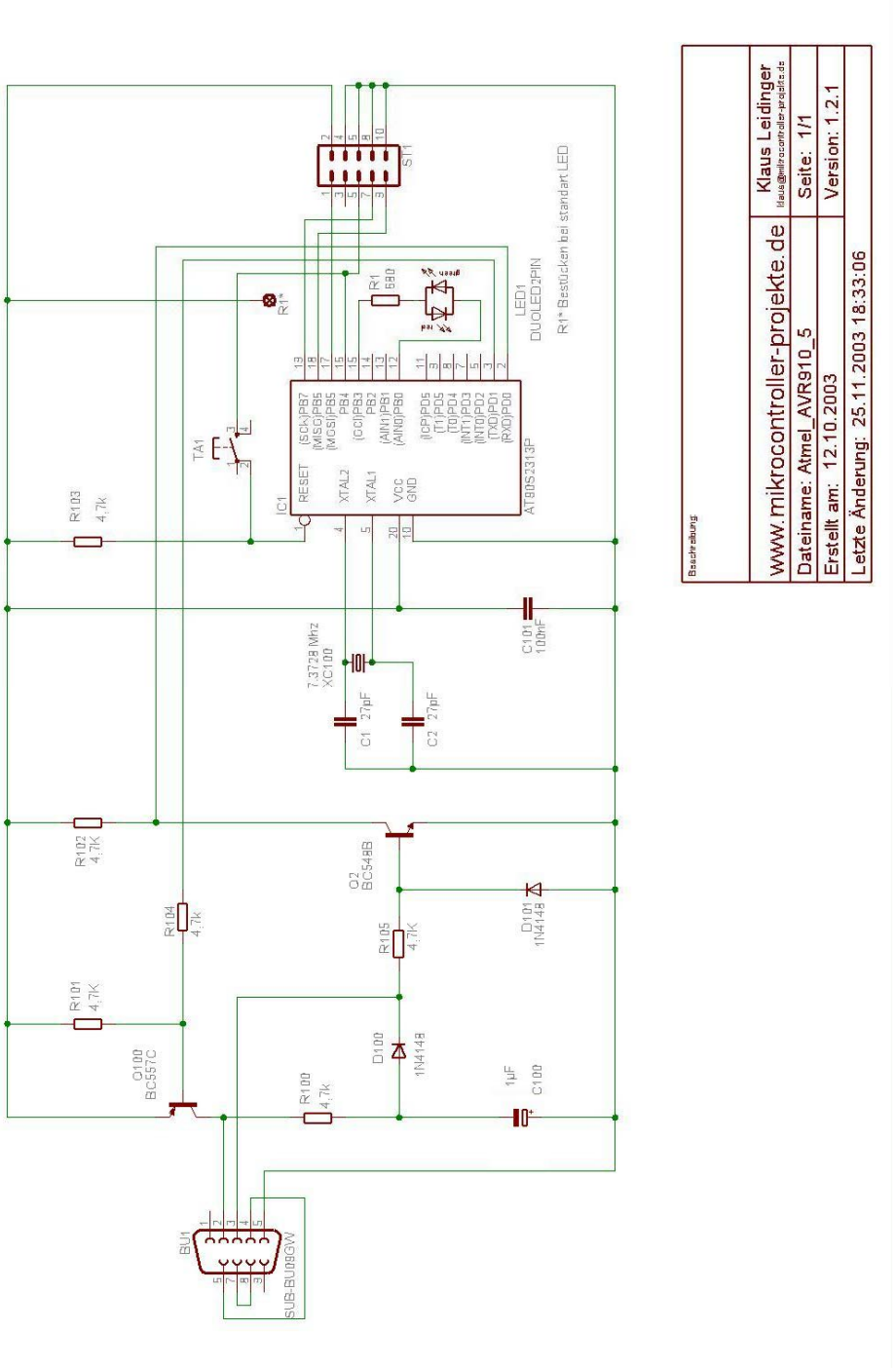
[4] Firemní literatura: Katalogové listy firmy Picomax www.picomax.com. 2007

[5] Firemní literatura: Katalogové listy firmy Maxim http://www.maxim-ic.com, 2007

[6] HW.CZ, Čtečka karet pro bezkontaktní identifikaci ID-12 Picomax, dostupné na http://hw.cz/Produkty/Obecne-produkty/ART878-Ctecka-karet-pro-bezkontaktni-identifikaci-ID-12-Picomax.html, 2007

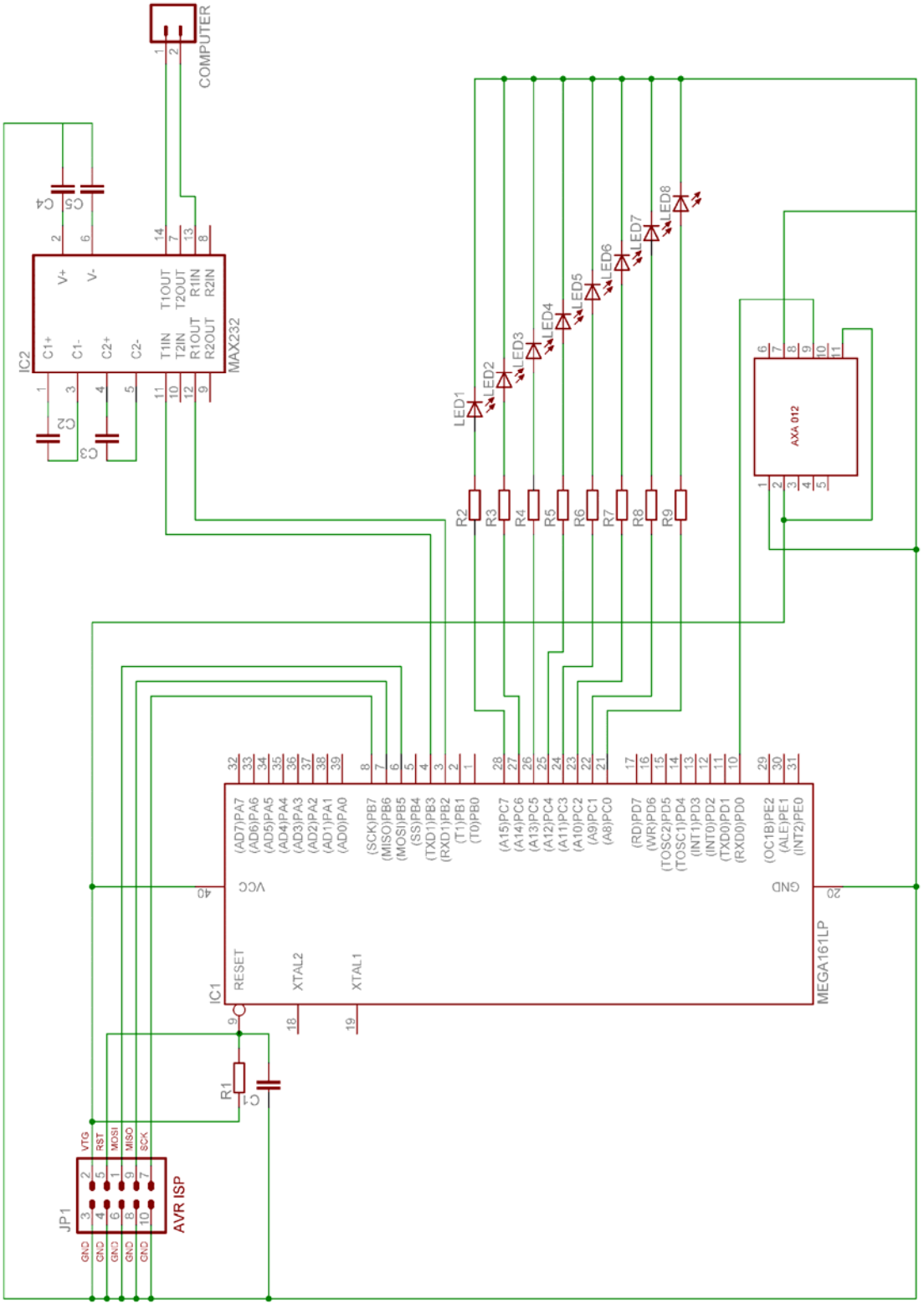
10. PŘÍLOHA

10.1 SCHÉMA ZAPOJENÍ PROGRAMÁTORU AVR910_5



Obrázek 10.1: Schéma zapojení programátoru AVR910_5

10.2 SCHÉMA ZAPOJENÍ MIKROKONTROLERU



Obrázek 10.2: Schéma zapojení mikrokontroleru

10.3 VÝPIS PROGRAMU

```
#include <io.h>
#include <iom162.h>
#include <signal.h>
#include <stdio.h>
#include <interrupt.h>

/*
// UCR - UART Control Register: definice bitu
// ------------------------------------------------------------
#define     RXCIE    0x80    // Bit 7
#define     TXCIE    0x40    // Bit 6
#define     UDRIE    0x20    // Bit 5
#define     RXEN     0x10    // Bit 4
#define     TXEN     0x08    // Bit 3
#define     UCSZ2    0x04    // Bit 2
#define     RXB8     0x02    // Bit 1
#define     TXB8     0x01    // Bit 0
*/

//Urceni BAUD rate
#define F_OSC 8000000                // Clock Speed
#define BAUD 9600
#define MYUBRR 51           // (F_OSC/(16*BAUD))-1



// Definice datoveho bufferu UART
//------------------------------------------------------------
#define BUFFER_SIZE     16     // velikost modulo 2: jen 2,4,8,16,32 bajtu
```

```
#define BUFFER_MASK     (BUFFER_SIZE-1)


#define MaxPocetKaret       100                     //Definice
maximalniho poctu ulozenych karet
#define NumberOfLevels      3                       //Definice     poctu
pristupovych urovni
#define NumberLenght        4                       //Definice     delky
cisla karty



// Obecne definice typu
//-------------------------------------------------------
typedef enum{ FALSE=0, TRUE=1} Boolean_t ;
typedef unsigned char pom0 [5];
typedef unsigned char pom1 [5];
typedef                 unsigned                char                PoleKaret
[NumberOfLevels][MaxPocetKaret][NumberLenght];
typedef unsigned char PocetKaret [NumberOfLevels];
typedef                 unsigned                char                 PoleVyhodnoceni
[NumberOfLevels][MaxPocetKaret][NumberLenght];              //Pole
vzhodnoceni, chtelo by to zavest boolean


typedef struct{ unsigned char UartData0[ BUFFER_SIZE ];
                unsigned char NumberOfBytes0;
                unsigned char UartStatus0;
                    unsigned char AccesLevel0 } Uart_t;

typedef struct{ unsigned char UartData1[ BUFFER_SIZE ];
```

```
                unsigned char NumberOfBytes1;
                unsigned char UartStatus1;
                    unsigned char AccesLevel1 } Uart_t;


        // Staticke promenne
        //-------------------------------------------------------
        static unsigned char            RxBuf0[ BUFFER_SIZE ];
        static volatile unsigned char       RxProducer0;  // Prijem: Int-Service je
vyrobce
        static volatile unsigned char     RxConsumer0;  // Prijem: Hlavni program je
konzument
        static unsigned char            TxBuf0[ BUFFER_SIZE ];
        static volatile unsigned char     TxProducer0;  // vysilani: Hlavni program je
vyrobce
        static volatile unsigned char       TxConsumer0;  // vysilani: Int-Service je
konzument

        static unsigned char            RxBuf1[ BUFFER_SIZE ];
        static volatile unsigned char       RxProducer1;  // Prijem: Int-Service je
vyrobce
        static volatile unsigned char     RxConsumer1;  // Prijem: Hlavni program je
konzument
        static unsigned char            TxBuf1[ BUFFER_SIZE ];
        static volatile unsigned char     TxProducer1;  // vysilani: Hlavni program je
vyrobce
        static volatile unsigned char       TxConsumer1;  // vysilani: Int-Service je
konzument

        //static volatile unsigned char     PocetKaret1;
```

```
//static volatile unsigned char       PocetKaret2;
//static volatile unsigned char       PocetKaret3;

/*
// Pomocne promenne
//-------------------------------------------------------
volatile unsigned char znak;
volatile unsigned int receive_done;
volatile unsigned int i=1;
*/

volatile unsigned char error=0x00;


// Prototypy funkci
//-------------------------------------------------------
//void InitUart( unsigned char baudrate );

Boolean_t ReceiveData0( Uart_t0 * Data0 );
Boolean_t TransmitData0( Uart_t0 * Data0 );

Boolean_t ReceiveData1( Uart_t1 * Data1 );
Boolean_t TransmitData1( Uart_t1 * Data1 );

void Delay( void );




void Delay( void )
```

```
{
 unsigned int i= 0x7FFF;
 while(--i);
}



// Inicializace portu C
//-------------------------------------------------------
void PORTC_Init ()
        {
        DDRC = 0xFF;
        PORTC = 0xAA;
        }



// Inicializace USART jednotky
//-------------------------------------------------------
void USART_Init(int ubrr)
{
        // nastavi baud rate
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;

        UBRR1H = (unsigned char)(ubrr>>8);
        UBRR1L = (unsigned char)ubrr;

        // Povoli prijimani  a odesilani
        UCSR0B = 0x98;
```

```
        UCSR1B = 0x98;

        // Nastavi format slova na: 8 bit data, 1 stop bit
        UCSR0C = 0x86;
        UCSR1C = 0x86;

        // Vynulovani indexu bufferu
        RxConsumer0    = 0;
        RxProducer0    = 0;
        TxConsumer0    = 0;
        TxProducer0    = 0;

        RxConsumer1    = 0;
        RxProducer1    = 0;
        TxConsumer1    = 0;
        TxProducer1    = 0;


    }

    //************************************************************
**********************************************************************
**********
    //
/
    //                                                  Obsluha    USART0
/
    //
/
```

```
//*************************************************************
*************************************************************
**************

//  Recieve-Interrupt Handler
//  -------------------------------------------------------

SIGNAL(USART_RXC0_vect)
{
   unsigned char data;

   data = UDR;              // cteni prijimanych dat
   RxBuf0[RxProducer0] = data;   // ulozeni prijateho bajtu do bufferu
   RxProducer0++;           // nastavit index bufferu
   RxProducer0 &= BUFFER_MASK0;    // Index omezit popr. na zacatek
bufferu

   if (RxProducer0 == RxConsumer0 )
   {
      // Chyba! prebeh prijimaciho bufferu
   }
}


// Prijimaci funkce UART
// -------------------------------------------------------
Boolean_t ReceiveData0( Uart_t0 * Data0 )
{
   unsigned char i=0;
```

```
    // Smycka k predani prijatych dat
    while (( RxConsumer0 != RxProducer0 ) && ( i < BUFFER_SIZE ) )
    {
      Data0->UartData0[i++] = RxBuf0[RxConsumer0++];

      RxConsumer0 &=BUFFER_MASK;      // omezit Consumer a ev. na
zacatek bufferu
    }

    Data0->NumberOfBytes0 = i ;

    if( i ) return TRUE ;
     else return FALSE ;
  }




  //  Transmit-Interrupt Handler
  //  ------------------------------------------------------------

  SIGNAL(USART_UDRE0_vect)
  {
    // Test, zda vysilani neni hotovo
    if ( TxConsumer0 != TxProducer0 )
    {
      ++TxConsumer0;              // aktualizovat index bufferu
      TxConsumer0 &= BUFFER_MASK;    // omezit index a ev. na zacatek
bufferu
      UDR0 = TxBuf0[TxConsumer0];    // predani bajtu
     }
```

```
    else
    {
      UCSR0B &= ~UDRIE0;          // Zablokovani preruseni od UDRE
    }
  }



  // Vysilaci funkce
  // -------------------------------------------------------
  Boolean_t TransmitData0( Uart_t0 * Data0 )
  {
    unsigned char i=0;

    while (( i < Data0->NumberOfBytes0 ) && ( i < BUFFER_SIZE ) )     //
predani prijimanych dat
    {
      TxBuf0[TxProducer0++] = Data0->UartData0[i++] ;

      TxProducer0 &= BUFFER_MASK;   // omezit Producer index a popr. na
zacatek bufferu
    }

    UDR0 = TxBuf0[TxConsumer0];       // spustit vysilani
    ++TxConsumer0;
    TxConsumer0 &= BUFFER_MASK;   // omezit Consumer index

    UCSRB0 |=  UDRIE0 ;
    Data0 -> NumberOfBytes0 = 0;
    return TRUE ;
  }
```

```
        //*************************************************************
************************************************************************
**************
        //
/
        //                                                  Obsluha    USART1
/
        //
/
        //*************************************************************
************************************************************************
**************


        //  Recieve-Interrupt Handler
        //  ------------------------------------------------------

        SIGNAL(USART_RXC1_vect)
        {
          unsigned char data;

          data = UDR1;              // cteni prijimanych dat
          RxBuf1[RxProducer1] = data;   // ulozeni prijateho bajtu do bufferu
          RxProducer1++;            // nastavit index bufferu
          RxProducer1 &= BUFFER_MASK;    // Index omezit popr. na zacatek
bufferu

          if (RxProducer1 == RxConsumer1 )
```

```
    {
      // Chyba! prebeh prijimaciho bufferu
    }
  }


  // Prijimaci funkce UART
  // -------------------------------------------------------
  Boolean_t ReceiveData1( Uart_t1 * Data1 )
  {
    unsigned char i=0;

    // Smycka k predani prijatych dat
    while (( RxConsumer1 != RxProducer1 ) && ( i < BUFFER_SIZE ) )
    {
      Data1->UartData1[i++] = RxBuf1[RxConsumer1++];

      RxConsumer1 &= BUFFER_MASK;     // omezit Consumer a ev. na
zacatek bufferu
    }

    Data1 -> NumberOfBytes1 = i ;

    if( i ) return TRUE ;
     else return FALSE ;
  }




  //  Transmit-Interrupt Handler
```

```
// ------------------------------------------------------

SIGNAL(USART_UDRE1_vect)
{
  // Test, zda vysilani neni hotovo
  if ( TxConsumer1 != TxProducer1 )
  {
    ++TxConsumer1;              // aktualizovat index bufferu
    TxConsumer1 &= BUFFER_MASK;   // omezit index a ev. na zacatek
bufferu
    UDR1 = TxBuf1[TxConsumer1];   // predani bajtu
   }
  else
  {
    UCSRB1 &= ~UDRIE1; // Zablokovani preruseni od UDRE
  }
}



// Vysilaci funkce
// ------------------------------------------------------
Boolean_t TransmitData1( Uart_t1 * Data1 )
{
  unsigned char i=0;

  while (( i < Data1->NumberOfBytes1 ) && ( i < BUFFER_SIZE ) )     //
predani prijimanych dat
  {
    TxBuf1[TxProducer1++] = Data1->UartData1[i++] ;
```

```
            TxProducer1 &= BUFFER_MASK;   // omezit Producer index a popr. na
zacatek bufferu
            }

            UDR1 = TxBuf1[TxConsumer1];       // spustit vysilani
            ++TxConsumer1;
            TxConsumer1 &= BUFFER_MASK;   // omezit Consumer index

            UCSRB1 |=  UDRIE1 ;
            Data1->NumberOfBytes1 = 0;
            return TRUE ;
        }


        //*************************************************************
************************************************************************
**************
        //
/
        //                                                    Hlavni  funkce
/
        //
/
        //*************************************************************
************************************************************************
**************



        void main( void )
        {
```

```
OSCCAL = 0xA2;            // Kalibrace vnitrniho oscilatoru procesoru
Uart_t UartData;

PORTC_Init();

USART_Init ( MYUBRR );      // Baudrate = 9600 pri frekvenci krystalu
8MHz
sei();                  // Povoleni vsech preruseni => Enable UART Interrupts
unsigned char j=0;
unsigned char k=0;
unsigned char l=0;

for( ;; )               // Nekonecna smycka
{



//-------- Zpracovani cisla karty prijate z PC ------------------------------------

   if( ReceiveData0( &UartData0 ) )
   {
      &UartData0 << 8;

      for (j=0;j=5;j++)
            &UartData0[j] -> pom0[j];

      NumberOfBytes0 == 5;

      if pom0[0]= 1
            {
            if PocetKaret[1] =< MaxPocetKaret
```

```
                {
                pom0 << 8;
                NumberOfBytes0 == NumberOfBytes0 - 1;
                PocetKaret[1]==PocetKaret[1]+1;

                for (j=0; j=NumberOfBytes0; j++)
                pom0[j] -> PoleKaret[1][PocetKaret[1]][j];
                }
        else error==0x03;
        }

if pom0[0]= 2
        {
        if PocetKaret[2] =< MaxPocetKaret
                {
                pom0 << 8;
                NumberOfBytes0 == NumberOfBytes0 - 1;
                PocetKaret[2]=PocetKaret[2]+1;

                for (j=0; j=NumberOfBytes0; j++)
                pom0[j] -> PoleKaret[2][PocetKaret[2]][j];
                }
        else error==0x03;
        }

if pom0[0]= 3
        {
        if PocetKaret[3] =< MaxPocetKaret
                {
                pom0 << 8;
                NumberOfBytes0 == NumberOfBytes0 - 1;
```

```
                PocetKaret[3]=PocetKaret[3]+1;

                for (j=0; j=NumberOfBytes0; j++)
                pom0[j] -> PoleKaret[3][PocetKaret[3]][j];
                }
        else error==0x03;
        }

//TransmitData0( & UartData0 );     // Echo prijateho bajtu
```

```
        //---------------------- Zpracovani cisla prijateho ze ctecky ----------------------
---------------
                }

                if( ReceiveData1( &UartData1 ) )
                {

                &UartData1 << 8;

                for (j=0;j=4;j++)
                        &UartData1[j] -> pom[j];
                NumberOfBytes1 == 4;
                for (j=0; j=NumberOfLevels; j++)
                        {
                        for (k=0; k=MaxPocetKaret; k++)
        //PoleKaret[NumberOfLevels][MaxPocetKaret][NumberLenght];
                                {
                                for (l=0; l=NumberLenght; l++)
                                        {
                                        if pom1[l] = PoleKaret[j][k][l]
                                                {
                                                PoleVyhodnoceni[l]==1;
                                                }
                                        else PoleVyhodnoceni[j][k][l]==0;
                                }
                repeatuntil
                PORTC = error;
                Delay();   */
           }
        }
```