VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

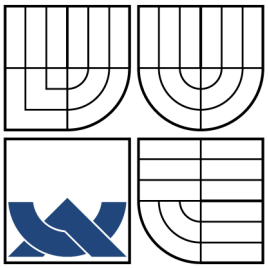
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIOELECTRONICS

# ZAŘÍZENÍ PRO MONITOROVÁNÍ SÉRIOVÉ KOMUNIKACE

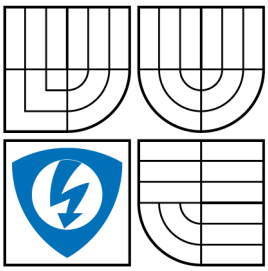
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Jan PERNÝ
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF RADIOELECTRONICS

# ZAŘÍZENÍ PRO MONITOROVÁNÍ SÉRIOVÉ KOMUNIKACE
SYSTEM FOR SERIAL COMMUNICATION MONITORING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Perný
AUTHOR

VEDOUCÍ PRÁCE — Doc. Jaromír Kolouch, CSc.
SUPERVISOR

BRNO 2008

## ABSTRAKT
Vývojáři, ať už hardwaru nebo softwaru, kteří musí řešit problémy při sériové komunikaci, potřebují nástroj k monitorování sériové komunikace. Tento nástroj musí přesně zachytit celou komunikaci. Pro pozdější analýzu je v záznamu nutný nějaký typ časových značek. Tato diplomová práce se zabývá programováním softwaru pro zařízení pro monitorování sériové komunikace s možností vkládání časových značek. Diskutovány jsou též možné typy časových značek a možnosti jejich vkládání.
Vytvořený software pro platformu FOX Board je schopen zachytávat a zaznamenávat komunikaci na portech RS232 a přidávat časové značky do záznamu podle třech nezávislých pravidel. Záznam je ukládán na běžný USB flash disk.

## KLÍČOVÁ SLOVA
monitorování, sériová komunikace, RS232, FOX Board, časová značka, záznam

## ABSTRACT
There is a need for a serial communication testing tool when software or hardware developers have to solve serial communication problems. This tool must accurately catch and record the whole communication. Some type of timestamps in the record is necessary for future analysis.
This diploma thesis deals with programming a serial communication monitoring equipment software with the possibility of the insertion of timestamps. Possible timestamps types and inserting methods are also discussed.
The programmed monitoring software for the FOX Board hardware platform is able to catch and record the serial communication at RS232 ports and add timestamps to the record in accordance to three independent rules. The record is saved to a standard USB flash drive.

## KEYWORDS
monitoring, serial communication, RS232, FOX Board, timestamps, record

# PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Zařízení pro monitorování sériové komunikace“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ..... .....

(podpis autora)

# PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Ing. Jaromíru Kolouchovi, CSc., Ing. Jindřichu Jeřábkovi, Ing. Jiřímu Gutmanovi a Ing. Jiřímu Křivánkovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.
V Brně dne 30. května 2008

.....
(podpis autora)

# BIBLIOGRAFICKÁ CITACE

PERNÝ, J. Zařízení pro monitorování sériové komunikace. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 55 s. Vedoucí diplomové práce doc. Ing. Jaromír Kolouch, CSc

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# ÚVOD

## Rozbor zadání

Úkolem této práce je vytvoření zařízení pro monitorování sériové komunikace a její záznam s vložením časových značek.

Zařízení má být schopno pořizovat záznam sériové komunikace z rozhraní RS-232 nebo RS-485 za použití vhodného převodníku. Maximální přenosová rychlost, se kterou má být schopno pracovat, byla stanovena na 115 kbit/s, ale pokusím se toto nebrat jako omezující faktor. Záznam má probíhat po dobu přibližně jednoho týdne.

Využití navrhovaného zařízení bude spočívat v servisní činnosti, kdy je třeba pořídit záznam komunikace různých přístrojů pro pozdější analýzu a nalezení případných chyb.

## Proč ne PC?

Zařízení pro monitorování sériové komunikace lze samozřejmě udělat i z obyčejného PC, nebo lépe z notebooku. Proč tedy vyvíjet něco jiného?

Počítač typu PC je charakterizován poměrně velkým příkonem, díky kterému musí být připojen do elektrické napájecí sítě, která však není všude dostupná. Ani notebook obsahující akumulátor nevydrží pracovat dostatečně dlouho. Napájecí šňůra nebo samotný počítač vzhledem ke své velikosti v případě použití přímo za provozu testovaného zařízení může překážet. Z těchto důvodu je nutné rozměrově malé zařízení s nízkým příkonem, napájené z baterií.

Dalším argumentem hovořícím proti klasickému počítači je nespolehlivost. Moderní operační systémy nejsou stavěny pro záznam dat v reálném čase, takže systém může na chvíli přestat reagovat. V takovém případě se může stát, že záznam buď nebude pořízen vůbec, nebo bude proveden špatně.

# 1 SÉRIOVÁ KOMUNIKACE

V sériovém přenosu dat jsou jednotlivé bity přenášeny stejným kanálem v časovém multiplexu. Paralelním přenosem dat je pak zpravidla nazýván takový typ přenosu, kdy jsou jednotlivé bity binárního slova přenášeny najednou v prostorovém multiplexu, tj. několika nezávislými kanály. Jednotlivá binární slova jsou však přenášena sériově.

Paralelního přenosu se užívá zpravidla na kratší vzdálenosti, kde je cena vodičů[1] ještě únosná. Výhodou je, že se přenáší celé binární slovo v jednom úseku časového multiplexu, a tak je vyšší přenosová rychlost proti sériovému přenosu. Sériový přenos je pak na delší vzdálenosti užíván z již zmíněných finančních důvodů a na krátké vzdálenosti v případě, že by vadilo velké množství vodičů z prostorového důvodu (například na plošném spoji).

Přenos dat může být synchronní, kde se užívá odděleného synchronizačního signálu, který taktuje časový multiplex – určuje, kdy jsou data platná. Je pro něj samozřejmě nutný další vodič, případně jeho zakódování do datového toku. Asynchronní přenos je rozdělen do sekvencí uvozených posloupností[2], podle které se přijímač vždy při jejím přijetí zasynchronizuje.

Synchronní přenos sice vyžaduje „drát navíc“, ale na přijímací straně již není nutný oscilátor, který by v rámci jedné sekvence určoval jednotlivé úseky časového multiplexu jako v případě přenosu asynchronního. Z existence oscilátoru u asynchronně pracujícího přijímače vyplývá, že vysílač nemůže měnit přenosovou rychlost například v závislosti na chybovosti. Synchronizační posloupnost také snižuje přenosovou rychlost.

Více informací o tomto tématu lze nalézt například v [2] a [1].

## 1.1 RS-232

Standard RS-232 byl normován u EIA[3] již v roce 1962 a jeho revize z roku 1969, známá jako RS-232C, platí dodnes. Později bylo rozhraní normováno znovu doporučením CCIT pod článkem V.24.

Co se týče napětí, rozhraní bylo definováno s nesymetrickými napěťovými úrovněmi podle tabulky 1.1.

Tab. 1.1: Napěťové úrovně RS-232C

| **Logická úroveň** | **Minimum** | **Maximum** | **Jednotka** |
|---|---|---|---|
| Logická 1 | −25 | −3 | V |
| Logická 0 | +3 | +25 | V |

Je nutno poznamenat, že zařízení je původně určeno pro komunikaci jednoho zařízení DTE (Data Terminal Equipment), například počítač, s jedním zařízením DCE

[1] Slovem vodič je myšleno jakékoliv propojení dvou zařízení, například i optické vlákno.
[2] Může to být i jediná hrana signálu.
[3] Electronic Industries Alliance

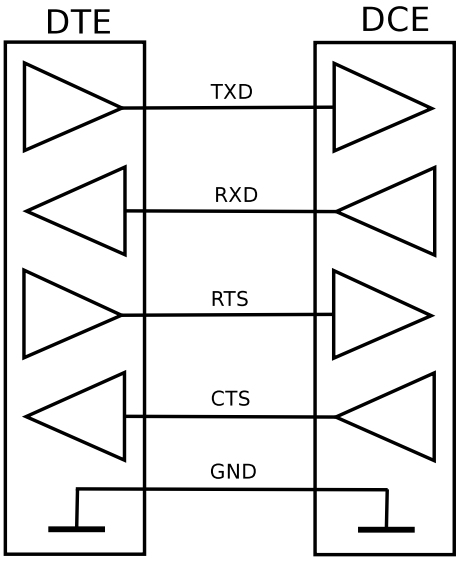
(Data Communications Equipment), například modem, které je připojeno k telefonní síti. Toto samozřejmě ovlivňuje používané signály uvedené v tabulce 1.2.

Tab. 1.2: Signály RS-232C

| **Signál** | **Význam** | **Použití** |
|---|---|---|
| **Datové signály** | | |
| RXD | Receive data | Vstup dat do zařízení DTE |
| TXD | Transmit data | Výstup dat ze zařízení DTE |
| **Stavové signály** | | |
| RTS | Request to send | DTE žádá o komunikaci |
| CTS | Clear to send | DCE potvrzuje, že je připraveno komunikovat |
| DSR | Data send ready | DCE je připraveno komunikovat |
| DTR | Data terminal ready | DTE je připraveno komunikovat |
| **Signály související s telefonní linkou** | | |
| RI | Ring indicator | Indikace vyzvánění |
| DCD | Data carrier detect | DCE oznamuje, že detekoval nosný kmitočet |
| **Ostatní signály** | | |
| SGND | Signal ground | Signálová zem |

Signály DSR a DTR se využívají, v případě že DCE data přeposílá a handshake probíhá i na vnější straně, k oznamovaní stavu vnější linky. Zdůrazněme též, že zařízení typu DCE na lince RXD vysílá a na lince TXD přijímá, takže není nutno linky „křížit“, pokud nepropojíme dvě zařízení stejného typu.

Stavové signály, které se původně používaly k řízení poloduplexní komunikace, se při plně duplexním provozu používají jen jako hlášení o (ne)připravenosti přijmout data, a to v různých kombinacích. Odtud také zřejmě plyne časté používání výrazu „Ready to send“ pro signál RTS.

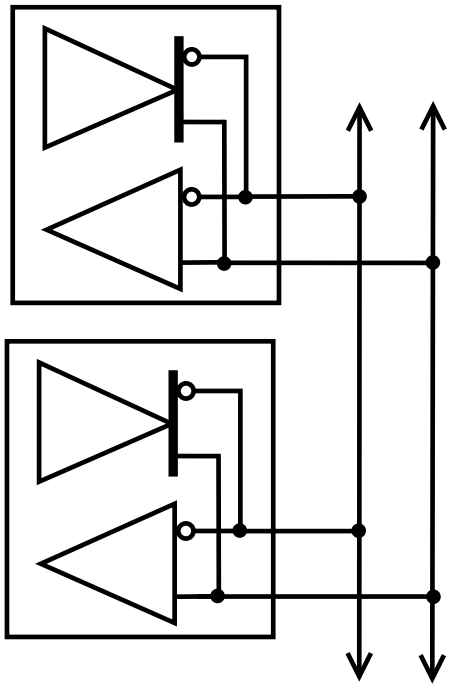



Obr. 1.1: Princip propojení dvou zařízení pomocí RS-232C

## 1.2 RS-485

Zatímco RS-232 používá nesymetrické úrovně napětí, RS-485 používá úrovní v symetrickém provedení. Na rozdíl od RS-232, které umožňovalo spojit dvě zařízení v duplexním provozu v nejjednodušším případě (bez handshaku) pomocí tří vodičů, je pomocí rozhraní RS-485 možno propojit[4] až 32 nezávislých zařízení jen pomocí dvou vodičů. Za tuto výhodu se platí nutností přepínání směru komunikace a jednoznačného určení, které z jednotlivých zařízení smí právě vysílat.

Obrázek 1.2 znázorňuje princip propojení dvou zařízení pomocí rozhraní RS-485, další by se k lince připojila stejným způsobem. Propojovací vodiče jsou zakončeny odporem 120Ω.



Obr. 1.2: Princip propojení několika zařízení pomocí RS-485

Velice pěkně je problematika rozhraní RS-232 i RS-485 popsána v [2], další informace lze nalézt též v [1].

[4]Při použití nízkoodběrových přijímačů až 256

# 2 ČASOVÉ ZNAČKY

Vložení časových značek do záznamu komunikace zvyšuje přehlednost záznamu a usnadňuje jeho pozdější analýzu, protože přidává další informaci. Už není známo jen pořadí, ale i časový odstup jednotlivých bloků dat.

Časovou značku lze vložit několika způsoby a který způsob bude výhodnější, záleží na typu zaznamenávané komunikace. Pojďme se podívat na jednotlivé způsoby. Z hlediska místa vložení značky je možno rozlišit

- vložení značky pouze na začátek,
- vkládání značky v pravidelných intervalech,
- vkládání značky na začátek bloku

a z hlediska formátu značky existuje

- absolutní značka,
- relativní značka a
- relativní značka s proměnnou přesností.

## 2.1 Vložení značky pouze na začátek

Pokud komunikace probíhá v pravidelných intervalech, stačí uložit záznam o času, kdy komunikace začala a ze znalosti intervalu lze dopočítat zbylé časové údaje. Výhodou tohoto způsobu je maximální úspora paměti. Naopak nevýhodou je malá univerzálnost a závislost na přesné znalosti komunikačního protokolu.

Značka je automaticky odlišena od dat, protože víme, že je na začátku a nemůže dojít k záměně. Tím, že je jen na začátku, ale vzniká problém při analýze, kdy je nutno všechny časové intervaly dopočítávat a v případě ztráty nebo poškození části záznamu dochází k znehodnocení všech následujících časových údajů.

## 2.2 Vkládání značky v pravidelných intervalech

Vkládání značky v pravidelných intervalech lze chápat v „časovém“ a „datovém“ pojetí.

Značku lze uložit vždy po určitém časovém intervalu. To má smysl jen tehdy, když komunikace neprobíhá v pravidelných intervalech, protože bychom ukládali zbytečně již známou, a tedy redundantní informaci. Pokud však komunikace neprobíhá pravidelně, ztrácíme informaci o časových poměrech uvnitř zvoleného intervalu ukládání. Mohou ale být aplikace, kde to nevadí.

Značku lze také ukládat po přijetí určitého počtu znaků (bytů). V případě pravidelnosti v komunikaci je toto plýtváním pamětí. V opačném případě dochází stejně jako u „časového“ pojetí, k časové nejistotě uvnitř intervalu.

Rozdíl mezi oběma způsoby je v nepřesnosti uvnitř ukládacího intervalu a ve využití paměti. Zatímco u „časového“ pojetí dojde k uložení časové značky i tehdy, když komunikace neprobíhá, a dochází tak k zbytečnému záznamu, u „datového“ pojetí se naopak může stát, že zvolený počet znaků bude přijat až po uplynutí velmi dlouhého intervalu, a časová nejistota v něm tak velice narůstá. „Časové“ pojetí tedy zajišťuje zvolenou maximální úroveň časové nejistoty a „datové“ pojetí zaručuje předem dané zvýšení nároků na paměť.

## 2.3 Vkládání značky na začátek bloku

V případě, že komunikaci lze rozdělit podle nějakého pravidla do logicky souvislých bloků, v nichž již není přesný čas jednotlivých částí bloku důležitý, je možno ukládat časovou značku jen na začátek každého bloku. Rozdělení do bloků většinou vyžaduje znalost komunikačního protokolu, a proto jej nelze užít vždy.

Za konec bloku lze považovat zvolený znak nebo kombinaci znaků [1], dovršení nějakého počtu znaků, změnu vysílajícího zařízení nebo delší (kratší) interval mezi znaky vysílanými v pravidelném intervalu. Ukončení bloku určitým znakem má nevýhodu v možnosti odlišnosti vyjádření konce u různých protokolů. Dovršení počtu znaků je velmi podobné „datovému“ pojetí ukládání časové značky v pravidelných intervalech, ale liší se v předpokladu, že komunikační protokol dělí komunikaci na bloky, v nichž už na přesném čase nezáleží. Změna vysílajícího zařízení je vetšinou indikována nějakým signálem, takže tento případ je velmi podobný případu s ukončením zvoleným znakem. Poslední možnost postihuje rozdělení na intervaly, kdy se vysílá a kdy je komunikační kanál nevyužit.

## 2.4 Absolutní značka

Pracovní název, který jsem zvolil, vychází z toho, že se ukládá absolutní čas. Znamená to uložení celého data od roku, přes dny a hodiny, až po sekundy tak, že vyjadřují skutečný čas události. Ukládání takového množství číselných údajů je jistě zbytečně náročné na paměť, zvlášť když se komunikuje často a číslo vyjadřující například roky je neměnné.

Nabízí se tedy ukládat zkrácenou absolutní značku. Na začátek záznamu je nutno vždy uložit plnou absolutní značku kvůli jednoznačnosti a dále stačí ukládat jen dopředu zvolené části – například hodiny, minuty a sekundy, nebo jen tu část značky, která se změnila od posledního záznamu. Ukládání zvolené části trpí stejným nedostatkem jako ukládání plné značky, protože se opět může ukládat několik čísel, ze kterých se mění jen jedno. Naopak ukládání pouze změněných čísel nutně vyžaduje uložení informace o tom, co vlastně značka vyjadřuje.

[1] Například carriage return (CR), line feed (LF) a další

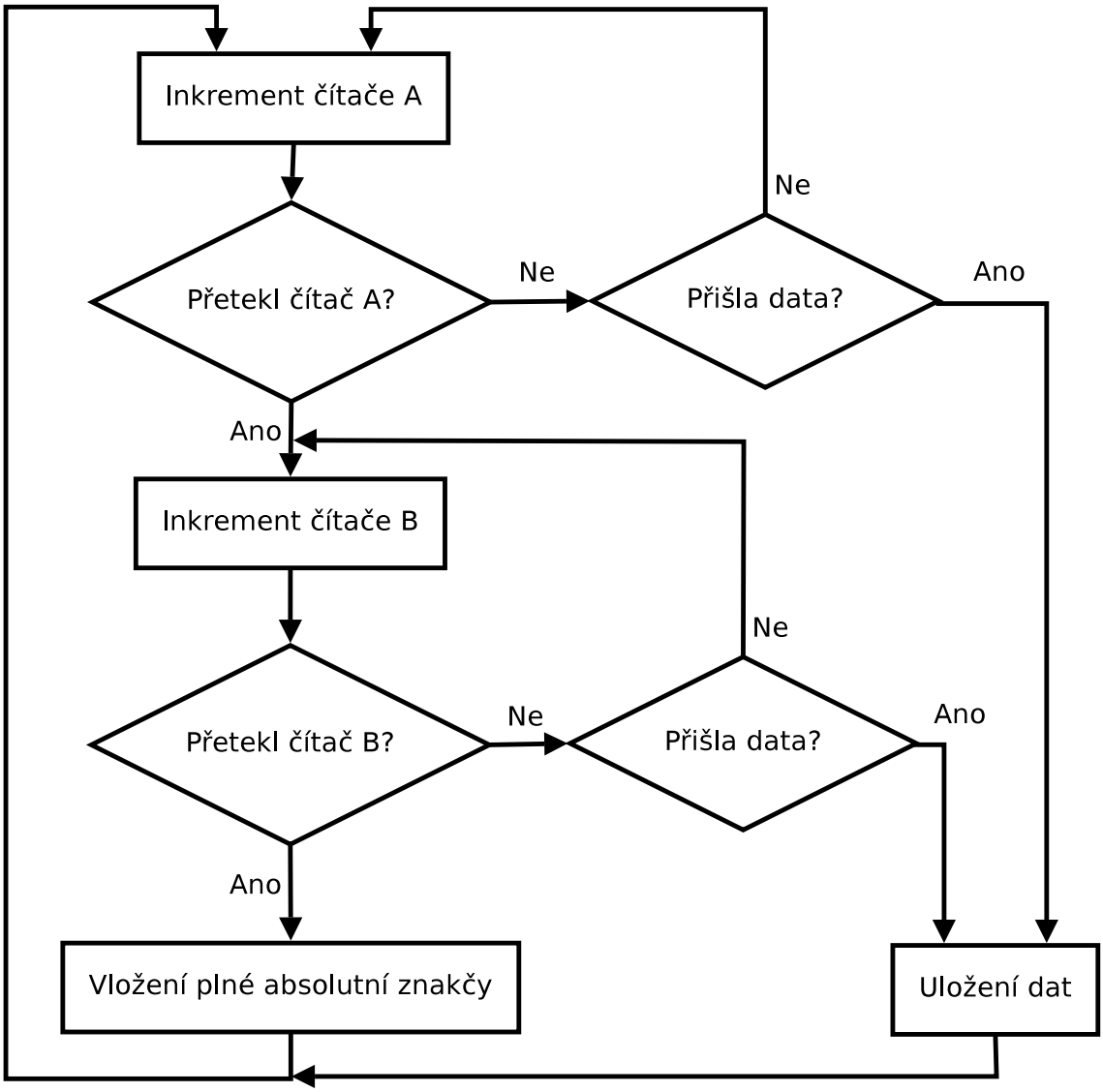
## 2.5 Relativní značka

Nabízí se i možnost ukládat čas relativně vzhledem k poslednímu záznamu, jak bylo naznačeno u zkrácené absolutní značky. Na začátek záznamu je opět nutno uložit plnou absolutní značku a dále se zaznamenává počet vhodně zvolených period od poslední zaznamenané komunikace. Protože má postihnout kratší interval, je vcelku pravděpodobné, že toto číslo bude malé a stačí k jeho vyjádření menší počet bitů, takže se ušetří paměť.

Může se ale stát, že komunikace bude probíhat s takovými přestávkami, že čítač obsahující počet period přeteče. Tento případ lze ošetřit zápisem absolutní značky a vynulováním čítače. Protože však žádná data nepřišla (jinak by se při záznamu čítač vynuloval, a nemohl tak přetéct), bude v záznamu jen samotná absolutní značka bez dat.

## 2.6 Relativní značka s proměnnou přesností

Možnost, jak elegantně vyřešit přetečení čítače period u relativní značky, nabízí proměnlivá přesnost časového záznamu. Jakmile přeteče čítač period, označme si ho pro snadnou orientaci $A$, začne čítat druhý čítač, označme si ho $B$, s mnohem větší periodou. Obsah čítače $A$ se zahodí a když se začne po odmlce opět komunikovat, zaznamená se obsah čítače $B$ s poznámkou, kterého čítače se číslo týká. Opět začne čítat čítač $A$ a pokud nepřeteče, používají se jeho hodnoty.



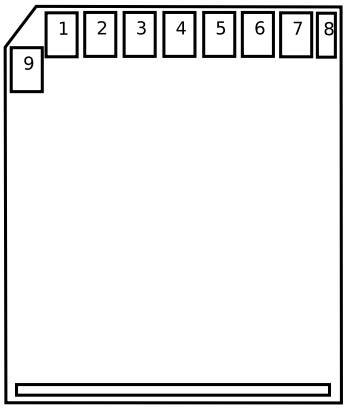

Obr. 2.1: Značka s proměnnou přesností – princip

# 3 PAMĚŤOVÉ KARTY SECURE DIGITAL

Paměťové karty se staly všeobecně rozšířeným médiem pro záznam dat a zvláště se uplatňují v oblasti digitální fotografie. V současné době je zřejmě nepoužívanějším a nejdostupnějším typ SD (Secure Digital) vyvinutý v letech 1999 až 2000 firmami Panasonic, SanDisc a Toshiba.

Standard je zpětně kompatibilní s kartou typu MMC (MultiMedia Card). Zatímco MMC je „otevřeným standardem“, specifikace SD byla zveřejněna teprve v dubnu 2006 (jako [9] případně též i [8]).

Rozměry obou typů karet jsou $24\text{mm} \times 32\text{mm} \times 1,4\text{mm}$. Komunikace může probíhat pomocí SPI (Serial Peripheral Interface), kdy hostitelský systém představuje zařízení typu master a paměťová karta zařízení typu slave, nebo v režimu „SD Card Interface“. V režimu SPI bude karta v mé práci používána, a proto se nadále budu zabývat pouze tímto režimem.




Obr. 3.1: Náčrt SD karty

Na obrázku 3.1 je náčrt SD karty z pohledu na stranu s přípojnými ploškami. Karta typu MMC se od karty typu SD liší tím, že nemá vývody číslo 8 a 9.

## 3.1 Připojení karty

Tabulka 3.1 udává názvy jednotlivých vývodů a ukazuje, jak připojit kartu k mikroprocesoru. Jestliže se funkce vývodů v režimu SPI liší od funkce v režimu SD Card Intreface, je za lomítkem uvedena funkce vývodu v režimu SPI. Kromě specifikací [9] a [8] jsem čerpal také z [7] a [6].

Komunikace probíhá pomocí takzvaných tokenů. Nejjednodušším z nich je Idle, který nenese žádnou informaci a je vyjádřen úrovní H po osm hodinových taktů. V případě, že běží hodinový signál, ale nejsou žádná data ani příkazy k přenesení, přenáší se Idle. Například se vyžaduje po každém příkazu osm taktů hodin, aby mohla karta zpracovat předchozí požadavek. Protože v této době musí SPI hostitelského systému generovat hodinový signál, ale nesmí vysílat další požadavky, vysílá Idle.

Speciálním typem tokenů jsou příkazy. Jde o posloupnost šesti bytů. První byte začíná úvodními bity 0,I a za nimi je vloženo 6 bitů reprezentujících vlastní příkaz.

Tab. 3.1: Vývody SD karty a jejich připojení k SPI

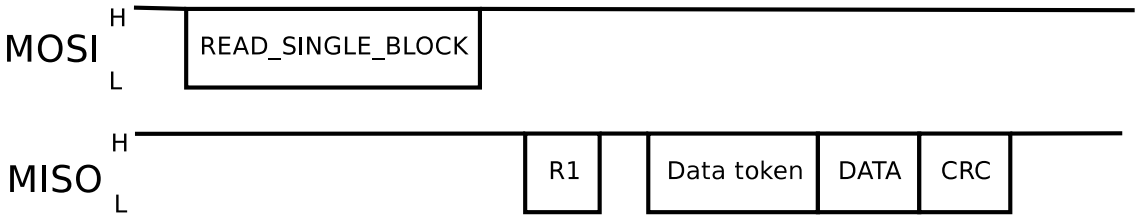
| Vývod | Označení | Připojení |
|---|---|---|
| 1 | DAT3/CS | SPI CS (Chip Select) vývod |
| 2 | CMD/DI | SPI MOSI vývod (Master Out, Slave In) |
| 3 | VSS | Zem |
| 4 | VDD | Napájecí napětí $+3,3$V |
| 5 | CLK | SPI vývod hodinového signálu |
| 6 | VSS | Zem |
| 7 | DAT0/DO | SPI MISO vývod (Master In, Slave Out) |
| 8 | DAT1 | Nepřipojeno |
| 9 | DAT2 | Nepřipojeno |

Následuje pět bytů argumentu a šestý byte. Ten obsahuje sedm bitů CRC-7 součtu a LSB je vždy nastaven do logické 1. Kontrola CRC je v režimu SPI pouze volitelná.

Karta na příkazy odpovídá nejčastěji pomocí odpovědi R1, což je byte začínající logickou 0 následovanou příznakovými bity. Odpovědi R2 a R3 se liší pouze délkou – R2 obsahuje 15 příznakových bitů a R3 je vlastně R1 následovaná obsahem OCR registru.

## 3.2 Čtení a zápis

Data token je odvysílán bezprostředně před přenášeným blokem dat o dopředu známé délce. Jestliže je například karta žádána o blok dat, vyšle hostitelský systém příkaz READ_SINGLE_BLOCK a následuje vyslání Idle, kdy karta zpracovává příkaz. Systém pokračuje ve vysílání Idle, dokud od karty nedostane odpověď (v tomto případě R1). Jestliže v odpovědi není signalizována chyba, vysílá opět Idle do doby, kdy přijme Data token. Po jeho přijmutí odvysílá Idle tokeny v počtu rovném délce bloku dat[1] a zároveň přijímá data. Po skončení příjmu opět odvysílá Idle, aby karta mohla případně dokončit interní operace.



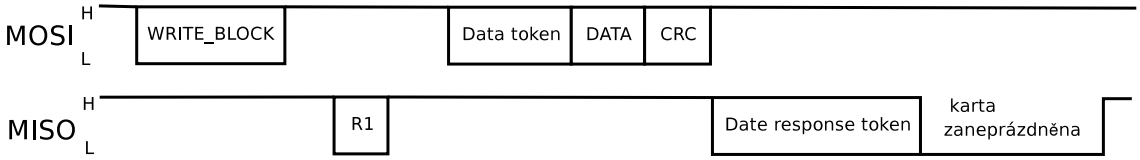

Obr. 3.2: Čtení dat z karty (jednoblokové)

V případě chyby karta místo Data tokenu odpovídá pomocí Data Error tokenu. Jde o byte sestávající se ze tří logických 0 následovaných příznakovými bity.

Při zápisu se pokračuje ve vysílání Idle, dokud karta odpovídá pomocí Data response tokenu. Jde o byte, jehož tři první bity nemají předepsanou hodnotu, další

[1] Nutno připočíst dva byty obsahující CRC-16 součet bloku dat.

je vždy logická 0, následují další tři bity obsahující informaci, zda data byla převzata a poslední je vždy v logické 1. Tento byte je následován byty obsahujícími samé logické 0 do doby, kdy karta skončí zápis dat do paměti flash. V této době karta samozřejmě nepřijímá žádná data nebo příkazy.




Obr. 3.3: Zápis dat na kartu (jednoblokový)

## 3.3 Inicializace karty

Karta je aktivována, když je CS v úrovni L. Jestliže je vložena do systému, mělo by před prvním použitím podle [6] předcházet nejméně 74 taktů hodinového signálu. Předstih přechodu signálu CS do aktivní úrovně před první aktivní hranou hodinového signálu [2] se pohybuje v řádu jednotek až desítek nanosekund. Přesně to lze zjistit čtením CSD registru.

Karta je nyní stále v režimu SD Card Interface a je třeba jí přepnout do režimu SPI. Toho lze dosáhnout odesláním příkazu GO_IDLE_STATE. Protože není karta v režimu SPI, je nutné, aby byl CRC součet správný. Příkaz GO_IDLE_STATE však nemá žádné parametry[3], a tak není nutné součet počítat[4]. Po odeslání příkazu se čeká na odpověď R1 (0x01 = Idle state).

Nyní je třeba kartu nechat provést interní inicializaci. Toho se dosáhne opakovaným posíláním příkazu SEND_OP_COND do té doby, než karta odpoví R1, ale tentokrát musí být nulové všechny příznaky (0x00).

Nyní je karta inicializována a připravena k práci. Minimálně je ale nutno přečíst ještě obsah registru CSD (Card Specific Data). Toho se dosáhne odesláním příkazu SEND_CSD. Karta musí odpověďět R1 (0x00), a poté následuje Data token (0xFE) a blok šestnácti bytů dat z registru CSD. Registr CSD obsahuje mimo jiné údaj o velikosti bloku pro zápis a velikosti bloku pro čtení o požadovaném předstihu CS před aktivní hranou hodinového signálu, zpožděních, proudovém odběru při čtení, proudovém odběru při zápisu nebo například bity zabraňující zápisu na kartu. Podrobný popis struktury registru CSD je dostupný v [7], proto budou dále zdůrazněny jen některé důležité části.

Velikost bloku pro čtení (označme například $BR$) je dána parametrem READ_BL_LEN, který je obsažen v bytu 5 (číslování začíná 0) v dolních čtyřech bitech, podle vztahu

$$BR = 2^{READ\_BL\_LEN}. \tag{3.1}$$

[2] Vzestupná hrana

[3] Data jsou zahazována.

[4] stačí odvysílat 0x40,0x00,0x00,0x00,0x00,0x95

Dalším parametrem je C_SIZE, který je fyzicky rozdělen do několika částí. Horní dva bity jsou uloženy v nejnižších dvou bitech bytu 6. Následujících 8 bitů je uloženo v bytu 7 a poslední, nejnižší, dva bity jsou uloženy v nejvyšších dvou bitech bytu 8. Nejnižší dva bity bytu 9 obsahují nejvyšší dva bity parametru C_SIZE_MULT a nejvyšší bit 10. bytu obsahuje nejnižší bit tohoto parametru. Z parametrů lze spočítat kapacitu karty jako

$$C = \left[(C\_SIZE + 1) \cdot 2^{(C\_SIZE\_MULT+2)}\right] \cdot BR. \tag{3.2}$$

# 4 COLDFIRE MCF5213

## 4.1 Coldfire

ColdFire jsou dvaatřicetibitové mikroprocesory s redukovanou instrukční sadou (RISC), určené pro aplikaci v sítí propojených řídících, medicinských a bezpečnostních systémech, v automatizaci v domácnosti nebo v průmyslu a jiných zařízeních. Stručný přehled vlastností a doporučené použití jednotlivých členů rodiny je možno nalézt v [5].

## 4.2 Výběr procesoru

Prvním kritériem výběru vhodného mikroprocesoru byly integrované periferie a výkon procesoru. Téměř všechny členy rodiny ColdFire obsahují alespoň jeden UART[1], který případně podporuje i DMA[2].

Pokusme se o hrubý odhad pro posouzení výkonu. Výkon nejslabšího z procesorů dosahuje 50 MIPS[3]. Má-li procesor zpracovávat data, musí být schopen vykonat dostatečné množství instrukcí tak, aby výsledek zpracování byl hotov před tím, než dostane další data ke zpracování. V případě, že by se data zpracovávala po jednotlivých bitech, může procesor s uvedeným výkonem při rychlosti 115 kbit/s vykonat více než 400 instrukcí, než bude muset zpracovávat další data. Vidíme, že výkon je dostačující.

Kritickým parametrem se stala dostupnost vývojových nástrojů. Pro vývoj lze použít prostředí Code Warrior od firmy Freescale, které je k dispozici ve třech verzích

- Standard Edition,
- Professional Edition a
- Special Edition.

Zatímco Standard a Professional Edition jsou placené, Special Edition je dodávána zdarma k vývojovým kitům a je také ke stažení na internetu. Je logické, že tato verze má jistá omezení. Prvním z nich je maximální velikost kódu, který lze naprogramovat do paměti flash, a to 131072 bytů. Druhé důležité omezení vyplývá z použití s vývojovým kitem – tato verze neumožňuje použití simulátoru. Odladění programu je tak nutno provést přímo na vývojovém kitu pomocí integrovaného hardwarového rozhraní.

Nakonec jsem zvolil procesor MCF5213, který obsahuje třikrát UART a je pro něho k dispozici vývojový kit.

[1] Universal Asynchronous/synchronous Receiver Transmitter
[2] Direct Memory Access
[3] Million Instructios Per Second

## 4.3 Softwarový pohled na MCF5213

Vzhledem k tomu, že k vývoji používám programovací jazyk C, softwarový pohled na mikroprocesor se mi velmi zjednoduší, a proto ho předkládám i zde ve zjednodušené formě. Velice podrobně je problematika probrána v [3] a zbytek lze nalézt v katalogovém listu [4].

Procesor obsahuje osm datových a osm adresních registrů pro bitové, bytové (8 bitů), 16-bitové a 32-bitové operace. Z toho vyplývá, že v C lze použít proměnné typu int8, int16 a int32 bez obav, že překladač vytvoří složitý a pomalý kód.

V instrukční sadě jsou obsaženy instrukce pro násobení a dělení jak $16 \times 16 \rightarrow 32$, tak i $32 \times 32 \rightarrow 32$, ať už se znaménkem, nebo bez. Bez obav tak lze v C použít násobení a dělení i jinými čísly než 2. Tyto instrukce neprovádí jednotka MAC [4], ale ALU.

Paměť programu je typu flash o velikosti 256kB a paměť dat programu typu SRAM je 32kB. O veškerou práci s nimi se stará překladač jazyka C.

## 4.4 Integrované periferie

Procesor má integrováno velké množství periferií, z nichž uvádím jen ty, které jsem použil nebo nějak ovlivnily moji práci.

### 4.4.1 GPIO

Procesor obsahuje jedenáct vstupně–výstupních bran pro všeobecné použití, každou po čtyřech až osmi vývodech. Jednotlivé vývody mohou mít nakonfigurovánu funkci vstupně-výstupní (GPIO), nebo mohou být přiřazeny některé z periferií (PRIMARY, ALTERNATE1 a ALTERNATE2 function). Při použití některé z periferií je třeba nastavit i příslušnou funkci vývodu v modulu GPIO.

### 4.4.2 UART

UART je periferií, která poskytuje hardwarovou podporu pro sériovou komunikaci s handshakem. Procesor MCF5213 obsahuje tři nezávislé UARTy, které mohou pracovat jak v asynchronním, tak i synchronním režimu. Hodinový signál může být buď přivedený externí, nebo interní, odvozený od taktovacího kmitočtu procesoru. V případě interního zdroje hodinového signálu platí vztah

$$Baudrate = \frac{f_{sys}}{32 \cdot divider}, \tag{4.1}$$

kde *Baudrate* je požadovaná přenosová rychlost, $f_{sys}$ je interní frekvence procesoru závislá na frekvenci oscilátoru a nastavení smyčky PLL a *divider* je číslo, které je nutno vložit do registrů UBG1 (horních 8 bitů) a UBG2 (dolních 8 bitů). Šestnáctibitovou předděličku lze použít i v případě vnějšího hodinového signálu. Samozřejmostí je nastavení počtu datových a stop bitů, automatické vkládání paritního bitu (sudá, lichá, žádná parita, nebo nastavení či nulování bitu) a jeho kontrola při příjmu.

[4] Multiply–Accumulate unit

Každý UART má k dispozici kromě posuvného registru pro příjem dat ještě trojitý FIFO zásobník, takže obslužný program má dosti času, aby přijatá data vybral. Pokud to nestihne, data jsou zahozena a je indikován stav overrun. Vysílač UARTu je vybaven pouze jednobytovou frontou, což znamená, že zatímco se vysílá jeden znak uložený v posuvném registru, může mikroprocesor připravit do fronty další.

Stavy front lze buď periodicky testovat čtením stavových bitů, nebo může naplnění, respektive vyprázdnění fronty generovat přerušení. U přijímače je možno zvolit, zda přerušení bude generovat přijetí každého jednoho znaku, nebo naplnění celého zásobníku. Přijatá data jsou vybavována signalizačními bity (parity, framing a overrun error) v případě, že nesouhlasí parita, nastavený počet bitů nebo přetekl zásobník.

Přijímač i vysílač mohou reagovat na handshakové signály nebo je generovat. Vzhledem k široké nastavitelnosti je pro konkrétní případ nutné prostudovat manuál [4].

### 4.4.3 PIT

Programmable Interrupt Timer je časovač, jak je z anglického názvu patrné, který vytváří požadavek na přerušení v nastavitelných časových intervalech. Jde o šestnáctibitový čítač dolů s předděličkou s nastavitelným rozsahem $2^0$ až $2^{15}$, jehož vstupem je vnitřní hodinový signál procesoru. Čítač může být volně bežící, kdy po dosažení hodnoty 0x0000 čítání pokračuje od hodnoty 0xFFFF, nebo může být interval hardwarově zkrácen, kdy je po načítání hodnoty 0x0000 do čítače přepsán obsah funkčního registru PMR.

Ve chvíli kdy čítač načítá do hodnoty 0x0000, je generován požadavek na přerušení. Obsah čítače je také v programu dostupný jako obsah registru PCNTR, který je doporučeno číst vcelku jako šestnáct bitů, aby bylo zaručeno, že se během čtení nezmění jeho obsah.

### 4.4.4 QSPI

Queued Serial Peripheral Interface je obousměrná synchronní sériová brána. Periferie je vybavena pamětí SRAM o velikosti 48 slov (po šestnácti bitech), která je programu přístupná pomocí adresovacího a datového registru. Tato paměť je rozdělena na tři části po šestnácti slovech. Jedna slouží pro uložení přijatých dat z kanálu MISO[5], ve druhé části jsou připravena data k odvysílání na kanál MOSI a třetí část obsahuje příkazy k ovládání čtyř nezávislých signálů chip select a konfiguraci chování QSPI. Délku vysílané zprávy odpovídající jedné pozici ve frontě lze nastavit od osmi do šestnácti bitů.

Jestliže probíhá komunikace, je přečten z paměti příkaz, který nastaví příslušné chip selecty do správné úrovně, data jsou odvysílána a zároveň jsou uložena ta přijatá. Při použití externího dekodéru lze připojit ke QSPI až šestnáct různých zařízení. Chip select se po odvysílání dat vrací do neaktivní úrovně. V případě, že

[5]V tomto procesoru je QSPI možno nastavit pouze do režimu MASTER, což znamená, že generuje hodinový signál a signály chip select.

tomu chceme zabránit, musíme v průběhu vysílání změnit nastavení aktivní úrovně, což je dosti nepraktické.

Fronta QSPI může být pomocí příkazů nakonfigurována do dvou režimů. V prvním z nich je nastaven začátek a konec fronty a po spuštění vysílání je fronta postupně odvysílána. Po dosažení konce se zastaví hodinový signál a chip selecty jsou uvedeny do neaktivní úrovně[6]. V druhém režimu po dosažení konce pokračuje vysílání od začátku fronty, který ovšem může být nastaven na libovolné místo v rámci paměti. Bohužel nastává problém s obsluhou takovýchto smyček, protože software musí zvládnout vybírat přijatá data z fronty a případně i měnit data určená k odvysílání.

Hodinový signál je odvozen od vnitřního hodinového signálu a je široce nastavitelný co se týče kmitočtu i přiřazení vzorkování a vkládání dat na výstup vzestupným a sestupným hranám.

### 4.4.5 GPT

General Purpose Timer je univerzální šestnáctibitový čítač se čtyřmi kanály nakonfigurovatelnými do režimu Output Compare, Input Capture a u kanálu tři navíc Pulse Accumulator, kdy jsou čítány externí pulzy. Taktovacím signálem je vnitřní hodinový signál podělený číslem 1 až 128 a čítač může být sesynchronizován s externí událostí.

### 4.4.6 Interrupt Controller Module

Řadič přerušení je prioritní, úrovňový a obsahuje 56 maskovatelných zdrojů přerušení od periferií. Každému zdroji je libovolně[7] přiřaditelná jedna z osmi hladin a v rámci hladiny priorita 1 až 7. Zdroj na vyšší hladině a s vyšší prioritou je obsloužen přednostně. Programátor musí při návrhu software zajistit, že nebude více zdrojům přerušení přiřazena stejná hladina a priorita.

[6] Je možno nastavit jejich návrat do neaktivní úrovně i po každém znaku.

[7] Periferie EPORT má svých sedm zdrojů přerušení napevno přiřazeno.

# 5 SOFTWARE PRO COLDFIRE

## 5.1 Inicializace periferií

První programem spouštěnou funkcí je `void proc_init(void)`. V ní nejprve proběhne nastavení funkce vývodů u vstupně–výstupních bran na funkce příslušející použitým periferiím, dále je nastaven PIT a řadič přerušení.

Pomocí standardní funkce `void mcf5xxx_set_handler (int vector, void (*handler) (void))` je přiřazena adresa začátku funkce obsluhující přerušení do tabulky vektorů přerušení. Například `mcf5xxx_set_handler(64+13, i_UART0)` zajistí, že obsluhu přerušení UART0 (zdroj přerušení 13 [1]) provede funkce `__interrupt__ void i_UART0(void)`.

Následně jsou podle postupu uvedeného v katalogovém listu nastaveny postupně všechny tři UARTy a jako poslední jsou povolena přerušení. Tím se zabrání jejich vzniku v částečně nastavených periferiích.

## 5.2 Systém reálného času

V zařízení je uložen neproměnný údaj, který nastaví uživatel jakožto okamžik spuštění zařízení. Jeho nastavením se vynuluje druhý záznam, který udává čas od posledního nastavení. Čas je uložen ve čtyřech osmibitových proměnných `unsigned short sec=0,min=0,hod=0,dnu=0`, jejichž význam je dán jejich názvem. Z definice proměnných je vidět, že po 255 dnech dojde k přetečení počítadla dnů. Tak dlouhý běh zařízení se ale nepředpokládá.

Aktualizace času je prováděna pomocí funkce `__interrupt__ void i_PIT (void)` obsluhující přerušení PIT0. Toto přerušení má nejvyšší prioritu, protože je velice krátké a případným přerušením jiného přerušení se příliš výkon přerušeného přerušení nezdrží. Nejvyšší priorita zajišťuje, že obsluha nebude ničím zdržována, a nevzniknou tak chyby v určení času.

Nastavení předděličky PIT je $2^{11}$, takže kmitočet inkrementace čítače spočteme podle vztahu

$$f_{PCNTR++} = \frac{80\text{MHz}}{2^{11}} = 39,0625\text{kHz} \tag{5.1}$$

a odpovídající perioda je $T = 25,6\mu$s. Toto je zároveň nejmenší rozlišitelný úsek času v zařízení.

Z důvodu zjednodušení výpočtu by bylo vhodné, aby PIT generoval přerušení v sekundových intervalech. Aby k tomu došlo, muselo by se přerušení generovat po načítání PCNTR do 39062,5, to však nejde. Čítač proto v prvním kroku čítá do 39062 a v druhém do 39063, takže každá první sekunda je kratší o $T/2$, ale každá druhá rozdíl vyrovná.

[1] Jde o třináctý zdroj maskovatelného přerušení a 77 zdroj přerušení. Proto se přičítá číslo 64.

## 5.3 Záznam sériové komunikace

Záznam duplexní komunikace na RS232 vyžaduje dva monitorovací vstupy. K tomu jsou použity UART1 a UART2 nakonfigurované jako přijímače. Při přijetí znaku je generováno přerušení, jehož obslužný podprogram znak uloží do speciálního zásobníku a případně k němu přidá časovou značku.

Aby nemohlo dojít k přehození přijatých znaků při přerušení obsluhy jednoho UARTu obsluhou druhého, maskuje UART2 na začátku svého obslužného podprogramu přerušení UART1 a na konci masku vrací zpět. Opačnou situaci řeší vyšší úroveň přerušení od UART1. Tím, že není UART okamžitě obsloužen, nedochází ke ztrátě přijatého znaku, protože všechny UARTy obsahují tříznakový vyrovnávací buffer.

Podívejme se detailně, jak přerušení probíhá. Na obrázku 5.1 je znázorněna základní struktura přerušení. V prvním kroku je třeba z jistit, ze které části UARTu požadavek na přerušení pochází a v případě, že nebyl přijat žádný znak, obsluhu ukončit.

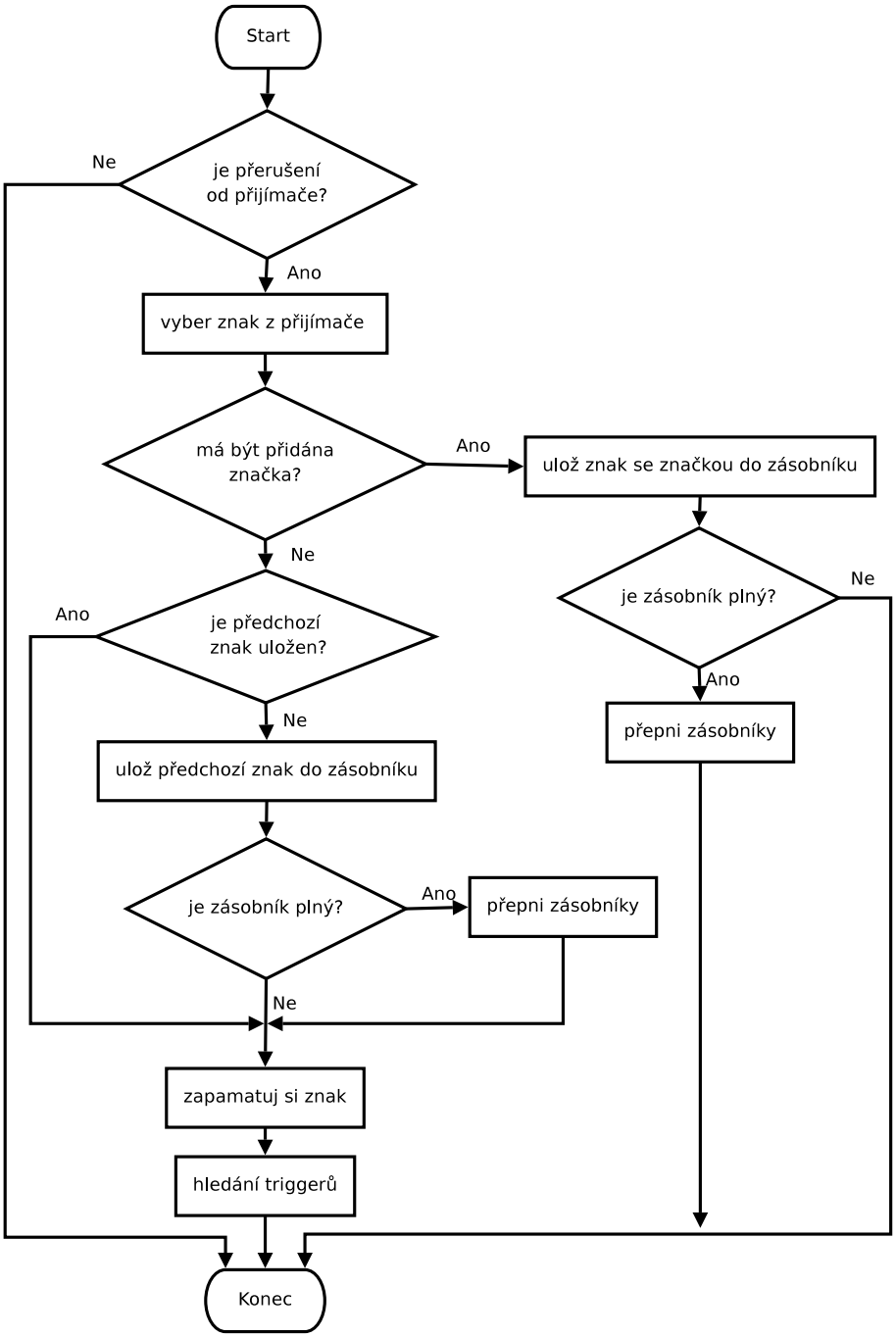
Přijatý znak je potom uložen do jednoho ze zásobníků společných pro oba kanály. Jeden ze zásobníků je vždy aktivní a jsou do něho ukládány znaky z obou kanálů s příznakem určujícím, ze kterého kanálu znak pochází. Druhý zásobník může mezi tím být v hlavní smyčce programu vyprazdňován. Pokus o implementaci kruhového bufferu jsem zamítl z důvodu náročnosti a také kvůli záznamu na paměťová média, ke kterým se většinou nepřistupuje po bytech, ale po blocích bytů. Jestliže je jeden zásobník plný, dochází k přepnutí na druhý.

Volba společného zásobníku pro oba kanály má proti odděleným zásobníkům výhodu ve znalosti přesného pořadí příjmu znaků bez nutnosti ukládat další informaci, a tedy efektivnější využití paměti. Lze pak snadno ze záznamu poznat, zda přijatá zpráva z jednoho kanálu byla reakcí na zprávu z kanálu druhého, a přišla tedy až po skončení první, nebo se obě prolínají a nesouvisí spolu.

## 5.4 Použití časových značek v programu

Časové značky jsou ukládány na začátcích bloků vymezených rozdílem času přijetí znaků větším než nastavená hodnota. Jestliže rozdíl dvou po sobě následujících značek je menší než jedna minuta, je značka ukládána jako relativní s rozlišením jedné periody inkrementace PCNTR, $T = 25,6\mu$s. Říkejme jí „Značka s plným rozlišením“. Po překročení minutového intervalu je uložena značka absolutní, vzniklá vyjádřením času v sekundách, té říkejme „Značka se sníženým rozlišením“. V binárním vyjádření značky zbylo ještě místo pro umístění příznaků, označujících typ značky (T) a příslušnost ke kanálu (C). Počítá se i s umístěním příznaků chybových stavů. Obě značky jsou graficky znázorněny na obrázku 5.2 a na obrázku 5.3 je blokové schema jejich vytvoření.

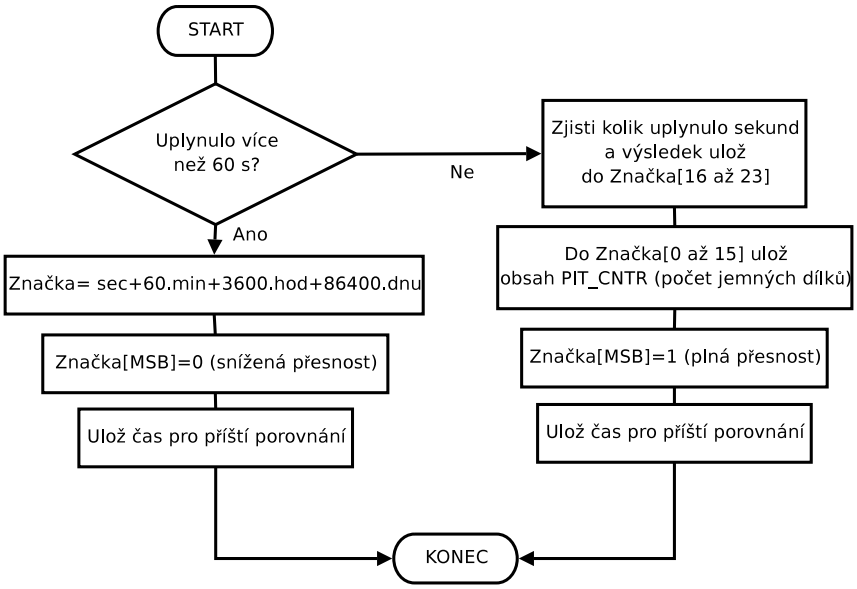
Dvojí systém značek je kompromisem mezi přesností v čase a velikostí záznamu. Jestliže hodinu neprobíhala žádná komunikace, nezáleží příliš na tom, jestli značka je s přesností na jednu sekundu nebo mikrosekundy. Naopak ve chvíli, kdy komunikace probíhá intenzívně a je třeba znát vztahy mezi jednotlivými úseky, je nutná větší přesnost, aby bylo možno říci, co je například odpovědí a co dotazem.




Obr. 5.1: Blokový diagram obsluhy přerušení UART1 nebo UART2

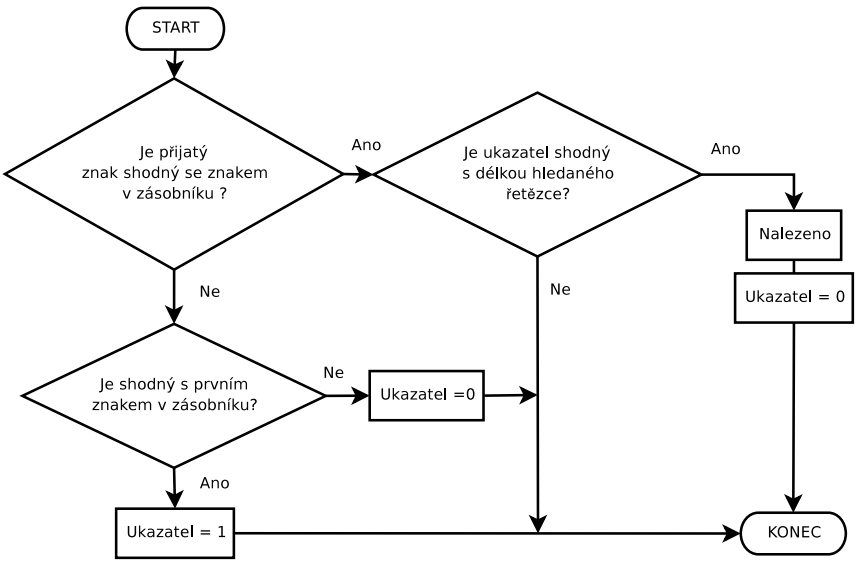
| bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 .. 16 | 15 ..... 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | Značka s plným rozlišením | | | | | | | | | |
| obsah | T=1 | C | 0 | 0 | 0 | 0 | 0 | X | $sec$ | obsah PCNTR |
| | Značka se sníženým rozlišením | | | | | | | | | |
| obsah | T=0 | C | 0 | 0 | 0 | 0 | 0 | $sec + 60 \cdot min + 3600 \cdot hod + 86400 \cdot dnu$ | | |

Obr. 5.2: Struktura časových značek




Obr. 5.3: Blokový diagram sestavení časové značky

Program po vzoru logických analyzátorů implementuje triggery – posloupnost znaků, která spustí nebo ukončí záznam. Jak je trigger hledán znázorňuje obrázek 5.4.




Obr. 5.4: Hledání konce zprávy

## 5.5 Záznam na SD/MMC kartu

Pro komunikaci s paměťovou kartou jsem napsal jednoduchý modul užívající QSPI periferii procesoru, který poskytuje funkce pro inicializaci QSPI i paměťové karty, pro zápis dat a pro čtení dat.

```
extern uint8 cti_blok(int32 cislo_bloku); /*precte blok 512B dat*/
extern uint8 zapis_blok(int32 cislo_bloku);/*zapise blok 512B dat*/
extern uint8 vsechny_inicializace_karty(void); /*provede
                                          inicializaci karty*/
```

Parametr `cislo_bloku` slouží jako adresa. Dále je poskytnuto několik globálních proměnných s parametry karty a proměnné `vuint8 kz0[512]` a `vuint8 kz1[512]` sloužící k předávání dat funkcím čtení a zápisu. Zároveň jsou to přepínané zásobníky, o nichž byla řeč v kapitole 5.3.

### 5.5.1 Inicializace karty

Funkce `uint8 vsechny_inicializace_karty(void)`, kterou je nutno volat po vložení karty, nejprve (voláním k tomu určené funkce) v paměti QSPI připraví frontu 10 příkazů, které ponechávají všechny chip select signály neaktivní a 10 datových polí s tokenem Idle. Odvysíláním se kartě poskytne 80 taktů hodin nutných podle [6].

Následuje volání funkce `void posli_6(uint8 p1, uint8 p2, uint8 p3, uint8 p4, uint8 p5, uint8 p6)` s parametry GO_IDLE_STATE, 0x00, 0x00, 0x00, 0x00, 0x95 , kde GO_IDLE_STATE je předdefinovaná konstanta s binární reprezentací tohoto příkazu a 0x95 je CRC-7 kontrolní součet. Funkce `posli_6` připraví QSPI frontu se svými parametry jako daty a spustí její vysílání. Aby bylo zajištěno, že fronta nebude poškozena voláním další funkce pracující s QSPI, je na konci funkce prázdná smyčka čekající do doby, než je vysílání fronty dokončeno.

Dále je volána funkce `uint8 cekej_na_odpoved(void)`, která periodicky vysílá token Idle a čeká na odpověď jinou než Idle. Návratovou hodnotou je odpověď karty. V případě dosažení jistého počtu pokusů je indikován neúspěch.

Postupné volání dvojice funkcí `posli_6` a `cekej_na_odpoved` je v modulu typickým dějem poslání příkazu a čekání na odpověď karty. Podle odpovědi je pak možno provést rozhodování.

Karta je nyní v režimu SPI a kontrola CRC-7 je neaktivní. V dalším kroku se stejným způsobem odešle kartě příkaz SEND_OP_COND a to se opakuje dokud není odpověď karty 0x00, nebo není dovršen maximální povolený počet neúspěšných pokusů.

Karta je nyní inicializována a je třeba zjistit její parametry. Po odeslání příkazu SEND_CSD a odpovědi R1 je pomocí volání `cekej_na_odpoved` zjišťováno, zda karta odvysílala Data token. V tom případě se provede volání funkce `void cti_16(void)`, která připraví frontu v paměti QSPI o délce šestnácti znaků, které jsou všechny Idle. Po jejich odvysílání jsou v paměti QSPI přijatá data, která si odtud může volající funkce přečíst. V tomto případě je to obsah registru CSD, ze kterého jsou získány údaje o kartě a uloženy do globálních proměnných pro další

použití. Je také nastavena globální proměnná `KARTA_INICIALIZOVANA=1`, pomocí níž funkce zápisu a čtení testují, zda již byla karta inicializována.

### 5.5.2 Čtení a zápis

Protože většina karet podporuje zápis a čtení pouze po blocích o velikosti 512B, je toto zvolena jako jediná možnost a karty, které tento způsob nepodporují, jsou při inicializaci odmítnuty[2].

K čtení z karty slouží funkce `uint8 cti_blok(int32 cislo_bloku)`, která přečte blok 512B dat. Adresou je přímo číslo bloku, které je interně přepočítáno vynásobením 512 na fyzickou adresu. Funkce čte data stejným způsobem jako inicializační funkce čte registr CSD, jen opakuje volání `cti_16` dvaatřicetkrát, čímž je přečteno 512B. Mezi každým voláním jsou přijatá data uložena do jednoho ze zásobníků `kz0`, `kz1`, kde jsou dále k dispozici.

Funkce zápisu pracuje na stejném principu, jen nevolá `cti_16`, ale sama plní frontu QSPI daty z jednoho ze zásobníků a odvysílá je.

### 5.5.3 Způsob uložení dat

Každý zapisovaný blok začíná hlavičkou o délce šestnáct bytů. Prvních pět bytů slouží k identifikaci obsazeného bloku. Jestliže obsahují znak 0x20, je blok obsazen, jinak je brán jako prázdný. Dalších šest bytů obsahuje zkopírované údaje o času spuštění záznamu ve formátu rok, měsíc, den, hodina, minuta, sekunda. Tři byty jsou nevyužity a polední dva obsahují údaj o velikosti bloku, který se využije v případě, že blok není plně obsazen.

Následují data ve stejné podobě, jako jsou uložena v zásobnících programu. Jestliže je ukládán znak, je nejprve uložen informační byte, viz obrázek 5.5 a následně přijatý znak.

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Význam | nevyužit | kanál | kanál | nevyužit | nevyužit | nevyužit | identifikace typu=1 | nesmí být využit |

Obr. 5.5: Struktura informačního byte

Využívá se při tom podobnosti struktury časové značky, která má informaci o kanálu uloženu v prvním bytu na stejných pozicích. Rezervace dvou bitů umožňuje budoucí rozšíření o další dva přijímací kanály. Bit identifikující typ uložených dat je u časové značky je logická 0.

Znak je uložen a jestliže k němu přísluší ještě časová značka, je uložena hned za něj. Při čtení se pak kontroluje identifikační bit a pokud je nastaven, je čten ještě následující byte a dvojice dekódována jako záznam znaku. Je-li naopak identifikační bit vynulován, jsou čteny ještě následující tři byty a čtveřice je dekódována jako časová značka. Následně se čte další byte a opět probíhá rozhodování, dokud není dosaženo konce bloku.

Pokud již místo v bloku dat nestačí k uložení dvojice respektive čtveřice bytů, je konec bloku ponechán volný a pokračuje se v novém bloku, který je samozřejmě

[2]Vzhledem k tomu, že většina pramenů, ze kterých jsem čerpal, ani jinou velikost bloku nepřipouští, nebude odmítnutí karty příliš časté.

opět opatřen výše popsanou hlavičkou. Jestliže je záznam ukončen, je zapsán poslední blok obsahující data a další, prázdný, blok je opatřen hlavičkou, která má na místě pěti identifikačních bytů znaky 0x00, tím je zaručeno, že konec záznamu bude rozpoznán.

## 5.6 Ovládání

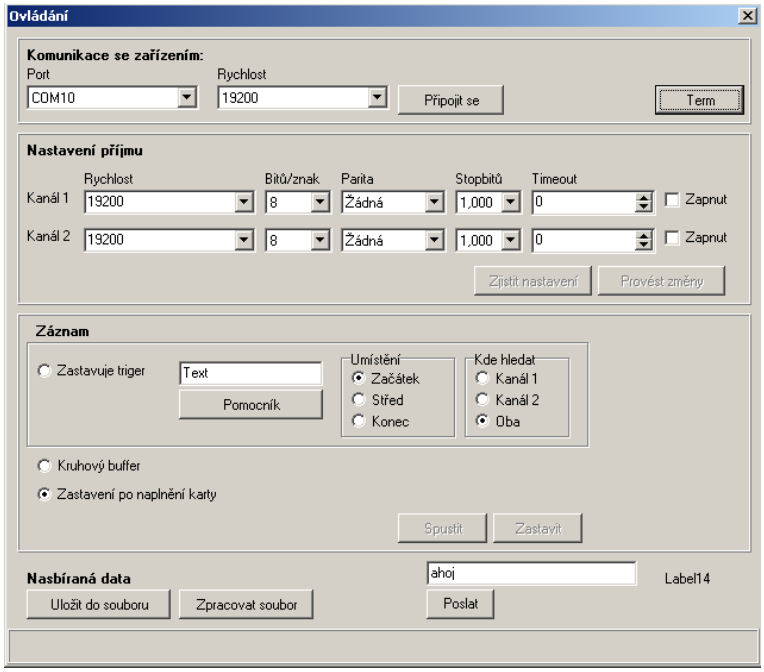
Pro nastavení a ovládání zařízení slouží UART0. Znaky z přijímače jsou ukládány do krátkého zásobníku a v případě přijetí kombinace `CR,LF` je začátek zásobníku porovnán s definovanými posloupnostmi znaků. Jestliže je nalezena shoda, je volána funkce provádějící příslušné nastavení. Jedním z „příkazů“ je `help`, vypisující nápovědu k ostatním příkazům:

```
==================================
MoSeK - monitor seriove komunikace
Jan Perny
==================================
Seznam prikazu:
kolikje - vypise aktualni cas od spusteni
jakyjeden - vypise cas spusteni
nastavden dd/mm/rr - nastavi den spusteni
nastavcas hh:mm:ss - nastavi cas spusteni
----------------------------------
zastavuart1 - zastavi zaznam z kanalu 1
spustuart1 - spusti zaznam z kanalu 1
zastavuart2 - zastavi zaznam z kanalu 2
spustuart2 - spusti zaznam z kanalu 2
----------------------------------
baudrateX YY ZZ - nastavi registry uartu X
    X=0 ovladaci kanal, X=1 nebo X=2 zaznamove kanaly
    UBG1 = YY = divider_msb, UBG2 = ZZ = divider_lsb
    uint16 divider=(80 000 000)/(32*baudrate)
    zadava se v hexadecimalnim vyjadreni");
umrX YY ZZ - nastavi registry uartu X
    X muze byt jen 1 nebo 2
    YY=UMR1 ZZ=UMR2, opet hexadecimalne
----------------------------------
prepnizasobnik - prepne zasobnik pro zaznam a druhy vyprazdni
konecblk XXrrrrr
    - nastaveni koncu zprav, po kterych ulozit casove znacky
    XX = jak bude dlouhy zadany retezec
    rrrr = uvedeny pocet (XX) znaku retezce
    pro vlozeni cr,lf >konecblk 02<a potvrdit
----------------------------------
vlozenakarta - inicializuje kartu
zaznamnakartu - zaznamenava prijata data na kartu, prepisuje
zaznambezkarty - prestane zaznamenavat na kartu
ctikartu - cte zaznam z karty, nelze delat pokud se na ni zaznamenava
----------------------------------
```

```
stav - vypis stavovych informaci
help - tato napoveda
==================================
```

UART0 se také velice osvědčil při ladění programu, kdy na něho byly vypisovány stavové informace sloužící k určení, kde se program nachází, chyby, které byly detekovány a dokonce i přijatá data a časové značky. Tento kanál lze případně použít i pro záznam dat pomocí počítače, pokud není vložena paměťová karta.

Aby se usnadnilo ovládání programu, bylo v programovacím jazyku Delphi vytvořeno jednoduché grafické rozhraní (obrázek 5.6), které podle zadaných voleb odešle příkazy programu, přečte a interpretuje jeho odpověď. Je také v součinnosti s programem, prostředkem ke čtení a uložení do počítače dat zaznamenaných na paměťovou kartu.




Obr. 5.6: Grafické rozhraní

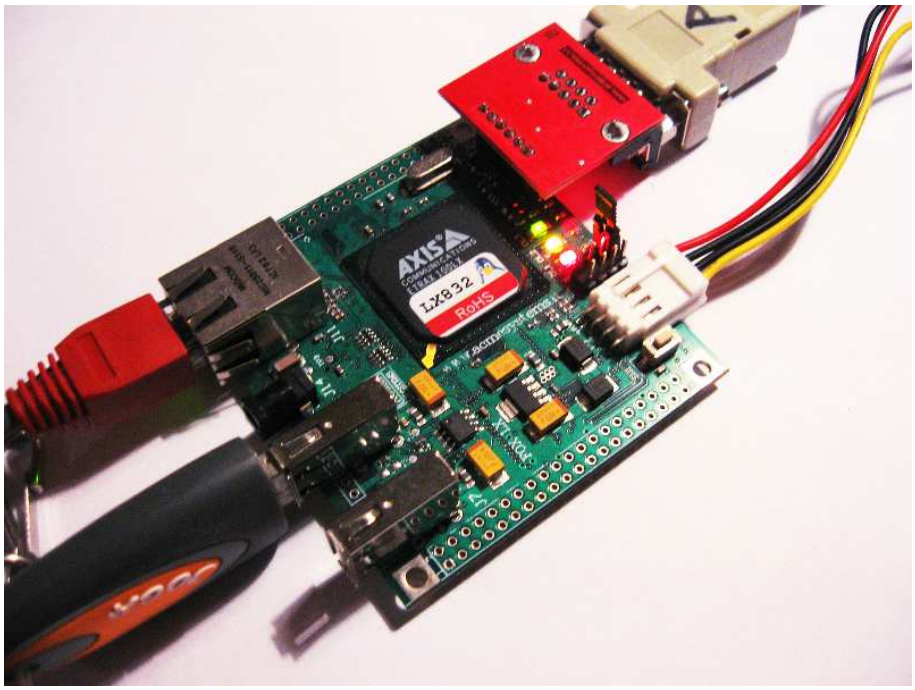
Při bližším pohledu na výpis příkazů zjistíme, že některá důležitá nastavení zde chybí. Je to způsobeno tím, že vývoj programu byl v tomto směru zastaven a došlo ke změně procesoru, jak je popsáno v kapitole 6.

# 6 ZMĚNA MIKROKONTROLÉRU

## 6.1 Důvody

Vzhledem k požadavku vzdáleného přístupu k zařízení přes internet byl stávající mikrokontrolér nevyhovující z důvodu neexistence hardwarové podpory ethernetu. Navíc se během práce ukázalo, že zaznamenávat data na paměťové médium tak, aby je bylo možné bez dalších pomůcek přečíst v běžném osobním počítači, vyžaduje implementaci známého souborového systému. Protože oba požadavky vyžadují vytvoření relativně složitého softwaru, rozhodl jsem se, že bude vhodné kromě změny hardware využít i služeb operačního systému.

## 6.2 Nová volba




Obr. 6.1: FOX Board

Volba padla na embedded počítač FOX Board (obrázek 6.1) firmy Acme systems s 32–bitovým RISC mikroprocesorem Axis ETRAX 100LX. Fox board má ethernet port, dva USB 1.1 porty a jeden sériový port, který může být doplněn dvěma dalšími[1]. Lze také použít USB hub a převodníky z USB na RS232, ale systém je nestíhá obsluhovat s dostatečnou rychlostí, takže mohou být užity jen pro malé přenosové rychlosti. Samozřejmostí jsou porty pro běžné použití, k nimž je připojeno jedno tlačítko a dvě použitelné LED diody.

Vybrané parametry jsou uvedeny v tabulce 6.1. Více lze nalézt v online dokumentaci [10] na stránkách výrobce.

[1]K dispozici jsou tři další, ale jeden z nich je blokován použitím USB

Tab. 6.1: Vybrané parametry Fox boardu

| **Parametr** | **Hodnota** |
|---|---|
| Velikost | 66 × 72 mm |
| Procesor | Axis ETRAX 100LX |
| | 100MIPS při 100MHz |
| RAM | 32MB |
| FLASH | 8MB |
| Napájecí napětí | 5V |
| Příkon | 1W |

Na sytému běží operační systém GNU/Linux, který poskytuje síťové služby telnet, FTP, HTTP a SSH. Vývoj aplikací v C je možný ve volně šiřitelném vývojovém prostředí, které neobsahuje žádné grafické nástroje, ale pouze nástroje nutné k překladu programu. Každý tak může použít svůj oblíbený textový editor pro tvorbu zdrojových kódů, které přeloží pomocí jednoduchého Makefile.

Na internetu je velmi užitečná dokumentace ke GNU LibC [11], z níž jsem velmi čerpal při vývoji programu. Rád bych zde vyzdvihl snadnou dostupnost veškeré dokumentace, kterou jsem potřeboval.

# 7 SOFTWARE PRO FOXBOARD

## 7.1 Požadované vlastnosti programu

Vlastnosti, které má vytvořený program mít, jsou podobné jako u předchozího. Tedy má být schopen zaznamenávat komunikaci na sériové lince při rychlosti 115200 Bd z několika kanálů. Do záznamu má přidávat časové značky podle volby uživatele a takto vytvořený záznamu ukládat. Záznam bude probíhat na standardní USB disk, který program připojí, nebo odpojí při stisku tlačítka, aby ho bylo možno nahradit za chodu jiným.

Časové značky se budou ukládat v pravidelném intervalu nebo k poslednímu znaku před časovou mezerou překračující nastavenou velikost a k následujícímu znaku. Poslední možností je nechat si značkou označovat konec uživatelem zvoleného slova nalezeného v proudu dat v kanálu. Nalezení slova může také v kanálu záznam zastavit, nebo naopak spustit, pokud si uživatel přeje zaznamenávat jen zajímavé úseky komunikace.

Všechna nastavení programu nutná pro jeho chod mu budou dodána v konfiguračním souboru. V případě chyby v konfiguračním souboru podá program hlášení a skončí. V případě chyb, které nejsou kritické, je bude program hlásit a snažit se pokračovat v činnosti.

## 7.2 Struktura programu

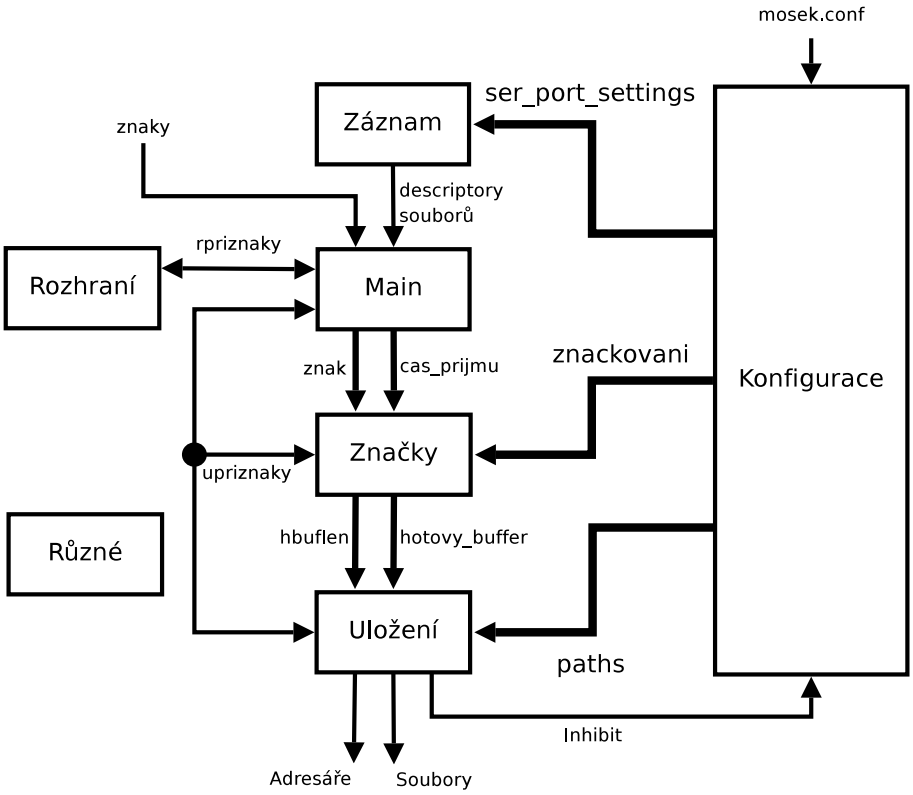
Program jsem rozdělil na části, říkejme jim moduly, které pokrývají nějakou logicky ucelenou část programu, například sběr, zpracování, uložení dat či konfiguraci.

Na obrázku 7.1 je uvedena bloková struktura uspořádání programu. Do programu vstupují soubor „mosek.conf“ a znaky přijímané ze sériových portů. Výstupem jsou soubory s daty uložené v adresáři na USB disku. Moduly mezi sebou komunikují pomocí jednobitových příznaků, případně dostávají nastavení od modulu Konfigurace. Modul Main na základě dodaných deskriptorů čte znaky ze sérových portů a spolu s časem přijetí je předává modulu Značky, který provede zpracování tak, aby nalezl triggery a vyhodnotil, zda se má přidat časová značka. Jakmile má potřebné informace, uloží data s přidanými informacemi do zásobníku `hotovy_buffer` a spolu s údajem o počtu bytů v zásobníku je předá modulu Uložení. Pro modul uložení jsou to již jen binární data, která má zapsat na disk.

## 7.3 Modul různé

Jak sám název napovídá, modul Různé obsahuje věci, které se nedaly nikam zařadit, nebo se používají ve více modulech. Proto s ním začínáme, abychom nemuseli stále vysvětlovat, že daná funkce či struktura je v něm uvedena.

V první řadě modul obsahuje ukazatel `ser_port_settings` na strukturu `ser_port_settings_T`, která obsahuje prvky pro uložení základních parametrů portů – přenosová rychlost, počet bitů, stopbitů a nastavení parity. Kromě toho je zde uložen i deskriptor souboru reprezentující sériový port a příznaky nastavení parametrů.




Obr. 7.1: Modulární struktura programu

Struktura je v paměti tolikrát s kolika porty – kanály – program pracuje. Paměť je dynamicky alokována při inicializaci.

Informace, jak se mají data z daného sériového portu zpracovávat, jsou uloženy v paměti v místě, kam ukazuje `znackovani`, což je ukazatel na strukturu `znackovani_T`. Struktura obsahuje tyto členy:

- `unsigned char stav_kanalu` obsahuje příznaky popisující stav kanálu a jaká zpracování se v něm mají provádět.
- `unsigned char pocet_triggeru` určuje počet triggerů, které v kanálu existují.
- `char **trigger` je dynamicky alokován a jsou v něm uloženy řetězce odpovídající triggerům.
- `unsigned char *trigger_type` určuje typ triggeru, ke kterému přísluší.
- `unsigned char *trigger_pos` slouží pro uložení stavu vyhledávání příslušného triggeru.
- `unsigned char *trigger_length` obsahuje údaj o délce příslušného triggeru. Jelikož trigger může být z libovolných znaků, nelze použít běžný nul terminated string.

- `struct timeval cas_znaku` sem je ukládána informace o tom, kdy byl přijat předchozí znak v tomto kanálu.
- `struct timeval delka_mezery` obsahuje čas mezi znaky, který je považován za maximum v rámci jednoho bloku.
- `struct timeval cas_znacky` obsahuje čas, ve kterém byla v kanálu naposled uložena časová značka.
- `struct timeval pravidelnost_znacky` obsahuje nastavení času vložení pravidelné značky.

Poslední strukturou, která však není v paměti tolikrát, kolik kanálů program zpracovává, ale pouze jednou, je `paths` typu `paths_T`. Obsahuje v sobě nul terminated stringy pro určení cesty pro data `data_path`, jméno blokového zařízení, které se bude montovat[1], cestu `mount_point`, kam se bude montovat, souborový systém na zařízení `fstype` a `altmntdev` řetězec určující, která zařízení zkoušet připojit v případě chyby.

Kromě těchto struktur modul obsahuje proměnnou `num_of_ports` obsahující počet kanálů, se kterými program pracuje a příznak `chyba_byla`, který je nastaven v případě, že program narazí na chybu, která mu dovolí dále pokračovat, ale musí se signalizovat, že nastala.

Modul poskytuje funkce:

- `void capitalize(char *this_string)`, sloužící k převodu malých písmen na velká. Je používána při konfiguraci.
- `unsigned char mystrtoint(const char *str,int *vysledek)` je funkce k převodu null terminated stringu na číslo. Jde o zapouzdření standartní funkce `strtol`, která však signalizuje chybu nevyhovujícím způsobem.
- `char *prelozstring(char *str, unsigned int length)` je funkce, která převádí řetězec znaků obsahující escape sekvence na řetězec obsahující jejich reprezentaci. Výstupem již není nul terminated string, a tak musí funkce vracet i délku řetězce.
- `void *mymalloc(size_t size)` je zapouzdření funkce `malloc` s ošetřením chyby při nedostatku paměti – program je ukončen.
- `void *myrealloc(void *ptr, size_t newsize)` je zapouzdření funkce `realloc` stejným způsobem jako je zapouzdřena `malloc`.
- `void init_error_loging(void)` je funkce zapouzdřující volání funkce `openlog` pro inicializaci syslogu.
- `unsigned char rozdilcasu(struct timeval *result, struct timeval time1, struct timeval time0)` je funkce sloužící k odečtu dvou struktur `timeval` obsahujících čas.

[1] V Linuxu se zařízení připojují pomocí příkazu mount.

- `unsigned char jevetsi(struct timeval time1, struct timeval time0)` je funkce pro porovnávání struktur `timeval`.

## 7.4 Modul záznam

Modul záznam slouží především ke konfiguraci, otevření a zavření sériových portů. Obsahuje tak pouze funkce `s_port_open(struct ser_port_settings_T *nastaveni)` pro konfiguraci a otevření sériového portu a `s_port_close (struct ser_port_settings_T *nastaveni)` pro jeho uzavření. Protože v Linuxu se se všemi zařízeními pracuje jako se souborem, je otevření, čtení, zápis a uzavření standardním postupem volání funkcí `open`, `read`, `write` a `close`, které jsou popsány v [11].

Funkce `s_port_open` tak otevře soubor, jehož název je jí dodán ve struktuře `ser_port_settings` a uloží si deskriptor vrácený funkcí `open`. Jakmile je port úspěšně otevřen, je pomocí funkce `tcgetattr`[2] přečtena struktura `termios serial_flags` obsahující veškerá nastavení portu, dokumentace je opět v [11]. Podle parametrů ve struktuře `ser_port_settings` jsou parametry pozměněny a pomocí funkce `tcsetattr` jsou nastavení provedena. Jen přenosová rychlost je nastavena pomocí funkcí `cfsetospeed` a `cfsetispeed`.

Funkce `s_port_close(struct ser_port_settings_T *nastaveni)` je pouze zapouzdřením funkce `close` s ošetřením chyb.

## 7.5 Modul konfigurace

Jedinou z vnějšku viditelnou funkcí tohoto modulu je `int zpracuj_soubor (char *filename)`, která je volána z modulu Main a jejím parametrem je název konfiguračního souboru. Ten je otevřen standardní funkcí `fopen` a dále zpracován jak naznačuje obrázek 7.2.

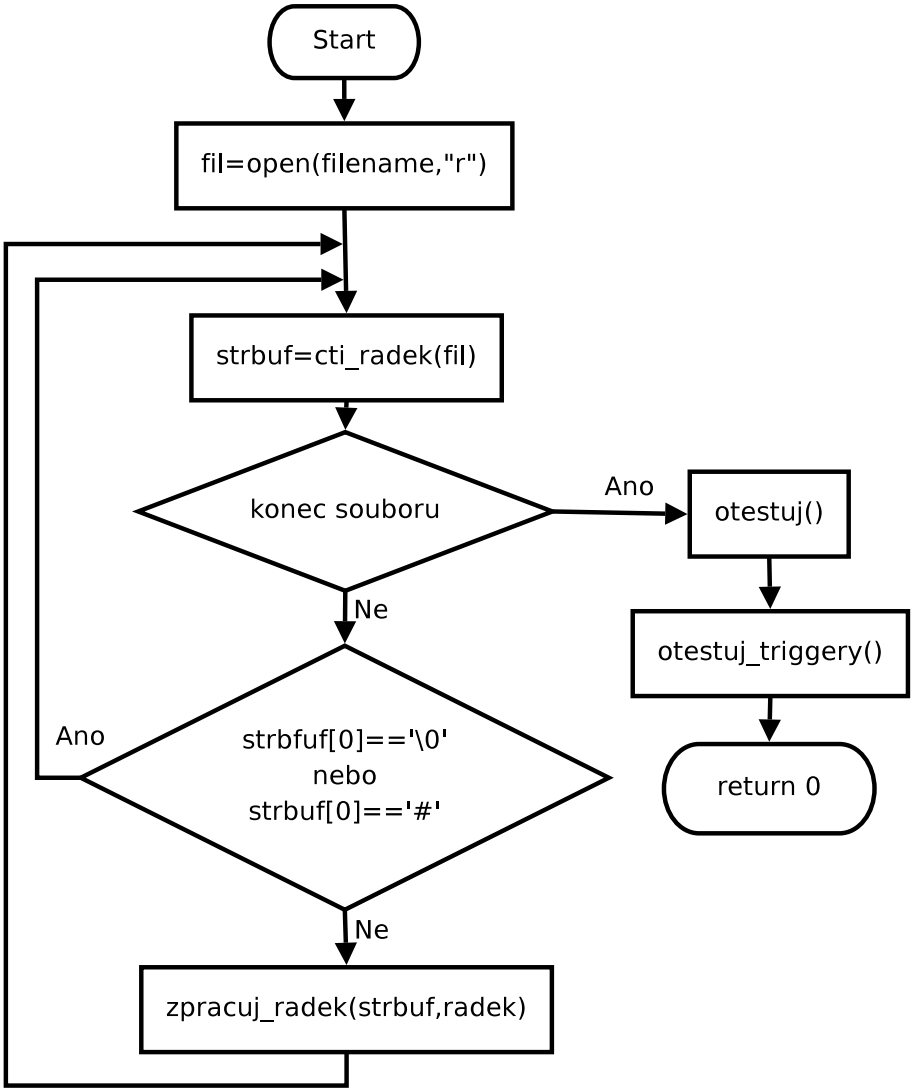
První v řetězci zpracování je funkce `char *cti_radek(FILE *fr)`, která čte ze souboru znaky dokud nenarazí na konec řádku a vkládá je do zásobníku, který si v případě potřeby zvětší. Nad tímto zásobníkem pak pracuje funkce `void zpracuj_radek(char *radek, unsigned int cislo)`, které se v druhém parametru předává také číslo řádku, aby bylo případně možné vypsat, na kterém řádku se vyskytla chyba. `Zpracuj_radek` není volána, pokud je řádek prázdný nebo je komentářový[3].

Jestliže je zpracován již celý soubor, jsou volány funkce `void otestuj (void)` a `void otestuj_triggery(void)`, které kontrolují, zda přečtená nastavení neobsahují chyby a zda byly zadány všechny potřebné údaje. Nyní se podrobněji věnujme funkci `zpracuj_radek`.

Při zpracování řádku je používána funkce `void najdislovo(char *kde, char **slovo,char **zbytek)`, která postupně rozebere řádek na jednotlivá slova – tokeny. V prvním parametru je jí dodán řetězec, kde má hledat a další dva jsou ukazatele na místo, kam má uložit výsledek. „kde“ je prohledáván znak po znaku,

[2]Rovněž je popsána v [11]

[3]Začíná znakem „#“.




Obr. 7.2: Zpracování konfiguračního souboru

dokud není nalezena nemezera[4], tedy začátek tokenu. Ukazatel na nalezený znak je vložen do „*slovo“ a hledání pokračuje dále, dokud jsou nacházeny další nemezery. Ukazatel na první následující mezeru je vložen do „*zbytek“, při čemž je mezera nahrazena znakem null pro ukončení stringu. V případě nenalezení, nebo konce řádku jsou vraceny nulové ukazatele. Funkci je možné volat ve smyčce způsobem `najdislovo(zbytek,&slovo,&zbytek);` a po každém zavolání bude v „slovo“ následující token ze řádku.

Popis konfiguračního souboru je uveden v kapitole 8.1.

[4]Cokoliv kromě znaků „ “, „\f“, „\n“, „\r“, „\t“, „\v“ a „=“, který mezi ně byl navíc.

## 7.6 Modul značky

Modul značky z vnějšího pohledu poskytuje jen dvě funkce – inicializační funkci `unsigned char init_znacky(void)` a pracovní funkci `void oznackuj_znak (char znak,struct timeval cas_prijeti,unsigned char kanal)` k zpracování přijatého znaku, jež je naznačeno na obrázku 7.3. Jak je vidět, funkce nejprve zjišťuje, zda se bude z nějakého důvodu přidávat k přijatému znaku časová značka a pokud ano, nastaví si příslušné příznaky.

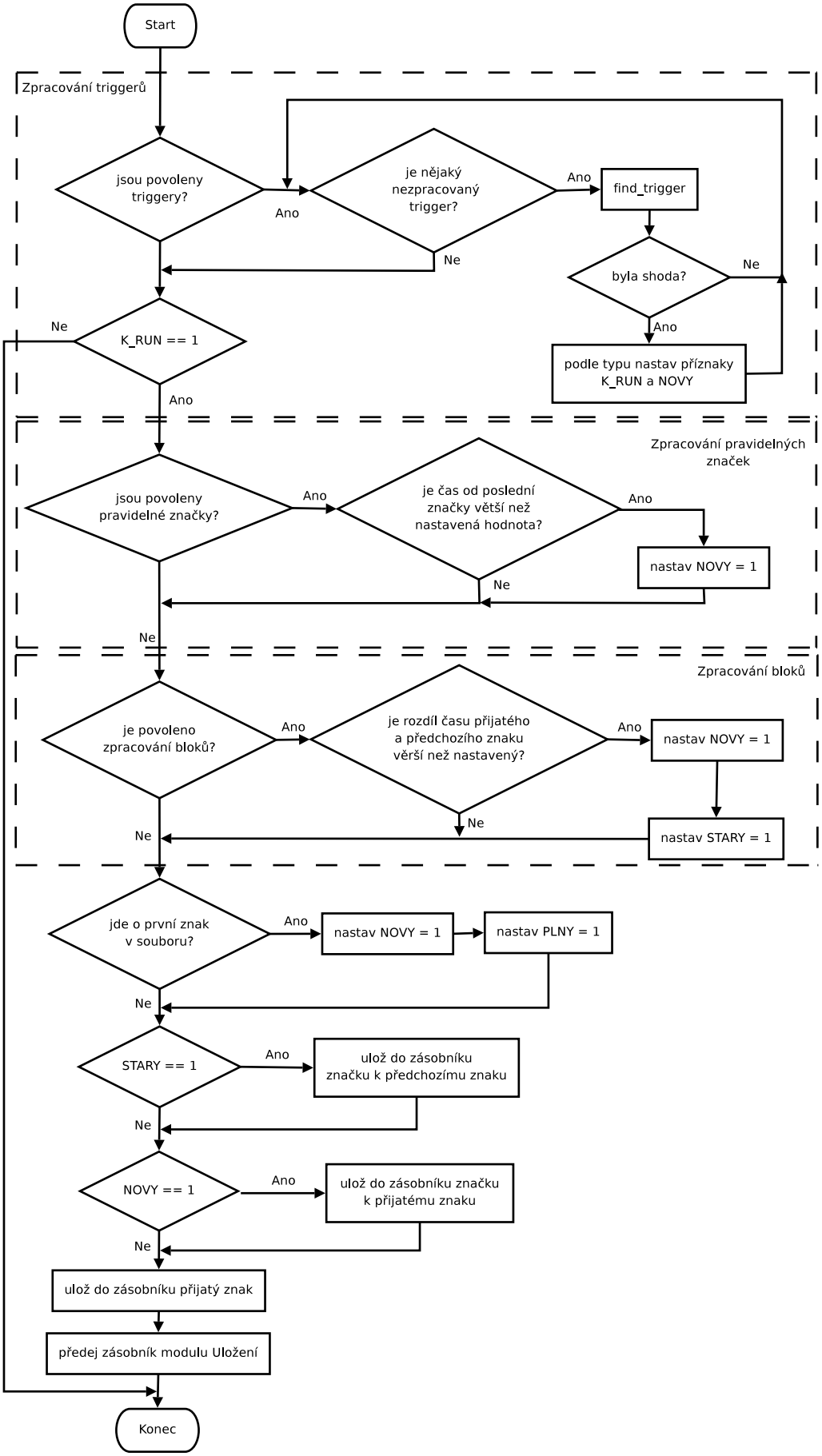
Zastavení příjmu z určitého kanálu je ovládáno příznakem „K_RUN“, který způsobí ukončení funkce před zpracováním znaku, avšak až po tom, co byly zpracovány triggery. To je nutné kvůli vyhledávání triggerů k opětovnému povolení příjmu. Hledání triggeru je provedeno způsobem uvedeným v kapitole 5.4 na obrázku 5.4.

Na základě nastavených příznaků je pak volána funkce `void vytvor_znacku (struct timeval cas, unsigned char kanal, unsigned char plna)`, která vytvoří a do zásobníku „hotovy_buffer“ vloží časovou značku. Značka může být dvou typů:

- Absolutní značka – udávající čas příjmu znaku v počtu sekund od Unixové epochy (1.1.1970 00:00:00, což lze nalézt v [12]).
- Relativní značka – udávající čas příjmu znaku v sekundách a násobcích 32$\mu$s od předchozí časové značky. Značka tedy obsahuje dva číselné údaje – počet sekund a počet násobků 32 mikrosekund. Výsledný čas je součtem obou údajů a času uvedeného v předchozí časové značce.

Která ze značek bude uložena, je rozhodnuto na základě rozdílu času předchozí a nyní ukládané značky. Jestliže je rozdíl větší než jedna minuta, je ukládána absolutní časová značka a pokud je menší, je ukládána relativní.

Výpis zdrojového kódu funkce `vytvor_znacku` je možno nalézt jako přílohu A.




Obr. 7.3: Zpracování v modulu Značky

## 7.7 Modul uložení

Modul Uložení tvoří rozhraní mezi systémem souborů a programem. Jeho úkolem je připojovat a odpojovat USB disky, vytvářet na nich adresáře a soubory pro uložení dat a data do souborů zapisovat. Pro komunikaci s ostatními moduly poskytuje globální proměnnou „upriznaky“.

Z funkcí poskytuje inicializační funkci `void ulozeni_init(void)`, funkci pro ukládání dat zpracovaných v modulu Značky `void uputdata(unsigned char *dbuffer,unsigned char length)`, funkce pro připojení a odpojení USB disku `char preruspromontovani (void)` a `char pokracujpomontovani (void)` a funkce použité při vyhledávání konfiguračního souboru při startu programu `char najdi_lomitko (char *vcem, char **tadyje)` a `void startmntfind(char *filenamebuff)`.

Funkce `uputdata` vezme data ze vstupního zásobníku a zkopíruje je do interního zásobníku „ubuffer“. V případě, že je zásobník prázdný, což značí, že bude zakládán nový soubor, zaznamená aktuální čas, ze kterého bude odvozeno jméno souboru. Jestliže byl zásobník naplněn nad stanovenou mez, je volána interní funkce `ulozdata` za účelem uložení dat do souboru. Je-li zásobník „ubuffer“ naplněn a nelze jej zvětšit (viz. dále), funkce data ze vstupního zásobníku nikam nekopíruje, ale zahazuje je.

Volaná funkce `ulozdata` nejprve zkontroluje, zda je otevřen soubor pro zápis dat a jestliže není a je připojen USB disk, vytvoří ho a otevře. Jestliže není možné soubor otevřít, například z důvodu nepřipojení disku, je zvětšen zásobník „ubuffer“ o hodnotu UBLENGTH = 1024B. [5] V případě, že by byl při zvětšování zásobníku překročen maximální počet bloků *maxbuf* zadaný v konfiguračním souboru (viz. 8.1), je nastaven příznak, že bylo dosažena maxima a buffer nelze dále zvětšovat a ke zvětšení nedojde.

V případě, že je soubor otevřen, nebo se jej otevřít podaří, jsou data zapsána a jestliže byl zásobník „ubuffer“ zvětšen z výchozí velikosti, je paměť uvolněna a zásobník se zmenší na výchozí velikost. V případě chyby při zápisu dat je postup shodný s postupem při neotevřeném souboru a pokud je detekováno na základě vyhodnocení typu chyby, že byl vytržen USB disk, považuje jej program za odpojený uživatelem, čeká na vložení nového a potvrzení vložení tlačítkem.

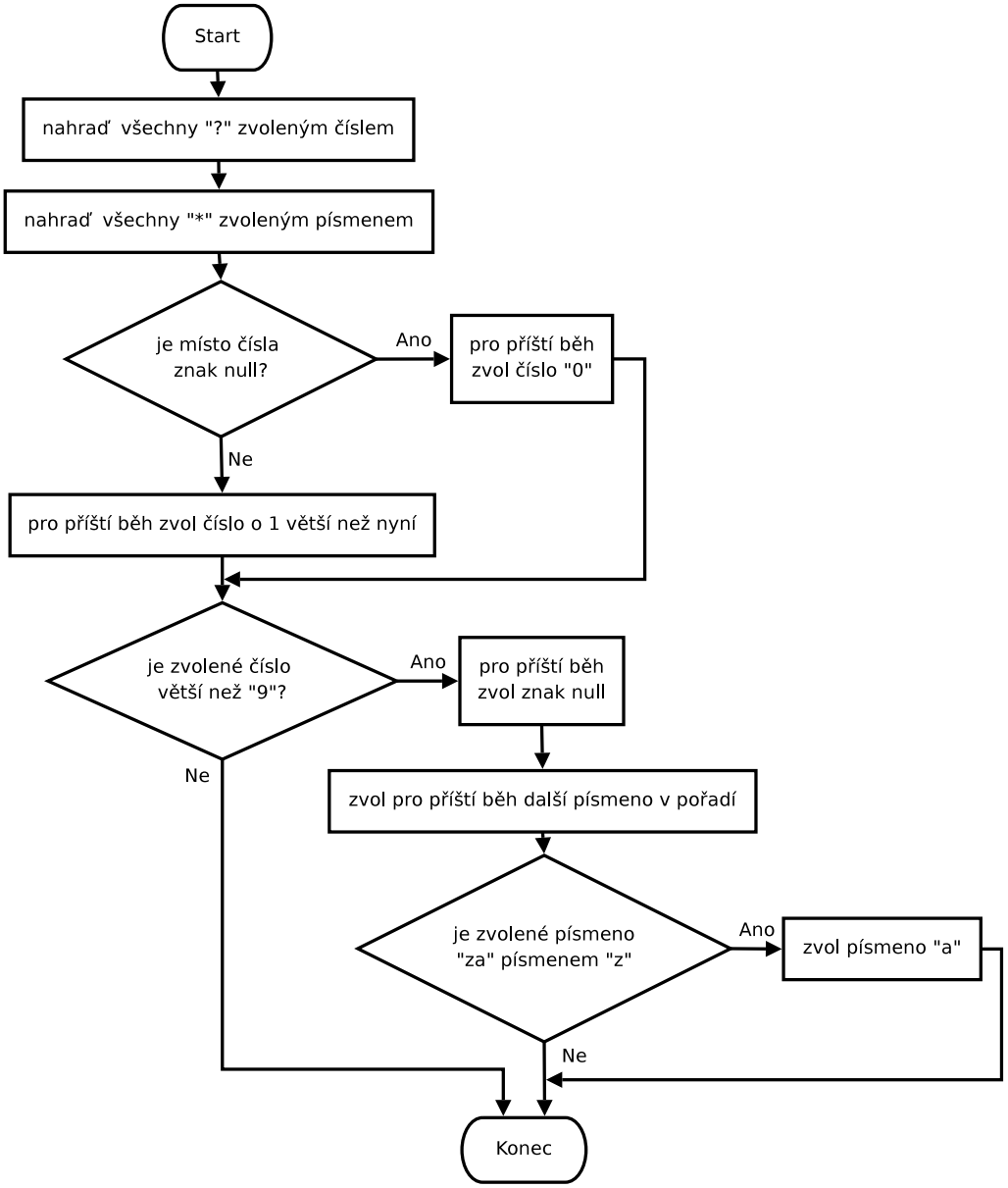
V případě, že byl soubor s daty založen před časem větším než je zadáno pomocí *savetime* v konfiguračním souboru, je soubor po zápisu uzavřen a nastavením příznaku je funkci `uputdata` signalizováno, že má opět zaznamenat čas pro vytvoření jména souboru, který bude založen při příštím volání `ulozdata`. Jména souborů jsou ve tvaru YYYY-MM-DD_HH-MM-SS.dat a v případě, že v adresáři již takový soubor existuje, jsou data přidávána na jeho konec.

Upozorněme zde na funkci `uloz_zapis`, která funguje stejně jako funkce `uloz`, ale soubor s daty zavírá vždy. Je volána funkcí `preruspromontovani`, aby byla všechna data uložena před odpojením USB disku (při zavírání souboru je volána funkce `fsync`, která pozdrží chod programu, dokud nejsou data bezpečně uložena na disku. Popsána je v [11]). V případě, že je v konfiguračním souboru nastaveno, že se zapisují data na médium, které se neodpojuje, funkce `preruspromontovani` zde skončí, jinak je volána funkce `odmontuj`, která je zapouzdřením funkce `umount2` (popsána je opět

[5] Na začátku je velikost zásobníku UBLENGTH + UBRES. Kde UBRES = 32B je rezerva zásobníku.

v [11]) s ošetřením chyb. Jestliže odpojení skončí neúspěchem, je pokus opakován na základě informace uložené v „upriznaky“ při příjmu dalšího znaku přímo z funkce `main` v modulu Main pomocí dalšího volání `preruspromontovani`.

Funkce `pokracujpomontovani` se, pokud je povoleno připojování zařízení v konfiguračním souboru, pokusí připojit USB disk a pokud se jí to nepodaří, opakuje se její volání a počítají se neúspěšné pokusy. Jestliže bylo dosaženo pěti neúspěšných pokusů, mezi kterými byla minimálně jednosekundová prodleva (při kratší se nepočítají), je cesta k blokovému zařízení zadaná v konfiguračním souboru prohlášena za neplatnou a program pomocí funkce `wildtranslate` zkouší vytvořit novou cestu z parametru *altmntdev* zadaného konfiguračním souboru.




Obr. 7.4: Funkce wildtranslate

`Wildtranslate` vyhledává a zaměňuje řetězci znaky „*“ a „?“ za znaky „a“ až

„z“, respektive za čísla 0 až 9 a znak null, jak je naznačeno na obrázku 7.4. Všechny nalezené znaky nahradí stejně!

V případě úspěchu při připojování pokračuje funkce `pokracujpomontovani` kontrolou adresářů z cesty *datapath* zadané v konfiguračním souboru. Pokud neexistují, nebo k nim nemá program přístupová práva, pokusí se je program vytvořit, respektive práva změnit a pokud se to nepodaří, program končí.

Funkce `najdi_lomitko` slouží k rozdělení cest na dílčí úseky a jde v ní jen o jednoduché vyhledání znaku „/“ v řetězci.

Funkce `startmntfind` se pokouší podobným způsobem jako funkce `pokracujpomontovani` připojit USB disk, avšak díky neexistenci konfiguračního souboru jí nelze uživatelsky zadat, co má zkoušet. Zvolené vstupní parametry by v konfiguračním souboru byly zapsány takto:
*mntpoint = /mnt/0*
*altmntdev = /dev/sd*?*
*fstype = vfat*
Při zkoušení jsou otestovány všechny možnosti, pak se jednu sekundu počká a opět se zkouší. Po dvaceti neúspěšných pokusech program skončí. Jestliže se podaří USB disk připojit, funkce hledá v jeho kořenovém adresáři soubor „mosek.conf“. Nenalezení souboru znamená konec programu. V případě úspěchu je program nakonfigurován z nalezeného souboru, ale nastavení pro připojování zařízení jsou ignorována.

## 7.8 Modul rozhraní

Modul Rozhraní slouží k obsluze ovládacího rozhraní zařízení, které je tvořeno dvěma LED diodami a jedním tlačítkem. Modul je vytvořen jako dvouúrovňový, kdy první úroveň pracuje přímo s hardwarem a je jako jediná část programu nepřenositelná na osobní počítač typu PC[6]. Na této úrovni jsou poskytovány čtyři služby zahrnující inicializaci portů pro všeobecné použití (`void init_tl_led(void)`), rozsvícení a zhasnutí připojené LED diody (`void led1_on(void)`, `void led1_off(void)`, `void led2_on(void)` a `void led2_off(void)`) a zjištění stavu tlačítka (`unsigned char tlacitko(void)`). V druhé úrovni jsou již jen volány zmíněné funkce a nezáleží na tom, na jakém hardwaru program běží.

Tlačítko i LED diody jsou připojeny na portu A, a to červená LED (led1) na vývodu PA3, žlutá LED (led2), která je paralelně spojena se žlutou LED, umístěnou v ethernet konektoru na vývodu PA2. Tlačítko je připojeno na vývod PA1. Dokumentace je uvedena [10] a lze v ní najít, že je použita negativní logika – diody svítí v logické `0` a stisknuté tlačítko také vrací logickou `0`.

Samotná obsluha portu je podle dokumentace výrobce [10] možná dvěma způsoby

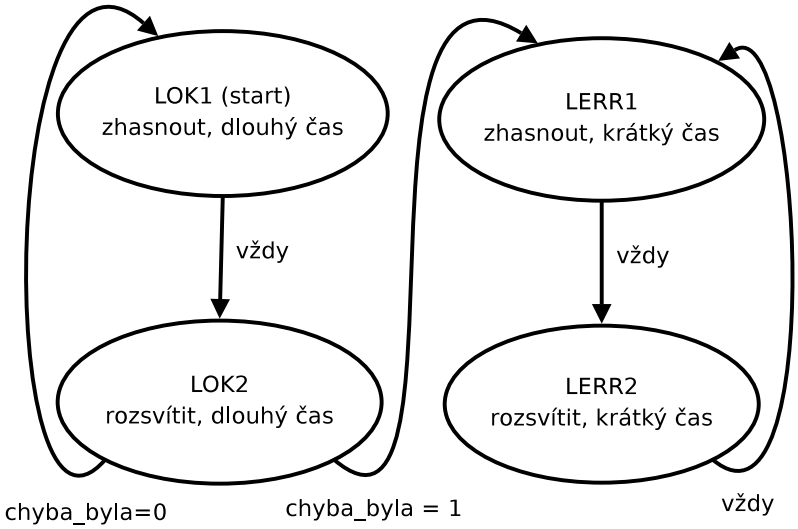
1. čtením a zápisem do speciálních souborů pro porty v adresáři /dev, nebo
2. použitím výrobcem dodané knihovny „linux/gpio_syscalls.h“, která porty ovládá přímo prostřednictvím jádra systému.

[6] Aby bylo možno ladit program na počítači, vytvořil jsem zároveň i verzi pro PC.

Protože je rychlejší a snadnější na implementaci, rozhodl jsem se pro druhou z možností, která se skládá jen ze správného volání funkcí `gpiosetdir`, `gpiosetbits`, `gpioclearbits` a `gpiogetbits`. Argumenty funkcí určují, se kterým portem a jeho vývodem pracujeme. Bližší popis je rovněž uveden v [10].

Verze pro PC místo LED diod používá výpis například `printf("LED1 on");`. Pro napodobení tlačítka byla napsána funkce `getch()`, která vrací číslo naposled stisknuté klávesy a vypíná kanonický mód linuxového terminálu [7], takže znaky jsou čteny okamžitě a ne až po stisknutí klávesy enter. Jestliže je při volání funkce `tlacitko` stisknuta nějaká[8] klávesa, vrací funkce hodnotu 1 = stisknuto, jinak hodnotu 0.

Druhá úroveň poskytuje inicializační funkci (`void rozhrani_init (void)`) a funkci obsluhy rozhraní (`struct timeval rfunguj (void)`). Návratovou hodnotou je čas, po kterém by měla být funkce volána příště a předává se jako timeout funkci `select`. Jestliže je funkce volána dříve, vrátí kratší čas tak, aby byla volána ve správném čase[9] a dále se neprovádí její kód.



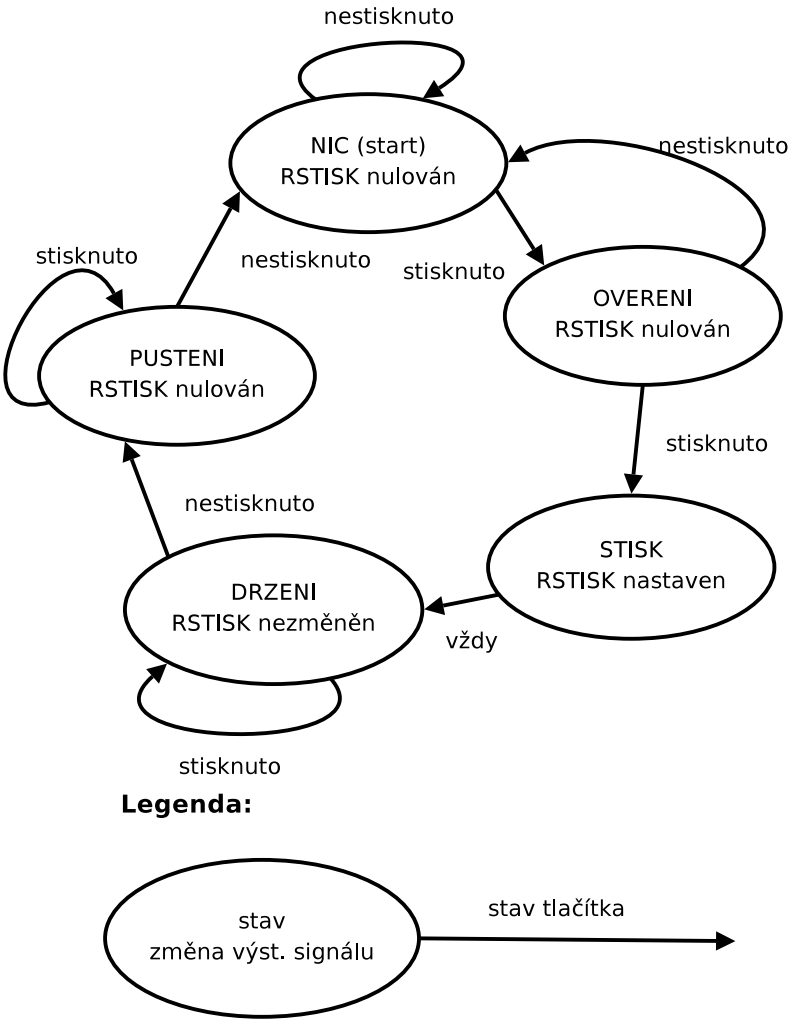

Obr. 7.5: Stavový automat pro LED1 v modulu rozhraní

Funkce obsahuje jednoduchý stavový automat o čtyřech stavech s blokovým diagramem uvedeným na obrázku 7.5, jehož výstupem je signál pro rozsvícení či zhasnutí první LED a čas, za který by se měla funkce opět volat. Jestliže je funkce volána v čase, kdy měla být, dojde k přechodu v automatu a k dalšímu vyhodnocování.

[7] stejným způsobem jako u sériového portu, jen se pracuje se stdin

[8] Uvažujme jen klávesy vracející znak.

[9] Pozdější volání je považováno za volání ve správném čase.




Obr. 7.6: Stavový automat pro ošetření zákmitů tlačítek

Jestliže je nastaven příznak „RLED2“ v proměnné „rpriznaky“, je rozsvícena oranžová LED dioda (led2) a naopak je-li vynulován, je dioda zhasnuta. Také dojde k vyhodnocení druhého stavového automatu sloužícího k ošetření zákmitů tlačítka, který je znázorněn na obrázku 7.6. Příznak „RSTISK“ je dále nulovám v modulu Main při obsluze stisku tlačítka.

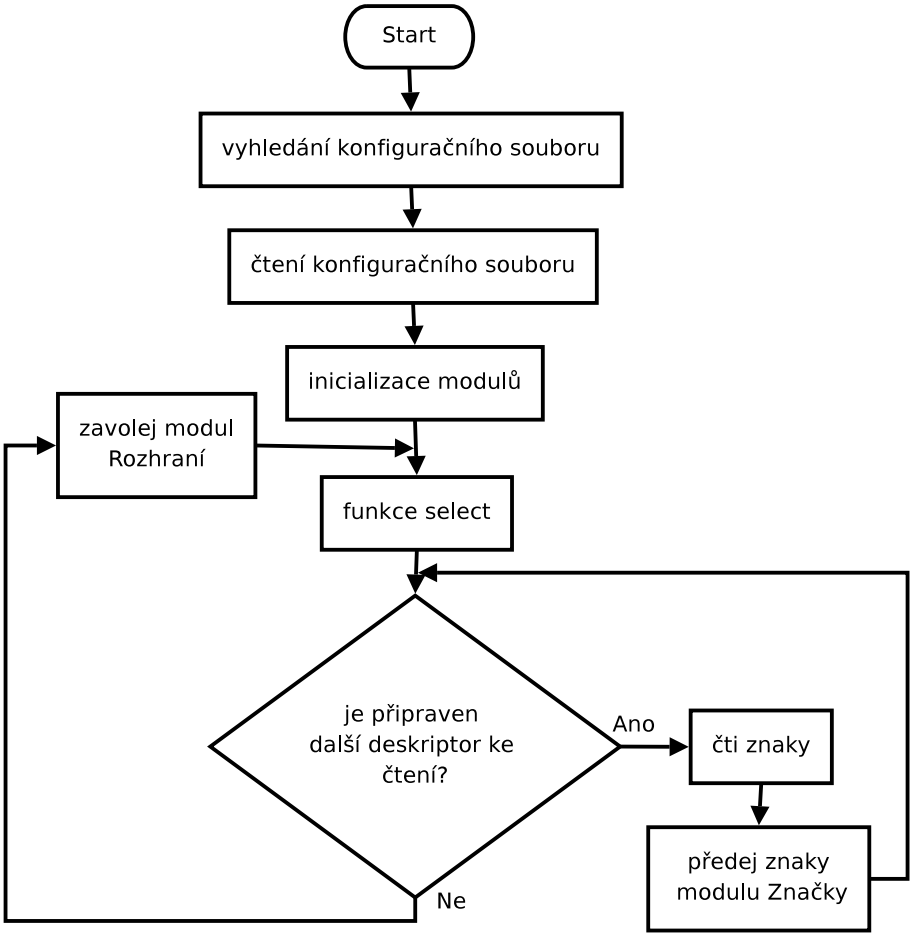
## 7.9 Hlavní modul – main

Modul Main obsahuje funkci `main`, která poté, co je program spuštěn, zavolá funkci pro čtení konfigurace `zpracuj_soubor`, inicializační funkce ostatních modulů, otevře porty a v nekonečné smyčce z nich čte znaky předávajíc je modulu Značky, jak je naznačeno na obrázku 7.7.

Funkce main před voláním `zpracuj_soubor` nejprve změní pracovní adresář na adresář, kde se nachází samotný program, a pomocí makra `accsess`[10] otestuje exis-

[10] zdokumentováno v [11]

tenci souboru „mosek.conf“. Jestliže je soubor nalezen v aktuálním adresáři, je otevřen pomocí `zpracuj_soubor`. Pokud však neexistuje, je nejprve volána funkce `startmntfind`, která se pokusí připojit USB disk a v kořenovém adresáři najít soubor „mosek.conf“, který je následně předán funkci `zpracuj_soubor`. Nepodaří-li se soubor najít ani v kořenu USB disku, či se nepodaří disk připojit, program ohlásí chybu a ukončí se.




Obr. 7.7: Funkce main modulu Main

Funkce `select` volaná v nekonečné smyčce uspí program na zadanou dobu, nebo do chvíle, kdy je připraven ke čtení jeden nebo více ze zadaných deskriptorů. Jakmile se program probudí, přečtou se data z portů připravených pro čtení pomocí funkce `read` a předají se modulu Značky k dalšímu zpracování. Nakonec se obslouží modul Rozhraní a program je opět uspán. Protože jsou funkce `select` a `read` pomalé, není možné data číst po jednotlivých znacích, ale musí se číst větší bloky, aby se čas zpracování v průměru zkrátil. Zpracování v modulu Značky je asi desetkrát rychlejší než volání funkce `read` a asi stokrát rychlejší než zpracování funkce `select`.

Aby bylo možno zjišťovat časové poměry, byl napsán další modul, Eltime, poskytující funkci `void eltime_start(void)` pro spuštění stopek, `struct timeval eltime_stop(char *label)` pro jejich zastavení a výpis času a funkci `void init_eltime(char *path)` pro inicializaci modulu. Je-li ukazatel path roven NULL, probíhá výstup na stderr, jinak do souboru specifikovaného cestou.

Funkce `init_eltime` provede kalibraci voláním funkcí `eltime_start` a `eltime_stop` a zjištěním času nutného k jejich zpracování, který je pak odčítán od výsledku ve funkci `eltime_stop`.

Funkce `main` volá funkci `rfunguj`, aby zjistila timeout pro funkci `select` a zároveň aby probíhala obsluha uživatelského rozhraní. Jestliže je při tom nastaven příznak „RSTISK“, je na základě příznaku „NAMOUNTOVANO“ rozhodnuto, zda USB disk odpojit nebo připojit a podle toho je volána příslušná funkce modulu Uložení. Příznak „RSTISK“ je při tom vynulován a příznak „RLED2“ je nastaven, či vynulován, aby měl stejnou hodnotu jako příznak „NAMOUNTOVANO“, ovšem až v dalším cyklu, aby odpovídal skutečnému stavu.

# 8 DOKUMENTACE PROGRAMU

## 8.1 Konfigurační soubor „mosek.conf“

Soubor mosek.conf obsahuje všechna nastavení programu. Každé nastavení je formátu „parametr = nastavení“, přičemž parametr může být následován jedním nebo dvěma čísly. V takovém případě první vyjadřuje číslo kanálu (sériového portu), který má být takto nastaven a druhé číslo vyjadřuje číslo triggeru v daném kanálu, pokud nastavujeme vlastnosti triggeru. Výjimkou z tohoto pravidla je parametr *enable*, který místo druhého čísla vyžaduje jedno z klíčových slov.

Program zpracovává soubor po řádcích, takže parametry a nastavení musí být na stejném řádku a naopak nemůže být více parametrů nastaveno na jednom řádku. Prázdné řádky a řádky začínající znakem „#“ jsou přeskakovány. Mezery v libovolném množství jsou přeskakovány a na jejich počtech tedy nezáleží. Znak „=“ mezi parametrem a nastavením je pouze doporučený[1]. Program nerozlišuje mezi velkými a malými písmeny, jen zadané cesty reprezentuje tak, jak jsou.

Prvním druhem parametrů jsou parametry týkající se nastavení ukládání dat. Nemají za sebou žádné číslo, na jejich pořadí nezáleží a mohou se objevit kdekoliv v konfiguračním souboru. Smí se však vyskytnout jen jednou. Jsou to:

- *maxbuf = číslo* Parametr určuje maximální počet bloků o velikosti 1kB, které smí program alokovat pro zásobník dat v modulu Uložení, pokud se mu nedaří ukládat do souboru, a musí tak zvětšovat svůj zásobník. Paměť je uvolněna v případě, kdy ji již není potřeba a zásobník se vrací k velikosti 1kB. V případě, že je alokována maximální povolená paměť, program další přijaté znaky zahazuje.

- *savetime = počet sekund* Jestliže byl předchozí soubor založen před více sekundami, než je zadáno, je při dalším ukládání uzavřen a je založen nový soubor. Soubory jsou pojmenovávány datem a časem svého založení.

- *mntpoint = cesta* Tento parametr nastavuje cestu, do které se připojí zařízení (USB disk). Pro aktuální adresář lze použít „.“, tečku.

- *datapath = cesta* Specifikovaná cesta je rekurzivně vytvořena v adresáři *mntpoint* pokud neexistuje. Ve výsledném adresáři jsou pak ukládána data.

- *mntdevice = cesta* Je cesta specifikující soubor reprezentující v Linuxu připojované zařízení – například „/dev/sda1“ pro první partition na USB disku.

- *altmntdev = předpis* Tento parametr se využije v případě, že se opakovaně nepodaří zařízení specifikované v *mntdevice* připojit. V takovém případě program vezme za cestu řetězec *předpis*, nahradí v něm všechny znaky „*“ písmenem „a“ a všechny znaky „?“ znakem null, čímž řetězec ukončí v místě prvního „?“ a zkusí připojit zařízení reprezentované výsledkem. V případě, že se ani

[1]Program zachází se znaky „=“ jako s mezerou, a lze je tedy mezerou nahradit, ale pro přehlednost je doporučeno je použít.

to nepodaří, nahradí „?“ postupně čísly 1 až 9[2]. Pokud se ani tehdy zařízení nepodaří připojit, je „*“ nahrazena písmenem „b“ a se znakem „?“ se zachází jako u písmene „a“. Jestliže jsou vyzkoušena všechna písmena, pokračuje se od začátku. Parametr *mntdevice* se pak již nikdy nepoužije a je nahrazen prvním z úspěšných pokusů.

Příklad: „altmntdev = /dev/sd*?“. Při neúspěchu připojit zařízení se zkouší v tomto pořadí /dev/sda, /dev/sda0, /dev/sda1 . . . /dev/sda9, /dev/sdb, /dev/sdb0 . . .

- *fstype = souborový systém* Parametr určuje jaký souborový sytém se nachází na připojovaném zařízení. Pokud za znakem „=“ nenásledují jen mezery nebo konec řádku (tedy nic), je jako parametr použito „vfat“, což značí souborový sytém FAT. Další možné souborové systémy lze nalézt v manuálových stránkách Linuxového programu mount [13].

- *mountovat = znak* Tento parametr programu říká, zda se má vůbec pokoušet zařízení připojovat. Znak může být i součástí nějakého slova, například „Yes“, důležité je pouze první písmeno ve slově. Znaky „A“, „a“, „Y“, nebo „y“ říkají, že program má připojovat a odpojovat zařízení. Hodnoty „N“, nebo „n“, že se o to nemá pokoušet. Program pak pouze zkusí vytvořit cestu *datapath* v místě *mntpoint*, ale nezkouší nic připojovat ani odpojovat a parametry *mntdevice* a *altmntdev* ignoruje. Této možnosti využijeme v případě běhu programu na PC, kdy data ukládá na harddisk a bylo by nemilé jej odpojit.

Dalším typem parametrů jsou parametry nastavující vlastnosti sériových portů. Na jejich pořadí nezáleží až na jedinou výjimku, a to parametr *kanalu*, který musí předcházet všem ostatním z této skupiny a také všem ze skupiny nastavení zpracování. V souboru se *kanalu* nesmí objevit vícekrát. Ostatní parametry pak musí být uvedeny tolikrát, kolik kanálů používáme. Parametry jsou:

- *kanalu = číslo* Zadané číslo musí být v rozsahu 1 až 32 a říká, kolik sériových portů bude v programu použito. Ve chvíli, kdy je tento parametr zadán, si program alokuje paměť, do které bude vkládat nastavení zadané dalšími parametry a to je i důvod, proč jim tento musí předcházet.

- *device číslo = cesta* Číslo v rozsahu nula až počet kanálů zadaný v *kanalu* zmenšený o jedna[3] říká, který z kanálů parametrem nastavujeme. Cesta je cestou k souboru reprezentujícímu sériový port – například „/dev/ttyS0“ nebo „/dev/ttyUSB1“.

- *baudrate číslo = rychlost* Číslo má stejný význam jako u *device*. Parametr nastavuje přenosovou rychlost v daném kanálu. Za rychlost musí být dosazena jedna z možností „50Bd“, „75Bd“, „150Bd“, „200Bd“, „300Bd“, „600Bd“, „1200Bd“, „1800Bd“, „2400Bd“, „4800Bd“, „9600Bd“, „19200Bd“, „38400Bd“, „57600Bd“, „115200Bd“, „230400Bd“, nebo „460800Bd“. Není možné vypustit jednotku ani ji oddělit mezerou.

[2] zde již není řetězec ukončován

[3] číslovaní začíná nulou

- *bitu číslo = počet* Číslo má opět význam jako u *device*. Počet zvolený z možností „5“, „6“, „7“ a „8“ vyjadřuje, kolik bitů má jeden přenesený znak bez uvažování parity a startbitu a stopbitů.

- *stopbitu číslo = počet* Nastavuje v daném kanálu počet stopbitů. Možnosti jsou „1“ a „2“.

- *parita číslo = písmeno* Nastavuje v daném kanálu paritu na lichou pro písmeno „o“, „O“, „l“ nebo „L“, na sudou pro „e“, „E“ nebo „s“ a „S“ a žádnou pro „n“, „N“, „z“ nebo „Z“. Písmeno může být opět součástí nějakého slova – například „Zadna“.

- *kanal číslo = slovo* Říká, zda má po startu program v daném kanálu znaky ukládat, nebo zahazovat a čekat na startovací trigger. Pokud je slovo „start“, jsou znaky ukládány již od začátku. Naopak pokud je slovo „stop“, jsou zahazovány, dokud se neobjeví zvolený startovací trigger.

Poslední skupinou jsou parametry pro nastavení zpracování přijatých dat. Na jejich pořadí opět nezáleží, jen nesmí předcházet parametru *kanalu* a parametry *trigger* a *triggertype* nesmí předcházet parametru *triggeru*. Až na *trigger* a *triggertype* mají u sebe všechny jedno číslo říkající, který z kanálů nastavujeme. *Trigger* a *triggertype* následují dvě čísla, z nichž první opět říká, o který kanál jde a druhé je číslem triggeru v daném kanálu – například „trigger 1 88 = qwert“ nastaví trigger číslo 88 v prvním kanálu na slovo „qwert“. Speciální výjimkou je pak *enable*.

- *enable číslo slovo = znak* Parametr povoluje, nebo zakazuje v kanálu určeném číslem[4] zpracování určené jedním ze slov „znacky“, „triggery“ a „bloky“. Znakem může být „A“, „a“, „Y“ nebo „y“ pro povolení a „n“ nebo „N“ pro zákaz a smí být opět začátkem nějakého slova. „Znacky“ říká, že povolujeme, či zakazujeme vkládání značky v pravidelném intervalu. „Bloky“ povoluje, nebo zakazuje přidávání značky na začátek a konec souvislého bloku znaků a „triggery“ povoluje, nebo zakazuje zpracování triggerů. Nastavení jsou platná jen pro kanál určený číslem.

- *znacky_s číslo = sekundy* spolu se *znacky_us* určují minimální čas pro pravidelné vložení značky. Pokud je přijat v kanálu znak a zjistí se, že doba poslední od časové značky v tomto kanálu je menší než takto nastavený čas, je k přijatému znaku přiřazena časová značka. [5]

- *znacky_us číslo = mikrosekundy* Mikrosekundy v rozsahu 0 až 999999 se přičtou k počtu sekund zadaných v *znacky_s*.

- *bloky_s číslo = sekundy* spolu s *bloky_us* určují čas mezi dvěma znaky, po jehož překročení je k druhému z nich přidána dvojice značek s časy obou znaků.[6]

[4] popsáno výše

[5] Toto zpracování je povoleno pomocí „enable číslo znacky = A“.

[6] Toto zpracování se povoluje pomocí „enable číslo bloky = A“.

- *bloky_us číslo = mikrosekundy* Mikrosekundy v rozsahu 0 až 999999 se opět přičtou k počtu sekund zadaných v *bloky_s*.

- *triggeru číslo = počet* Nastavuje počet triggerů v daném kanálu. Počet triggerů pro každý kanál není limitován, jen musí být všechny správně nastaveny. Parametry *trigger* a *triggertype* musí být uvedeny až za tímto parametrem a musí se vyskytnout přesně tolikrát, kolik bylo nastaveno triggerů.

- *trigger číslo1 číslo2 = řetězec* Parametr nastaví posloupnost znaků, která se vyhledává v přijatých znacích kanálu daném číslo1. Číslo2 může nabývat hodnot 0 až počet triggerů nastavený pomocí *triggeru* zmenšený o jedna a říká, který trigger v daném kanálu nastavujeme. Řetězec je pak posloupností znaků, u které program zachovává velikost písmen. Pokud v řetězci narazí na znak „\“, podívá se za něho a

  – následuje-li ještě jeden znak „\“, uloží si pouze jeden znak „\“ a druhý je zahozen.

  – Následuje-li číslo 0 až 255, uloží si ASCII znak s dekadickým vyjádřením stejným jako číslo a znak „\“ a číslo zahodí.

  – Následuje-li znak „x“ nebo „X“ a za ním číslo v hexadecimálním formátu v rozsahu 0 až FF (případně ff), je opět uložen ASCII znak vyjádřený číslem a sekvence zahozena.

  Například „trigger číslo = \\5abcd\x3a\48“ bude přeložen jako „\5abcd:0“[7]. Uvedení čísla mimo rozsah není ošetřeno a je na uživateli, aby zadal číslo správně. Například „\481“ nebude přeloženo na „01“. Je nutné zadat „\48\49“, nebo vzhledem k tomu, že to jsou tisknutelné znaky, je možné i „01“.

- *triggertype číslo1 číslo2 = slovo* Parametr slouží k nastavení akce, která se provede, když je příslušný trigger nalezen. Význam čísla1 a čísla2 je stejný jako u *trigger*. Slovo může být „stop“ pro zakázání ukládání znaků v tomto (číslo1) kanálu, „start“ pro opětovné povolení a „znacky“ pro uložení časové značky s posledním znakem vyhledaného řetězce. V případě zastavení ukládání je uložena část vyhledaného řetězce až na poslední znak. Při povolení jsou zahozeny všechny znaky řetězce až na poslední, který je uložen. Není ošetřeno chování programu v případě, kdy existují dva triggery se stejným vyhledávaným řetězcem, nastavené na „start“ a „stop“.

## 8.2 Formát uložených dat

Každý záznam do souboru se skládá ze dvou částí – identifikačního byte, jehož struktura je naznačena na obrázku 8.1, a datové části o velikosti jeden až čtyři byty.

[7] První dvě \ jsou přeložena jako jedno, \x3a jako : a \48 jako 0

| Bit | Náze | Popis |
|---|---|---|
| 7 | Typ značky | 0 = absolutní, 1 = relativní |
| 6 | Číslo kanálu | MSB |
| 5 | | |
| 4 | | Číslo 0 až 31 |
| 3 | | |
| 2 | | LSB |
| 1 | Typ dat | 0 = časová značka, 1 = data |
| 0 | Označkován | 1 = předcházela časová značka |

Obr. 8.1: Struktura identifikačního byte

*Typ značky* určuje, zda jde o značku absolutní, nebo relativní a je nastaven (logická 1), jestliže jde o relativní značku. Bit je ignorován, pokud není *typ dat* časová značka.

*Číslo kanálu* je binárně (přirozený binární kód, big endian) uložené číslo přiřazené sériovému portu v konfiguračním souboru.

*Typ dat* říká, zda datové pole obsahuje záznam o čase, časovou značku, nebo přijatá data z daného kanálu. Datové pole je tříbytové pro značku relativní, čtyřbytové pro značku absolutní a jednobytové pro přijatá data. Bit je nastaven, jestliže jsou ukládána data a vynulován, jestliže je ukládána časová značka.

*Označkován* je bit, který může být při zpracování záznamu ignorován. Je nastaven, jestliže není typ dat časová značka, ale záznam s časovou značkou předcházel tomuto záznamu a vztahuje se k němu.

Pro *datové pole* platí následující:

- Jsou-li ukládána data (*typ dat = 1*), je *datové pole* dlouhé jeden byte a obsahuje data tak, jak byla přečtena ze sériového portu.
- Je-li ukládána absolutní značka (*typ dat = 0*, *typ značky = 0*) je *datové pole* dlouhé čtyři byty a obsahuje číslo (přirozený binární kód, big endian) vyjadřující unixový čas [12].
- Je-li ukládána relativní značka (*typ dat = 0*, *typ značky = 1*) je datové pole dlouhé tři byty. Obsahuje údaj o čase, který uběhl od času uloženého v předchozí značce. Čas je vyjádřen jako celistvý počet sekund uložený v prvním (horním) bytu a počet násobků třicetidvou mikrosekund uložený v dolních dvou bytech (přirozený binární kód, big endian).

Příklad programu na čtení dat je uveden v příloze B.

## 8.3 Ovládání a chování programu

Před spuštěním programu je nutné připravit konfigurační soubor, popsaný v kapitole 8.1, a umístit jej buď do adresáře společně s programem, nebo do kořenového adresáře USB disku.

Po spuštění programu je jeho běh signalizován blikáním červenou LED diodou a jestliže se podařilo připojit USB disk i rozsvícenou žlutou diodou. Blikání červené diody s periodou jedné sekundy značí bezporuchový stav. Jestliže dioda bliká rychle (interval 0,3 s), došlo k chybě, která nezapříčinila konec programu.

Jestliže dioda jen svítí, či nesvítí, došlo ke kritické chybě a program již neběží. Pro zjištění příčin je možné použít záznam z démonu syslog, pokud byl nakonfigurován, aby ukládal hlášení přicházející z „local1“.

Odpojení, či připojení záznamového média se provede stiskem tlačítka. Pokud svítí žlutá LED dioda, je médium připojeno a nesmí být vyjmuto, jinak hrozí ztráta nebo poškození dat.

## 8.4 Chyby

Kritickou chybu, která způsobila konec programu, program před ukončením vypíše na stderr. Pokud šlo o chybu v konfiguračním souboru, je součástí hlášení i číslo řádku. Před ukončením se program ještě pokusí uložit data a odpojit případný USB disk. Chybu, po které program mohl pokračovat, signalizuje zkrácením blikání červené LED diody ze sekundového intervalu na třetinu. Všechny chyby, informace a varování jsou předávány démonu syslog, který je podle nastavení může zahazovat, ukládat do souboru, zasílat přes internet nebo vypisovat uživateli.

Algoritmus vyhledávání triggerů (obrázek 5.4) nedokáže najít trigger, pokud před ním předchází posloupnost znaků shodná se začátkem triggeru. Například máme-li vyhledávat posloupnost „112“ nebo „kokoska“ a dostáváme-li například znaky „1112“, respektive „kokokoska“.

Funkce `select` je velmi pomalá a funkce `read` nevrací najednou více než 512 bytů a díky tomu program není schopen přijímat požadovanou rychlostí 115200 Bd, ale jen 57600 Bd, to však s velkou rezervou. I přesto je velmi dobře použitelný pro monitorování komunikace do této rychlosti včetně a v případě, že nejde o kontinuální provoz[8], je možno jej použít i při rychlosti 115200 Bd.

Vzhledem k tomu, že při monitorování více kanálů je funkce `select` volána pouze jednou a čtení jsou provedena tolikrát, kolik je kanálů, klesá maximální rychlost jinak než přímo úměrně počtu kanálů.

[8]Například přenos souboru většího než 50kB

# ZÁVĚR

Rád bych zde shrnul svoji činnost v průběhu vypracování této práce.

Seznámil jsem se s procesorem MCF5213 a v jazyce C pro něho napsal program pro záznam sériové komunikace ve dvou simplexních kanálech, nebo jednom duplexním při rychlosti 115200 Bd. Program je schopen přidávat k přijatým datům časové značky a výsledek uložit na připojenou kartu typu SD nebo MMC. K programu existuje grafické rozhraní pro Windows, které data z připojené karty vyčte.

Protože se během vývoje ukázalo, že daný procesor nedostačuje plně požadavkům, změnil jsem platformu a program začal psát pro FOX Board. Výsledný program běží pod operačním systémem Linux, je schopen přijímat data teoreticky až z 32 kanálů, ovšem jen do rychlosti 57600 Bd. Zaznamenávaná data mohou být opatřována časovými značkami v závislosti na konfiguraci podle až tří různých nezávislých pravidel a jsou ukládána na připojený USB disk.

Maximální rychlost by zřejmě bylo možné zvýšit přepsáním modulu Main tak, aby běžel ve více vláknech, z nichž by každé četlo data z jednoho portu. Tím by bylo možné vynechat funkci `select`.

K programu existuje i verze, kterou je možné spustit na počítači typu PC pod operačním systémem Linux.

# LITERATURA


[1] VLACH, J.: *Počítačová rozhraní*. Ben, Praha 1995

[2] *Soubor informací o RS232* [online]. 2003 [cit. 2007-05-06]. Dostupný z WWW: http://rs232.hw.cz

[3] *ColdFire Family Programmerś Reference Manual* [online]. 03/2003, Revision 3 [cit. 2007-05-06]. Dostupný z WWW: http://www.freescale.com/files/dsp/doc/ref_manual/CFPRM.pdf

[4] *MCF5213 ColdFire Integrated Microcontroller Reference Manual* [online]. 05/2005 , Revision 1 [cit. 2007-05-06]. Dostupný z WWW: http://www.freescale.com/files/32bit/doc/ref_manual/MCF5213RM.pdf.

[5] *ColdFire Overview Brochure* [online]. [cit. 2007-05-06]. Dostupný z WWW: http://www.freescale.com/files/microcontrollers/ doc/brochure/ColdFire_Overview.pdf.

[6] MATULA, J.: Multimedia karty a čo s nimi. *Praktická elektronika*. 2007, roč. 2007, č. 04, s. 19-22.

[7] *SD Media Format Expands the MAXQ2000ś Space for Nonvolatile Data Storage : APPLICATION NOTE 3969* [online]. Dec 01, 2006 [cit. 2007-05-06]. Dostupný z WWW: http://www.maxim-ic.com/appnotes.cfm/appnote_number/3969. http://pdfserv.maxim-ic.com/en/an/AN3969.pdf.

[8] *SanDisk Secure Digital Card : Product Manual*[online]. 2003. SanDisk Corporation , December 2003, Version 1.9 [cit. 2007-05-06]. Dostupný z WWW: http://www.sandisk.com/pdf/oem/ProdManualSDCardv1.9.pdf.

[9] *SD Specifications : Part 1 - Physical Layer Simplified Specification* [online]. September 25, 2006 , Version 2.0 [cit. 2007-05-06]. Dostupný z WWW: http://www.sdcard.org/sd_memorycard/Simplified%20Physical %20Layer%20Specification.PDF.

[10] *FOX Board LX documentation index* [online]. [2005] [cit. 2008-05-13]. Dostupný z WWW: http://acmesystems.it/?id=14.

[11] *The GNU C Library Reference Manual : for Version 2.7 of the GNU C Library* [online]. 2007-09-09 [cit. 2008-05-10]. Dostupný z WWW: http://www.gnu.org/software/libc/manual/html_node/index.html#Top.

[12] *Unix time* [online]. [2008] [cit. 2008-05-20]. Dostupný z WWW: http://en.wikipedia.org/wiki/Unix_time.

[13] *Manuálové stránky programu mount*. 2004-12-16. Součást linuxové distribuce Ubuntu 7.10

# A FUNKCE VYTVOR_ZNACKU

```
void vytvor_znacku(struct timeval cas, unsigned char kanal, unsigned char plna)
{
  struct timeval pomcas;
  //urceni rozdilu casu posledni a NOVY ukladane znacky
  if(rozdilcasu(&pomcas, cas, cas_posledni_znacky))
  {
    chyba_byla=1;
    syslog(LOG_DEBUG,"DEBUG: Doslo k nekauzalite pri porovnavani casu posledni znacky,
                             nekde je neobjevena chyba!");
  }
  if(jevetsi(pomcas,minuta)||plna==1)
  //neukladali jsme znacku dele jak minutu (muze se stat, pokud nejsou zadne znaky)
  {
    //absolutni, take jiz plna znacka
    hotovy_buffer[hbuflen++]=0x00|B_IDENTIFIKACE_ZNACKA| //toto je znacka
                          B_ZNACKA_SNIZENA|        //se snizenou presnosti
                          ((kanal&&0x1F)<<2);      //dolnich 5 bitu cisla kanalu
    hotovy_buffer[hbuflen++]= (unsigned char)               //pretypujeme - z intu na char
              ((((unsigned int)cas.tv_sec)  //pretypujeme - musime s tim do doby ulozeni
                                             //pracovat jako s intem
                       &0xFF000000)                    //maskujeme si zajimay byte
                       >>24);              //presuneme se ho nekam, kde se s nim da neco delat
    hotovy_buffer[hbuflen++]= (unsigned char)                  //pretypujeme - z intu na char
                       ((((unsigned int)cas.tv_sec)  //pretypujeme
                       &0x00FF0000)                    //maskujeme si zajimay byte
                       >>16);              //presuneme se ho nekam, kde se s nim da neco delat
    hotovy_buffer[hbuflen++]= (unsigned char)                  //pretypujeme - z intu na char
                       ((((unsigned int)cas.tv_sec)  //pretypujeme
                       &0x0000FF00)                    //maskujeme si zajimay byte
                       >>8);               //presuneme se ho nekam, kde se s nim da neco delat
    hotovy_buffer[hbuflen++]= (unsigned char)                  //pretypujeme - z intu na char
                        ((((unsigned int)cas.tv_sec)  //pretypujeme
                       &0x000000FF));                   //maskujeme si zajimay byte
  }
  else
  {
    //relativni/plna znacka
 //neubehlo vice jak minuta od posledni znacky - ulozime relativni znacku s "plnou presnosti"
    hotovy_buffer[hbuflen++]=0x00|B_IDENTIFIKACE_ZNACKA|   //toto je znacka
                          B_ZNACKA_PLNA|           //s plnou presnosti
                          ((kanal&&0x1F)<<2);      //dolnich 5 bitu cisla kanalu
    hotovy_buffer[hbuflen++]=((unsigned char)pomcas.tv_sec); //pocet sekund
    //ted jeste pocet mikrosekund (pocet 32us dilku ve skutecnosti)
    hotovy_buffer[hbuflen++]=(unsigned char)                  //pretypujeme
                       (((((unsigned int)pomcas.tv_usec) //prevedeme na typ int
                       >>5)  //podelime 32   -nejmensi perioda je 32us - mame jen 16 bit, musi
                  // se tam vejit 1e6 us - ulozime si tedy jen pocet 32us dilu (kompromis)
                       & 0x0000FF00)                   //vymaskujeme prislusny byte
                       >>8);                           //presuneme si ho na spravnou pozici
    hotovy_buffer[hbuflen++]=(unsigned char)                  //pretypujeme
                       (((((unsigned int)pomcas.tv_usec) //prevedeme na typ int
                       >>5) //podelime 32   -nejmensi perioda je 32us
                       & 0x000000FF));                 //vymaskujeme prislusny byte
  }
  znackovani[kanal].cas_znacky=cas; //musime si ulozit cas posledni znacky
                                    //pro porovnavani
  cas_posledni_znacky=cas;
}
```

Z typografických důvodů bylo pozměněno odsazení textu a zkráceny některé komentáře.

# B ZDROJOVÝ KÓD PROGRAMU READDATA

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define B_ZNACKA_PLNA 0x80          //jde o znacku s plnou presnosti, relativi
#define B_IDENTIFIKACE_ZNAK 0x02    //jde o znak, ne o znacku
#define B_OZNACKOVANO 0x01          //predchozi ulozena znacka patri k tomuto znaku
#define B_KANAL_MASKA 0x7C          //na techto pozicich je ulozeno cislo kanalu
int main(int argc, char *argv[])
{
  FILE *f;
  unsigned char d;
  int c; //sem cteme jednotlive byty ze souboru
  unsigned int znacka; //sem si precteme znacku
  time_t cas;
  char *conv;
  if(argc<2)
  {
    fprintf(stderr,"Zadejte soubor, ktery ma byt zpracovan\r\n");
    return(EXIT_FAILURE);
  }
  f=fopen(argv[1],"r"); //otevreme pro cteni
  if(f==NULL)           //chyba otevirani
  {
    fprintf(stderr,"Chyba pri otevirani souboru [%s]\r\n",argv[1]);
    exit(EXIT_FAILURE);
  }
  fprintf(stderr,"\r\n");
  printf("Kanal\tCas\t\tKanal\tZnak\tHexa");
  printf("\r\n");
  while((c=fgetc(f))!=EOF)//cteme prvni znak
  {
    znacka=0; //vynulovat promennou
    if((c&B_IDENTIFIKACE_ZNAK)==0) //mame znacku (celkem budeme cist 4 nebo 5 bytu)
    {
      printf("\r\n%d\t",(c&B_KANAL_MASKA)>>2); //zjistime a vypiseme cislo kanalu
      if((c&B_ZNACKA_PLNA)!=0) //plna presnost - 4 byty
      {
        c=fgetc(f); //cteme pocet sekund
        if(c==EOF)//chyba
        {
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        d=(unsigned char)c;
        c=fgetc(f); //cteme prvni byte (MSB) poctu mikrosekund
        if(c==EOF)//chyba
        {
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        znacka|=(unsigned char)c;  //precetli jsme MSB a vlozili ho do znacka
        znacka<<=8; //posuneme ho abychom mohli vlozit dalsi
        c=fgetc(f); //cteme druhy byte (LSB) poctu mikrosekund
        if(c==EOF)//chyba
        {
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        znacka|=(unsigned char)c;  //precetli jsme MSB a vlozili ho do znacka
        znacka<<=5; //jsou to nasobky 32 mikrosekund
        printf("+%4d,%6d\t",d,znacka); //vypiseme sekundy a mikrosekundy od predchozi znacky
      }
      else //snizena presnost
      {
        c=fgetc(f); //cteme prvni byte (MSB)
        if(c==EOF)//chyba
        {
```

```
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        znacka|=(unsigned char)c;  //precetli jsme MSB a vlozili ho do znacka
        znacka<<=8; //posuneme ho abychom mohli vlozit dalsi
        c=fgetc(f); //cteme druhy byte ()
        if(c==EOF)//chyba
        {
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        znacka|=(unsigned char)c;  //precetli jsme MSB a vlozili ho do znacka
        znacka<<=8; //posuneme ho abychom mohli vlozit dalsi
        c=fgetc(f); //cteme druhy byte ()
        if(c==EOF)//chyba
        {
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        znacka|=(unsigned char)c;  //precetli jsme MSB a vlozili ho do znacka
        znacka<<=8; //posuneme ho abychom mohli vlozit dalsi
        c=fgetc(f); //cteme prvni treti (LSB)
        if(c==EOF)//chyba
        {
          fprintf(stderr,"Soubor neocekavane skoncil\r\n");
          exit(EXIT_FAILURE);
        }
        znacka|=(unsigned char)c;  //precetli jsme MSB a vlozili ho do znacka
//        mame sestavenou znacku = pocet sekund od startu
        printf(" %d\t\t",znacka); //vypismeme pocet sekund od startu programu
//        cas=znacka;
//        conv=ctime(&cas);
//        conv[(strlen(conv)-1)]='\0';
//        printf("%s",conv);
      }
    }
    else //mame znak (celkem budeme cist 2 bytu)
    {
      if((c&B_OZNACKOVANO)==0) //ke znaku nepatrila predchozi znacka
      {
        printf("\r\n\t\t\t"); //netiskneme ho za znacku
      }
      else //ke znaku patrila predchozi znacka
      {
        //neodradkujeme - tiskneme za znacku
      }
      printf("%d",(c&B_KANAL_MASKA)>>2); //zjistime a vypiseme cislo kanalu
      c=fgetc(f); //cteme dalsi znak
      if(c==EOF)//chyba
      {
        fprintf(stderr,"Soubor neocekavane skoncil\r\n");
        exit(EXIT_FAILURE);
      }
      if((c>31)&&(c<127)) //stabdartni tisknutelny ASCII znak
      {
        printf("\t%c\t0x%02X", (unsigned char)c, (unsigned char)c); //vypisme precteny znak
      }
      else
      {
        printf("\t\t0x%02X",  (unsigned char)c); //vypisme precteny znak
      }
    }
  }
  printf("\r\n");
  fprintf(stderr,"Konec souboru\r\n");
  exit(EXIT_SUCCESS);
}
```