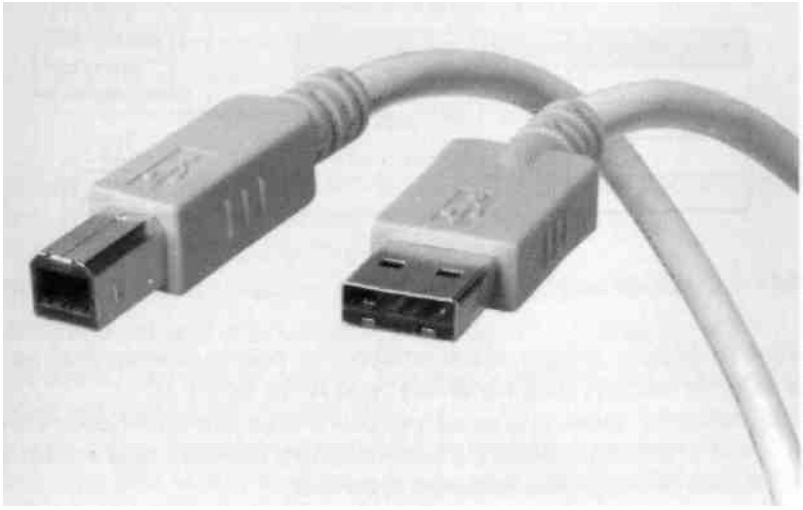
ZÁKLADNÍ POJMY USB



V této kapitole probereme a vysvětlíme základní pojmy spojené s USB. Jedná se o teoretický úvod nezbytný pro další práci s USB.

1.1 PARAMETRY USB

Základní vlastnosti:

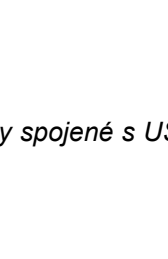
- sériové rozhraní,
- přenosové rychlosti: 1,5 Mb/s; 12 Mb/s a 480 Mb/s (viz kapitolu 1.3),
- možnost připojování zařízení na relativně velkou vzdálenost (až 5 m),
- možnost napájet zařízení přímo z konektoru (běžně lze odebrat 100 mA, po speciálním přihlášení až 500 mA),
- velký počet připojitelných zařízení (při použití hubu až 127),
- podpora Plug&Play (připojování a odpojování zařízení za provozu),
- podpora operačními systémy Windows 98/2000/Me/XP (také Linux, MAC OS-8, OS-9, OS-X).

1.2 KONEKTORY A KABELY

USB je sériová sběrnice, data se tedy přenášejí po jednotlivých bitech a to diferenčně (pro snížení vlivu rušení), datové vodiče nesou vzájemně negované signály. Napěťové úrovně jsou v rozsahu 0 až 3,3 V.

USB konektor obsahuje pouze 4 vývody (viz *obr. 1.1).*

Číslo vývodu	Význam
1	$^{+}5V(U_{CC})$
2	Data+ (přímá data)
3	Data- (negovaná data)
4	GND (zem)

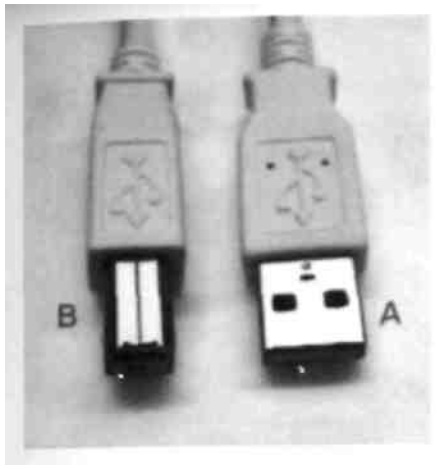


Obr. 1.1 USB konektory

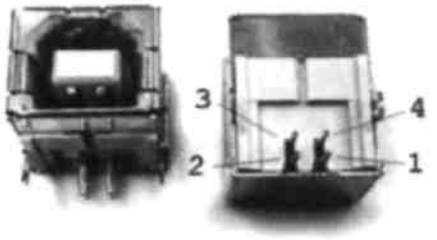
Počítač je vybaven jedním (starší notebooky), dvěma (standardně) nebo čtyřmi a více (novější modely) USB konektory typu A (viz *obr. 1.1).*

Menší zařízení (hlavně myši) používají pevně připojený kabel zakončený konektorem typu A. Rychlejší zařízení s odnímatelným kabelem mají konektor typu B a k počítači se připojují USB kabelem typu A-B.

Zřídka se používá i kabel typu **A-A** (obvykle pro spojení dvou počítačů; laplink). Vyžaduje to však použití speciálního hardware nebo základní desky.



Obr. 1.2 USB konektory A a B



Obr. 1.3 Konektor typu B do plošného spoje

1.3 VERZE USB A PŘENOSOVÉ RYCHLOSTI

USB zařízení pracují ve verzi 1.1a nověji ve verzi 2.0. Tyto standardy se z vnějšího pohledu odlišují hlavně přenosovými rychlostmi (viz *tab. 1.1).*

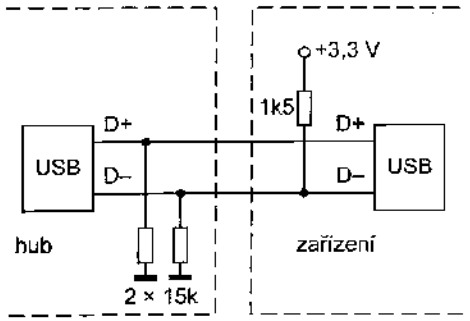
Tab. 1.1 Standardy USB a přenosové rychlosti

Rychlost	Přenosový výkon	Standard
Low Speed	**1,5 Mb/s**	USB 1.1/2.0
Full Speed	**12 Mb/s**	USB 1.1/2.0
High Speed	**480 Mb/s**	USB 2.0

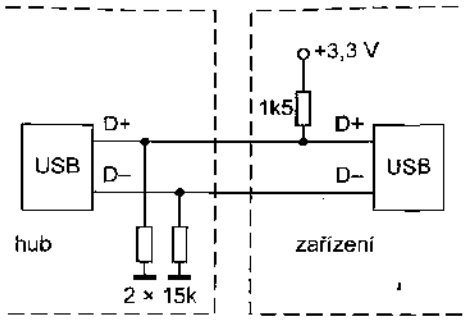
Uvedené přenosové výkony platí pro jedno zařízení. Je-li k počítači připojeno více zařízení, rozděluje se šířka pásma mezi jednotlivá zařízení. Použitou přenosovou rychlost definují sama zařízení:

- Low Speed zařízení připojuje pull-up rezistor 1,5 kQ mezi D- a 3,3 V,
- Full Speed zařízení připojuje pull-up rezistor 1,5 kQ mezi D+ a 3,3 V,
- High Speed zařízení se detekují stejně jako zařízení Full Speed s tím, že změna rychlosti se řeší programově.

Připojení pull-upů na D+ nebo D-zároveň hubu sděluje, zeje připojeno zařízení, protože jinak jsou linky taženy směrem k 0 V pomocí pull-downů (snižovacích rezistorů) velikosti 15 kQ (viz *obr. 1.4* a *obr. 1.5).*



Obr. 1.4 Low Speed zařízení



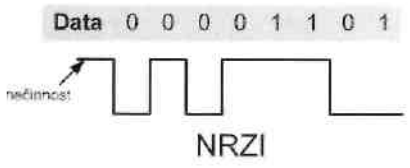
Obr. 1.5 Full Speed nebo High Speed zařízení

1.4 PŘENOS DAT A PŘÍBUZNÉ POJMY

USB je jednomasterová sběrnice, všechny aktivity vycházejí z počítače. Data se vysílají v paketech o délce 8 až 64 (1024 pro izochronní přenos) bajtů. Počítač může požadovat data od zařízení, ale žádné zařízení nemůže začít vysílat samo od sebe.

Veškerý přenos dat se uskutečňuje po rámcích o délce přesně 1 ms. Uvnitř těchto rámců mohou být postupně zpracovávány pakety pro několik zařízení. Spolu se mohou vyskytovat pakety různých rychlostí. Obrací-li se počítač na více zařízení, zajišťuje rozdělení paketů hub (rozbočovač), který také zabraňuje tomu, aby se signály s vyššími rychlostmi předaly na pomalá zařízení.

Slavě (podřízené zařízení) se musí zasynchronizovat na datový tok. Protože hodinový signál není přenášen po zvláštní lince, získávají se hodiny přenosu přímo z datového signálu. K tomu se používá metoda NRZI (Non Retům To Zero). Nuly v datech vedou ke změně úrovně, jedničky nechávají úrovně beze změny (viz *obr. 1.6).*

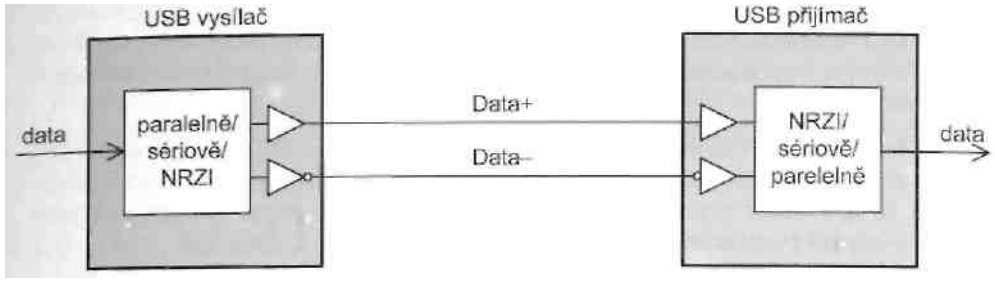


Obr. 1.6 NRZI kódování

Kódování a dekódování signálů je čistě hardwarovou záležitostí. Přijímač musí být schopen získat hodinový signál, přijmout a detekovat data. Speciální prostředky zajišťují, aby nedocházelo ke ztrátě synchronizace.

Obsahuje-li původní datový tok šest po sobě jdoucích 1, přidá vysílač automaticky jednu 0 proto, aby se vynutila změna úrovně. Tato operace je nutná pro obnovení hodinového kmitočtu z datového signálu. Této technice se říká bit-stuffing (vsouvání bitů). Přijímač pak tuto nadbytečnou 0 odstraní (bit-unstuffing).

Každý paket obsahuje za účelem synchronizace speciální bajt, tzv. sync-bajt (00000001b). Přijímač pak v důsledku NRZI a vsouvání bitů vidí 8 střídajících se bitových stavů, na které se snadno zasynchronizuje. Během následujícího přenosu musí být synchronizace zachována.



Obr. 1.7 Vysílač a přijímač USB

Vysílač i přijímač se v každé součástce objevují společně. Zařízení obsahuje jednotku SIE (Seriál Interface Engine), která přebírá vlastní práci. K výměně dat mezi SIE a zbytkem zařízení slouží buffery (vyrovnávací paměti) FIFO. Architektura FIFO představuje paměti schopné postupně přijímat a vysílat data podobně jako posuvné registry (údaje zapsané vysílačem do FIFO se vysílají v pořadí zápisu, údaje přečtené přijímačem se čtou v pořadí příjmu). FIFO umožňuje vzájemně sladit rozdílné rychlosti USB sběrnice a USB zařízení.

Existují mikrokontroléry, u nichž je SIE jejich přímou součástí (například AN2131). Existují také speciální obvody (PDIUSBD11, PDIUSBD12), které se k mikrokontroléru připojují například přes sběrnici I^2C.

Zařízení USB má obecně několik pamětí FIFO, jejichž prostřednictvím je možno přenášet data. K adrese zařízení se pak navíc přidává adresa tzv. koncového bodu (endpointu). Tato adresa udává, kam se data mají uložit nebo odkud se mají vyzvednout (udávají použitou FIFO). Například myš má vždy endpoint 0 (používá se při inicializaci) a endpoint 1 (sem mikrokontrolér zapisuje užitková data a počítač si je vybírá).

USB programy tvoří tzv. trubice (pipes) k jednotlivým endpointům. Jedna pipe pak představuje logický kanál k jednomu endpointu v jednom zařízení. Pipe si tedy můžeme představit jako datový kanál tvořený jedním vodičem (ve skutečnosti jsou však data z pipe přenášena jako datové pakety v milisekundových rámcích a hardware je směruje do reálné paměti podle jejich endpointu). Jedno zařízení může současně používat více pipes, takže přenosová rychlost pak vzroste.

1.5 TYPY PŘENOSŮ NA USB

Pro USB jsou definovány čtyři typy přenosu dat: ■ Řídicí přenos (Control transfer) - používají se k řízení hardware, pracují s vysokou prioritou a automatickým zabezpečením chyb; přenosová rychlost je vysoká, na jeden dotaz lze přenést až 64 bajtů,

- Přenos přes přerušení (Interrupt-Transfer) - používají zařízení, která periodicky vysílají menší množství dat (například myš nebo klávesnice). Počítač se periodicky (například každých 10 ms) dotazuje na nová data. Typicky se najeden dotaz přenese 8 bajtů,
- Hromadný přenos (Bulk-Transfer) -je vhodný pro přenos velkých množství dat se zabezpečením. Priorita přenosu je nízká, takže tento typ není vhodný pro časově kritické operace (typické použití: tiskárna, skener, externí ZIP),
- Izochronní přenos (Isochronous-Transfer) - je vhodný pro přenos velkých množství dat definovanou rychlostí (nejvyšší priorita) bez jejich zabezpečení. Je vhodný pro systémy, kdy je chyba přenosu menším zlem než jeho výpadek (například vnější zvukové karty).

1.6 ENUMERACE - ROZPOZNÁVÁNÍ ZAŘÍZENÍ

USB podporuje Plug&Play, takže každé USB zařízení, které připojíme, musí být automaticky rozpoznáno operačním systémem.

Po připojení zařízení se operační systém dotáže na parametry USB zařízení. To je nutné proto, aby byl vybrán odpovídající ovladač (při prvním připojení se ovladač také často instaluje). Potom je zařízení ohlášeno a obdrží svou sběrnicovou adresu. Tyto operace provádí operační systém zcela samostatně, ani uživatelský program ani uživatel nemusí nic dělat (snad kromě vložení diskety s ovladačem při prvním připojení zařízení).

Enumerace (vyčítání parametrů) zařízení spočívá v tom, že se operační systém dotazuje nově připojeného zařízení na určité parametry ve formě tzv. deskriptorů (přesně definovaných bloků dat). Počítač tato data požaduje prostřednictvím odpovídajících řídicích dotazů na endpoint 0.

Hub rozpozná připojení nového zařízení tak, že dojde ke „zdvihnutí" linky D+ nebo D-, potom se provedou následující kroky:

1) Hub informuje hostitelský počítač (host), že je připojeno nové zařízení,
2) host se dotáže hubu, na který port je zařízení připojeno,
3) host nyní ví, na který port je zařízení připojeno a vydá příkaz tento port aktivovat a provést reset USB sběrnice,
4) hub vyvolá USB reset (nulovací signál) o délce 10 ms a uvolní pro zařízení proud 100 mA. Jednotka SIE následně vyvolá reset mikrokontroléru a tak je zařízení připraveno,
5) než zařízení obdrží vlastní sběrnicovou adresu, je možno se na něj obracet přes implicitní adresu 0. Host čte první bajty deskriptoru zařízení, aby stanovil délku datových paketů,
6) host přiřadí zařízení jeho sběrnicovou adresu,
7) host si pomocí nové sběmicové adresy načte všechny informace obsažené v deskriptoru zařízení,

8) host přiřadí zařízení jednu z možných konfigurací. Zařízení pak může odebírat tolik proudu, kolik je stanoveno v aktivovaném konfiguračním deskriptoru. Tím je tedy připraveno k použití.

Řídicí dotazy jsou uloženy do endpointu 0. Po analýze se rozpozná druh dotazu, určité klíčové bajty datového paketu pak definují požadavek na deskriptor zařízení. Mikrokontrolér odpoví zápisem deskriptoru do výstupního endpointu 0, odkud si je hostitel přečte.

Po prvním přístupu obdrží zařízení definitivní sběrnicovou adresu, která se musí zapsat do SIE proto, aby mohly být přijímány následující datové pakety směrované na zařízení.

1.7 NEJDŮLEŽITĚJŠÍ POLOŽKY DESKRIPTORU ZAŘÍZENÍ

Deskriptor zařízení je přesně definován například v [19]. Na tomto místě se pouze seznámíme s významem nejdůležitějších položek deskriptoru.

Tab. 1.2 Deskriptor zařízení

Položka	Význam
VID (Vendor ID)	číselný identifikátor výrobce (16bitové číslo přidělované organizaci USB)
PID (Product ID)	číselný identifikátor výrobku (16bitové číslo určené výrobcem)
Manufacturer ID	řetězec identifikující výrobce
Manufacturer	řetězec popisující výrobce
Product	řetězec popisující výrobek
Seriál Number	řetězec sériového čísla (umožní připojit několik stejných výrobků)
počet konfigurací	počet konfiguračních deskriptorů, které se například liší odběrem

Kromě deskriptoru zařízení se používají konfigurační deskriptory (obsahují například informace o požadovaném odběru).

1.8 HUBY - ROZBOČOVAČE

Pro připojení několika zařízení k jednomu portu počítače se používá hub - rozbočovač. Rozlišujeme dva typy hubu:

- kořenový hub - obsahuje je jej každý počítač, umožní mu realizaci dvou (i více) portů,
- externí hub s jedním portem pro připojení k počítači (upstream port) a čtyři i více porty pro spojení se zařízeními či dalšími huby.

Každý hub a jeho kabel zvyšuje zpoždění signálu, které nesmí překročit jistou maximální hodnotu. Proto lze maximálně zapojit 7 hubu, maximálně 127 zařízení.

Hub také zásobuje zařízení napájecím napětím. Při startu může každé USB zařízení odebrat maximálně 100 mA. Pokud to nestačí, žádá o zvýšený proud pomocí konfiguračního deskriptoru, celkově lze odebrat proud maximálně 500 mA.

Externí hub může pro zařízení napájené ze sběrnice USB dodávat proud pouze 100 mA, protože celkový odběr nesmí přesáhnout 500 mA a samotný hub má také určitou spotřebu.

POPIS OBVODU FT232BM

*Firma **FTDI Chip** vyrábí obvody **FT232BM** a **FT245BM** pracující jako konvertory USB - UART a USB - FIFO. Spojují tedy možnosti sběrnice USB spolu s jednoduchým připojením vnějších zařízení, protože komunikace sériovým asynchronním kanálem nebo po paralelní sběrnici je jistě snazší než po sběrnici USB. Cenová relace je do 200 Kč za kus.*

Obvody **FT232BM** a **FT245BM** jsou již druhou generací populárních USB konvertorů. Tyto součástky však pouze nepřidávají nové funkce do svých předchůdců **(FT8U232AM** a **FT8U245AM),** ale navíc zachovávají částečnou vývodovou kompatibilitu a redukují počet vnějších součástek. Tím se snižují náklady na vývoj a výrobu zařízení a otevírají se nové možnosti v dalších aplikačních oblastech.

V dalším textu se zaměříme na konvertor **FT232BM** a to z několika důvodů (na doprovodném CD-ROM naleznete popis všech obvodů vyráběných firmou FTDI Chip):

- řada A (tj. obvody **FT8U232AM** a **FT8U245AM)** se v současnosti jeví jako zastaralá (nepodporuje například režim **Bit Bang),**
- použití ve spojitosti s mikrokontroléry je jednodušší než u obvodů **FT245BM** (mikrokontroléry jsou obvykle vybaveny jednotkou **UART,** navíc komunikace po dvou linkách zabere méně vývodů než 8bitová paralelní komunikace),
- **FT232BM** může pracovat i v paralelním módu (v režimu **Bit Bang).**

2.1 ZÁKLADNÍ VLASTNOSTI FT232BM

Tato kapitola uvádí stručně výčet vlastností obvodu FT232BM.

Hardwarové vlastnosti:

- jednočipový převodník USB <^ UART,
- plný handshake a plné rozhraní signálů modemu,
- podpora 7/8bitového přenosu, 1/2 stop-bitů a parity (lichá/sudá/značená/ mezerová/bez parity),
- přenosová rychlost nastavitelná v širokých mezích:
  - 300 Bd až 3 MBd (TTL),
  - 300 Bd až 1 MBd (RS232),
  - 300 Bd až 3 MBd (RS422/RS485),
- přijímací buffer hloubky 384 B, vysílací buffer hloubky 128 B (zajištění vysoké propustnosti dat),
- nastavitelný time-out přijímače,
- podpora X-On/X-Off handshake,
- zabudovaná podpora pro událostní znaky a přerušení linky,
- automaticky řízený vysílací buffer pro rozhraní RS485,
- podpora režimů USB suspend/resume pomocí signálů SLEEP# a RI#,
- podpora pro napájení USB zařízení s vysokým odběrem pomocí signálu PWREN#,

- integrovaný konvertor úrovní UART a řídicích signálů pro 5 V a 3,3 V logiku,
- integrovaný regulátor 3,3 V pro USB obvody,
- integrovaný obvod Power-On Reset,
- integrovaná násobička kmitočtu ze 6 na 48 MHz (fázový závěs PLL),
- režimy přenosů Bulk a izochronní USB,
- jednoduché napájení v rozsahu 4,4 až 5,25 V,
- kompatibilita se standardy USB 1.1a USB 2.0 (částečná, nedokáže zajistit např. přenosovou rychlost 480 Mb/s),
- možnost uložení VID, PID, sériového čísla a popisu výrobku do vnější E^2PROM,
- E^2PROM programovatelná přímo v aplikaci přes USB.

***VCP ovladače** pro operační systémy (umožňují práci obvodu FT232BM jako virtuálního sériového portu):*

- Windows 98 (včetně Windows 98 SE),
- Windows 2000/Me/XP,
- Windows CE,
- MAC OS-8, OS-9, OS-X,
- Linux 2.40 a vyšší.

***D2XX ovladače** (přímé ovladače, poskytují další funkce pro operační systémy):*

- Windows 98 (včetně Windows 98 SE),
- Windows 2000/Me/XP.

Aplikační oblasti:

- převodníky USB <=> RS232, USB <=> RS422/RS485,
- rozhraní pro mikrokontroléry připojené přes USB,
- přenos dat do programovatelných součástek přes USB,
- USB hardwarové modemy,
- USB bezdrátové modemy,
- a mnoho dalšího.

2.2 ROZŠÍŘENÍ SCHOPNOSTÍ V ŘADĚ B

Tato kapitola shrnuje rozšíření zavedená s druhou generací obvodů proti předchůdci FT8U232AM.

Integrovaný obvod Power-On Reset (POR)

Obvod nyní obsahuje vnitřní funkci POR. Zachovaný vývod RESET# (vstup) lze použít pro reset vnějším zařízením. Obvykle se RESET# ponechává nezapojený

(není to však doporučeno) nebo se připojuje přímo k VCC. Dále byl přidán vývod RSTOUT# (výstup), který dovoluje generovat nulovací signál pro vnější mikrokon-trolér nebo jiné součástky (dříve byl vývod odpovídající RSTOUT# použit jako TEST).

Integrovaný obvod RCCLK

U předchozí verze byl požadován vnější RC obvod, který zajistil ustálení kmitů oscilátoru a PLL. Nyní je tento obvod zabudován na čipu. Dřívější vývod RCCLK je nyní označen jako TEST a při normálním provozu musí být spojen s GND.

Integrovaný konvertor úrovní UART a řídicích signálů modemu

Předchozí verze pracovala pouze s 5V logikou. Nyní je k dispozici vývod VCCIO dovolující definovat napěťové úrovně jako 3,3 V nebo 5 V (není tedy nutný vnější konvertor úrovní).

Zdokonalený power management pro napájení zařízení s vysokou spotřebou přímo z USB

Předchozí verze měly vývod USBEN, který se stal aktivním po úspěšné enumera-ci USB. Pro zajištění řízení spotřeby musel být tento signál spojen s vývody SLEEP# a RESET#. Tato operace je nyní řešena přímo na čipu a USBEN je nahrazen signálem PWREN#, který lze použít pro přímé řízení P-kanálového MOSFETu (v aplikacích, kde je požadováno připojování větších zátěží).

Nová volba vytvořená na základě E^2PROM dovoluje připojenému zařízení jemně odpojit jeho linky UART rozhraní v okamžiku odpojení napájení (PWREN# = 1). Protože napětí se po jeho odpojení snižuje pozvolně (projeví se působení filtračních a blokovacích kondenzátorů), je nutno zařízení bezpečně resetovat přes vývod RSTOUT# = 1.

Nízký odběr v režimu USB suspend

Integrovaný RCCLK spolu s návrhem čipu snižuje odebíraný proud v režimu USB suspend na 200 uA (vyjma odběru 1,5k pull-upu na vývodu USBDP). To přináší větší rezervu pro periferie (limit odběru pro režim USB suspend je 500 uA).

Podpora izochronního přenosu dat

USB bulk přenos je obvykle nejlepším výběrem přenosu dat, ale přesné časování přenosu není zaručeno. Pro aplikace, kde je správné časování přenosu důležitější než datová integrita (například při přenosu audio nebo video dat s úzkou šíří pásma), lze zvolit izochronní přenos (přesné časování s možností porušení datové integrity). Ovlivní se volbou v E^2PROM.

Programovatelný time-out přijímacího bufferu

V předchozí verzi byl time-out přijímacího bufferu použitý pro spláchnutí zbývajících dat z přijímacího bufferu pevný (16 ms). Nyní je time-out programovatelný přes USB od 1 do 255 ms (po 1 ms krocích), takže dovoluje lepší optimalizaci pro protokoly vyžadující kratší doby odezvy u kratších datových paketů.

Pevné časování TXDEN

Časování TXDEN je nyní pevné, čímž se odstraní vnější zpoždění dříve potřebné pro RS485 aplikace na vysokých přenosových rychlostech. TXDEN nyní pracuje korektně v průběhu vysílání přerušovací podmínky.

Uvolněné blokování VCC

Nová verze vkládá blokovací kondenzátory na čip. Tím se sice sníží rušení s ohledem na normy FCC, CE a EMI, zjednoduší návrh desky, ale neodstraní nutnost použití vnějších blokovacích kondenzátorů!

Zdokonalená předdělička přenosových rychlostí

Předchozí verze předděličky podporovala dělení (n+0), (n+0,125), (n+0,25) a (n+0,5), kde n je celé číslo v rozsahu 2 až 16384.

Nyní byly připojeny další dělicí poměry: (n+0,375), (n+0,625), (n+0,75) a (n+0,875), které usnadňují volbu nových přenosových rychlostí (hlavně vysokých hodnot).

Dělení předděličky 1

Předchozí verze měla problém v případě, když byla celočíselná část dělitele rov-na 1 (n = 1). Druhá generace tento problém vyřešila a při n = 1 dosahuje přenosové rychlosti 2 MBd. Dokonce je možno nastavit n = 0, přenosová rychlost je pak 3 MBd.

Režim Bit Bang

Druhá generace má nový režim označený jako Bit Bang. V Bit Bang režimu se 8 řídicích linek UART přepne na 8bitový paralelní vstupně/výstupní port. Datové pakety posílané do obvodu se sekvenčně posílají na rozhraní rychlostí danou nastavením předděličky přenosové rychlosti. Obvod lze pak použít jako standardní obecně použitelný vstupně/výstupní řadič například pro ovládání světel, relé a spínačů a dalších zajímavých aplikací. Například je možné připojit zařízení obsahující programovatelné obvody FPGA výrobců Altera nebo Xilinx a programovat jejich SRAM.

FPGA je po resetu bez funkce, aplikační program může v režimu Bit Bang provést download konfiguračních dat do FPGA a tím definovat jeho funkci. Poté se lze přepnout do normálního režimu a naprogramovaný FPGA pak může komunikovat s počítačem přes USB. Režim Bit Bang tedy dává schopnost vytváření generických USB periferií, jejichž funkce se mění aplikačním programem. Hardwarový návrh založený na FPGA lze snadno upgradovat nebo zcela měnit pomocí konfiguračního souboru FPGA.

Nižší počet podpůrných součástek

^ Podobně jako došlo k eliminaci RC článku u vývodu RCCLK, byl eliminován požadavek 100k pull-upu pro vývod EECS. Když je obvod FT232BM použit bez konfigurační E^2PROM, lze vývody EECS, EESK a EEDATA ponechat nezapojené.

Pro obvody, které požadují dlouhý resetovací interval (součástka je resetována z vnějšku použitím resetovacího generátoru nebo je reset řízen přes vývod mikro-kontroléru), není požadován vnější tranzistorový obvod a 1,5k pull-up na vývodu USBDP se připojí proti RSTOUT# místo proti 3,3 V (vývod RSTOUT# má totiž úrovně 3,3 V logiky!).

Rozšířená podpora E^2PROM

Předchozí generace obvodů podporovaly pouze E^2PROM typu 93C46 s organizací 16 bitů. Nové obvody jsou schopny pracovat i s typy 93C56 a 93C66. Nadbytečný prostor není používán obvodem a lze jej využít pro vnější mikrokontrolér v průběhu resetu FT232BM.

Standard USB 2.0

Nová volba poskytovaná E^2PROM dovoluje, aby obvod FT232BM vracel deskriptor na úrovni standardu USB 2.0 nebo USB 1.1.

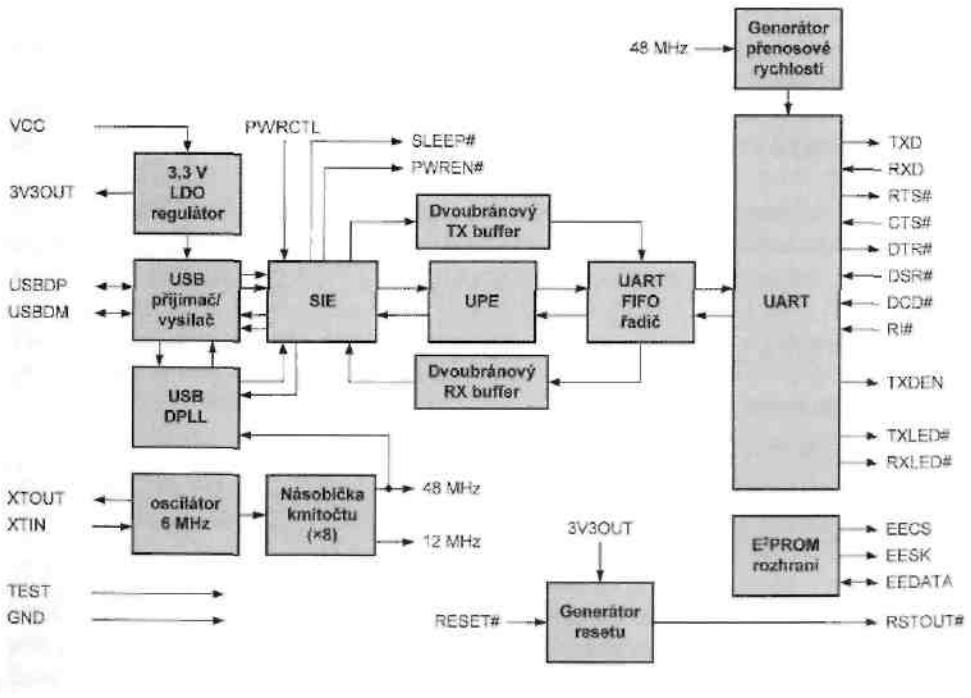
Nicméně přenosová rychlost nemůže být 480 Mb/s (high speed), ale maximálně 12 Mb/s (full speed). Prakticky dosažené rychlosti jsou však mnohem nižší.

Podpora více obvodů bez nutnosti použít E^2PROM

Pokud je připojen obvod FT232BM bez E^2PROM (nebo s prázdným či neplatným obsahem), nevkládá FT232BM sériové číslo do deskriptoru USB. To podle výrobce dovoluje, aby bylo k jednomu počítači připojeno více obvodů FT232BM i bez E^2PROM. Nicméně stále je doporučeno E^2PROM používat, protože bez ní lze zařízení identifikovat pouze z adresy USB portu, ke kterému je připojeno. Praktické testy však ukazují, že použití E^2PROM je nutností.

2.3 BLOKOVÉ SCHÉMA FT232BM

Na *obr. 2.1* je uvedeno blokové schéma obvodu FT232BM.



Obr. 2.1 Blokové schéma obvodu FT232BM

3 3V LDO regulátor (LDO - Low Drop-Out)

Generuje 3,3 V referenční napětí pro buzení USB vysílače. Vyžaduje vnější blokovací kondenzátor připojený mezi vývody 3V30UT a GND. Také zajišťuje napětí 3,3 V pro vývod RSTOUT#.

Lze jej použít i pro buzení vnějších logických obvodů pracujících s 3,3V logikou do odběru 5 mA. *USB přijímač/vysílač*

Poskytuje fyzické rozhraní pro USB kabel.

USB DPLL

Provádí detekci hodinového a datového signálu z příchozího NRZI kódování USB.

Oscilátor 6 MHz

Generuje referenční hodinový kmitočet 6 MHz, který je odvozen z vnějšího krystalu nebo keramického rezonátoru.

Násobička hodin

Ze 6 MHz vytváří referenční kmitočet 12 MHz pro SIE, UPE a UART FIFO. Také generuje 48 MHz referenční hodiny pro USB DPLL a generátor přenosové rychlosti.

SIE (Seriál Interface Engine)

Provádí paralelně-sériovou a sériově-paralelní konverzi USB dat. Ve shodě se standardem USB 1.1 zajišťuje vkládání a vyjímání synchronizačních bitů a CRC5/ CRC16 generaci/testování v datovém proudu USB.

UPE (USB Protocol Engine)

Spravuje datový proud z řídicího koncového bodu USB.

Dvoubránový TX buffer (128 B)

Data z výstupního koncového budu USB se ukládají do dvoubránového TX bufferu odkud jsou vyjímána vnějším vysílacím registrem UART pod správou UART FIFO řadiče.

Dvoubránový RX buffer (384 B)

Data z přijímacího UART registru se ukládají do dvoubránového RX bufferu před tím, než jsou vyjmuta SIE při dotazu na data ze vstupního koncového bodu.

UART FIFO řadič

Ovládá přenos dat mezi RX/TX buffery a vysílacím/přijímacím registrem UART. *UART*

Zajišťuje 7/8bitovou paralelně-sériovou a sériově-paralelní konverzi dat na RS232 (RS422/RS485) rozhraní. Řídicí signály podporované jednotkou UART zahrnují RTS, CTS, DSR, DTR, DCD a RI. UART poskytuje aktivační signál vysílače (TXDEN) k ovládání RS485 vysílačů.

UART podporuje RTS/CTS, DSR/DTR a X-On/X-Off handshaking. Je-li hand-shaking vyžadován, je řešen hardwarově proto, aby se dosáhlo co nejkratších odezev. UART také podporuje RS232 přerušení a detekci stavu linek.

Generátor přenosové rychlosti

Obsahuje 14bitovou předdělička a 3bitový registr pro jemné nastavení přenosové rychlosti (dělí celým číslem + zlomek). Lze naprogramovat přenosové rychlosti od 300 Bd do 3 MBd.

Generátor resetu

Poskytuje spolehlivý reset po připojení napájení (power-on reset). Přídavný vstup RESET# a výstup RSTOUT# dává ostatním zařízením možnost resetovat obvod FT232BM nebo se nechat resetovat od něj.

V průběhu resetu je RSTOUT# ve vysoké impedanci, jinak je buzen ze zabudovaného regulátoru 3,3 V. RSTOUT# může být použit pro řízení 1,5k pull-upu na vývodu USBDP, je-li vyžadována zpožděná USB enumerace. Také může být použit pro reset vnějších obvodů. RSTOUT# zůstane ve stavu vysoké impedance zhruba 2 ms po tom, co VCC překročí 3,5 V a současně běží oscilátor a současně je RESET#vlog. 1.

RESET# by měl být připojen na VCC, jinak je vyžadováno připojit na něj reseto-vací obvod.

E^2PROM rozhraní

Přestože může obvod FT232BM pracovat i bez vnější E^2PROM 93C46 (93C56 nebo 93C66), doporučuje se tuto paměť připojit.

E^2PROM slouží k uložení VID, PID, sériového čísla, řetězce popisu výrobku a hodnoty odebíraného proudu. E^2PROM je také vyžadována v případě, že je k počítači připojen více než jeden obvod FT232BM (unikátní sériové číslo se pak sváže s unikátní virtuálním sériovým portem).

Další parametry zahrnují „Remote Wake Up", izochronní přenos dat, „Soft Pull Down Power-Off" a deskriptor na úrovni standartu USB 1.1 nebo USB 2.0.

E^2PROM musí být v 16bitové šíři jako například u Microchip 93LC46B nebo ekvi-valentní. Musí být schopna pracovat na rychlosti 1 Mb/s při napájení 4,4 až 5,25 V.

E^2PROM je programovatelná přímo na desce pomocí speciálního programu nebo funkcemi uživatelského rozhraní. To dovoluje osadit desku prázdnou E^2PROM a naprogramovat ji přímo při vývoji.

Není-li E^2PROM připojena (nebo je prázdná), použije obvod FT232BM výchozí hodnoty VID, PID, popisu výrobku a proudového odběru. V tomto případě nebude USB deskriptor obvodu obsahovat sériové číslo.

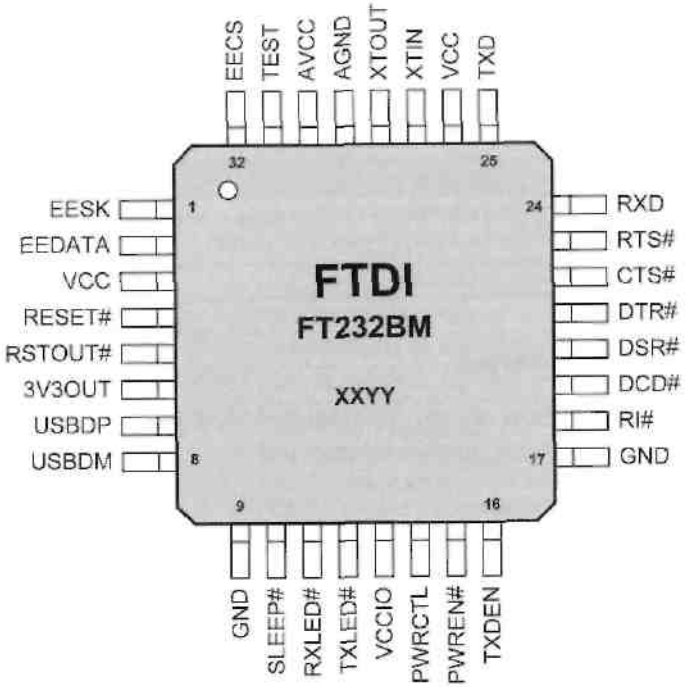
Více je uvedeno v kapitole 11.

2.4 POPIS VÝVODŮ FT232BM

Rozložení vývodů obvodu na pouzdru LQFP-32 uvádí *obr. 2.2.* Pro popis se vývody sdružují do skupin podle své funkce.

2.4.1 UART rozhraní

UART rozhraní obsahuje datové linky UART a řídicí signály modemu.



Obr. 2.2 Rozložení vývodů obvodu FT232BM na pouzdru LQFP-32 Tab.

2.1 UART rozhraní

Vývod	Signál	Typ	Popis
25	TXD	výstup	výstup vysílaných dat
24	RXD	vstup	vstup přijímaných dat
23	RTS#	výstup	signály modemu
22	CTS#	vstup	
21	DTR#	výstup	
20	DSR#	vstup	
19	DCD#	vstup	
18	RI#	vstup	
16	TXDEN	výstup	aktivuje vysílač dat pro RS485

Signál RI# se používá pro Remote Wakeup. Je-li volba Remote Wakeup aktivována (v E^2PROM), lze přechod RI# do log. 0 použít k aktivaci hostitelského řadiče z režimu USB suspend.

Po přepnutí do režimu Bit Bang lze každý z těchto vývodů konfigurovat nezávisle na ostatních jako vstup nebo výstup.

2.4.2 USB rozhraní

USB rozhraní obsahuje datové signály pro připojení k USB.

Tab.2.2 USB rozhraní

Vývod	**Signál**	**Typ**	**Popis**
7	USBDP	vstup/výstup	USB signál D+ (vyžaduje 1,5k pull-up směrem k 3V30UT nebo RSTOUT#)
8	USB DM	vstup/výstup	USB signál D-

2.4.3 E^{2}PROM rozhraní

E^{2}PROM rozhraní obsahuje signály pro připojení konfigurační E^{2}PROM. Všechny tyto vývody jsou po resetu ve třetím stavu.

Tab.2.3 E^{2}PROM rozhraní

Vývod	Signál	**Typ**	**Popis**
32	EECS	vstup/výstup	Chip Select paměti
1	EESK	výstup	hodinový signál paměti
2	EEDATA	vstup/výstup	datový vstup/výstup; datové vývody paměti propojte přes odpor 2k2, výstup musí být opatřen pull-upem 10k.

2.4.4 Řízení spotřeby

Řízení spotřeby podporuje režim USB suspend a možnost napájet zařízení přímo z USB.

Tab. 2.4 Řízení spotřeby

Vývod	**Signál**	**Typ**	Popis
10	SLEEP#	výstup	přejde do log. 0 v režimu USB suspend. Typicky se používá jako power-down (vypínač) vnějšího konvertotu TTL na RS232 pro aplikace typu konvertor USB <=> RS232
15	PWREN#	výstup	přejde do log. 0 po provedení konfigurace skrze USB; v režimu USB suspend je v log. 1. Typicky se používá pro řízení odběru vnější logiky pomoci vnějšího P-kanálového MOSFETu (musí být přizpůsoben logickým úrovním)
14	PWRCTL	vstup	PWRCTL = 0, napájeno z USB; PWRCTL = 1, napájeno z vlastního (vnějšího) zdroje

2.4.5 Pomocné signály

Pomocné signály zahrnují reset, vývody pro připojení krystalu nebo vnějšího zdroje synchronizace a další speciální funkce.

Tab. 2.5 Pomocné signály (OC značí otevřený kolektor)

Vývod	Signál	**Typ**	**Popis**
4	RESET#	vstup	lze použít pro reset FT232BM vnějším obvodem, v opačném případě připojíme na VCC
5	RSTOUT#	výstup	výstup zabudovaného generátoru resetu; zůstává ve vysoké impedanci asi 2 ms po náběhu VCC nad 3,5 V (když současně běží krystalový oscilátor), potom se napojí na vnitřní 3,3 V regulátor; aktivace RESET# = 0 vede k přechodu RSTOUT# do vysoké impedance, RSTOUT# není ovlivněn řešetem vyvolaným USB sběrnicí
12	TXLED#	výstup (OC)	vytvoří impulz do log. 0 při vyslání dat na USB
11	RXLED#	výstup (OC)	vytvoří impulz do log. 0 při příjmu dat přes USB
27	**XTIN**	vstup	vstup 6MHz oscilátoru; lze napájet z vnějšího zdroje hodin (práh přepínání je VCC/2; takže při buzení z vnějšího obvodu musí mít signál CMOS úrovně 5 V, nebo přepínací úroveň okolo 2,5 V)
28	XTOUT	výstup	výstup 6MHz oscilátoru (lze použít pro vnější obvody); v režimu USB suspend je oscilátor zastaven
31	TEST	vstup	TEST = 1 pro normální režim; TEST = 0 pro testování funkce obvodu (provádí výrobce)

2.4.6 Napájecí vývody

Napájecí vývody slouží pro připojení napájecího napětí obvodu, definici napěťových úrovní rozhraní UART a přivedení vyhlazeného napájecího napětí pro zabudovanou násobičku kmitočtu (analogová část obvodu).

Tab. 2.6 Napájecí vývody

Vývod	**Signál**	**Typ**	**Popis**
6	3V30UT	výstup	výstup zabudovaného regulátoru; musí být zablokován kondenzátorem kapacity 33 nF; přednostně je tento regulátor určen k napájení vnitřní logiky, lze jej však použít i pro napájení vnějších obvodů do odběru 5 mA
3.26	VCC	napájení	napájení 5 V jádra a zabudovaného regulátoru; rozsah 4,4 až 5,25 V
13	VCCIO	napájení	napájecí napěti pro UART rozhraní v rozsahu 3 až 5,25 V (vývody 10,11,12,14,15,16, 18 až 25); pokud UART spolupracuje s obvody 3,3 V logiky, připojíme VCCIO na vnější 3,3 V zdroj, jinak jej spojíme s VCC (pak mají vývody UART rozhraní 5 V úrovně)
9,17	GND	napájení	napájecí a signálová zem
30	AVCC	napájení	napájení zabudované násobičky hodin (analogová část)
29	AGND	napájení	napájení zabudované násobičky hodin (analogová část)

2.5 ROZMĚRY A ZNAČENÍ POUZDRA LQFP-32

Přesné rozměry pouzdra LQFP naleznou zájemci v daíasheetu obvodu FT232BM v kapitole 5.

Značení obsahuje v prvním dvoučíslí rok výroby (například pro rok 2002 platí, XX = 02). Druhé dvoučíslí obsahuje týden výroby (například YY = 45). Viz *obr. 2.2.*

2.6 MEZNÍ ÚDAJE

Na tomto místě jsou uvedeny jen mezní údaje obvodu FT232BM, charakteristické údaje naleznete v datasheetu obvodu FT232BM v kapitole 6.

Tab. 2.7 Mezní údaje

Parametr	Povolený rozsah
Skladovací teplota	-65°Caž+150°C
Provozní teplota	0 °C až +70 °C
Napájecí napětí	-0.5 V až +6 V
Vstupní napětí	-0,5 V až VCC+ 0,5 V
Výstupní proud	max. 24 mA
Výkonová ztráta (VCC = 5,25 V)	500 mW

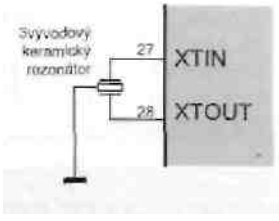
2.7 PŘÍKLADY ZAPOJENÍ OBVODU FT232BM

Tato kapitola ukazuje několik praktických příkladů, jak připojovat obvod FT232BM k vnějším součástkám.

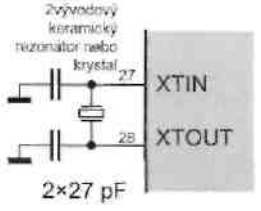
2.7.1 Připojení oscilátoru

Obr. 2.3 uvádí připojení třívývodového keramického rezonátoru (například typ Murata CSTLS6M00G53). Rezonátor má zabudované zatěžovací kondenzátory a tak nejsou nutné žádné další součástky. Typická přesnost je ±0,5 %, což je v souladu se specifikací USB.

Obr. 2.4 uvádí použití krystalu 6 MHz nebo dvouvývodového keramického rezonátoru. V tomto případě se musí použít vnější zatěžovací kondenzátory. Pro většinu krystalů je vhodná kapacita 27 pF.



Obr. 2.3 Připojení keramického rezonátoru



Obr. 2.4 Připojení krystalu

2.7.2 Připojení E²PROM

Obr. 2.5 uvádí připojení sériové E²PROM typu 93C46 (nebo 93C56, 93C66) k obvodu FT232BM.

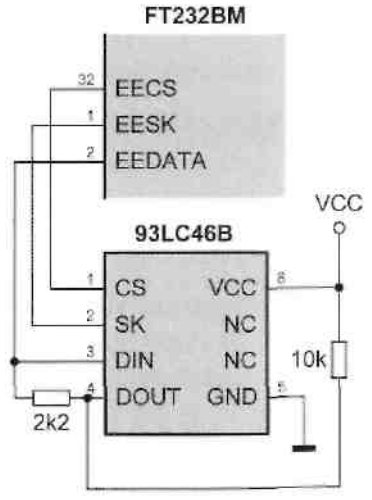
Vývod EECS je přímo napojen na signál CS paměti.

Vývod EESK je přímo napojen na signál SK paměti.

Vývod EEDATA je napojen přímo na datový vstup (DIN), na tento vývod je připojen i datový výstup (DOUT) paměti. Aby se zabránilo konfliktům, je připojení provedeno přes odpor doporučené hodnoty 2k2.

Při resetu FT232BM (vyvolaném buď připojením napájení nebo přes USB sběrnici) se testuje, zda je E²PROM připojena a zda obsahuje platná data. Pokud je obojí splněno, jsou data E²PROM použita pro definici deskriptorů USB. V opačném případě se použije výchozí hodnota.

E²PROM potvrzuje platný požadavek stažením signálu DOUT do log. 0. Pro test této podmínky je nezbytný pull-up obvyklé hodnoty 10k. Pokud není příkaz potvrzen, je tedy vývod EEDATA vytažen směrem k log. 1 a tak FT232BM detekuje neplatný příkaz (pozná typ paměti) nebo nepřipojení paměti.



Obr. 2.5 Připojení E²PROM

Existují dvě varianty E²PROM, které se liší šíří slova (8 nebo 16 bitů). FT232BM vyžaduje E²PROM 16bitové šíře, například **93LC46B** od Microchipu (taková paměť je používána v dále uvedených konstrukcích). E²PROM musí být také schopna číst data rychlostí 1 Mb/s při napájení 4,4 až 5.25 V (to splňuje většina dostupných součástek).

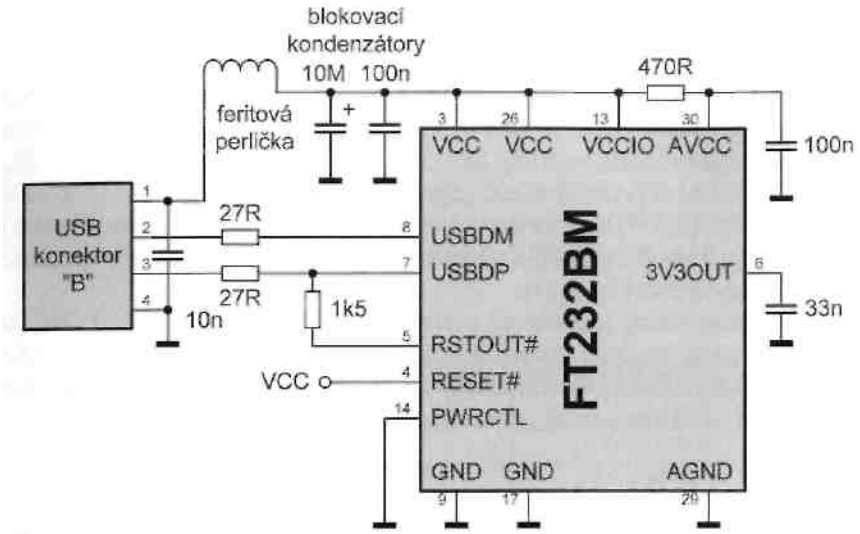
Vývody 6 a 7 E²PROM se často ponechávají nezapojené, někteří výrobci je používají pro výběr šířky slova (vývod ORG, šíře je volitelná 8 nebo 16 bitů) nebo pro testovací účely.

Je možné „sdílet" E²PROM mezi FT232BM a jejím vnějším obvodem (například mikrokontrolérem). Nicméně je to možné pouze v okamžiku, kdy není FT232BM resetován (v této chvíli se čte obsah, mikrokontrolér by měl mít příslušné vývody ve třetím stavu).

Obr. 2.6 Napájení aplikace z USB sběrnice

2.7.3 Napájení aplikace z USB sběrnice

Na *obr.* 2.6 je ukázáno typické připojení zařízení napájeného z USB sběrnice.



Základní pravidla:

a) po připojení nesmí zařízení odebírat více než 100 mA,

b) v režimu USB suspend nesmí být spotřeba vyšší než 500 uA,

c) zařízení se spotřebou vyšší než 100 mA mohou použít vývod PWREN# pro připojení vyšší zátěže po úspěšně provedené enumeraci (viz kapitolu 2.4.4),

d) zařízení odebírající více než 100 mA nemůže být připojeno k USB přes hub,

e) žádné zařízení nemůže odebrat více než 500 mA.

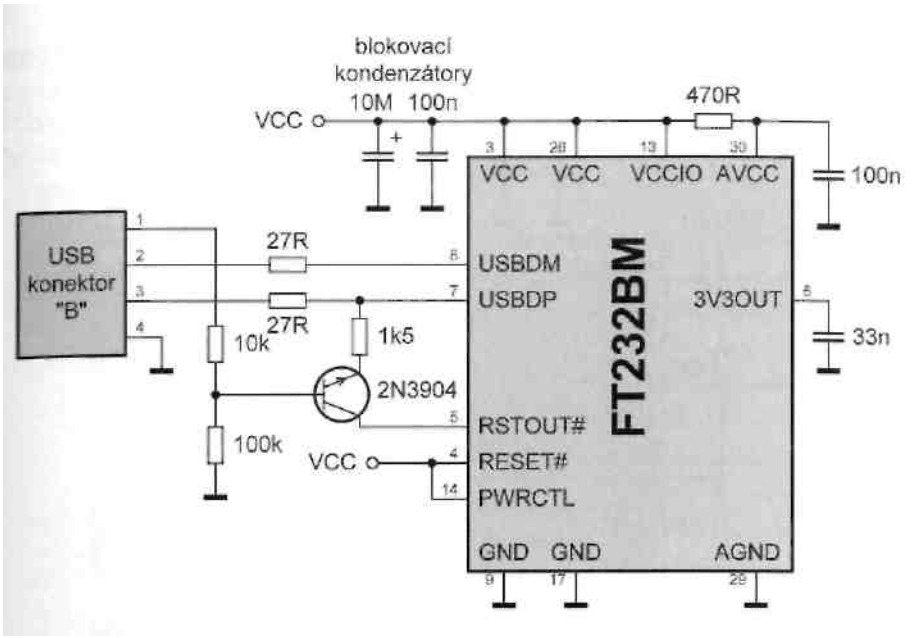
Vývod **PWRCTL** se připojí na log. 0 proto, aby se FT232BM sdělilo, že má být použit deskriptor odběru zařízení uložený v E^2PROM. Feritová perlička (připojená do série s napájením získaným z USB sběrnice) snižuje rušení FT232BM a připojených obvodů (EMI). Hodnota indukčnosti je závislá na celkovém odebíraném proudu (výrobce doporučuje navštívit stránky **www.steward.com,** kde jsou uvedeny parametry některých feritových perliček pro USB aplikace).

2.7.4 Napájení aplikace z vlastního zdroje (1)

Na *obr. 2.7* je uvedeno typické zapojení používající samostatný napájecí zdroj (není napájeno z USB).

Základní pravidla:

a) samostatně napájené zařízení nemůže způsobovat odpojení proudu USB sběrnice, když je vypnut USB hub nebo hostitel,



Obr. 2.7 Klasické řešení napájení z vnějšího zdroje

b) samostatně napájené zařízení bude mít dostatek proudu pro normální operaci a vyhoví i režimu USB suspend,

c) samostatně napájené zařízení lze použít libovolným USB hostitelem nebo hubem.

Vývod **PWRCTL** připojený na log. 1 sděluje, že se nepoužije napájecí deskriptor. Napájecí deskriptor uložený v E^2PROM pak lze naprogramovat na 0 mA.

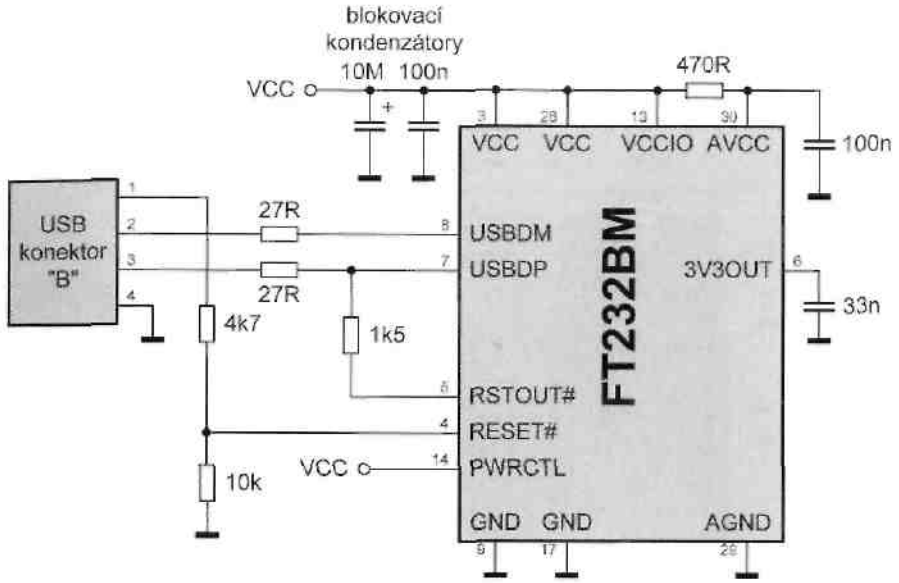
K dosažení požadavku a) musí být obvod 1,5k pull-upu modifikován, aby se zabránilo odběru proudu z vývodu USBDP přes tento zdvihací odpor, když je USB hostitel nebo hub vypnut. Pokud tato úprava nebude provedena, může to vést k chybám USB hostitele nebo řadiče.

NPN tranzistor **2N3906** (lze použít libovolný NPN tranzistor) slouží jako sonda přítomnosti napětí na USB sběrnici. Jedná se o emitorový sledovač, který je saturován, pokud je na USB sběrnici napětí. Pak se pull-up 1,5k připojí na vývod RSTOUT#. Když je USB napájení odpojeno, je tranzistor vypnut, takže nedochází k proudovému odběru z linky USBDP.

2.7.5 Napájení aplikace z vlastního zdroje (2)

Na *obr. 2.8* je uvedena jiná varianta zapojení, které je napájeno z vlastního zdroje.

Nyní je 1,5k pull-up zapojen přímo mezi USBDP a RSTOUT#. Napájení získané ze sběrnice USB je použito k řízení vývodu RESET#. Když je tedy USB hostitel nebo hub vypnut, bude RESET# v log. 0 a obvod zůstane v resetu. Když je RESET# = 0 je také RSTOUT# v log. 0, takže USBDP neodebírá z 1,5k pull-upu žádný proud.



Obr. 2.8 Jiná možnost zapojení při napájení z vnějšího zdroje

Když je FT232BM v resetu, jsou vývody UART rozhraní ve třetím stavu. Tyto vývody mají zabudovány 200 kQ zdvihací rezistory proti VCCIO a tak jsou „slabě" taženy k log. 1.

Porovnání obou řešení

- *Obr. 2.7* - FT232BM je „živý". Když se napájení na USB portu odpojí, nebude žádná aktivita na USB sběrnici a zařízení přejde do nízkoodběrového režimu spánku během několika milisekund. V této konfiguraci je vývod RE-SET# stále k dispozici pro další účely.
- *Obr. 2.8*- FT232BM je držen v resetu, když se napájení USB sběrnice odpojí. V resetu stále běží 6MHz oscilátor a zařízení má tedy určitý nezanedbatelný odběr.

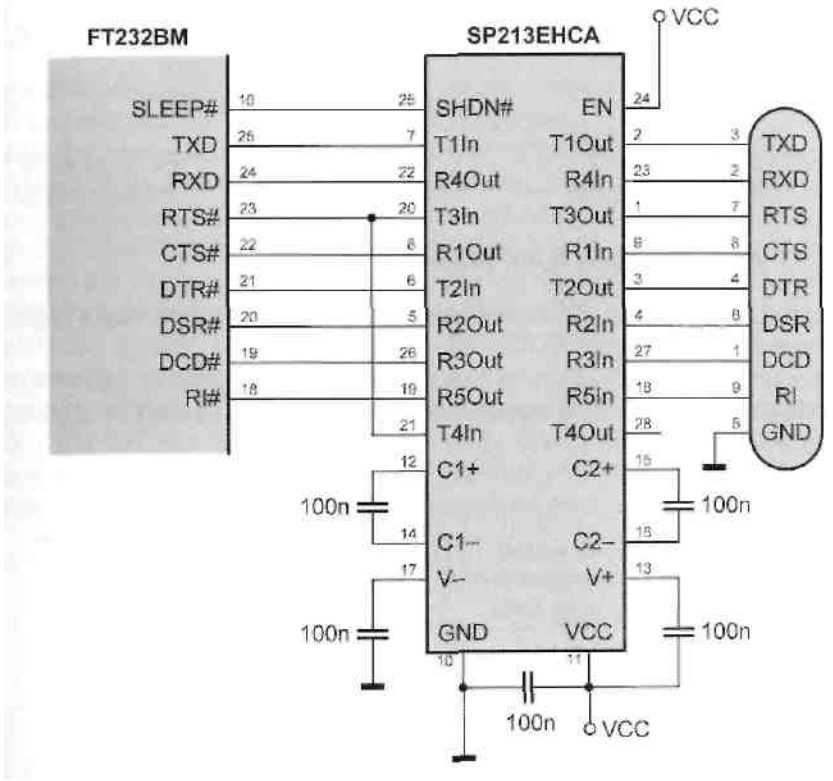
2.7.6 Zapojení UART rozhraní

Obr. 2.9 ukazuje, jak připojit UART rozhraní FT232BM k převodníku úrovní TTL na RS232 za účelem vytvoření konvertoru USB <=> RS232.

Je použit obvod populární série „213", který má 4 vysílače a 5 přijímačů ve 28vý-vodovém pouzdru SSOP (pro plošnou montáž) a zabudovaný napěťový převodník pro konverzi napájecího napětí 5 V na ±9 V. Významnou vlastností obvodu je vývod SHDN#, kterým lze řídit spotřebu v režimu USB suspend.

V příkladu zapojení je použit obvod SP213EHCA (Sipex) poskytující přenosovou rychlost až 500 kBd. Pro nižší přenosové rychlosti lze použít i obvody

SP213ECA, MAX213CAI nebo ADM213E, které pracují až do 115,2 kBd. Naopak obvod MAX3245CAI zajistí přenosovou rychlost až 1 MBd (není však vývodově kompatibilní s dříve uvedenými obvody; navíc má vývod SHDN aktivní v log. 1, takže se musí připojit místo na SLEEP# na PWREN#).



Obr. 2.9 Konvertor USB o *RS232*

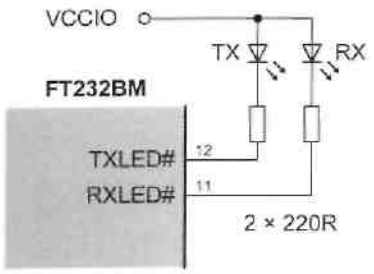
V datasheetu obvodu FT232BM jsou také uvedeny konvertory USB o RS422 a USB <=> RS485, které nebudeme popisovat.

2.7.7 Rozhraní pro připojení indikačních LED

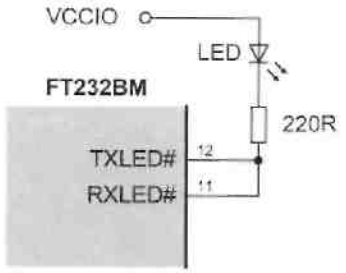
FT232BM má dva vývody určené pro připojení LED indikujících příjem nebo vysílání dat.

Při příjmu/vysílání přejde vývod RXLED#/TXLED# ze stavu vysoké impedance do log. 0 (jedná se o výstup typu otevřený kolektor), takže lze indikovat přenos dat. Pro prodloužení impulzu na délku, kterou je uživatel schopen sledovat, je použit zabudovaný digitální monostabilní klopný obvod.

Pokud chceme každou z operací sledovat zvlášť, lze zapojit dvě LED (viz *obr. 2.10).* Pokud nám stačí sledovat aktivitu sběrnice obecně, můžeme oba výstupy spojit a ovládat jimi jedinou LED.



Obr. 2.10 Připojení dvou indikačních LED

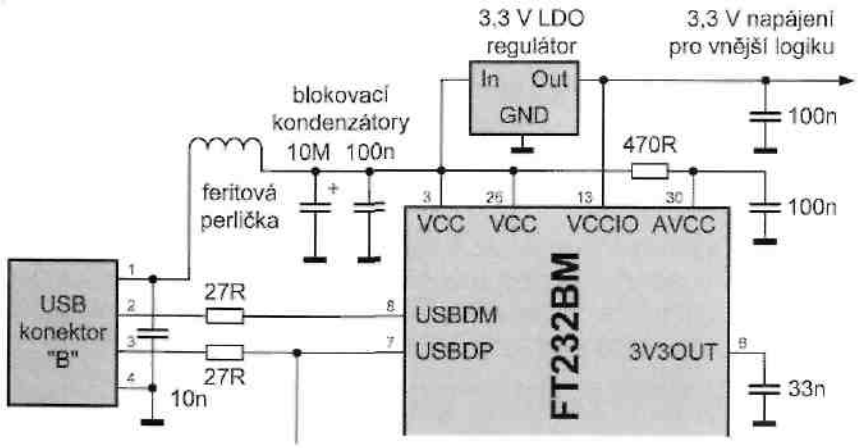


Obr. 2.11 Použití jediné indikační LED

2.7.8 Rozhraní pro 3,3V logiku

Na *obr. 2.12* je uvedeno, jak zapojit obvod FT232BM pro spolupráci s logickými obvody napájenými z 3,3 V.

Napájecí napětí pro 3,3V logiku je získáno stabilizátorem přímo z USB sběrnice. Vývod **VCCIO** je připojen na výstup tohoto stabilizátoru, takže UART rozhraní pracuje v 3,3 V úrovních.



Obr. 2.12 Připojení obvodů s 3,3V logikou

Protože je zařízení napájeno z USB sběrnice, musí být splněny určité podmínky:

a) stabilizátor musí být LDO typu (s malým průchozím úbytkem), musí totiž zajistit výstupní napětí 3,3 V i při vstupní hodnotě 4,4 V,

b) vlastní odběr stabilizátoru nemůže překročit maximální hodnotu proudu v režimu USB suspend (500 uA).

Těmto požadavkům vyhoví například stabilizátor **TC55** (Microchip), který je schopen dodat výstupní proud až 250 mA, vlastní spotřeba je 1 uA.

Když je FT232BM napájen z vnějšího zdroje, lze vývod VCCIO připojit jednoduše na 3,3 V

V některých případech je odběr 3,3 V části tak malý (<5 mA), že lze použít 3,3 V regulátor zabudovaný v FT232BM (vývod 3V30UT).

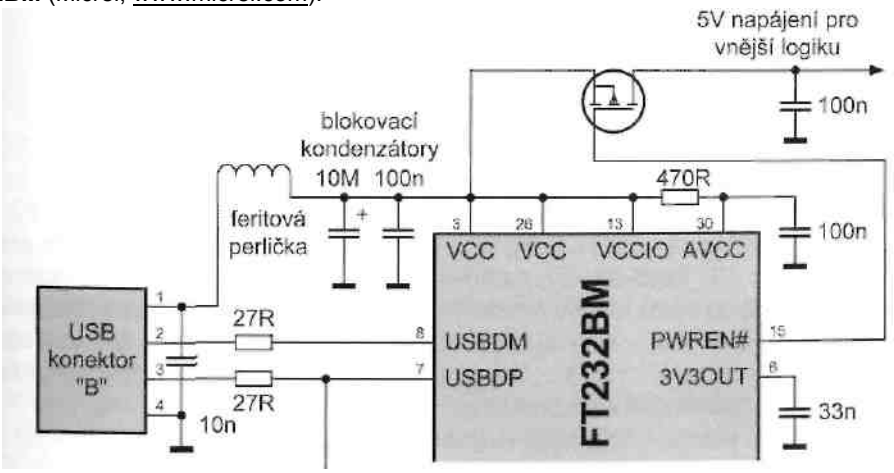
2.7.9 Řízení spotřeby vnějších obvodů

Součástky napájené z USB sběrnice musí být schopny snížit svůj odběr v režimu USB suspend pod 500 uA (včetně vnějších obvodů).

Některé obvody jsou vybaveny vývodem POWEREN#, pak stačí přivést takový vývod na výstup FT232BM označený **PWREN#.** U obvodů, které takto vybaveny nejsou, poskytuje FT232BM jinou cestu, jak snížit odběr v režimu USB suspend.

Obr. 2.13 uvádí použití P-kanálového MOSFET, který je kompatibilní s logickými úrovněmi, pro řízení spotřeby vnějších logických obvodů. Například lze použít typ **NDT456P** (www.fairchildsem.com). V dále uvedených konstrukcích byl použit na českém trhu snadno dostupný tranzistor **IRFD9120** (především pro své relativně malé rozměry).

Řešení se spínacím tranzistorem je vhodné pro řízení spotřeby do maximálního odběru 100 mA. Pro odběry nad 100 mA nebo zátěže generující při zapnutí proudové špičky, je vhodné místo MOSFET používat obvody typu „soft-start". Příkladem je obvod **MIC2025-2BM** (Micrel, www.micrel.com).



Obr. 2.13 Řízení odběru vnější logiky

Podmínky:

a) řízená logika musí mít vlastní resetovací obvod, který ji resetuje při připojení napětí následujícím po ukončení režimu USB suspend,

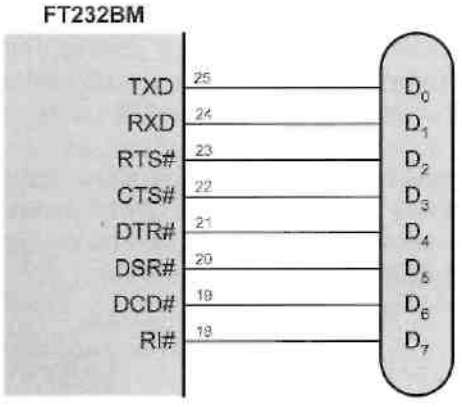
b) nastavte volbu „soft pull-up" v E^2PROM,

c) je-li VCCIO napájen z 3,3 V, nemůže dojít současně s odpojením vnější logiky k odpojení tohoto vývodu (PWREN# je totiž buzen z vývodu 3V30UT).

Buď se musí spínač zapojit mezi výstup 3,3V regulátoru a vnější 3,3V logiku nebo lze VCCIO napájet přímo z vývodu 3V30UT.

2.8 REŽIM BIT BANG

Obvod FT232BM se může programově přepnout do speciálního režimu, kdy je standardní funkce nahrazena možností přímého řízení všech 8 datových linek. Tento režim se označuje jako Bit Bang. Datové linky obvodu FT232BM lze pak chápat jako 8bitovou vstupně/výstupní sběrnici. Tento režim je vhodný například pro programování FPGA nebo ovládání jednodušších zařízení bez potřeby použít vnější mikrokontrolér.



Obr. 2.14 Chování linek UART v režimu Bit Bang

Definice vývodů v režimu Bit Bang je zřejmá z *obr. 2.14.* Každý vývod lze programově (funkcí FT_SetBitMode) nastavit do vstupního nebo výstupního režimu.

Rychlost přenosu bajtů uložených do výstupního bufferu odpovídá 16násobku zvolené přenosové rychlosti, to zároveň určuje vzorkovací kmitočet, se kterým se data ukládají do vstupního bufferu. Například přenosové rychlosti 9600 Bd odpovídá 153 600 B/s (jeden bajt za 6,5 us).

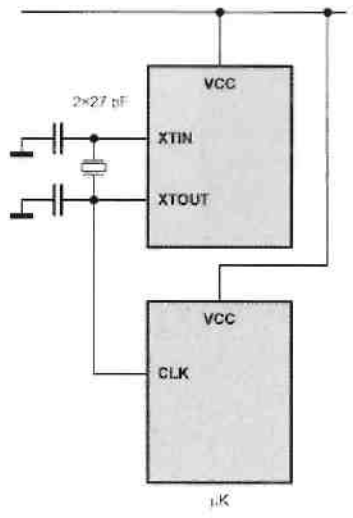
Příklad použití režimu Bit Bang je uveden v kapitole 5.7, viz také kap. 3.4.

2.9 POUŽITÍ PRACOVNÍHO KRYSTALU PRO MIKROKONTROLÉR

Možná vás napadlo, zda je možno použít krystal 6 **MHz** (potřebuje jej obvod FT232BM) například pro taktování použitého mikrokontroléru. Záměr je zřejmý-ušetřit jednu součástku a tak zlevnit cenu konstrukce.

Teoreticky je celé řešení snadné, prostě připojíme taktovací vstup mikrokontroléru na vývod XTOUT obvodu FT232BM. Nesmí být však překročeno dovolené zatížení. Jinak se oscilace utlumí a nebude pracovat ani jedna z obou součástek. Je jasné, že vstupní odběr většiny mikrokontroléru je zanedbatelný. Problém však vznikne v případě, zeje napájení mikrokontroléru odpojeno (spínacím tranzistorem, viz *obr. 2.13)* v době enumerace zařízení. Pak se projeví vliv ochranných diod, které způsobí zvýšený odběr, rozpad oscilací a v konečném důsledku neúspěšnou enumeraci.

Na *obr. 2.15* je ukázáno klasické připojení hodinového vstupu mikrokontroléru na vývod XTOUT. Problém rozpadu oscilací není řešen.

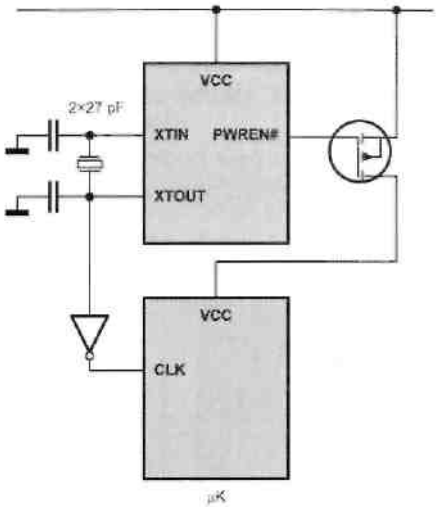


Obr. 2.15 Klasické připojení mikrokontroléru

Obr. 2.16 ukazuje jednu z možností, jak lze odpojovat napájení pro mikrokontrolér a zároveň zabránit rozpadu oscilací. Řešení je založeno na použití oddělovacího hradla (například invertoru 74HCT04 nebo 74HC04), které je připojeno na napájecí napětí konvertoru FT232BM. Přetížení výstupu tohoto hradla v době odpojení napájecího napětí pro mikrokontrolér pak není na závadu (při použití jednotného napájení 5V).

V dále uvedených konstrukcích je upřednostněna forma dle *obr. 2.15.* Cena oddělovacího hradla je totiž srovnatelná s cenou krystalu.

V řadě případů je dávána přednost použití dvou krystalů. Jedná se o konstrukce s mikrokontrolérem AT90S2313, protože ten může být taktován kmitočtem až 10 MHz (byla snaha využít plnou rychlost).



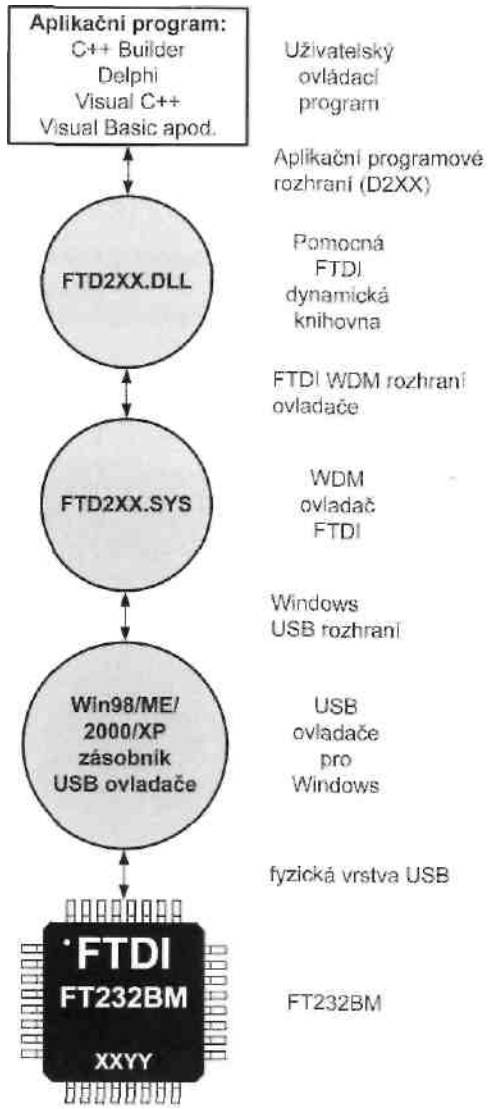
Obr. 2.16 Jedno z možných připojení při odpojováni napájení pro mikrokontrolér

3

*Pro ovládání zařízení na bázi obvodu **FT232BM** lze používat ovladače VCP (Virtual Communication Port) nebo D2XX. Prakticky stejným způsobem lze ovládat i obvod FT245BM.*

***VCP** ovladače umožňují ovládat zařízení jako virtuální sériový port (pokud jste již někdy vytvářeli aplikace pro sériový port, je tento přechod velmi jednoduchý - zařízení je sice připojeno k USB, ale chápe se jako by bylo připojeno k sériovému portu). Použití VCP ovladače je ukázáno v kapitole 12.*

*Tato kapitola je překladem referenční příručky ovládacího rozhraní **D2XX.** Rozhraní D2XX poskytuje mnohem zajímavější možnosti ovládání než VCP.*



Obr. 3.1 Architektura ovladače D2XX

3.1 ARCHITEKTURA D2XX

D2XX je přímý ovladač pro Windows, který umožňuje aplikačnímu programu komunikovat s FT232BM pomocí kódu uloženého v dynamické knihovně (DLL).

D2XX obsahuje WDM ovladač (FTD2XX.SYS), který komunikuje se zařízením přes Windows USB zásobník a DLL (FTD2XX.DLL), která je rozhraním pro aplikační programy vytvářené ve vývojových prostředích Visual C++, C++ Builder, Delphi, Visual Basic apod.

Součástí instalace jsou instalační soubory **(INF),** program na odinstalování a programátorská příručka v anglickém originále.

Nová verze D2XX ovladače obsahuje mnoho rozšíření, funkce jsou rozděleny do čtyř skupin:

- Klasické rozhraní seskupuje původní D2XX funkce, které byly pro zpětnou kompatibilitu zachovány. Jedná se o funkce, které poskytují snadný přístup k USB zařízením.
- Rozhraní E^2PROM dovoluje aplikaci číst/zapisovat do konfigurační E^2PROM 93C46 včetně možnosti použití volného prostoru pro uložení aplikačně specifických údajů.
- FT232BM rozšíření dovoluje plně využít schopností druhé generace součástek (režim Bit Bang a izochronní režim).
- FT_Win32 API definuje funkce odpovídající původním Win32 API voláním pro práci se sériovým portem. Dávají tedy možnost snadno a rychle přejít od aplikace vyvinuté pro sériový port na aplikaci s obvodem FT232BM připojeným na USB sběrnici. Klasické rozhraní a FT_Win32 API představují dva alternativní přístupy, můžete si vybrat, který chcete použít. Nedoporučuje se obě skupiny funkcí vzájemně míchat.

3.2 KLASICKÉ FUNKCE

Klasické funkce zajišťují základní možnosti přístupu na zařízení vybavené obvodem FT232BM.

Před tím, než se začne se zařízením pracovat, musíme získat jeho handle. Otevření zařízení (získáni jeho handle) zajišťují funkce FT_Open a FT_OpenEx. Poté je možno zasílat/číst data funkcemi FT_Write a WT_Read. Po dokončení operací lze zařízení zavřít voláním funkce FT_Close.

Kromě toho jsou k dispozici další funkce: FT_ResetDevice (provede USB reset zařízení), FT_Purge (vyprázdní přijímací nebo vysílací buffer), FT_SetTimeouts (nastaví komunikační time-outy), FT_GetQueueStatus (zjistí stav přijímací fronty), FT_GetStatus (zjistí stav zařízení).

Dále je možno získat výčet všech zařízení připojených k počítači pomocí funkce FTjJstDevices. Lze nastavovat charakteristiky přenosu dat, hw/sw handshake, řídicí signály modemu pomocí funkcí: FT_SetBaudRate (nestandardní přenosové rych-•osti pak pomocí funkce FT_SetDivisor), FT_SetDataCharacteristics, FT_SetFlowControl, FT_SetDtr, FT_ClrDtr, FT_SetRts, FT_ClrRts, FT_SetBreakOn, FT_SetBrakOff, FT_SetEventNotification, FT_GetModemStatus, FT_SetChars.

3.2.1 Typ FT_STATUS

Všechny níže uvedené funkce indikují úspěšnost svého provedení pomocí návratové hodnoty typu FT_STATUS. V případě úspěchu vrací konstantu FT_OK, při chybě pak jinou hodnotu dle *tab. 3.1* (tím lze rozpoznat příčinu chyby).

Tab. 3.1 Možné hodnoty FT_STATUS

Hodnota	**Význam**
FT_OK	operace úspěšně provedena
FTJNVALIDJHANDLE	neplatný handle zařízení
FT_DEVICE_NOT_FOUND	zařízení nenalezeno
FT_DEVICE_NOT_OPENED	zařízení neotevřeno
FT_IO_ERROR	vstupně/výstupní operace selhala
FT_INSUFFUCIENT_RESOURCES	nedostatečné systémové zdroje
FT_INVALID_PARAMETR	neplatný parametr operace
FT_INVALID_BAUD_RATE	neplatná přenosová rychlost
FT_DEVICE_NOT_OPENED_FOR_ERASE	není otevřeno pro smazání
FT_DEVICE_NOT_OPENED_FOR_WRITE	není otevřeno pro zápis
FT_FAILED_TO_WRITE_DEVICE	selhání zápisu
FT_EEPROM_READ_FAILED	selhání při čtení E^2PROM
FT_EEPROM_WRITE_FAILED	selhání při zápisu do E^2PROM
FT_EEPROM_ERASE_FAILED	selhání při mazání E^2PROM
FT_EEPROM_NOT_PRESENT	E^2PROM není připojena
FT_EEPROM_NOT_PROGRAMMED	E^2PROM není naprogramována
FT_INVALID_ARGS	neplatný argument
FT_OTHER_ERROR	jiná chyba

Tab. 3.2 Možné hodnoty dwFlags a jejich význam

dwFlags	**Význam**
FT_LIST_NUMBER_ONLY	Zjistí počet připojených zařízení; **pvArgl** je interpretován jako ukazatel na proměnnou typu **DWORD,** kam se uloží počet připojených zařízení
FT_OPEN_BY_SERIAL_NUMBER	Zjistí sériové číslo zařízení
FT_OPEN_BY_DESCRIPTION	Zjisti popis výrobku
FT_LIST_BY_INDEX	Lze použít pro zjištění informačních řetězců pro jedno zařízení; **pvArgl** musí obsahovat pořadové číslo (index) zařízeni (čísluje se od 0), **pvArg2** je interpretován jako adresa řetězce pro příjem informačních dat
FT_LIST_ALL	Lze použít pro zjištění informačních řetězců všech připojených zařízení; **pvArgl** ukazatel na pole adres řetězců pro načtení informačních dat (poslední adresa musí být **NULL),** **pvArg2** počet položek pole odkázaného **pvArgl**

3.2.2 FT ListDevices

Získá informace o aktuálně připojených zařízeních (počet připojených zařízení, řetězce sériových čísel zařízení a popisů výrobků). Příklad je uveden v kapitole 5.5. Hlavička:

```
FT_STATUS FT_ListDevices(
        PVOID pvArgl, PVOID pvArg2, DWORD dwFlags);
```

Parametry:

- pvArgl, pwArg2 - závislé na dwFlags,
- dwFlags - určuje formát zjišťované informace. Možné hodnoty viz *tab. 3.2.* FT_OPEN_BY_SERIAL_NUMBER nebo FT_OPEN_BY_DESCRIPTION se musí kombinovat s FT_LIST_BY_INDEX nebo FT_LIST_ALL.

3.2.3 FT_Open

Otevře zařízení a vrátí handle, přes který jej lze ovládat. Hlavička:

```
FT_STATUS FT_Open(
        INT iDevice, FTJHANDLE *ftHandle);
```

Parametry:

- iDevice - 0, pokud se otevírá jedno zařízení; případně 1, 2, atd. pro více zařízení,
- ftHandle - adresa proměnné typu FT_HANDLE pro příjem handle otevřeného zařízení.

POZNÁMKY: Úspěšnost otevření určuje návratová hodnota FT_OK.

Funkce FT_Open neumožňuje otevřít určité zařízení (uvedením jeho jména), v takovém případě použijte funkci FTOpenEx.

3.2.4 FTOpenEx

Otevře pojmenované zařízení a vrátí handle, přes který jej lze ovládat. Hlavička:

```
FT_STATUS FT_OpenEx(
        PVOID pvArgl, DWORD dwFlags, FT_HANDLE *ftHandle);
```

Parametry:

- pvArgl - význam závisí na hodnotě dwFlags,
- dwFlags - určuje, zda je zařízení identifikováno řetězcem sériového čísla (FT_OPEN_BY_SERIAL_NUMBER) nebo řetězcem popisu (FT_OPEN_BY_DESCRIPTION) zadaným v pvArgl,
- ftHandle - adresa proměnné typu FTJHANDLE pro příjem handle otevřeného zařízení.

POZNÁMKY: *Úspěšnost otevření určuje návratová hodnota* ***FT_OK.*** *Tato funkce je dobře použitelná pro otevření konkrétního zařízení.*

3.2.5 FT_GetDeviceInffo

Zjistí rozšířené informace o zařízení. Hlavička:

```
FT_STATUS FT_GetDeviceInfo(
        FT_HANDLE ftHandle, FT_DEVICE *pftType,
        LPDWORD lpdwID, PCHAR pcSerialNumber,
        PCHAR pcDescription, PVOID pvDummy);
```

Parametry:

- ftHandle - handle zařízení vrácený funkcí FT_Open nebo FT_OpenEx,
- pftType - adresa proměnné typu FT_DEVICE pro uložení typu zařízení (možné hodnoty jsou: FT_DEVICE_BM, FT_DEVICE_AM, FT_DEVICE_100AX),
- lpdwID - adresa proměnné typu DWORD pro uložení ID zařízení (dolní slovo obsahuje PID, horní slovo pak VID),
- pcSerialNumber - znakový řetězec pro příjem sériového čísla zařízení,
- pcDescription - znakový řetězec pro příjem popisu zařízení,
- pvDummy - vyhrazeno pro pozdější použití, nastavte na NULL.

3.2.6 FT_Close

Zavře zařízení určené jeho handle. Hlavička:

```
FT_STATUS FT_Close(
        FTJHANDLE ftHandle);
```

Parametry:

- ftHandle - handle zavíraného zařízení.

3.2.7 FT_Read

Čte data ze zařízení. Hlavička:

```
FT_STATUS FT_Read(
        FT_HANDLE ftHandle, LPVOID lpBuffer,
        DWORD dwBytesToRead, LPDWORD lpdwBytesReturned);
```

Parametry:

- ftHandle - handle zařízení, ze kterého se má číst,
- lpBuffer - adresa bufferu pro příjem dat,

- dwBytesToRead - počet bajtů, které chceme přečíst,
- lpdwBytesReturned - adresa proměnné typu DWORD pro příjem skutečného počtu přečtených bajtů

POZNÁMKY: *Funkce* ***FTRead*** *vždy vrací počet načtených bajtů v proměnné, na kterou ukazuje* ***lpdwBytesReturned.***

Tato funkce se nevrací, dokud není přečteno ***dwBytesToRead*** *bajtů do bufferu odkázaného* ***lpBuffer.*** *Počet bajtů uložených aktuálně v přijímací frontě lze zjistit voláním funkcí* ***FT_GetStatus*** *nebo* ***FT_GetQueueStatus.*** *Zjištěný počet bajtů lze pak předat do parametru* ***dwBytesToRead.*** *Pak se* ***FT_Read*** *vrátí okamžitě.*

Pokud je určen time-out pro čtecí operaci předchozím voláním funkce ***FT_SetTimeouts,*** *vrací se* ***FT_Read*** *po vypršení tohoto intervalu nebo pokud se přečte žádaný počet bajtů. Nastane-li time-out, přečte* ***FT_Read*** *dostupná data do bufferu a vrátí hodnotu* ***FT_OK***

Aplikace může použít návratovou hodnotu a hodnotu uloženou na adresu danou ***lpdwBytesReturned*** *k tomu, aby posoudila výsledek operace, Vrátí li* ***FT_Read*** *hodnotu* ***FT_OK*** *a* ***lpdwBytesReturned*** *=* ***dwBytesToRead,*** *proběhlo volání normálně (načetl se požadovaný počet bajtů). Vrací-li* ***FT_Read*** *hodnotu* ***FT_OK*** *a* ***lpdwBytesReturned < dwBytesToRead,*** *vypršel time-out (načetlo se pouze* ***lpdwBytesReturned*** *bajtů).*

Vrátí-li funkce ***FT_Read*** *hodnotu* ***FT_IO_ERROR,*** *došlo k hrubé chybě (například odpojení USB zařízení).*

3.2.8 FT_Write

Zapisuje data do zařízení. Hlavička:

```
FT_STATUS FT_Write(
        FTJHANDLE ftHandle, LPVOID lpBuffer,
        DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten);
```

Parametry:

- ftHandle - handle zařízení, na které se má zapisovat,
- lpBuffer - adresa bufferu zapisovaných dat,
- dwBytesToWrite - počet bajtů, které chceme zapsat,
- lpdwBytesWritten - adresa proměnné typu DWORD pro příjem skutečného počtu zapsaných bajtů.

3.2.9 FT **ResetDevice**

Pošle nulovací příkaz (USB reset) na zařízení. Hlavička:

FT_STATUS FT_ResetDevice(
FT_HANDLE ftHandle);

Parametry:

- ftHandle - handle zařízení, které se má resetovat.

3.2.10 FT SetBaudRate

Nastaví přenosovou rychlost zařízení. Hlavička:
FT_STATUS FT_SetBaudRate(
FT_HANDLE ftHandle, DWORD dwBaudRate);

Parametry:

- ftHandle-handle zařízení,
- dwBaudRate - požadovaná přenosová rychlost v Baudech.

POZNÁMKA: *Funkce* ***FT_SetBaudRate*** *umožňuje nastavení standardních přenosových rychlostí. Při nevhodné volbě vrátí* ***FT_INVALID_BAUD_RATE.*** *Pro nestandardní přenosové rychlosti použijte funkci* ***FT_SetDMsor.***

3.2.11 FT_SetDivisor

Nastaví přenosovou rychlost zařízení (vhodné pro nestandardní přenosové rychlosti).
Hlavička: FT_STATUS FT_SetDivisor(
FT_HANDLE ftHandle, USHORT usDivisor);

Parametry:

- ftHandle - handle zařízení,
- usDivisor - požadované nastavení děličky z 3 MHz.

POZNÁMKA: *Příklad použití nestandardní přenosové rychlosti je uveden v kapitole 8.1.*

3.2.12 FT SetDataCharacteristics

Nastaví charakteristiky přenosu dat pro zvolené zařízení. Hlavička:
FT_STATUS FT_SetDataCharacteristics(
FT_HANDLE ftHandle, UCHAR uWordLength, UCHAR
uStopBits, UCHAR uParity);

Parametry:

- ftHandle - handle zařízení,

- uWordLength - délka znaku, musí být FT_BITS_8 nebo FT_BITS_7 (8 nebo 7 bitů),
- uStopBits - počet stop-bitů, musí být FT_STOP_BITS_1 nebo FT_STOP_BITS_2 (1 nebo 2 stop-bity),
- uParity - použitá parita, musí být FT_PARITY_NONE, FT_PARITY_ODD, FT_PARITY_EVEN, FT_PARITY_MARK nebo FT_PARITY_SPACE (nepoužita, lichá, sudá, značená, mezerová).

3.2.13 FT_SetFlowControl

Nastaví řízení toku dat pro zvolené zařízení. Hlavička:
FT_STATUS FT_SetFlowControl(
FT_HANDLE ftHandle, USHORT uFlowControl, UCHAR
uXon, UCHAR uXoff);

Parametry:

- ftHandle - handle zařízení,
- uFlowControl - určuje zvolený způsob hardwarového řízení toku dat, musí být FT_FLOW_NONE, FT_FLOW_RTS_CTS, FT_FLOW_DTR_DSR nebo FT_FLOW_XON_XOFF (žádný, pomocí signálů RTS a CTS, pomocí signálů DTR a DSR, pomocí zapínacího a vypínacího znaku),
- uXon - zapínací znak pro uFlowControl = FT_FLOW_XON_XOFF,
- uXoff - vypínací znak pro uFlowControl = FT_FLOW_XON_XOFF

3.2.14 FT_SetDtr a FT_ClrDtr

Tyto funkce uvedou linku DTR# do log. 0 nebo do log. 1. Hlavičky:
FT_STATUS FT_SetDtr(
FTJHANDLE ftHandle);
FT_STATUS FT_ClrDtr(
FT_HANDLE ftHandle);

Parametry:

- ftHandle-handle zařízení.

3.2.15 FT_SetRts a FT_ClrRts

tyto funkce uvedou linku RTS# do log. 0 nebo do log. 1. Hlavičky:
FT_STATUS FT_SetRts(
FT_HANDLE ftHandle);
FT_STATUS FT_ClrRts(
FTJHANDLE ftHandle);

Parametry:

- ftHandle - handle zařízení.

3.2.16 FT_SetBreakOn a FT SetBreakOff

Tyto funkce uvedou linku **TXD** do log. 0 nebo do log. 1. Hlavičky:

FT_STATUS FT_SetBreakOn(FT_HANDLE ftHandle);

FT_STATUS FT_SetBreakOff(FT_HANDLE ftHandle);

Parametry:

- **ftHandle** - handle zařízení.

3.2.17 FT GetModemStatus

Zjistí stav linek modemu. Hlavička:

FT_STATUS FT_GetModemStatus(FT_HANDLE ftHandle, LPDWORD lpdwModemStatus);

Parametry:

- **ftHandle** - handle zařízení,
- **lpdwModemStatus** - adresa proměnné typu **DWORD** pro příjem stavu modemu. Dekódování stavu jednotlivých linek provádíme pomocí logického součinu mezi **lpdwModemStatus** a symboly dle *tab. 3.3.*

Tab. 3.3 Symboly pro zjišťování stavu linek modemu

Symbol	Testuje linku
MS_CTS_ON	CTS#
MS_DSR_ON	DSR#
MS_RING_ON	**Ri#**
MS_RLSD_ON	DCD#

POZNÁMKA: *Například stav linky* ***CTS#*** *získáme po volání funkce* ***FT_GetModemStatus*** *takto:* ***lpdwModemStatus&MS_CTS_ON.***

Je-li příslušný příznak vynulován, značí to, zeje linka v log. 1. Pokud je příznak nastaven, je linka v log. 0.

Příklad použití je uveden v kapitole 5.6.2.

3.2.18 FT_SetChars

Nastaví speciální (řídicí) znaky zařízení. Hlavička:

FT_STATUS FT_SetChars(FT_HANDLE ftHandle, UCHAR uEventCh, UCHAR uEventChEn, UCHAR uErrorCh, UCHAR uErrorChEn);

Parametry:

m **ftHandle** - handle zařízení,

- **uEventCh** - událostní znak (viz funkci **FT_SetEventNotification),**
- **uEventChEn -** pro **uEventCh = 0** je událostní znak vypnut (při hodnotě různé od Oje zapnut),
- **uErrorCh** - znak chyby přenosu,
- **uErrorChEn -** pro **uErrorChEn = 0** je chybový znak vypnut (při hodnotě různé od Oje zapnut).

3.2.19 FT_Purge

Vyprázdní přijímací/vysílací buffery zařízení. Hlavička:

FT_STATUS FT_Purge(FT_HANDLE ftHandle, DWORD dwMask);

Parametry:

- **ftHandle** - handle zařízení,
- **dwMask-** libovolná kombinace (získá se operátorem logického součtu |) symbolů **FT_PURGE_RX** (vyprázdní přijímací buffer) a **FT_PURGE_TX** (vyprázdní vysílací buffer).

3.2.20 FT_SetTimeouts

Nastaví time-outy pro operace čtení a zápisu. Hlavička: **FT_STATUS FT_SetTimeouts(FT_HANDLE ftHandle, DWORD dwReadTimeout, DWORD dwWriteTimeout);**

Parametry:

- **ftHandle** - handle zařízení,
- **dwReadTimeout** - time-out pro operaci čtení v milisekundách,
- **dwWriteTimeout** - time-out pro operaci zápisu v milisekundách.

3.2.21 FT GetQueueStatus

Zjistí počet znaků uložených v přijímací frontě. Hlavička: **FT_STATUS FT_GetQueueStatus(FT_HANDLE ftHandle, LPDWORD lpdwAmountInRxQueue);**

Parametry:

- ftHandle - handle zařízení,
- lpdwAmountlnRxQueue - adresa proměnné typu DWORD pro zjištění počtu znaků uložených v přijímací frontě.

3.2.22 FT_GetStatus

Zjistí aktuální stav zařízení včetně počtu znaků uložených v přijímací a vysílací frontě.
Hlavička: FT_STATUS FT_GetStatus(

FT_HANDLE ftHandle,

LPDWORD lpdwAmountlnRxQueue,

LPDWORD lpdwAmountlnTxQueue,

LPDWORD lpdwEventStatus);

Parametry:

- ftHandle - handle zařízení,
- lpdwAmountlnRxQueue - adresa proměnné typu DWORD pro zjištění počtu znaků uložených v přijímací frontě,
- lpdwAmountlnTxQueue - adresa proměnné typu DWORD pro zjištění počtu znaků uložených ve vysílací frontě,
- lpdwEventStatus - adresa proměnné typu DWORD pro zjištění aktuálního stavu zařízení (viz kapitolu 3.2.23).

POZNÁMKA: *Funkce* ***FT_GetStatus*** *se používá společně s níže popsanou funkcí* ***FTJSetEventNotification.***

3.2.23 FT SetEventNotification

Nastaví podmínky pro notifikaci události. Hlavička:
FT_STATUS FT_SetEventNotification(FTJHANDLE
ftHandle, DWORD dwEventMask, PVOID pvArg);

Parametry:

- ftHandle - handle zařízení,
- dwEventMask - sledované podmínky (viz *tab. 3.4),*
- pvArg - handle události.

Tab. 3.4 Podmínky, které lze sledovat funkcí FT_SetEventNotification

Symbol	Podmínka
FT_EVENT_RXCHAR	událost se nastaví, když je zařízením přijat událostní znak definovaný funkcí **FT_SetChars**
FT_EVENT_MODEM_STATUS	událost se nastaví, když dojde ke změně stavu linek modemu

POZNÁMKY: *Aplikace může použít funkci* ***FT_SetEventNotification*** *pro blokování aplikačního vlákna dokud nenastane jedna z určených podmínek. Aplikace obvykle vytvoří událost a voláním této funkce blokuje své provádění dokud nenastane jedna z určených podmínek.*

dwEventMask *je bitová maska popisující události, které jsou aplikací sledovány. pvArgje brán jako handle události, kterou dříve vytvořila aplikace. Potom, co nastane jedna ze stanovených podmínek, je událost signalizována.*

Příklad naleznete v kapitole 5.6.3.

3.3 FUNKCE PRO PROGRAMOVÁNÍ E^{2}PROM

FT232BM má vloženu podporu pro programování E^{2}PROM pomocí funkcí FT_EE_Program (programování), FT_EE_Read (čtení obsahu). Příklad je uveden v kapitole 11.

Část paměti, která není použita FT232BM, je k dispozici uživateli (mluvíme o uživatelské oblasti). Funkce FT_EE_UASize zjistí velikost uživatelské oblasti, funkce FT_EE_UAWrite a FT_EE_UARead umožňují zápis a čtení do této oblasti.

3.3.1 FT_EE_Program

Naprogramuje E^{2}PROM. Hlavička: FT_STATUS
FT_EE_Program(FTJHANDLE ftHandle,
PFT_PROGRAM_DATA lpData);

Parametry:

- ftHandle - handle zařízení,
- lpData - adresa struktury FT_PROGRAM_DATA, která obsahuje programovací data.

Definice struktury FT_PROGRAM_DATA:

```
typedef struct{
    WORD Vendorld;          //VID = 0x0403
    WORD Productid;         //PID = 0x6001
    char "Manufacturer;     //výrobce="FTDI"
```

```
    char *ManufacturerId;           //identifikátor výrobce="FT"
    char *Description;              //popis výrobku
    char *SerialNumber;             //sériové číslo
    WORD MaxPower;                  //maximální odběr (0 až 500 mA)
    WORD PnP;                       //Plug&Play (1 zapnuto)
    WORD SelfPowered;               //napájení (1 vnější, 0 z USB)
    WORD RemoteWakeup;              //Remote Wakeup (1 zapnuto)
    //rozšíření zavedená pro FT232BM:
    bool Rev4;                      //aktivuje další položky (1 aktivováno)
    bool IsoIn;                     //vstupní endpoint je izochronní (1)
    bool IsoOut;                    //výstupní endpoint je izochronní (1)
    bool PullDownEnable;            //aktivace pull-down rezistorů (1)
    bool SerNumEnable;              //sériové číslo použito (1)
    bool USBVersionEnable;          //verze USB použita (1)
    WORD USBVersion;                //verze USB (0x0200 pro USB 2.0)
} FT_PROGRAM_DATA;                  //(0x0101 pro USB1.0)
```

POZNÁMKY: *Data se zapíšou do E²PROM a poté se čtou zpět, tím se provede verifikace zápisu.*

Je-li položka ***SerialNumber = NULL*** *(nebo když* ***SerialNumber*** *obsahuje prázdný řetězec), je sériové číslo odvozeno z* ***ManufacturerId*** *a aktuálního data a času.*

3.3.2 FT_EE_Read

Čte obsah E²PROM. Hlavička: FT_STATUS
FT_EE_Read(FT_HANDLE ftHandle,
PFT_PROGRAM_DATA lpData);

Parametry:

- ftHandle - handle zařízení,
- lpData - adresa struktury FT_PROGRAM_DATA, která obsahuje načtená data.

POZNÁMKA: *Řetězcové proměnné musí být dostatečně veliké, aby pojaly údaje! Doporučené hodnoty: Manufacturer = 32 znaků, ManufacturerId = 16 znaků, Description = 64 znaků a SerialNumber =16 znaků.*

Délka řetězců Manufacturer + Description zapsaných v E²PROM nesmí být větší než 40 znaků (týká se funkce ***FT_EE_Program).***

3.3.3 FT_EE_UASize

Zjistí velikost uživatelské oblasti E²PROM. Hlavička:


```
FT_STATUS     FT_EE_UASize(
        FT_HANDLE     ftHandle,
        LPDWORD lpdwSize);
```

Parametry:

- ftHandle - handle zařízení,
- lpdwSize - adresa proměnné typu DWORD, která obsahuje zjištěnou velikost uživatelské oblasti E²PROM.

3.3.4 FT_EE_UARead

Provede čtení z uživatelské oblasti E²PROM. Hlavička:

```
FT_STATUS FT_EE_UARead(
        FT_HANDLE ftHandle, PUCHAR pucData, DWORD
        dwDataLen, LPDWORD lpdwBytesRead);
```

Parametry:

- ftHandle - handle zařízení,
- pucData - adresa bufferu pro načtená data,
- dwDataLen - délka bufferu odkázaného pucData v bajtech,
- lpdwBytesRead - adresa proměnné typu DWORD, kam se uloží počet skutečně přečtených bajtů (když je velikost uživatelské oblasti menší než dwDataLen).

3.3.5 FT_EE_UAWrite

Provede zápis do uživatelské oblasti E²PROM. Hlavička:
FT_STATUS FT_EE_UAWrite(FTJHANDLE ftHandle, PUCHAR
pucData, DWORD dwDataLen);

Parametry:

- ftHandle - handle zařízení,
- pucData - adresa bufferu, který obsahuje zapisovaná data,
- dwDataLen - délka zapisovaných dat v bajtech.

3.4 ROZŠÍŘENÉ FUNKCE

Rozšířené funkce zajišťují podporu obvodů řady B. Umožňují například nastavit time-out přijímače a aktivovat režim Bit Bang.

Režim Bit Bang

Obvody **FT232BM** se mohou přepnout do speciálního režimu, kdy je standardní funkce nahrazena režimem označeným **Bit Bang.**

V tomto režimu se 8 datových/řídicích linek chová jako 8bitová obousměrná sběrnice. Tento režim je vhodný hlavně pro konfiguraci programovatelných součástek.

Přepnutí do režimu Bit Bang zajišťuje funkce **FT_SetBitMode** (zároveň volí funkci každého vývodu jako vstupu nebo výstupu; každý vývod lze naprogramovat jako vstupní nebo výstupní nezávisle na ostatních).

Libovolná data posílaná běžným způsobem (funkcí **FT_Write)** se automaticky vybavují na výstupních datových vývodech. Rychlost vystavování dat odpovídá zvolené přenosové rychlosti (nastaví se funkcí **FT_SetBaudRate).** Rychlost přenosu je 16násobkem zvolené přenosové rychlosti (až 3 Mb/s). Pokud nepřicházejí nová data, stav linek se nemění.

Čtení stavu linek může proběhnout klasicky pomocí funkce **FT_Read,** data jsou v tomto případě vzorkována 16násobkem zvolené přenosové rychlosti (jako při zápisu) a ukládána do fronty. Okamžitý stav vstupů lze zjistit voláním funkce **FT_GetBitMode.** Příklad je uveden v kapitole 5.7.1.

3.4.1 FT_SetBitMode

Aktivuje nebo deaktivuje režim **Bit Bang.** Hlavička:

FT_STATUS FT_SetBitMode(FT_HANDLE ftHandle,
UCHAR ucMask, UCHAR ucEnable);

Parametry:

- **ftHandle** - handle zařízení,
- **ucMask** - bitová maska definující pro každý vývod UART rozhraní jeho směr. Nastavený bit odpovídá výstupu, vynulovaný bit zase vstupu. Například **0x80** přepne vývod **RI#** do výstupního směru, ostatní vývody pracují jako vstupy (viz *obr. 2.14),*
- **ucEnable** - zapíná **(ucEnable** = 1) nebo vypíná **(ucEnable** = **0)** režim Bit Bang.

3.4.2 FT GetBitMode

Čte stav vstupů konfigurovaných v režimu **Bit Bang.** Hlavička:

FT_STATUS FT_GetBitMode(FT_HANDLE ftHandle, PUCHAR
pucStatus);

Parametry:

- **ftHandle-**handle zařízení,

- **pucStatus** - adresa proměnné typu **UCHAR** pro příjem aktuálního stavu vývodů UART rozhraní konfigurovaných do režimu Bit Bang.

3.4.3 FT_SetLatencyTimer

Nastaví zpožďovací čítač. Hlavička:

FT_STATUS FT_Setl_atencyTimer(
FTJHANDLE ftHandle, UCHAR ucTimer);

Parametry:

- **ftHandle** - handle zařízení,
- **ucTimer** - žádaný interval zpožďovacího čítače v ms (v rozsahu 1 až 255).

POZNÁMKA: *U obvodu **FT8U232AM** byl time-out použitý pro spláchnutí zbývajících dat z přijímacího bufferu pevný (16 ms). Obvod **FT232BM** má tento interval programovatelný od 1 do 255 ms (po 1 ms). To dovoluje lépe optimalizovat přenosový protokol hlavně s ohledem na pakety malé délky.*

3.4.4 FT_GetLatencyTimer

Zjistí aktuální hodnotu zpožďovacího čítače. Hlavička:

FT_STATUS FT_Getl_atencyTimer(FTJHANDLE ftHandle,
PUCHAR pucTimer);

Parametry:

- **ftHandle** - handle zařízení,
- **pucTimer** - adresa proměnné typu **UCHAR** pro načtení hodnoty zpožďovacího čítače.

3.4.5 FT SetUSBParameters

Nastaví přenosovou šíři pro USB požadavek. Hlavička: **FT_STATUS**
FT_SetUSBParameters(FT_HANDLE ftHandle, DWORD
dwlnTransferSize, DWORD dwOutTransferSize);

Parametry:

- **ftHandle-**handle zařízení,
- **dwlnTransferSize** - přenosová šíře pro vstupní USB požadavek,
- **dwOutTransferSize** - přenosová šíře pro výstupní USB požadavek.

POZNÁMKA: *Dříve byla velikost USB požadavku pevná (4096 B). Tato funkce umožňuje velikost USB požadavku měnit tak, aby lépe vyhověla konkrétní aplikaci.*

3.5 FUNKCE FT WIN32 API

Nová verze D2XX rozhraní obsahuje funkce odvozené od **Win32 API** a **Win32 COMM API.** To umožňuje snadný přechod od programů zapsaných pro **VCP** (virtuální sériový port) na používání rozhraní D2XX.

Před přístupem na zařízení jej musíme otevřít voláním funkce **FT_W32_CreateFile.** Zápis a čtení je realizováno funkcemi **FT_W32_WriteFIle** a **FT_W32_ReadFile.** Při ukončení práce se zařízením jej lze zavřít voláním funkce **FT_W32_CloseHandle.**

3.5.1 FT_W32_CreateFile

Funkce otevře pojmenované zařízení a vrátí jeho handle. Hlavička:

FT_HANDLE FT_W32_CreateFile(
LPCSTR lpszName, DWORD dwAccess,
DWORD dwShareMode,
LPSECURITY_ATTRIBUTES lpSecurityAttributes,
DWORD dwCreate, DWORD dwAttrsAndFlags, HANDLE hTemplate);

Parametry:

- **lpszName** - jméno zařízení. Může se jednat o sériové číslo nebo popis výrobku (viz **dwAttrsAndFlags),**
- **dwAccess** - definuje přístup na zařízení. Může být **GENERIC_READ** (pro čtení) nebo **GENERIC_WRITE** (zápis) či obojí (kombinuje se operátorem |),
- **dwShareMode** - definuje sdílení zařízení. Platí, že zařízení nelze sdílet. Proto musí být **dwShareMode = 0,**
- **lpSecurityAttributes** - tento parametr nemá význam, musí být **lpSecurityAttributes = NULL,**
- **dwCreate** - způsob vytvoření. Musí být **dwCreate = OPEN_EXISTING,**
- **dwAttrsAndFlags** - atributy a příznaky souboru. Lze používat **FILE_ATTRIBUTE_NORMAL** a **FILE_FLAG_OVERLAPPED** (otevření pro překryvné operace). Dále se takto definuje význam parametru **lpszName.** Pokud se použije **FT_OPEN_BY_SERIAL_NUMBER,** má **lpszName** význam sériového čísla zařízení. Pokud je použit symbol **FT_OPEN_BY_DESCRIPTION,** má **lpszName** význam popisu výrobku. Sčítaní příznaků se provádí operátorem |,
- **hTemplate** - nepoužívá se, nastavte na **NULL,**

- **návratová hodnota** - při úspěchu vrací handle zařízení, jinak hodnotu **INVALID_HANDLE_VALUE.**

POZNÁMKA: *Tato funkce umožňuje otevřít zařízení pro překryvné (asynchronní operace).*

Definice struktury OVERLAPPED (má význam pro funkce FT_W32_ReadFile, FT_W32_WriteFile,FT_W32_GetOverlappedResult,FT_W32_WaitCommEvent):

```
typedef struct{
    DWORD Internal;          //vyhrazeno pro operační systém
                             //(systémově definovaný stavový příznak)
    DWORD IntemalHigh;       //vyhrazeno pro operační systém
                             //(počet přenesených bajtů)
    DWORD Offset;            //nižší slovo relativní pozice přenosu
                             //(nelze měnit)
    DWORD OffsetHigh;        //vyšší slovo relativní pozice přenosu
                             //(nelze měnit)
    HANDLE hEvent;           //handle události signalizované
                             //při dokončení překryvné operace }
OVERLAPPED;
```

3.5.2 FT_W32_CloseHandle

Zavře zařízení určené jeho handle. Hlavička: **BOOL FT_W32_CloseHandle(FT_HANDLE ftHandle);**

Parametry:

- **ftHandle** - handle zavíraného zařízení,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení operace.

3.5.3 FT_W32_ReadFile

Čte data ze zařízení. Hlavička:

BOOL FT_W32_ReadFile(
FTJHANDLE ftHandle, LPVOID lpBuffer,
DWORD dwBytesToRead, LPDWORD lpdwBytesReturned,
LPOVERLAPPED lpOverlapped);

Parametry:

- **ftHandle** - handle zařízení,
- **lpBuffer** - adresa bufferu pro příjem dat,
- **dwBytesToRead** - počet čtených bajtů,

- **lpdwBytesReturned** - adresa proměnné typu **DWORD** pro uložení počtu skutečně přečtených bajtů,
- **lpOverlapped** - adresa překryvné struktury **OVERLAPPED** (je-li použit překryvný vstup),
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení operace.

Aktivace překryvné operace:

*Tato funkce podporuje překryvný i nepřekryvný vstup. Je-li zařízení otevřeno s příznakem **FILE_FLAG_OVERLAPPED,** musí **lpOverlapped** obsahovat adresu překryvné struktury. V opačném případě je **lpOverlapped = NULL.***

Nepřekryvný vstup (normální režim)

Je-li **lpOverlapped = NULL,** jedná se o nepřekryvný vstup. V tomto případě je v parametru **lpdwBytesReturned** odkázán počet přečtených bajtů.

Funkce **FT_W32_ReadFile** se nevrací, dokud se nepřečte **dwBytesToRead** bajtů. Takže může snadno dojít k zablokování programu. Lze však využít výsledků volání funkcí **FT_GetStatus** nebo **FT_GetQueueStatus** a načíst jen tolik bajtů, kolik je uloženo ve vstupní frontě.

Také lze nastavit tíme-out pro čtení. Pokud interval time-outu vyprší před načtením žádaného počtu bajtů **(dwBytesToRead),** funkce se vrátí a do bufferu **(lpBuffer)** zapíše jen tolik bajtů, kolik bylo k dispozicí. V tom případě také platí **lpdwBytesReturned < dwBytesToRead.** Pokud se žádaný počet bajtů přečte před vypršením time-outu, bude **lpdwBytesReturned = dwBytesToRead.**

Je třeba poznamenat, že nezávisle na vypršení time-outu, vrací funkce **FT_W32_ReadFile** vždy hodnotu **TRUE.**

Překryvný vstup (asynchronní operace)

Pokud je zařízení otevřeno jako překryvné a současně obsahuje **lpOverlapped** adresu platné **OVERLAPPED** struktury, jedná se o překryvný vstup.

Když je zařízení otevřeno pro překryvnou operaci, může aplikace vyčkávat na odezvu a současně provádět jinou práci. Pokud nejsou data v požadovaném rozsahu k dispozici ve frontě, vrací se funkce **FT_W32_ReadFile** okamžitě s návratovou hodnotu **FALŠE** (0), která má značit chybu.

Aplikace pak může skutečnou příčinu chyby zjistit voláním funkce **FT_W32_GetLastError.** Pokud je výsledkem hodnota **ERROR_IO_PENDING,** značí to, že překryvná operace stále běží. Také lze použít výsledek funkce **FT_W32_GetOverlappedResult** (zjistí, kolik bajtů se již načetlo).

I v případě překryvného vstupu je v parametru **lpdwBytesReturned** odkázán počet přečtených bajtů.

3.5.4 FT_W32_WriteFile

Zapíše data na zařízení. Hlavička: **BOOL**

FT_W32_WriteFile(

FT_HANDLE ftHandle, LPVOID lpBuffer,

DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten,

LPOVERLAPPED lpOverlapped);

Parametry:

- **ftHandle** - handle zařízení,
- **lpBuffer** - adresa bufferu, který obsahuje zapisovaná data,
- **dwBytesToWrite** - počet zapisovaných bajtů,
- **lpdwBytesWritten** - adresa proměnné typu **DWORD** pro uložení počtu skutečně zapsaných bajtů,
- **lpOverlapped** - adresa překryvné struktury **OVERLAPPED** (je-li použit překryvný výstup),
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení operace.

Aktivace překryvné operace:

*Tato funkce podporuje překryvný i nepřekryvný výstup. Je-li zařízení otevřeno s příznakem **FILE_FLAG_OVERLAPPED,** musí **lpOverlapped** obsahovat adresu překryvné struktury. V opačném případě je **lpOverlapped = NULL.***

Nepřekryvný výstup (normální režim)

Je-li **lpOverlapped = NULL,** jedná se o nepřekryvný výstup. V tomto případě je v parametru **lpdwBytesWritten** odkázán počet zapsaných bajtů.

Funkce **FT_W32_WriteFile** se nevrací, dokud se nezapíše **dwBytesToWrite** bajtů. Takže může snadno dojít k zablokování programu. Lze však nastavit time-out pro zápis. Pokud interval time-outu vyprší před zápisem žádaného počtu bajtů **(dwBytesToWrite),** funkce se vrátí předčasně. V tom případě také platí **lpdwBytesWritten < dwBytesToWrite.** Pokud se žádaný počet bajtů zapíše před vypršením time-outu, bude **lpdwBytesWritten = dwBytesToWrite.**

Je třeba poznamenat, že nezávisle na vypršení time-outu, vrací funkce **FT_W32_WriteFile** vždy hodnotu **TRUE.**

Překryvný výstup (asynchronní operace)

Pokud je zařízení otevřeno jako překryvné a současně obsahuje **lpOverlapped** adresu platné **OVERLAPPED** struktury, jedná se o překryvný výstup.

Když je zařízení otevřeno pro překryvnou operaci, může aplikace vyčkávat na odezvu a současně provádět jinou práci. Data se nyní zapisují „na pozadí běžící aplikace" a funkce **FT_W32_WriteFile** se vrací okamžitě s návratovou hodnotou **FALŠE** (0), která má značit chybu.

Aplikace pak může skutečnou příčinu chyby zjistit voláním funkce **FT_W32_GetLastError.** Pokud je výsledkem hodnota **ERROR_IO_PENDING,** značí to, že překryvná operace stále běží. Také lze použít výsledek funkce **FT_W32_GetOverlappedResult** (zjistí, kolik bajtů se již zapsalo).

I v případě překryvného vystupuje v parametru **lpdwBytesWritten** odkázán počet zapsaných bajtů.

3.5.5 FT_W32_GetLastError

Vrací kód poslední chyby, která nastala u zařízení. Hlavička: **DWORD FT_W32_GetLastError(FT_HANDLE ftHandle);**

Parametry:

- **ftHandle-**handle zařízení,
- **návratová hodnota** - kód poslední chyby (viz *tab. 3.1).*

POZNÁMKA: *Tato funkce se používá obvykle v souvislostí s překryvným vstupem nebo výstupem dat prováděným funkcemi* ***FT_W32_ReadFile*** *nebo* ***FT_W32_WriteFile*** *proto, aby se určilo, zda překryvná operace běží nebo že došlo k jejímu selhání.*

3.5.6 FT W32 GetOverlappedResult

Zjistí stav probíhající překryvné operace. Hlavička: **BOOL FT_W32_GetOverlappedResult(FT_HANDLE ftHandle, LPOVERLAPPED lpOverlapped, LPDWORD lpdwBytesTransferred, BOOL bWait);**

Parametry:

- **ftHandle** - handle zařízení,
- **lpOverlapped** - adresa překryvné struktury **OVERLAPPED,** která byla použita pro spuštění překryvné operace,
- **lpdwBytesTransferred** - adresa proměnné typu **DWORD** pro příjem počtu bajtů přenesených v průběhu překryvné operace,
- **bWait** — je-li **bWait = TRUE,** funkce se nevrací, dokud není překryvná operace dokončena,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

POZNÁMKA: *Tato funkce se používá v souvislosti s překryvným vstupem nebo výstupem dat prováděným funkcemi* ***FT_W32_ReadFile*** *nebo* ***FT_W32_WriteFile*** *proto, aby se určilo, kolik bajtů již bylo přeneseno.*

3.5.7 FT W32 ClearCommBreak

Uvede komunikační linku (TXD) do nepřerušeného stavu (log. 1). Hlavička: **BOOL FT_W32_ClearCommBreak(FT_HANDLE ftHandle);**

Parametry:

- **ftHandle** - handle zařízení,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

3.5.8 FT_W32_SetCommBreak

Uvede komunikační linku (TXD) do přerušeného stavu (log. 0). Hlavička: **BOOL FT_W32_SetCommBreak(FT_HANDLE ftHandle);**

Parametry:

- **ftHandle** - handle zařízení,
- **návratová hodnota** - **TRUE** (1) indikuje úspěšné provedení funkce.

3.5.9 FT_W32_EscapeCommFunction

Provede rozšířenou funkci zařízení. Hlavička:
BOOL FT_W32_EscapeCommFunction(
FTJHANDLE ftHandle, DWORD dwFunc);

Parametry:

- **ftHandle** - handle zařízení,
- **dwFunc** - definuje rozšířenou funkci dle *tab. 3.5* (pouze jedna varianta, nelze slučovat),
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

Tab. 3.5 Kódy rozšířených funkcí

Symbol	Význam
CLRDTR	DTR# = 1
SETDTR	DTR# = 0
CLRRTS	RTS# = 1
SETRTS	RTS# = 0
CLRBREAK	TXD = 1
SETBREAK	TXD = 0

3.5.10 FT W32 GetCommModemStatus

Zjistí stav linek modemu. Hlavička:
BOOL FT_W32_GetModemStatus(
FT_HANDLE ftHandle, LPDWORD lpdwStat);

Parametry:

m **ftHandle** - handle zařízení,

- **lpdwStat** - ukazatel na proměnnou typu **DWORD** pro příjem stavu modemu. Dekódování stavu jednotlivých linek provádíme pomocí logického součinu mezi ***lpdwModemStatus** a symboly dle *tab. 3.3,*
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

POZNÁMKA: *Funkce pracuje podobně jako výše popsaná funkce* ***FT_GetModemStatus*** *(viz kapitolu 3.2.17).*

3.5.11 FT_W32_ClearCommError

Zjistí informace o chybě komunikace a aktuálním stavu zařízení. Hlavička: **BOOL FT_W32_ClearCommError(FT_HANDLE ftHandle, LPDWORD lpdwErrors, LPFTCOMSTAT lpftComstat);**

Parametry:

- **ftHandle** - handle zařízení,
- **lpdwErrors** - adresa proměnné typu **DWORD** pro zjištění chyby komunikace (maska definující chybu; viz *tab. 3.6),*
- **lpftComstat** - adresa struktury **FTCOMSTAT** pro zjištění stavu zařízení,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

Tab. 3.6 Symboly pro zjišťování chyby komunikace

Symbol	Popis chyby
CE_BREAK	přerušeni linky (BREAK stav)
CE_FRAME	chyba rámce (například chybný počet stop-bitu)
CE_RXOVER	přetečení vstupního bufferu
CE_TXFULL	výstupní buffer je plný
CE_RXPARITY	chyba parity přijatého znaku

POZNÁMKA: *Například přeplnění vstupního bufferu testujeme po volání funkce* ***FT_W32_ClearCommError*** *takto:* ***lpdwErrors&CE_RXOVER.***

Definice struktury FTCOMSTAT (nejpoužívanější jsou poslední dvě položky):

```
typedef struct{
    DWORD fCtsHold : 1;          //vysílač čeká na CTS
    DWORD fDsrHold : 1;          //vysílač čeká na DTR
    DWORD fRIsdHold : 1;         //vysílač čeká na DSR
    DWORD fXoffHold : 1;         //vysílač čeká, protože byl přijat Xoff
```



```
    DWORD fXoffSent: 1;          //vysílač čeká, protože byl vyslán Xoff
    DWORD fEof: 1;               //byl přijat znak konce souboru
    DWORD fTxim : 1;             //vysílaný znak byl vložen do fronty
                                  //přednostně
    DWORD fReserved : 25;        //vyhrazeno
    DWORD cblnQue;               //počet znaků ve vstupní frontě
    DWORD cbOutQue;              //počet znaků ve výstupní frontě
} FTCOMSTAT;
```

3.5.12 FT_W32_SetCommState

Konfiguruje přenos podle obsahu DCB struktury. Hlavička: **BOOL FT_W32_SetCommState(FT_HANDLE ftHandle, LPFTDCB lpftDcb);**

Parametry:

- **ftHandle-**handle zařízení,
- **lpftDcb** - adresa struktury **FTDCB** pro konfiguraci přenosu,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

Definice struktury FTDCB (popsané jsou pouze nejdůležitější položky):

```
typedef struct{
    DWORD DCBIength;             //velikost struktury, použijte sizeof(FTDCB)
    DWORD BaudRate;              //přenosová rychlost v Bd
    DWORD fBinary: 1;            //binární režim (netestuje konec souboru)
    DWORD fParity: 1;            //povolení kontroly parity
    DWORD fOutxCtsFlow:1;        //CTS handshaking
    DWORD fOutxDsrFIow: 1;       //DSR handshaking
    DWORD fDtrControl:2;         //DTR řízení toku
    DWORD fDsrSensitivity:1; DWORD
    fTXContinueOnXoff: 1; DWORD
    fOutX: 1; DWORD flnX: 1; DWORD
    fErrorChar: 1; DWORD fNull: 1;
    DWORD fRtsControl:2;         //RTS řízení toku
    DWORD fAbortOnError:1;
    DWORDfDummy2:17; WORD
    wReserved; WORD XonLim;
    WORD XoffLim;
    BYTE ByteSize;               //délka znaku 7 nebo 8 bitů
    BYTE Parity;                 //parita: 0 až 4 = nic, lichá, sudá,
                                  //značená, mezerová
```

```
        BYTE StopBits;          //počet stop-bitů: 0,2 = 1,2 stop-bity
        char XonChar;           //definuje Xon znak
        char XoffChar;          //definuje Xoff znak
        char ErrorChar;
        char EofChar;           //znak konce souboru
        char EvtChar;           //událostní znak
        WORD wReservedl; }
FTDCB;
```

3.5.13 FT_W32_GetCommState

Zjistí konfiguraci přenosu. Hlavička: **BOOL FT_W32_GetCommState(FT_HANDLE ftHandle, LPFTDCB lpftDcb);**

Parametry:

- **ftHandle** - handle zařízení,
- **lpftDcb** - adresa struktury **FTDCB** pro načtení konfigurace přenosu (viz výše),
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

3.5.14 FT_W32_SetCommTimeouts

Konfiguruje time-outy pro čtení a zápis. Hlavička: **BOOL FT_W32_SetCommTimeouts(FT_HANDLE ftHandle, LPFTTIMEOUTS lpftTimeouts);**

Parametry:

- **ftHandle-**handle zařízení,
- **lpftTimeouts** - adresa struktury **FTTIMEOUTS** pro konfiguraci time-outů,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

Definice struktury FTTIMEOUTS:

```
typedef struct {
        DWORD ReadIntervalTimeout;          //maximální prodleva
                                            //mezi čtením dvou znaků DWORD
        ReadTotalTimeoutMultiplier; //násobitel počtu znaků pro čtení DWORD
        ReadTotalTimeoutConstant; //konstanta pro čtení (v ms) DWORD
        WhteTotalTimeoutMultiplier;  //násobitel počtu znaků pro zápis DWORD
        WriteTotalTimeoutConstant; //konstanta pro zápis (v ms)
} FTTIMEOUTS;
```


POZNÁMKA: *Interval time-outu při čtení je dán součinem počtu čtených znaků a hodnoty* ***ReadTotalTimeoutMultiplier,*** *k tomuto výrazu se ještě přičte konstantní část daná* ***ReadTotalTimeoutConstant.*** *Interval mezi čtením dvou po sobě jdoucích znaků nesmí překročit hodnotu* ***ReadIntervalTimeout.***

Interval time-outu při zápisu je dán součinem počtu zapisovaných znaků a hodnoty ***WriteTotalTimeoutMultiplier,*** *k tomuto výrazu se ještě přičte konstantní část daná* ***WriteTotalTimeoutConstant.***

Všechny údaje jsou v milisekundách.

3.5.15 FT_W32_GetCommTimeouts

Zjistí nastavení time-outů pro čtení a zápis. Hlavička: **BOOL FT_W32_GetCommTimeouts(FT_HANDLE ftHandle, LPFTTIMEOUTS lpftTimeouts);**

Parametry:

- **ftHandle** - handle zařízení,
- **lpftTimeouts** - adresa struktury **FTTIMEOUTS** načtení time-outů,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

3.5.16 FT_W32_SetupComm

Nastaví velikost přijímacích a vysílacích bufferů. Hlavička: **BOOL FT_W32_SetupComm(FT_HANDLE ftHandle, DWORD dwReadBufferSize, DWORD dwWriteBufferSize);**

Parametry:

- **ftHandle** - handle zařízení,
- **dwReadBufferSize** - délka vstupního bufferu v bajtech,
- **dwWriteBufferSize** - délka výstupního bufferu v bajtech,
- **návratová hodnota - TRUE** (1) indikuje úspěšné provedení funkce.

POZNÁMKA: *Funkce* ***FT_W32_SetupComm*** *nemá význam, protože je na odpovědnosti ovladače, aby alokoval přiměřenou velikost bufferů. Tato funkce je zavedena pouze pro kompatibilitu s WIN32 COMM API!!!*

3.5.17 FT_W32_PurgeComm

Vyprázdní zvolené buffery zařízení. Hlavička:

BOOL FT_W32_PurgeComm(
FT_HANDLE ftHandle, DWORD dwFlags);

Parametry:

- ftHandle - handle zařízení,
- dwFlags - určuje buffery, které se mají vyprázdnit dle *tab. 3.7* (lze provádět kombinace operátorem |),
- návratová hodnota - TRUE (1) indikuje úspěšné provedení funkce.

Tab. 3.7 Určení bufferu pro vyprázdnění

Symbol	Popis operace
PURGE_TXABORT	zruší nedokončený překryvný zápis
PURXE_RXABORT	zruší nedokončené překryvné čtení
PURGE_TXCLEAR	smaže výstupní buffer
PURGE_RXCLEAR	smaže vstupní buffer

3.5.18 FT_W32_SetCommMask

Určí událost, která je na zařízení monitorována. Hlavička: BOOL
FT_W32_SetCommMask(
FT_HANDLE ftHandle, DWORD dwMask);

Parametry:

- ftHandle - handle zařízení,
- dwMask - definuje monitorované události dle *tab. 3.8* (lze provádět kombinace operátorem |),
- návratová hodnota - TRUE (1) indikuje úspěšné provedení funkce.

POZNÁMKA: *Funkce* ***FT_W32_SetCommMask*** *definuje události, které budou monitorovány. Aplikace může volat funkci* ***FT_W32_WaitCommEvent*** *a tak čekat, až jedna z příslušných událostí nastane.*

3.5.19 FT **W32_WaitCommEvent**

Čeká na monitorovanou událost. Hlavička:

BOOL FT_W32_WaitCommEvent(
FT_HANDLE ftHandle, DWORD lpdwEvent,
LPOVERLAPPED lpOverlapped);

Parametry:

m ftHandle - handle zařízení,

- lpdwEvent - adresa proměnné typu DWORD pro příjem aktivované události,
- lpOverlapped - adresa struktury OVERLAPPED pro případ použití překryvných operací (pro nepřekryvné operace bude lpOverlapped = NULL),
- návratová hodnota - TRUE (1) indikuje úspěšné provedení funkce.

Tab. 3.8 Definice monitorovaných událostí

Symbol	Monitorovaná událost
EV_BREAK	přerušení linky (BREAK)
EV_CTS	změna stavu linky CTS
EV_DSR	změna stavu linky DSR
EV_ERR	chyba stavových linek
EV_RING	detekce vyzvánění (RING)
EV_RLSD	změna stavu linky DCD
EV_RXCHAR	byl přijat znak
EV_RXFLAG	byl přijat událostní znak
EV_TXEMPTY	vysílač je vyprázdněn

POZNÁMKY: *Monitorované události se stanoví předchozím voláním funkce* ***FT_W32_SetCommMask.***

Funkce podporuje překryvné i nepřekryvné provádění. V obou případech lze detekovanou událost testovat pomocí ****lpdwEvent,*** *operátoru & a symbolů dle tab. 3.8.*

Nepřekryvné operace

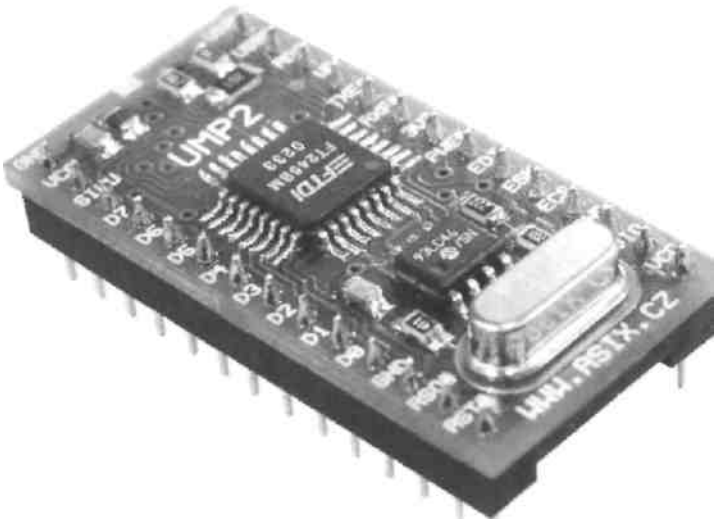
Pro nepřekryvný vstup/výstup platí, že se funkce nevrátí do té doby, než nastane jedna ze stanovených událostí.

Překryvné operace

Pokud je zařízení otevřeno pro překryvné operace nehrozí zablokování jako při normální režimu otevření. Když žádná ze specifikovaných událostí nenastane, vrátí funkce FT_W32_WaitCommEvent hodnotu FALŠE a následně volaná funkce FT_GetLastError vrací ERROR_IO_PENDING (překryvný dotaz stále probíhá). Také lze použít funkci FT_GetOverlappedResult. Pokud událost nastane, vrátí funkce FT_W32 WaitCommEvent hodnotu TRUE.

4

ZAŘÍZENÍ USB OD FIRMY ASIX



V této kapitole se velmi stručně seznámíme s přípravky na bázi obvodů FTDI, které nabízí firma ASIX. Tato firma je zároveň jedním z dodavatelů FTDI obvodů v České republice.

4.1 FIRMA ASIX

Firma ASIX se specializuje na výrobu vývojových prostředků pro práci s programovatelnými součástkami několika předních výrobců (Xilinx, Microchip atd.).

4.2 ZAJÍMAVÉ PŘÍPRAVKY

Pro nás jsou zajímavé přípravky používající obvody od firmy FTDI. Úplný přehled najdete na stránkách společnosti ASIX:

www.asix.cz

nebo na e-shopu.

Část z této nabídky je také uvedena v *tab. 4.1, jedná* se o modul osazený obvodem FT232BM a podpůrné přípravky.

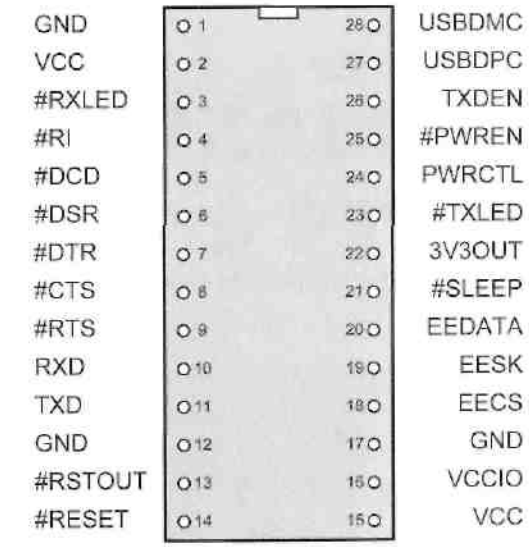
Tab. 4.1 Přehled přípravků spojených s obvodem FT232BM (ceny uvedeny bez DPH, platné k počátku října 2003)

Název	Popis	Náhled	Přibližná cena
UMS2	**modul** s **FT232BM** USB-UART pro vývoj a malosériovou výrobu		680 Kč
UCAB232 S	Kompaktní **převodník** USB – RS232 založený na obvodu **FT232BM** firmy FTDI. Provedení "S" - Standard.		880 Kč
UCAB232 E	Kompaktní **převodník** USB – RS232 založený na obvodu **FT232BM** firmy FTDI. Provedení "E" - Enhanced.		1380 Kč
USET1	sada vhodná pro realizace vlastních aplikací s USB portem.		350 Kč

Zájemci o amatérskou stavbu převodníku USB<=>RS-232 si jej mohou vyrobit sami, konstrukce je popsána v kapitolách 12.1 a 12.2.

4.3 MODUL UMS2

Pro amatérskou praxi je patrně nejdůležitější modul UMS2, který obsahuje obvod FT232BM spolu s konfigurační pamětí 93LC56/SN a ostatními součástkami. Takže se jedná o kompaktní modul, ke kterému se pouze připojí USB konektor. Hlavní výhodou je skutečnost, že celek je kontaktován do 28vývodového širokého pouzdra DIP a tak odpadá nutnost pájení součástek SMD. Viz *obr. 4.1.*



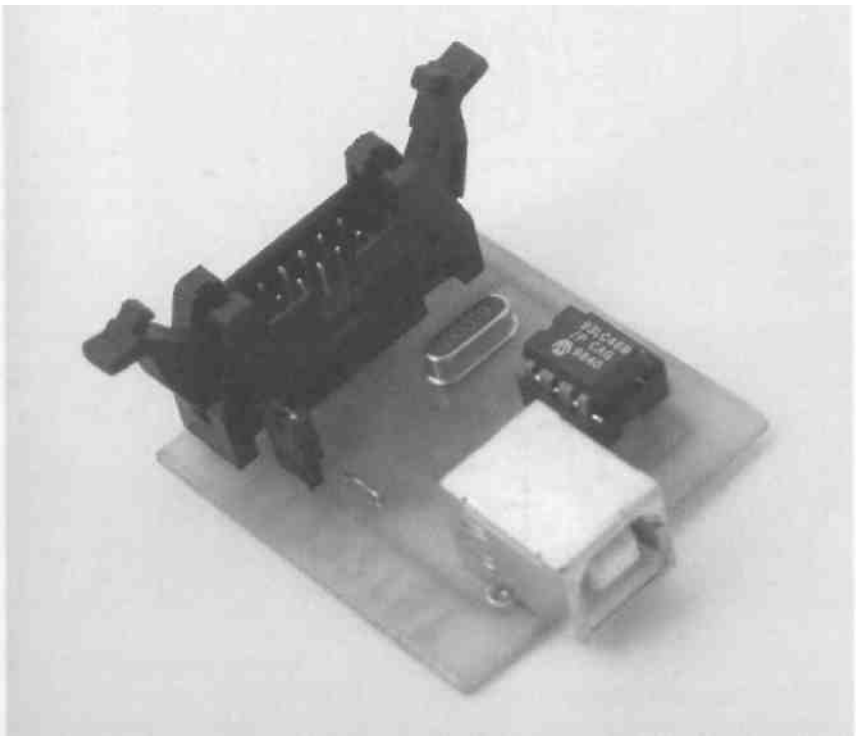
Obr. 4.1 Modul UMS2 a jeho vývody

Vývody odpovídají *obr. 2.2.* Označení USBDMC a USBDPC určuje, že ochranné rezistory hodnoty 27 Q pro vývody USBDM a USBDP jsou již na přípravku vloženy a takto vyvedeny na kontakty 28 a 27. Úplnou dokumentaci k modulu UMS2 najdete na doprovodném CD-ROM v adresáři DATASHEET\ASIX\UMS2.PDF.

Abych nepředbíhal výklad příkladů, je konstrukce založená na tomto modulu uvedena až v kapitole 12.4.

5

UNIVERZÁLNÍ MODUL S FT232BM



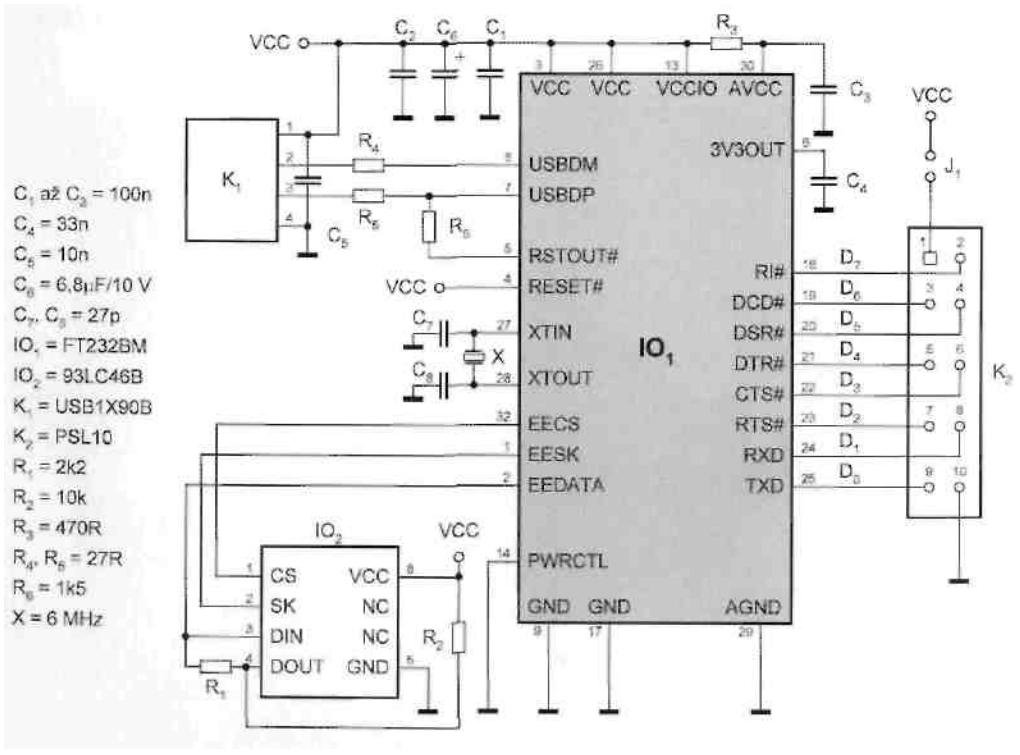
Než se pustíme do tvorby aplikací s obvodem FT232BM, bude vhodné ukázat typické použití funkcí ovládacího rozhraní z kapitoly 3. Pro úvodní testy si tedy vytvoříme jednoduchý modul s obvodem FT232BM, který umožní připojovat různá zařízení.

5.1 PŘÍPRAVEK FT232TST

Schéma zapojení testovacího přípravku vychází z popisu obvodu **FT232BM,** který byl uveden v kapitole 2. Pro univerzální použití byly vývody UART rozhraní připojeny na konektor K_2 **(PSL10).**

Rozložení jednotlivých vývodů na konektoru K_2 odpovídá rozložení, které jsem zavedl v knihách o mikrokontrolérech **ATMEL** ([1] až [3]). Takže bude velmi snadné připojovat tyto mikrokontroléry přímo k testovacímu přípravku!

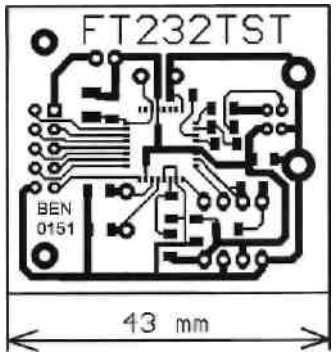
Jumper J, umožňuje odpojit napájecí napětí získané přípravkem přímo ze sběrnice USB. Pokud není na vývod 1 konektoru K_2 na straně připojovaného zařízení přivedeno napětí, nechá se jumper zapojen (tak lze napájet přípravek přímo z portu). Naopak, pokud má přípravek vlastní napájení a to je na vývodu 1 konektoru K_2, měl by být jumper vyjmut!



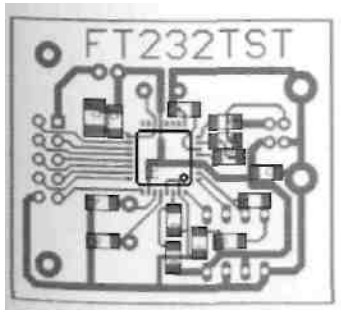
Obr. 5.1 Schéma zapojení přípravku FT232TST

Rozpis součástek pro přípravek FT232TST (cena asi 300 Kč).

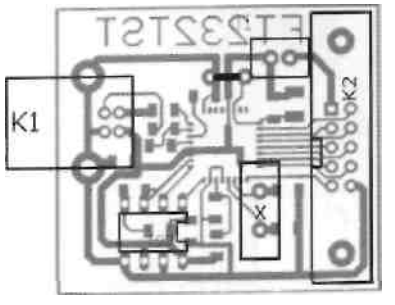
C_1 až C_3	CK+100NX7R	3 ks
C_4	CK+33N X7R	1 ks
C_5	CK+10NX7R	1 ks
C_6	CTS6M8/10VB	1 ks
C_7, C_8	CK+27P NPO	1 ks
10_1	FT232BM	1 ks
10_2	93LC46B	1 ks
K,	USB1X90B PCB	1 ks
K_2	PSL10	1 ks
R_1	RR+2K2 SMD	1 ks
R_2	RR+10KSMD	1 ks
R_3	RR+470R SMD	1 ks
R_4, R_5	RR+27R SMD	1 ks
R_6	RR+1K5SMD	1 ks
X	QM 6.000MHz	1 ks
J,	jumper	1 ks



Obr. 5.2 Výkres desky plošných spojů přípravku FT232TST (BEN 0151)



Obr. 5.3 Osazovací plánek přípravku FT232TST (strana spojů)



Obr. 5.4 Osazovací plánek přípravku FT232TST (strana součástek)

Vzhledem k tomu, že jsme desku plošných spojů museli navrhnout jako jednostrannou, použili jsme součástky SMD. Vlastně to byla nutnost, protože sám obvod FT232BM je v provedení SMD. SMD jsme napájeli ze strany spojů. Strana součástek obsahuje pouze oba konektory (K_1pro připojení USB a K_2 pro připojení řízeného zařízení), konfigurační E^2PROM v patici, krystal 6 MHz, jednu drátovou propojku a jumper.

Přípravek je možno objednat přímo od autora (viz kapitolu 14.4).

5.2 PLUG&PLAY OVLADAČ

Po pečlivém osazení součástek můžeme přikročit k důležitému kroku a sice k připojení přípravku přes USB kabel A-B k počítači.

Pokud je vše v pořádku, detekuje operační systém nové zařízení (viz *obr. 5.5).* Po stisku tlačítka **Další** se zobrazí dialog dle *obr. 5.6* (ponecháme nastavenou volbu nejvhodnějšího ovladače a pokračujeme tlačítkem **Další).**

Obr. 5.5 Operační systém detekoval obvod FT232BM

Obr. 5.6 Vyhledání nejvhodnějšího ovladače

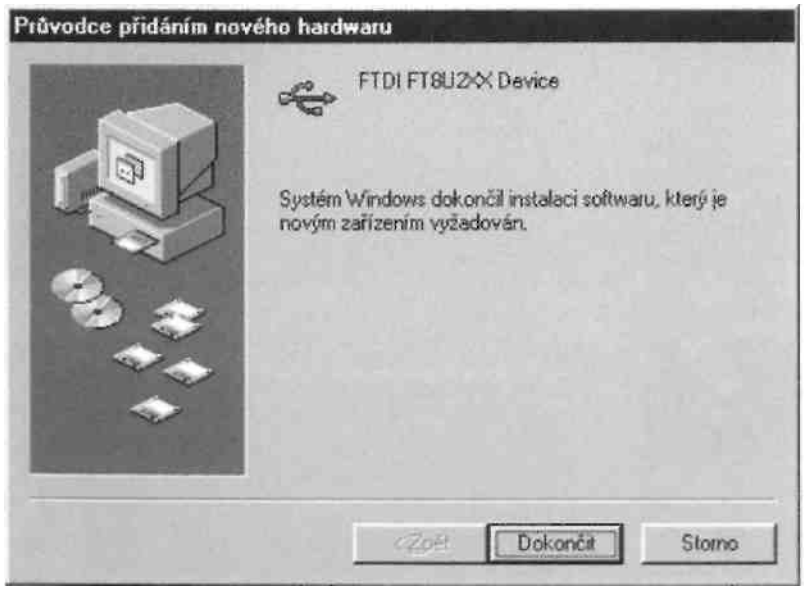
Nyní se operační systém zeptá na umístění souborů ovladače. Lze instalovat z diskety nebo zvolit adresář, kde jsou soubory ovladače umístěny (viz *obr. 5.7).*

Obr. 5.7 Určení adresáře se soubory ovladače

Po stisku tlačítka **Další** se operační systém pokusí soubory najít. Ovladače jsou umístěny na doprovodném CD-ROM v adresáři **FTDI\OVLADACE\D2XX\FTDI.**

Nyní je vše připraveno k instalaci, po stisku tlačítka **Další** v dialogu dle *obr. 5.8* se spustí instalace popsaná obsahem souboru **FTD2XX.INF.**

Obr. 5.8 Instalace ovladače D2XX



Obr. 5.9 Instalace je úspěšně dokončena

Pokud je vše v pořádku, dostaneme nakonec dialog dle *obr. 5.9,* který hlásí úspěšné dokončení instalace. Po stisku tlačítka **Dokončit** je možno zařízení okamžitě používat.

Můžeme se ještě „podívat" do správce zařízení **(Start|Nastavení|Ovládací panely|Systém),** zeje FT232BM skutečně instalován.

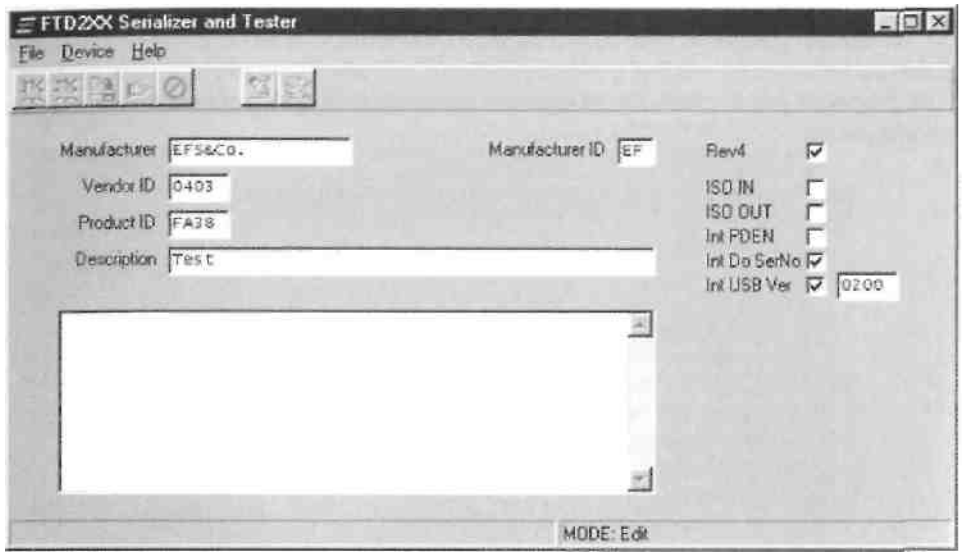
Obr. 5.10 Zařízení je zaregistrováno v systému

5.3 NAPROGRAMOVÁNÍ E^{2}PROM

Naprogramování E^{2}PROM je nutné proto, aby bylo možno zařízení jednoznačně identifikovat ovládacím programem. Tímto způsobem se odstraní problémy známé například z použití sériových nebo paralelních portů.

Programování lze provést utilitkou FTD2XXST.EXE, kterou naleznete na doprovodném CD-ROM. Také lze používat programátor uvedený v kapitole 11 (mě se osvědčil lépe, protože jeho ovládání je velmi snadné). Pokud program FTD2XXST.EXE nedetekuje obvod, je do registry nutno přidat soubor FTDI.REG. Najdete jej ve stejném adresáři jako uvedený program.

Po spuštění **FTD2XXST.EXE** vybereme položku menu **File|New.** Nyní je nutno vyplnit obsah jednotlivých políček dle *obr. 5.11.*



Obr. 5.11 Konfiguračn i hodnoty pro pří pra vek FT232 TS T

Význam jednotlivých položek (obr. 5.11):

- Manufacturer- řetězec popisu výrobce (libovolný text v rozsahu 2 až 38 znaků),
- ManufacturerID - dva znaky identifikující výrobce v případě, že necháte sériová čísla výrobků generovat automaticky (podle data a času),
- Vendor ID - identifikační hexačíslo výrobce (0403 odpovídá FTDI; pro získání vlastního VID musíte složit registrační poplatek organizaci USB: www.usb.org),
- Product ID - identifikační hexačíslo výrobku (6001 odpovídá FTDI; pro svou potřebu jsem od FTDI obdržel číslo FA38, proto jej budu používat ve všech konstrukcích),
- Description - popis výrobku, napište Test bez jakýchkoliv mezer,
- Rev4 - zpřístupňuje další položky (revize 4):
  - ISO IN - vstupní endpoint je izochronní,
  - ISO OUT - výstupní endpoint je izochronní,
  - Int PDEN - povolení pull-down rezistorů,
  - Int Do SerNo - povolení programování sériového čísla,
  - Int USB Ver - povolení verze USB deskriptoru (0200 značí USB 2.0). Nakonec

klepněte do políčka pod Description. Tak se vám zpřístupní položka menu Device|Advanced Setup. Po její aktivaci se zobrazí další dialog, kde se vyplňují pokročilá nastavení.

Pokročilá nastavení (obr. 5.12):

U Plug and Play - povolení Plug&Play enumerace dostupných zařízení,

- Fixed Seriál Number - povoluje zadat sériové číslo přímo (libovolný řetězec délky 8 znaků),
- Šelf Powered - zařízení je napájeno z vlastního zdroje (ne ze sběrnice USB),
- Remote Wakeup - podporuje Remote Wakeup,
- Max Power - maximální odběr z USB sběrnice v mA (pokud není zvoleno Šelf Powered). Maximálně lze zadat 500 mA pro host a 100 mA pro hub.

Nakonec program vyžaduje uložení dané konfigurace položkou menu File|Save. Po této operaci je možno provést programování E^2PROM aktivací položky menu Device|Program.

Nové nastavení se projeví až po znovupřipojení zařízení ke kabelu (odpojíme a pak připojíme).

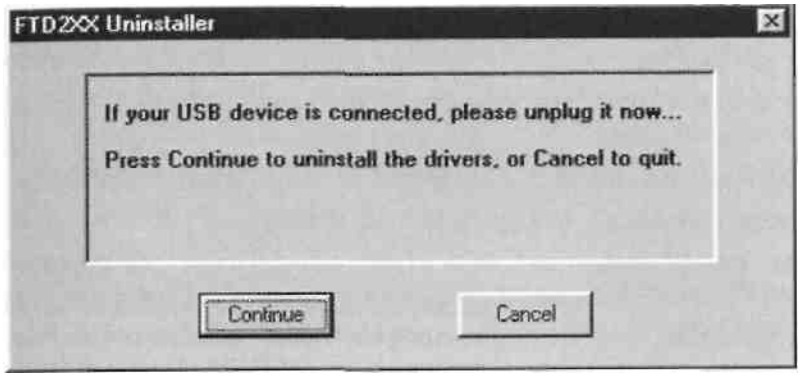
Obr. 5.12 Dialog pokročilých nastavení

5.4 AKTUALIZACE OVLADAČE PO ZMĚNĚ PID

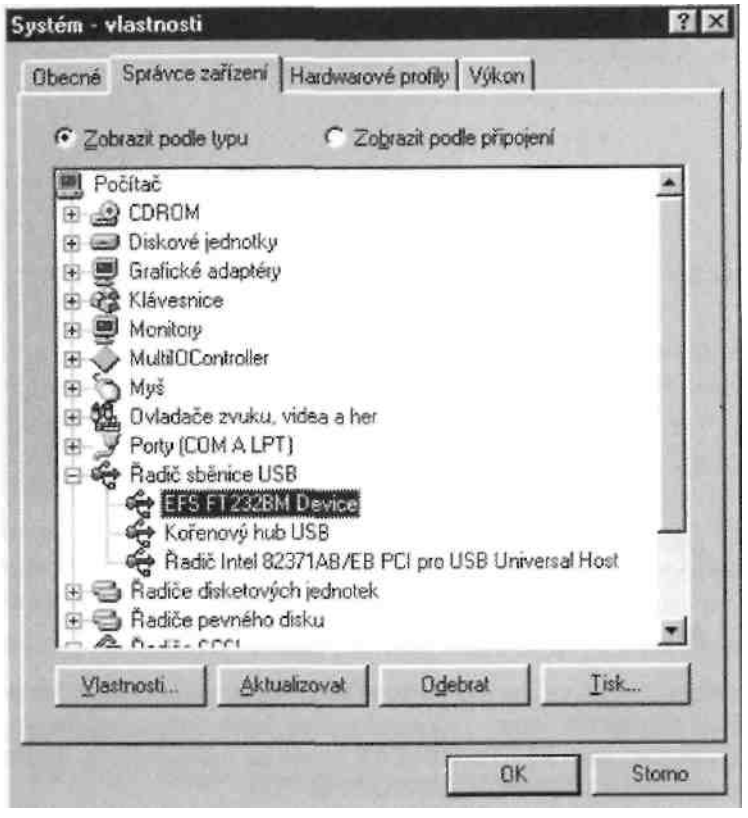
Aby bylo možno používat novou hodnou PID, musíme nejdříve odpojit zařízení a odinstalovat stávající ovladač (případně jej lze ponechat nainstalovaný). Odinstalování zajistí program FTD2XXUN.EXE (opět jej naleznete na doprovodném CD-ROM v adresáři FTDI\OVLADACE\D2XX\FTDI).

Po jeho spuštění jsme nejdříve dotázání, zda chceme odinstalování provést (viz *obr. 5.13),* což potvrdíme stiskem tlačítka Continue.

Potom připojíme přípravek FT232TST k USB. Zařízení se již ohlásí jako Test. Ovladače pro nové PID naleznete na doprovodném CD-ROM v adresáři FTDI\OVLADACE\D2XX\EFS. Nyní je zařízení ohlášeno jako EFS FT232BM Device (neboť tak jsem nastavil položky v instalačním souboru, další informace uvádí kapitola 13), viz *obr. 5.14.*



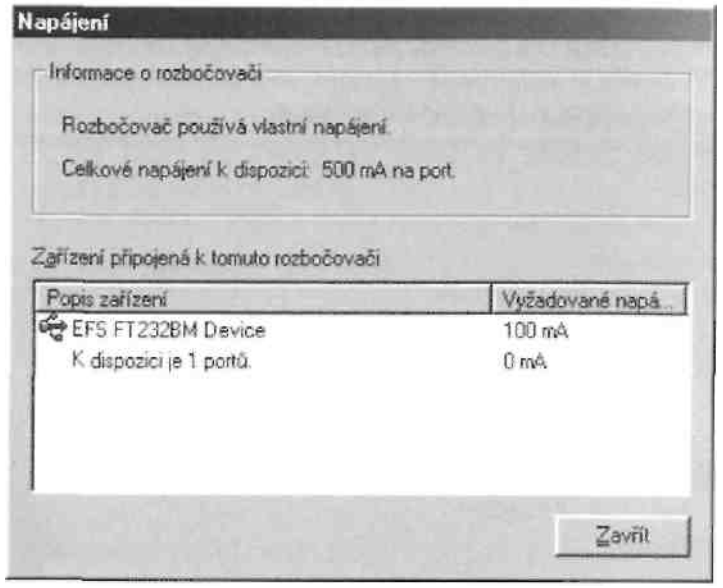
Obr. 5.13 Odinstalování stávajícího ovladače



Obr. 5.14 Nové zařízení je instalováno

Ještě můžeme ověřit zadaný proudový odběr. V dialogu dle *obr. 5.14* poklepejte na položku **Kořenový hub USB.** V nově zobrazeném dialogu vyberte kartu **Napájení** a stiskněte tlačítko **Vlastnosti napájení.** Výsledek je uveden na *obr. 5.15.*

Je na čase přikročit k ukázkám práce s přípravkem FT232TST.

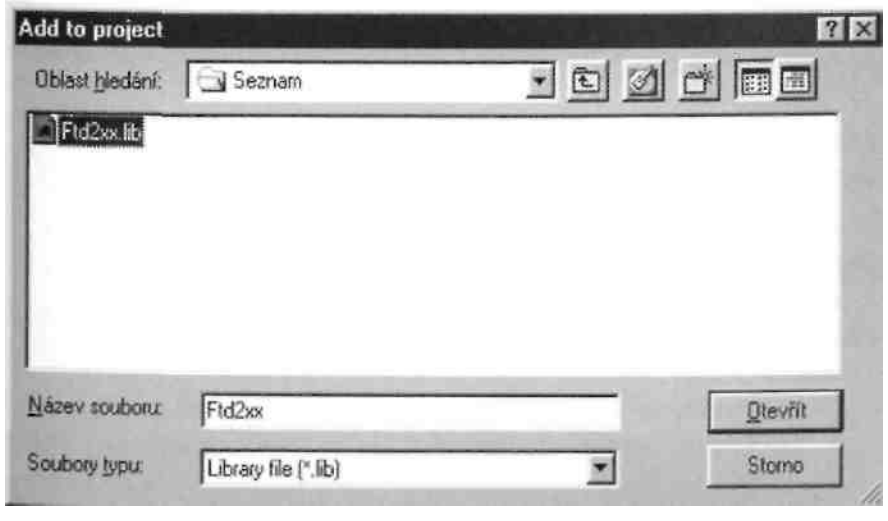


Obr. 5.15 Obsazené porty USB hubu a odběr připojeného zařízení

5.5 SEZNAM ZAŘÍZENÍ

První příklad ukazuje použití funkce **FT_ListDevice** pro zjištění počtu připojených zařízení, jejich popisu a sériových čísel.

Aplikace obsahuje komponentu **Timer** se jménem **Časovač** a komponentu **ListBox** se jménem **Seznam.** Po každém vypršení intervalu časovače (událost **Aktivace-Casovace)** se zjistí počet připojených zařízení, jejich popis a sériové číslo.



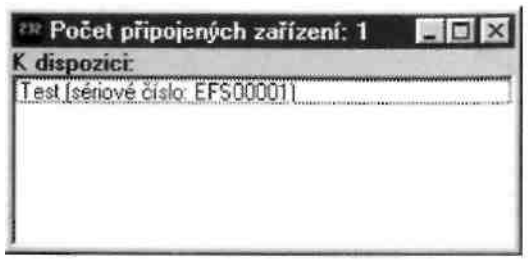
Obr. 5.16 Připojení importní knihovny do projektu

Pro použití dynamické knihovny **FTD2XX.DLL** musíme do projektu přidat importní knihovnu **FTD2XX.LIB** položkou menu **Project|Add To Project.** Viz *obr. 5.16.* Dynamická knihovna je instalována v systémovém adresáři operačního systému, proto ji nemusíte kopírovat do adresáře aplikace.

SEZNAMM.CPP:

```
// ---------------------------------------------------------------------------
#include   <vcl.h>
//hlavičkový   soubor   D2XX:
#include   "Ftd2xx.h"
#pragma   hdrstop
#include   "SeznamM.h"
// ---------------------------------------------------------------------------
#pragma   package(smart_init)
#pragma   resource   "*.dfm"
TForml   *Forml;
// ---------------------------------------------------------------------------
 _fastcall   TForml::TForml(TComponent*   Owner) :
             TForm(Owner)
{
    //první   aktualizace:
    AktivaceCasovace(NULL);
}
// ---------------------------------------------------------------------------
void    _fastcall   TForml::AktivaceCasovace(TObject   *Sender)
{
    //aktualizace   zajištěná   časovačem:
    FT_HANDLE   ftHandle; FT_STATUS   ftStatus;
    TStringList   *s=new   TStringList; DWORD
    Počet;
    //zjistí   počet   připojených   zařízení:
    ftStatus=FT_ListDevices(&Pocet,NULL,
        FT_LIST_NUMBER_ONLY); if
    (ftStatus==FT_OK) {
        AnsiString   Text;
        char   Buffer[64];
        //zjišťuje   popisy   zařízeni:
        for(DWORD    Index=0;Index<Pocet;Index++){
            ftStatus=FT_ListDevices((PVOID)Index,Buffer, FT   LIST   BY
              INDEX|FT   OPEN   BY   DESCRIPTION);
            if(ftStatus==FT_OK){
              Text=Buffer;
              //zjišťuje   sériové   číslo   zařízení:
              ftStatus=FT_ListDevices((PVOID)Index,Buffer,
                  FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);
              if(ftStatus==FT_OK){
                  //vytvoří   řádek   z   popisu   a   sériového   čisla: Text=Text
+AnsiString("   (sériové   číslo:   ") +Buffer+AnsiString( " ) " ) ;   s-
>Add(Text); } } } }
//zobrazí   počet   zařízeni:
Caption=AnsiString("Počet   připojených   zařízení:   ") +Pocet;

//změna   obsahu   Seznam,   když   došlo   ke   změně:
if(!Seznam->Items->Equals( s ) )
 Seznam->Items->Assign( s ); delete   s; }
```

Nejdříve se zjistí počet připojených zařízení a uloží se do proměnné **Počet.** Potom se zjišťují popisy a sériová čísla jednotlivých zařízení a ukládají se do pracovního seznamu s. Pokud je obsah listboxu **Seznam** odlišný od obsahu seznamu s, provede se zápis nových hodnot. Tak se tedy aktualizuje seznam připojených zařízení. Aktualizace je řízena časovačem, který má interval 1 s.



Obr. 5.17 Aplikace v akci

POZNÁMKA: *Další programy budou vytvořeny na podobném základě, budou tedy rovněž vkládat importní knihovnu* ***FTD2XX.LIB*** *do projektu.*

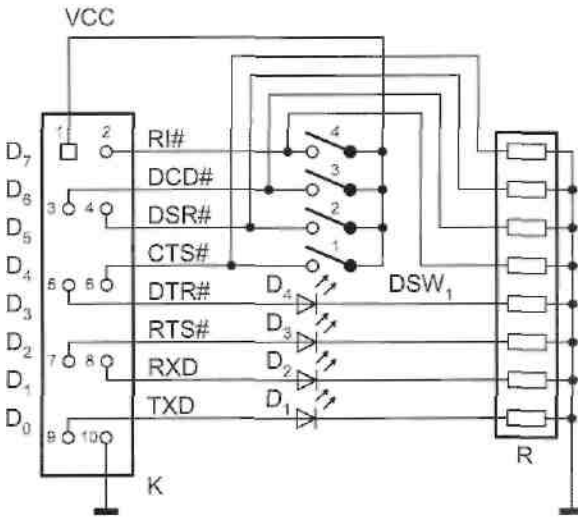
5.6 PŘÍMÉ ŘÍZENÍ LINEK MODEMU

Níže uvedené příklady ukazují přímé řízení linek modemu. Takto máme k dispozici 3 výstupy a 4 vstupy (přesně jako u sériového portu počítače). V mnoha případech takový počet ovládacích linek postačuje a není tedy nutno připojit mikrokontrolér, který by používal klasický asynchronní přenos dat.

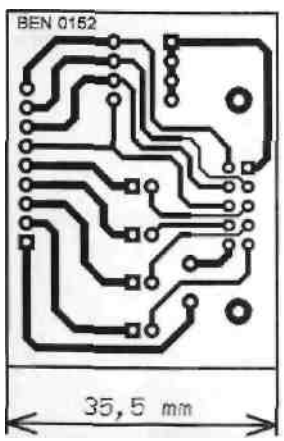
5.6.1 Přípravek FTDITEST

Než si ukážeme ovládání linek modemu, musíme vytvořit jednoduchý testovací přípravek, který umožní sledovat stav výstupů a nastavovat stav vstupů.

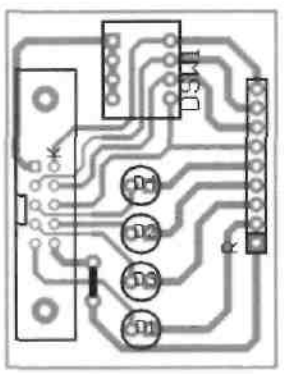
Na výstupy připojíme LED, vstupy budou opatřeny spínači. Výsledný přípravek jsem nazval **FTDITEST** a schéma zapojení je uvedeno na *obr. 5.18.*



Obr. 5.18 Schéma zapojení přípravku FTDITEST



Obr. 5.19 Výkres desky plošných spojů přípravku FTDITEST (BEN 0152)



Obr. 5.20 Osazovací plánek přípravku FTDITEST

LED svítí při log. 1. Vstupy jsou při rozepnutých spínačích taženy k log. 0, při sepnutí se přečte log. 1. Pro zmenšení rozměrů byly použity spínače ze 4násobného DIP označeného jako DSW, a dále odporová síť 8 odporů označená R.

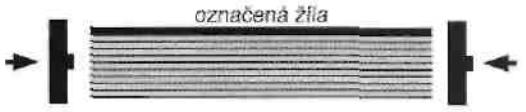
Připomeňme, že linka RXD není výstupní (při přímém řízení ji však nelze použít ani jako vstup), použití LED označené jako D_2 bude ukázáno v kapitole 5.7.1.

Rozpis součástek pro přípravek FTDITEST (cena asi 60 Kč):

K	PSL10	1 ks
D, až D_4	LED 5MM 200 MCD	4 ks
DSW,	DIP4X	1 ks
R	RR 8X470R (odporová síť 8 odporů 470 R)	1 ks

Konstrukce **propojovacího kablíku** je zřejmá z *obr. 5.21.* Potřebujeme dva konektory **PFL10** a kousek (cca 15 cm) plochého kabelu s 10 žilami (AWG28-10). Oba konektory posadíme na kabel dle obrázku. Pak jeden konektor po druhém vložíme do svěráčku a stáhneme. Tím je mechanicky připevníme ke kabelu.

Obr. 5.21 Konstrukce propojovacího kablíku se dvěma konektory PFL10



POZNÁMKA: *Vzhledem k tomu, že přípravek **FTDITEST** nemá vlastní napájení, musí být při jeho připojení k přípravku **FT232BM** vložen jumper J1 U níže uvedených případech připojíme přípravek FTDITEST k přípravku FT232TST Pomocí checkboxů TXD,RTSaDTRIze ovládat LED. Checkboxy RI, DCD, DSR a CTS sledují stav spínačů.*

5.6.2 Použití časovače

První příklad ukazující ovládání linek modemu je založen na použití časovače, který testuje stav vstupních linek s intervalem 100 ms. Zároveň je ukázáno otevření zařízení funkcí **FT_OpenEx.**

TESTLINEKM.H:

```
#ifndef   TestLinekMH
#define   TestLinekMH


// ------------------------------------------------------------------------------------------
class   TForml   :   public   TForm {
```

```
public:// User declarations
    _fastcall TForml(TComponent* Owner);
    _fastcall -TForml();
    FT_HANDLE ftHandle;                    //handle otevřeného zařízení
};
// ---------------------------------------------------------------------------
extern PACKAGE TForml *Forml;
// ---------------------------------------------------------------------------
#endif
```

TESTLINEKM.CPP:

```
#include <vcl.h>
#include "Ftd2xx.h"
#pragma hdrstop
#include "TestLinekM.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForml *Forml;
//---------------------------------------------------------------------------
_ fastcall TForml::TForml(TComponent* Owner)
        : TForm(Owner) {
    FT_STATUS ftStatus;

    //otevře zařízeni Test:
    ftStatus=FT_OpenEx("Test",
        FT_OPEN_BY_DESCRIPTION,&ftHandle);
    if(ftStatus!=FT__OK)                    //test úspěšnosti
        throw Exception("Zařízení test neni připojeno!");

    //vynuluje linky DTR#, TXD a RTS#:
    DTRClick(NULL);
    TXDClick(NULL);
    RTSClick(NULL);
}
//---------------------------------------------------------------------------
_ fastcall TForml: :-TForml()
i
    //zavře zařízeni:
    FT_Close(ftHandle); }
```

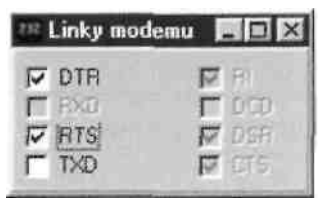


```
//---------------------------------------------------------------------------
void _fastcall TForml::DTRClick(TObject *Sender)
{
    //ovládá DTR#:
    if(!DTR->Checked)
        FT_SetDtr(ftHandle); else
        FT_ClrDtr(ftHandle);
}
//---------------------------------------------------------------------------
void ___fastcall TForml::TXDClick(TObject *Sender)
{
    //ovládá TXD:
    if ( !TXD->Checked)
        FT_SetBreakOn(ftHandle); else
        FT_SetBreakOff(ftHandle);
}
//---------------------------------------------------------------------------
void _fastcall TForml::RTSClick(TObject *Sender)
{
    //ovládá RTS#:
    if(!RTS->Checked)
        FT_SetRts(ftHandle);
    else
        FT_ClrRts(ftHandle);
}
// ---------------------------------------------------------------------------
void ___fastcall TForml::AktivaceCasovace(TObject *Sender)
{
    //periodické čteni stavu vstupů:
    DWORD Stav;

    //zjištěni stavu vstupních linek:
    FT_GetModemStatus(ftHandle,&Stav);

    //dekódováni:
    DCD->Checked=!(Stav&MS_RLSD_ON);
CTS->Checked=!(Stav&MS_CTS_ON); DSR-
>Checked=!(Stav&MS_DSR_ON); RI-
>Checked=!(Stav&MS_RING_ON); }
```

Všimněte si, že zapisované a čtené hodnoty jsou mezi přípravkem a programem vždy negovány. Například pro DTR# = 0 musíme volat funkci **FT_SetDtr,** kdežto pro DTR# = 1 voláme zase **FT_ClrDtr.**

Podobně při čtení linky **CTS#** vymaskujeme příslušnou hodnotu z výsledku funkce **FT_GetModemStatus** a před zobrazením znegujeme.

Toto chování odpovídá označení signálů (CTS# vlastně představuje $\overline{CTS}$), negace je nutná pro součinnost s konvertory úrovní RS-232C (viz kapitolu 12).



Obr. 5.22 Aplikace v akci

***POZNÁMKA:** Jinou možností, jak ovládat výstupní linky modemu, je použití funkce **FT_W32_EscapeCommFunction** (viz kapitolu 5.8).*

5.6.3 Kvazipřerušení (čekání na signalizaci události)

Použití časovače pro sledování stavu vstupních linek je poměrně jednoduché. V mnohých případech je ale výhodnější vytvořit aplikaci tak, aby pracovala podobně, jako když procesor dostane přerušení. Tedy aby se aktualizace checkboxů provedla jen v tom případě, že dojde ke změně stavu alespoň jedné vstupní linky.

K tomuto efektu lze využít funkci **FT_SetNotificationEvent.**

Zápis zdrojového textu je prakticky stejný jako v kapitole 5.6.2, místo obslužné událostí časovače je definována funkce **Aktualizace.** Pro realizaci kvazipřerušení je použito samostatné aplikační vlákno **EventThread.** Definice je v souborech **VLAKNO.H** a **VLAKNO.CPP**

KPM.H:

```
#ifndef KPMH
#define KPMH


#include "Vlákno.h"
#include "Ftd2xx.li"
//---------------------------------------------------------------------------
class TForml : public TForm {
```



```
public:// User declarations
    __fastcall TForml(TComponent* Owner);
    __fastcall -TForml();
    void ___fastcall Aktualizace();
    TEventThread *EventThread;
    FT_HANDLE ftHandle;
};
//---------------------------------------------------------------------------
extern PACKAGE TForml *Forml;
//---------------------------------------------------------------------------
#endif
```

KPM.CPP:

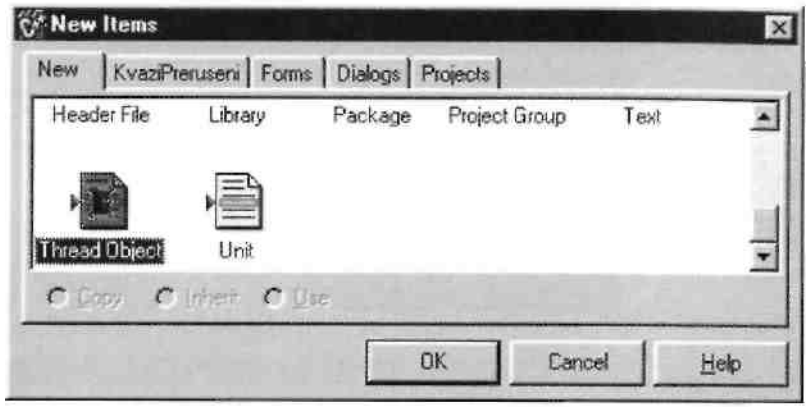
```
#include <vcl.h>
#pragma hdrstop
#include "KPM.h"
//---------------------------------------------------------------------------
tpragma package(smart_init)
#pragma resource "*.dfm"
TForml *Forml;
//---------------------------------------------------------------------------
__fastcall TForml::TForml(TComponent* Owner)
                    : TForm(Owner) {
  FT_STATUS ftStatus;
  //otevře zařízeni Test:
  ftStatus=FT_OpenEx("Test",
     FT_OPEN_BY_DESCRIPTION,&ftHandle); if (ftStatus!=FT_OK) throw
  Exception("Zařízeni test neni připojeno!");

  //vynulováni výstupních linek:
  DTRClick(NULL); TXDClick(NULL);
  RTSClick(NULL);

  //počáteční stav vstupních linek:
  Aktualizace();
  //spuštěni vlákna, které sleduje vstupy:
  EventThread=new TEventThread(falše);
}
//---------------------------------------------------------------------------
```

```
 _fastcall   TForml::-TForml()
{
    //ukončení   činnosti   vlákna:
    delete   EventThread;
    //zavření   zařízení:
    FT_Close(ftHandle);
}
//---------------------------------------------------------------------------
void    _fastcall   TForml::DTRClick(TObject   *Sender)
{
    if(!DTR->Checked)
        FT_SetDtr(ftHandle); else
        FT_ClrDtr(ftHandle);
}
//---------------------------------------------------------------------------
void    _fastcall   TForml::TXDClick(TObject   *Sender)
{
    if ( !TXD->Checked)
        FT_SetBreakOn(ftHandle); else
        FT_SetBreakOff(ftHandle);
}
// ---------------------------------------------------------------------------
void  ____fastcall   TForml::RTSClick(TObject   *Sender)
{
    if(!RTS->Checked)
        FT_SetRts(ftHandle);
    else
        FT_ClrRts(ftHandle);
}
// ---------------------------------------------------------------------------
void    _fastcall   TForml::Aktualizace() { DWORD
Stav;

    //aktualizuje   stav   vstupů:
    FT_GetModemStatus(ftHandle,&Stav); DCD-
    >Checked=!(Stav&MS_RLSD_ON); CTS-
    >Checked=!(Stav&MS_CTS_ON); DSR-
    >Checked=!(Stav&MS   DSR  ON);
```



```
RI->Checked=!(Stav&MS_RING_ON); }
```

Nové aplikační vlákno vytvoříme tak, že aktivujeme položku menu **File|New** a z dialogu vybereme **Thread Object.** Viz *obr. 5.23.*



Obr. 5.23 Dialog pro vytvoření vlákna

Zobrazí se nám dialog dle *obr. 5.24* a zde uvedeme název třídy vlákna, v našem případě **TEventThread.**

Obr. 5.24 Dialog pro stanovení třídy vlákna

Zde je definice metod vlákna:

```
VLÁKNO.H:
#ifndef  VlaknoH
#define   VlaknoH
// ---------------------------------------------------------------------------
#include   <Classes.hpp>
// ---------------------------------------------------------------------------
class   TEventThread   :   public   TThread {
```

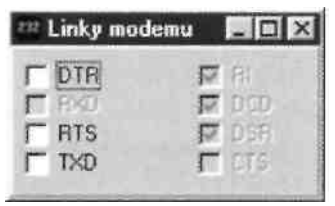
```
private:
    HANDLE   hEvent;                          //handle   události
    DWORD   Status;                           //stav   operace
protected:
    void    __ fastcall   Execute();          //jádro   obsluhy  vlákna
public:
        fastcall   TEventThread(bool   CreateSuspended); _fastcall   -
     TEventThread() ;
};
// ---------------------------------------------------------------------------
#endif
```

VLÁKNO.CPP:

```
#include   <vcl.h>
#pragma   hdrstop
#include   "Vlákno.h"
#include   "KPM.h"
#pragma   package(smart_init)
// ---------------------------------------------------------------------------
 _fastcall   TEventThread::TEventThread(bool   CreateSuspended) :
             TThread(CreateSuspended)
{
    //založeni   události:
    hEvent=CreateEvent(NULL,FALŠE,FALŠE,"");
    if(!hEvent)   //test   selháni:
        throw   Exception("Vlákno   nelze   vytvořit");

    //událost   bude   monitorovat   změnu   stavu   linek   modemu:
    FT_SetEventNotification(Forml->ftHandle,
        FT_EVENT_MODEM_STATUS,hEvent);
}
// ---------------------------------------------------------------------------
_ fastcall   TEventThread::-TEventThread()
{
    //zrušeni   události:
    CloseHandle(hEvent);
}
//---------------------------------------------------------------------------
void ____fastcall   TEventThread::Execute()
{
    //jádro   obsluhy   vlákna
    while(!Terminated){
```



```
        //nekonečný   cyklus   (ukonči   se   s   koncem   aplikace)

        //test   signalizace   hEvent,   time-out   100   ms
        if(WAIT_OBJECT_0==WaitForSingleObject(hEvent,100))
//signalizováno->vyvolej   Aktualizace: Synchronize(Forml-
>Aktualizace); } }
```



Obr. 5.25 Aplikace v akci

POZNÁMKA: Pro realizaci byly použity další funkce Win32 API, které nelze na tomto místě popsat, protože přesahují rámec této knihy. Informace lze najít v nápovědě Windows SDK nebo v [12].

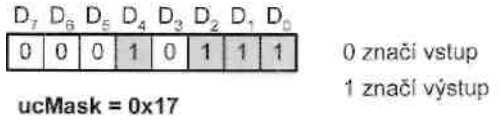
5.7 ŘÍZENÍ LINEK V REŽIMU BIT BANG

Tato kapitola ukazuje použití režimu Bit Bang (linky UART rozhraní přejdou do paralelního režimu) na jednoduchém příkladu.

5.7.1 Použití přípravku FTDITEST

V tomto příkladu si ukážeme, jak použít výše popsaný přípravek FTDITEST v režimu Bit Bang. Viz *obr. 5.18.*

Po otevření zařízení musíme aktivovat režim Bit Bang voláním funkce FT_SetBitMode, zde se zároveň stanoví, které vývody jsou konfigurovány jako vstupní a které jako výstupní (situace musí odpovídat zapojení přípravku FTDITEST). Viz *obr. 5.26.*



Obr. 5.26 Konfigurační hodnota pro FT_SetBitMode

Nakonec nastavíme přenosovou rychlost na 300 Bd. Reakce na ovládací checkboxy vstupů a výstupů je nyní zapsána do společné obsluhy časovače. Zápis dat je proveden funkcí FT_Write, čtení pak pomocí funkce FT_GetBitMode.

BITBANG1M.H:

```
#ifndef BitBangIMH
#define BitBangIMH


//---------------------------------------------------------------------------
class TForml : public TForm {


public:// User declarations
    _ fastcall TForml(TComponent* Owner);
    _fastcall -TForml();
    FT HANDLE ftHandle;          //handle zařízeni
    FT_STATUS ftStatus;          //stav provedené operace
};
//---------------------------------------------------------------------------
extern PACKAGE TForml *Forml;
//---------------------------------------------------------------------------
#endif
```

BITBANG1M.CPP:

```
#include <vcl.h>
#include "Ftd2xx.h"
#pragma hdrstop
#include "BitBangIM.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForml *Forml;
//---------------------------------------------------------------------------
_ fastcall TForml::TForml(TComponent* Owner)
        : TForm(Owner) {
    //otevře zařízeni Test:
    ftStatus=FT_OpenEx("Test",
        FT_OPEN_BY_DESCRIPTION,&ftHandle); if
    (ftStatus!=FT_OK)
        throw Exception("Zařízeni Test neni připojeno!");

    //přepne do režimu Bit Bang:
```



```
    ftStatus=FT_SetBitMode(ftHandle,0x17,1);
    if(ftStatus!=FT_OK)
    throw Exception("Zařízeni nelze přepnout" " do režimu Bit
    Bang!"); //nastaveni komunikační rychlosti:
    ftStatus=FT_SetBaudRate(ftHandle,300); if (ftStatus!=FT_0K)
        throw Exception("Nelze nastavit přenosovou" " rychlost
            300 Bd!");
}
//---------------------------------------------------------------------------
 _fastcall TForml::-TForml()
{
    //zrušeni režimu Bit Bang:
    FT_SetBitMode(ftHandle,0x00, 0) ; //zavře
    zařízeni: FT_Close(ftHandle);
}
//---------------------------------------------------------------------------
void _fastcall TForml::AktivaceCasovace(TObject *Sender)
{
    //aktualiace vstupů a výstupů: DWORD
    počet;

//sestavení dat pro odeslání na výstupy: UCHAR
data
    =D4->Checked*16
    +D2->Checked*4
    +DI->Checked*2
    +D0->Checked;

//odesláni výstupů:
ftStatus=FT_Write(ftHandle,&data,1,&pocet);
if(ftStatus!=FT_OK){
    Casovac->Enabled=false;
MessageBox(Handle,"Zápis selhal", "Test Bit
Bang",MB_ICONHAND) ; }
//čteni vstupů:
ftStatus=FT_GetBitMode(ftHandle,&data);
if(ftStatus!=FT_OK){
    Časovač->Enabled=false;
```

```
    MessageBox(Handle,"Čtení    selhalo", "Test   Bit
        Bang",MB_ICONHAND);
}
else{
    //dekódování   vstupů: D7-
    >Checked=data&0x8 0; D6-
    >Checked=data&0x4 0; D5-
    >Checked=data&0x20; D3-
    >Checked=data&0x08;
}}
```

Obr. 5.27 Aplikace v akci

POZNÁMKA: V režimu Bit Bang nejsou linky negovány!

5.8 TESTOVACÍ PŘÍPRAVEK S D/A PŘEVODNÍKEM TC1320

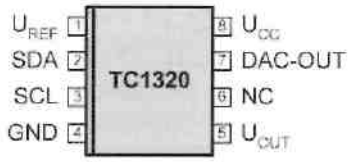
V kapitole 10 je uvedena konstrukce měřicí desky. Jednou z částí této desky je D/A převodník TC1320 od firmy Microchip. Vzhledem ktomu, že se jedná o relativně neznámý obvod, bude třeba uvést krátký popis. Tento popis je spojen s ukázkou tvorby levného D/A převodníku s rozlišením 8 bitů a pracovním rozsahem 0 až 2,5 V.

5.8.1 Stručný popis obvodu TC1320

Obvod TC1320 je D/A převodník od firmy Microchip ovládaný sériovou sběrnicí, klíčové vlastnosti (cena cca 50 Kč):

- 8bitové rozlišení,
- integrální nelinearita ±2 LSB, diferenciální nelinearita ±0,8 LSB,
- jednoduché napájení v rozsahu 2,7 až 5,5 V,
- zabudovaná funkce Power-On reset (vynulování po připojení napájecího napětí),
- komunikace pomocí sběrnice I^2C,

- klidový odběr 350 uA, snížený odběr 0,5 uA,
- k dispozici pouze v 8vývodových pouzdrech pro plošnou montáž: SOIC (značení TC1320EOA) nebo MSOP (TC1320EUA).

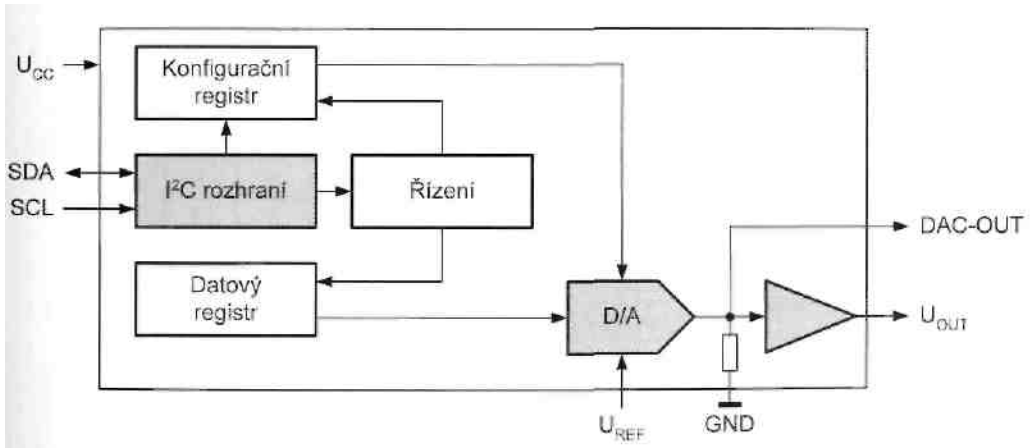


Funkce vývodů:

U_{cc} - napájecí napětí, **GND** - signálová a napájecí zem, U_{REF} - referenční napětí, DAC-OUT - přímý výstup, U_{0UT} - buferovaný výstup, SDA, SCL - linky sběrnice I^2C

Obr. 5.28 Rozložení vývodů na pouzdrech SOIC a MSOP

Vnitřní schéma obvodu TC1320 je uvedeno na *obr. 5.29.* Napájení v rozsahu 2,7 až 5,5 V se připojuje mezi vývody **GND** a U_{cc}. Referenční napětí odpovídá vývodu U_{REF} a musí být v rozsahu 0 až U_{cc} - 1,2 V.



Obr. 5.29 Vnitřní zapojení obvodu TC1320

Výstupy jsou dva. První výstup je označen DAC-OUT, jedná se o přímý výstup D/A převodníku, který má poměrně velký výstupní odpor (lze jej zatěžovat pouze vysokým odporem). Druhý výstup je označen U_{0UT} a odpovídá výstupu ze sledovače (buferovaný výstup).

Dále je obvod opatřen linkami sériové sběrnice I^2C (SDA a **SCL).**

Mezní a charakteristické údaje

Mezní a charakteristické údaje jsou uvedeny formou *tab. 5.1 a tab. 5.2* (překročení mezních údajů může vést k trvalému poškození obvodu; nelze dosáhnout dvou a více mezních hodnot současně).

Parametry sběrnice PC

Pro úspěšnou komunikaci s obvodem TC1320 je důležité dodržet potřebné předstihy a přesahy dat na sběrnici I^2C.

Tab. 5.1 Mezní údaje obvodu TC1320

Parametr	Hodnota
Napájecí napětí	max. +6 V
Napětí libovolného vývodu	GND – 0,3 V až U_{CC} + 0,3 V
Provozní teplota	–40 °C až +85 °C

Tab. 5.2 Statické a dynamické údaje obvodu TC1320

Symbol	Parametr	min	typ	max	jed.
U_{CC}	Napájecí napětí	2,7	–	5,5	V
I_{CC}	Klidový odběr	–	350	500	µA
I_{CC-STB}	Odběr v režimu standby	–	0,1	1	µA
R	Rozlišení	–	–	8	bitů
INL	Integrální nelinearita	–	–	±2	LSB
DNL	Diferenciální nelinearita	–	–	±0,8	LSB
U_{OS}	Napěťový ofset výstupu	–	±0,3	±8	mV
U_{REF}	Referenční napětí	0 až U_{CC} – 1,2			V
I_{OUT}	Výstupní proud	–	2	–	mA
I_{SC}	Výstupní zkratový proud	–	20	50	mA
SWR	Rychlost přeběhu	–	0,8	–	V/µs
t_S	Doba ustálení	–	10	–	µs
U_{IH}	Úroveň vstupní log. 1	2,4	–	U_{CC}	V
U_{IL}	Úroveň vstupní log. 0	0	–	0,6	V
U_{OL}	Úroveň výstupní log. 0	–	–	0,4	V

Tab. 5.3 Parametry sběrnice I^2C

Symbol	Parametr	min	typ	max	jed.
f_{SMB}	Kmitočet sběrnice	10	–	100	kHz
t_R	Doba náběhu	–	–	300	ns
t_F	Doba poklesu	–	–	1000	ns
t_{LOW}	Doba trvání log. 0 na SCL	4,7	–	–	µs
t_{HIGH}	Doba trvání log. 1 na SCL	4	–	–	µs
t_{SETUP}	Doba předstihu dat na SDA	100	–	–	ns
t_{HOLD}	Doba přesahu dat na SDA	100	–	–	ns

Výstupní napětí

Výstupní napětí obvodu TC1320 je dáno vztahem (N je vstupní číslo):

$$\frac{N \cdot U_{REF}}{256}$$

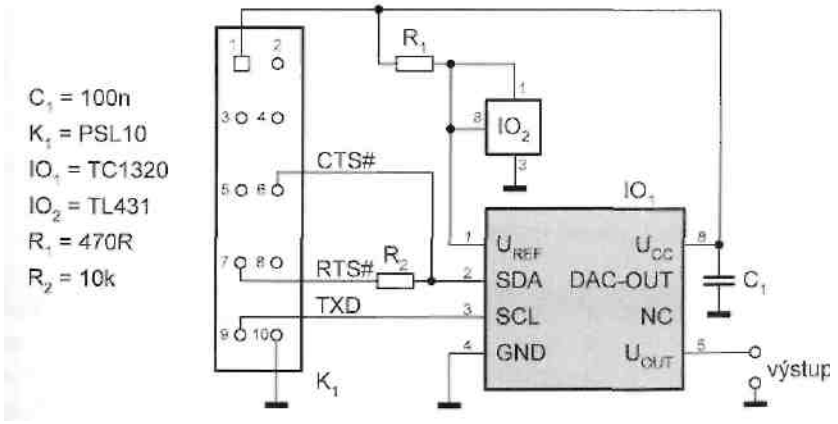
Stručný popis komunikace po sběrnici PC

Zápis dat do datového registru obvodu TC1320 probíhá takto (podrobnější popis sběrnice I^2C lze nalézt například v [1], [2], [3] nebo [15] nebo v datasheetu obvodu na doprovodném CD-ROM):

- nejdříve je třeba vyslat stav sběrnice označovaný jako **START** (sestupná hrana SDA při SCL=1),
- poté musí být vyslána 8bitová **adresa** oslovovaného obvodu (pro TC1320 je to hodnota 1001 0000b), adresa je obvodem potvrzena stažením linky SDA po dobu dalšího hodinového cyklu SCL (tento stav sběrnice se označuje jako **ACK),**
- následuje vyslání **příkazu** (při posílání dat má příkaz hodnotu 0000 0000b), následuje **ACK,**
- nakonec se vyšlou **data** pro D/A převodník následovaná **ACK,**
- sekvenci uzavírá **STOP** (náběžná hrana SDA při SCL = 1).

5.8.2 Testovací přípravek TC1320

Schéma zapojení testovacího přípravku pro obvod **TC1320** je uvedeno na *obr. 5.30.*



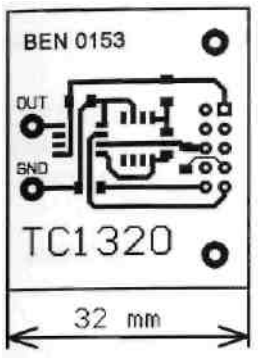
Obr. 5.30 Schéma zapojení přípravku TC1320

Referenční napětí (zhruba 2,5 V) je získáno pomocí referenčního zdroje **TL431** (IO_2), rezistor *R^* definuje pracovní proud zhruba 4 mA. Napájecí napětí ze sběrnice USB je blokováno kondenzátorem C_v

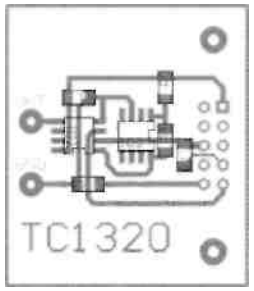
Připojení linek je řešeno takto: **SCL** odpovídá **TXD, SDA** pak odpovídá **RTS#** (ve výstupním směru) a **CTS#** (při čtení). Možné kolizi výstupů obvodu **TC1320** (IO0 s obvodem FT232BM brání rezistor R_2.

POZNÁMKA: *Při připojení testovací desky **TC1320** k přípravku **FT232TST,** musí být J_1 vložen (přípravek **TC1320** nemá vlastní napájení).*

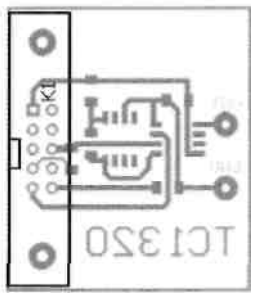
Výkres desky plošných spojů a osazovací plánky uvádí *obr. 5.31* až *obr. 5.33.* Mimo konektoru K., (PSL10) jsou použity výhradně součástky pro povrchovou montáž (SMD). Ostatně obvod TC1320 je vyráběn pouze jako SMD.



Obr. 5.31 Výkres desky plošných spojů přípravku TC1320 (BEN 0153)



Obr. 5.32 Osazovací plánek (strana spojů)



Obr. 5.33 Osazovací plánek (strana součástek)

Rozpis součástek pro přípravek TC1320:

C_1	CK+100NX7R TC1320EOA TL431 SMD PSL10	1 ks 1
io1	RR+470R SMD RR+10KSMD RR+OR SMD	ks 1
$I0_2$		ks 1
K_1		ks 1
Ri R_2		ks
=L		1 ks
		2 ks

5.8.3 Příklad použití funkcí FT_Win32 API

Na příkladu komunikace s obvodem TC1320 budou zároveň předvedeny základní funkce FT_Win32 API (popis viz kapitolu 3.4).

Nejdříve je nutno otevřít zařízení funkcí FT_W32_CreateFile, k identifikaci jsem použil jméno zařízení (souvisí s ním atribut **FT_OPEN_BY_DESCRIPTION).** Opět získáme handle zařízení typu FTJHANDLE Zavření zařízení se provádí voláním funkce FT_W32_CloseHandle.

Zapojení přípravku bylo voleno tak, aby bylo možno ovládat vývody obvodu TC1320 přímo jako linky modemu (není tedy nutno přepínat do režimu Bit Bang). Výstupní linky **TXD,** RTS# a **DTR#** jsou pak řízeny funkcí FT_W32_EscapeCommFunction. Opět připomeňme fakt programové negace linek. Například volání funkce:

FT_W32_EscapeCommFunction(ftHandle,SETRTS);

způsobí vynulování linky **RTS#.** Pro zkrácení zápisu metod **SDA** a SCL byl použit ternární operátor (?:). Méně znalým programátorů připomenu, že zápis:

FT_W32_EscapeCommFunction(ftHandle,b?CLRRTS:SETRTS); má stejný efekt jako: if (b)

FT_W32_EscapeCommFunction(ftHandle,CLRRTS); **else**
FT_W32_EscapeCommFunction(ftHandle,CLRRTS);

Dále byla použita funkce FT_W32_GetCommModemStatus. Touto funkcí se čte stav vývodu SDA (tedy linky **CTS#)** v okamžiku potvrzení komunikace. Podobně jako u funkce FT_GetModemStatus (viz kapitolu 5.6) je třeba načíst stav všech vstupních linek modemu a poté provést vymaskování příslušného bitu operátorem & (opět působí programová negace).

TC1320M.CPP:

```
// ---------------------------------------------------------------------------
#include   <vcl.h>
#include   "Ftd2xx.h"
#pragma   hdrstop
#include    "TC1320M.h"
// ---------------------------------------------------------------------------
#pragma   package(smart_init)
#pragma   resource   "*.dfm"
TForml   *Forml;
// ---------------------------------------------------------------------------
__fastcall   TForml::TForm1TComponent*   Owner)
                :   TForm(Owner) {
    //otevře   zařízeni   Test:
    ftHandle=FT_W32_CreateFile(
    "Test",
    GENERIC_READ|GENERIC_WRITE,
```

```
    0,
    NULL,
    OPEN_EXISTING,
    FT_OPEN_BY_DESCRIPTION,
    0);
    //test úspěšného otevření:
    if(ftHandle==INVALID_HANDLE_VALUE)
        throw Exception("Zařízeni test neni připojeno!");
}
//---------------------------------------------------------------------------
 _fastcall TForml::-TForm1)
{
    //zavřeni: FT_W32_CloseHandle(ftHandle) ;
}
//---------------------------------------------------------------------------
void __ fastcall TForm1:SDA(bool b)
{
    //ovládá SDA:
    FT_W32_EscapeCoiranFunction (
        ftHandle,b?CLRRTS:SETRTS);
}
//---------------------------------------------------------------------------
void __ fastcall TForm1:SCL(bool b)
{
    //ovládá SCL:
    FT_W32_EscapeCommFunction(
        ftHandle,b?CLRBREAK:SETBREAK);
}
//---------------------------------------------------------------------------
void __ fastcall TForm1:Start()
{
    //vyšle sekvenci START:
    SDA(l);
    SCL (1) ;
    Sleep(l);
    SDA(O);
    Sleep(l);
    SCL(O);
    Sleep(l); }
```



```
//---------------------------------------------------------------------------
void ___fastcall TForm1:Posli(Byte b)
{
    //vyšle bajt b, testuje odpověď obvodu: DWORD d;

    //pošle 8 bitů bajtu b:
    for(int i=0;i<8;i++){
        SDA(b&0x80);            //pošli nejvyšší bit
        SCL(l);                 //potvrď náběžnou hranou SCL
        Sleep(l);               //ustáleni
        b<<=l;                  //posuň bity vlevo
        SCL(O);                 //příprava na další bit
        Sleep(l);
    }

    //test odpovědi obvodu:
    SDA(l);
    SCL(l);
    FT_W32_GetCoinmModemStatus (ftHandle, &d) ;
    if(!(d&MS_CTS_ON))          //test ACK
        throw Exception("Obvod neni připojen");
    Sleep(l);
    SCL(O);
    Sleep(l);
}
//---------------------------------------------------------------------------
void ___fastcall TForm1:Stop()
{
    //vyšle sekvenci STOP:
    SDA(O) ;
    Sleep(l);
    SCL(l);
    Sleep(l);
    SDA(l);
}
//---------------------------------------------------------------------------
void ___fastcall TForm1:ScrollBarScroll(TObject *Sender,
    TScrollCode ScrollCode, int &ScrollPos) {
    //příprava dat pro odesláni:
    Byte Adresa=0x90;
```

```
Byte   Prikaz=0;
Byte   Data=ScrollBar->Position;
double  Uref=2.52;                    //skutečná hodnota ref. napěti
Label->Caption=FormatFloat("0.00   V",Data*Uref/256) /

//zápis,  jen  když  je  scroll  bar  uvolněn:
if(ScrollCode==scEndScroll){
    //celý  zápis  do  obvodu  TC1320:
    Start();                          //START
    Posli(Adresa);                    //adresa=1001   0000b
    Posli(Prikaz);                    //přikaz=0000   0000b
    Posli (Data) ;                    //N
    Stop();                           //STOP
}
```

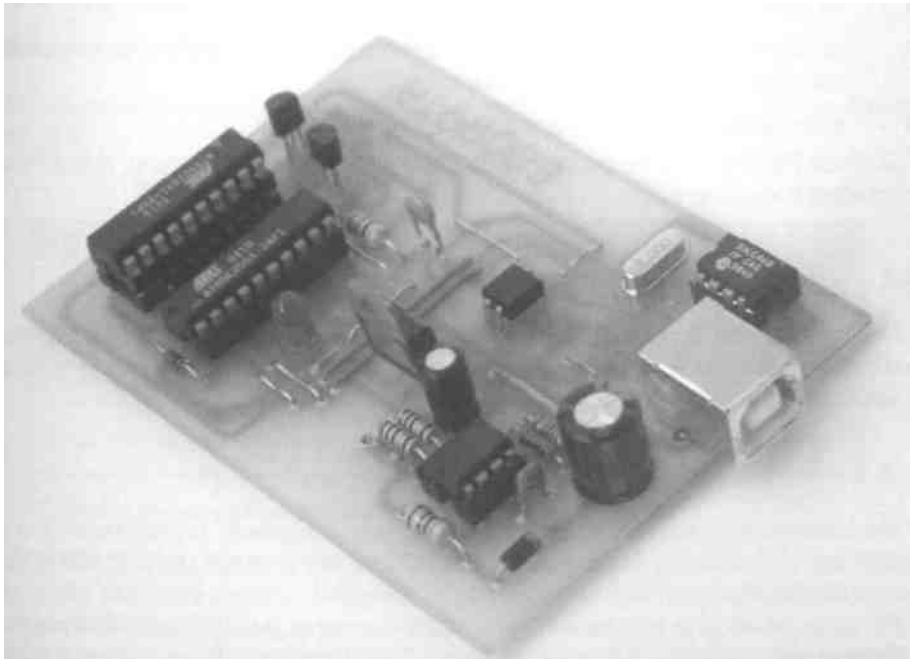
Obr. 5.34 Aplikace v akci

Pokud není přípravek připojen, je o tom uživatel informován. Tento stav se rozezná testováním stavu **ACK.** Viz *obr. 5.35.*

Obr. 5.35 Stav, když je přípravek odpojen

5.9 DALŠÍ PŘÍKLADY

Další příklady jsou uvedeny zejména v kapitole 8. Jedná se o ukázky asynchronní sériové komunikace mikrokontrolérů **AT89C2051** nebo **AT90S2313** s obvodem FT232BM.



*ATPROG 3.0 je již třetí verzí programátoru mikrokontroléru **AT89C2051.** Jedná se o upravené zapojení programátoru, které bylo uvedeno v [1] a [16].*

Místo připojení k sériovému portu je použit port USB. Dále je programátor napájen přímo z USB, což výrazně zjednodušuje manipulaci s ním.

6.1 VÝCHOZÍ IDEA

Musím se přiznat, že původní ideou bylo vytvořit zajímavou aplikaci pro tuto knihu. Aplikaci, na které by se ukázaly další možnosti poskytované obvodem **FT232BM.**

V [1] a [16] byl popsán programátor **ATPROG 2.1,** který se připojoval k sériovému portu. Tento programátor byl vytvořen na základě mikrokontroléru **AT89C2051** (tímto mikrokontrolérem se programoval jiný, který byl zasunut v programovací patici) a posuvného registru **4094.**

Programování bylo poměrně rychlé. V nové konstrukci jsem však chtěl provést dosti výrazná vylepšení:

- vyhnout se nutnosti použít obvod 4094 (obsloužit všechny programovací linky pouze mikrokontrolérem případně pomocí řídicích linek modemu),
- napájet programátor ze sběrnice USB (programovací napětí 12 V se získá pomocí měniče),
- možnost přerušit programování a čtení obsahu Flash (původní verze programovacího mikrokontroléru to „nedovedla"),
- možnost používat zámky pro Flash (ATPROG 2.1 nepodporoval),
- použití krystalu 6 MHz (synchronizační zdroj pro FT232BM) i programovacím mikrokontrolérem (určité snížení ceny),
- srovnatelná rychlost programování jako u ATPROG 2.1.

Je třeba říci, že všechny požadované vlastnosti se podařilo naplnit, popis řešení je uveden v kapitole 6.3.

6.2 PROGRAMOVACÍ ALGORITMUS

Než se pustíme do popisu vlastní realizace programátoru **ATPROG 3.0,** bude nezbytně nutné uvést programovací algoritmus a hlavně *tab. 6.1.* Ta je totiž pro výklad zapojení rozhodující.

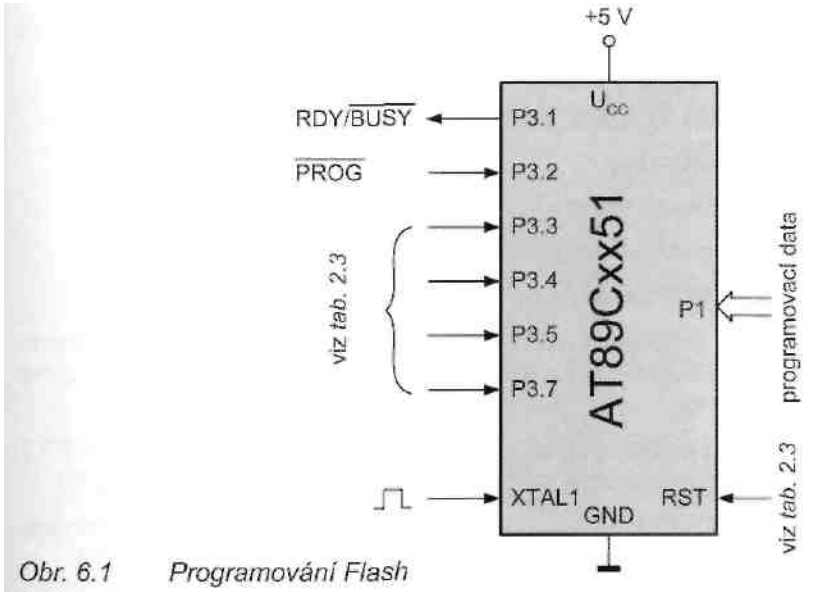
6.2.1 Paměť Flash

Mikrokontroléry typu AT89Cx051 mají programovou paměť realizovánu jako paměť Flash (E^2PROM) o kapacitách: 1 KB **(AT89C1051), 2 KB (AT89C2051)** nebo 4 KB **(AT89C4051).**

Při koupi obvodu je tato paměť ve smazaném stavu (každá buňka má hodnotu FFh) a je tedy připravena k programování. Pokud je paměť Flash naprogramována, musí se před novým programováním (elektricky) smazat.

6.2.2 Vnitřní adresový čítač

Mikrokontrolér obsahuje vnitřní adresový čítač paměti Flash. Tento čítač se nuluje (dosáhne hodnoty OOOOh) po příchodu náběžné hrany na vývod **RST** (reset). Obsah čítače je inkrementován (zvýšen o 1) po příchodu kladného impulzu (spojení náběžné a sestupné hrany) na vývod **XTAL1.**



Obr. 6.1 Programování Flash

6.2.3 Režimy programování

Mikrokontroléry AT89Cx051 rozlišují 5 programovacích režimů. Vše je zřejmé *z tab. 6.1.*

Tab. 6.1 Režimy programování

Režim		RST	P3.2	P3.3	P3.4	P3.5	P3.7
Čtení signatury		1	1	0	0	0	0
Čtení programových dat		1	1	0	0	1	1
Mazání obsahu		12 V	⎍*	1	0	0	0
Zápis programových dat		12 V	⎍	0	1	1	1
Zápis zámku	Bit – 1	12 V	⎍	1	1	1	1
	Bit – 2	12 V	⎍	1	1	0	0

* mazací impulz musí trvat alespoň 10 ms

Všimněte si, že hodnoty linek **P3.5** a **P3.7** jsou ve všech případech stejné! Není tedy nutno vésti tyto signály odděleně, tím se uspoří jeden vývod programovacího mikrokontroléru.

Dále je zajímavé, že na vývod **RST** je programovací napětí (12 V) připojeno jen tehdy, když je alespoň jedna z linek **P3.3** nebo **P3.4** v log. 1. Takže spínání progra-

movacího napětí může probíhat právě pomocí těchto linek (opět ušetříme jeden vývod programovacího mikrokontroléru).

Smazání obsahu

Obsah Flash je mazán elektricky použitím odpovídající kombinace dle *tab. 6.1.* Vývod P3.2 musí být držen v log. 0 alespoň po dobu 10 ms. Po smazání je obsah všech buněk roven FFh. Mazání je nutné před každým programováním.

Zápis programových dat

Doporučený algoritmus programování je tento:

1) připojení napájení: U_{cc} připojit na 5 V, RST = XTAL1 = 0 V,
2) RST = log. 1, P3.2 = log. 1,
3) nastavení kombinace vývodů P3.3 až P3.7 dle *tab. 6.1,*
4) přivedení bajtu z adresy OOOOh na port P1,
5) náběh RST na 12 V (programovací napětí) pro povolení programování,
6) záporný impulz (kombinace sestupné a náběžné hrany) na P3.2 pro potvrzení zápisu, zapisovací cyklus je časován automaticky a trvá zhruba 1,2 ms (maximálně 2 ms),
7) kontrola zápisu (verifikace) připojením RST na log. 1 a vývodů P3.3 až P3.7 dle *tab. 6.1* (Čtení programových dat), data jsou k dispozicí na portu P1,
8) k programování dalšího bajtu musíme vytvořit kladný impulz (kombinace náběžné a sestupné hrany) pro XTAL1 a připojit nová data na port P1,
9) opakování kroků 5 až 8 tak dlouho, dokud se nezapíše celý program,
10) odpojení napájení: RST = XTAL1 = 0 V, U_{cc} = 0 V.

Průběh programování je monitorován linkou P3.1, která je shozena do log. 0 po aplikaci náběžné hrany P3.2. Po uplynutí doby nutné pro provedení zápisu (typicky 1,2 ms a maximálně 2 ms) se P3.1 vrátí na log. 1 a tím indikuje úspěšné provedení zápisu. Tato možnost však nebude použita (monitorování vývodu P3.1 by bylo patrně časově náročnější než vložení čekacího intervalu).

Zápis zámku

Zápis zámku probíhá prakticky stejně, jako zápis programových dat. Odlišnost spočívá v tom, že každý z bitů zámku je zapisován s jinou kombinací linek P3.3 až P3.7. Stav linek portu P1 je při programování zámků libovolný.

Význam obou zamykacích bitů je uveden formou *tab. 6.2.*

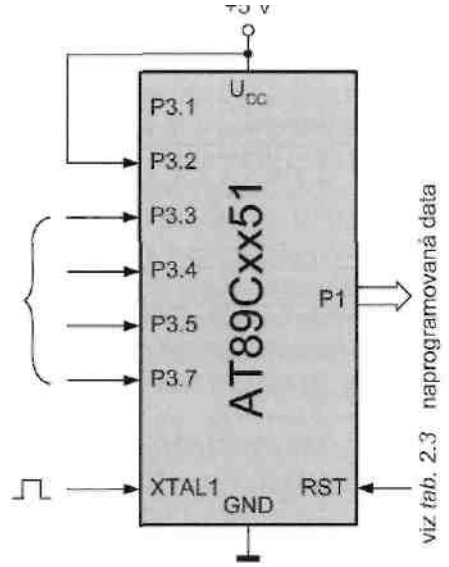
Tab. 6.2 Zamykací bity (N - bit nenaprogramován, P - bit naprogramován)

Zamyk	aci bity	Význam
LB1	LB2	
N	N	Obsah Flash není chráněn
P	N	Následné programování Flash není možné
P	P	Obsah Flash není možno programovat ani číst

Čtení obsahu Flash

Doporučený algoritmus čtení je tento:

1) připojení napájení: U_{cc} připojit na 5 V, RST = XTAL1 = 0 V,
2) nastavení kombinace vývodů P3.3 až P3.7 dle *tab. 6.1,*
3) data jsou k dispozici na portu P1,
4) pro čtení dalšího bajtu musíme vytvořit kladný impulz (kombinace náběžné a sestupné hrany) pro XTAL1,
5) opakování kroku 4 tak dlouho, dokud se nepřečte celá paměť.



Obr. 6.2 Čtení Flash

Čtení signatury

Signatura slouží k identifikaci obvodu, bajty signatury se čtou z adres OOOOh a 0001 h kombinací linek P3.3 až P3.7 dle *tab. 6.3.*

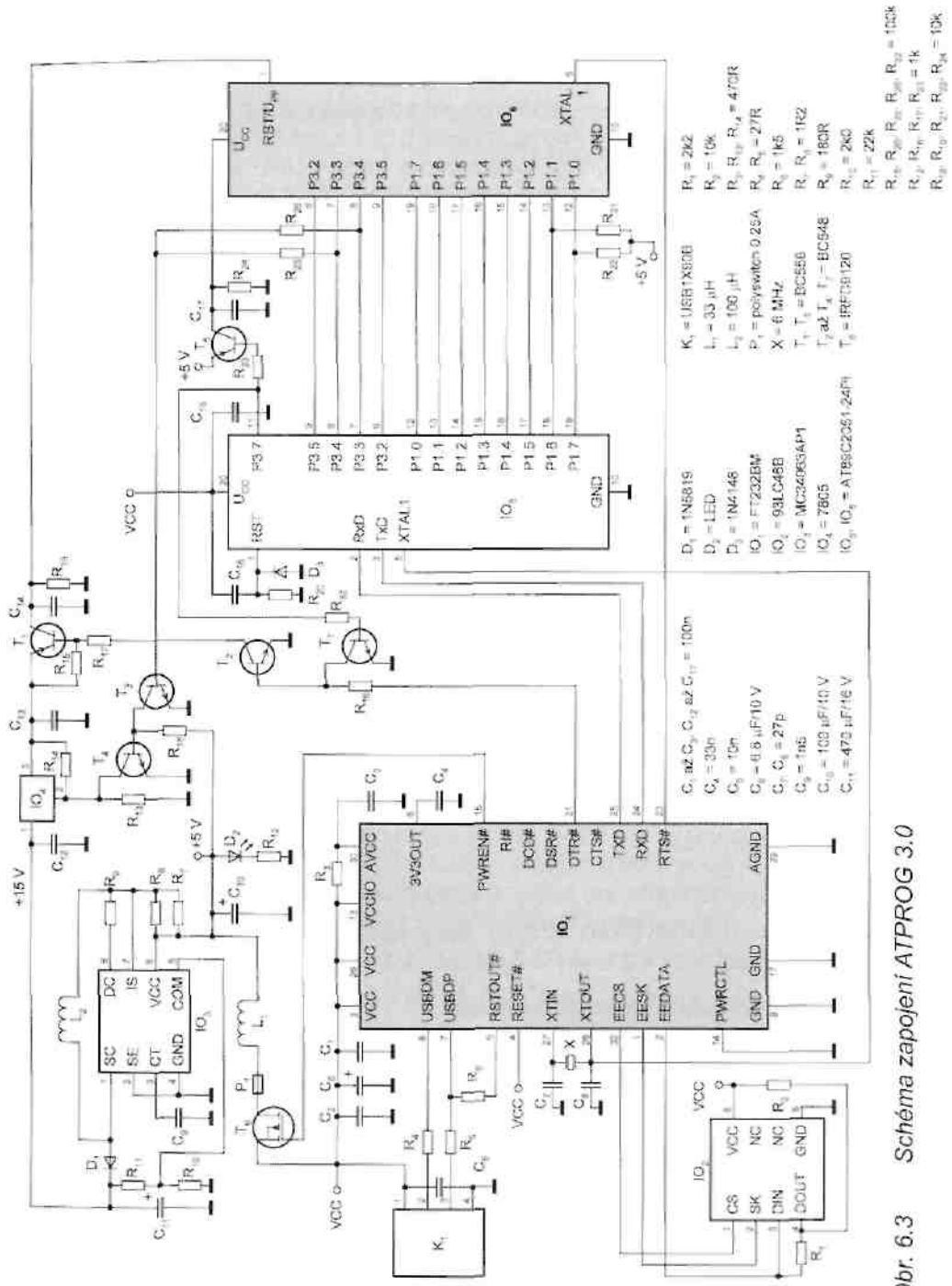
Tab. 6.3 Signatura a typ mikrokontroléru

0000h	0001 h	Typ mikrokontroléru
1Eh	12h	Atmel **AT89C1051**
1Eh	21h	Atmel **AT89C2051**
1Eh	41h	Atmel **AT89C4051**

6.3 VLASTNÍ REALIZACE

Schéma zapojení programátoru je uvedeno na *obr. 6.3.*

Základní část zapojení je stejná jako v kapitole 5. Jedná se o připojení obvodu **FT232BM** (IO_1) paměti **94LC46B** (IO_2) k USB konektoru.



Obr. 6.3 Schéma zapojení ATPROG 3.0

Výkonová část programátoru není připojena k napájení získanému z USB sběrnice přímo, ale připojuje je přes tranzistor PMOS T_6 **(IRFD9120).** Odběr je totiž vyšší než 100 mA. Typ tranzistoru byi vybrán především s ohledem na jeho relativně malé rozměry (není to sice typ určený pro spolupráci s číslicovými obvody, ale prahové napětí -2 až -4 V předpokládanému použití vyhovuje). Řídicí elektroda tohoto tranzistoru je ovládána vývodem **PWREN#.** Cívka *L,* zabraňuje průniku rušení z měniče zpět do počítače, polyswitch P, omezuje odebíraný proud na 250 mA.

Pro získání programovacího napětí (U_{PP}) 12 V byl použit měnič s obvodem **MC34063AP1** (IO_3). Výstupní napětí měniče je dáno poměrem odporů R_{10}, R„ a je zhruba 15 V. Odpory R_7, R_8 definují maximální proud odebíraný měničem (tedy i programátorem).

Protože programovaný mikrokontrolér (v patici IO_6) pracuje s hodnotami U_{PP} = 5 nebo 12 V, musel být připojen regulátor na základě stabilizátoru IO_4. Je-li na linkách P3.3 nebo P3.4 log. 1, je tranzistor T_3 (přes R_{25}, R_{26}) sepnut a tím je tranzistor T_4 vypnut. V tomto případě se výstupní napětí stabilizátoru 7805 (5 V) zvýší o napěťový úbytek na rezistoru R_{13} (teče jím klidový proud stabilizátoru plus proud odvozený z výstupního napětí stabilizátoru, dohromady asi 14 mA), což je asi 7 V. Výstupní napětí stabilizátoru je pak asi 12 V. Pokud jsou obě linky **P3.3** a **P3.4** v log. 0, je tranzistor T_3 vypnut a tedy tranzistor T_4 sepnut. Výstupní napětí je pak 5 V. Programovací napětí je však ještě ovládáno tranzistorem T,, který je spínán tranzistorem T_2. Báze tranzistoru T_2 je přes rezistor R_{16} připojena na linku **DTR#.** Je-li DTR# = 0, je programovací napětí odpojeno. Je-li DTR# = 1, je programovací napětí připojeno. Tranzistor T_7 navíc blokuje programovací napětí, pokud je odpojeno napájecí napětí (U_{cc}).

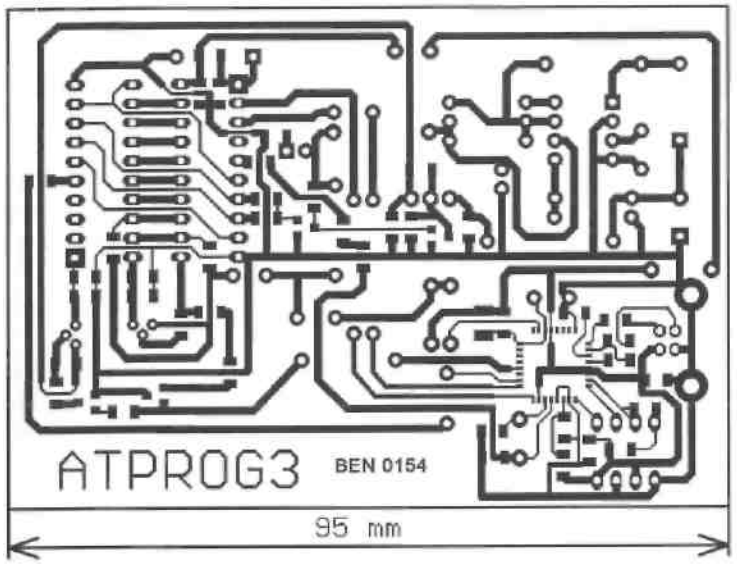
Programovací mikrokontrolér **AT89C2051** (IO_5) je připojen klasicky. Součástky R_{20}, C_{16}, D_3 tvoří jednoduchý resetovací obvod. Komunikace je zajištěna asynchronním sériovým kanálem, tedy linkami **RXD** (TxD) a **TXD** (RxD). Některé vývody IO_5 přímo ovládají vývody programovaného mikrokontroléru IO_6. Rezistory R_{21}, R_{22} představují pull-up rezistory linek P1.0, P1.1 mikrokontroléru IO_5. Jako zdroj hodinového kmitočtu je použit výstup **XTOUT** obvodu FT232BM (tedy 6 MHz), ušetří se tím krystal.

Tranzistor T_5 ovládá napájecí napětí programovaného mikrokontroléru. Napájecí napětí je připojeno při P3.7 = 0. Při P3.7 = 1 je napájecí napětí odpojeno a sepnutý tranzistor T_7 zároveň blokuje programovací napětí odvedením bázového proudu tranzistoru T_2.

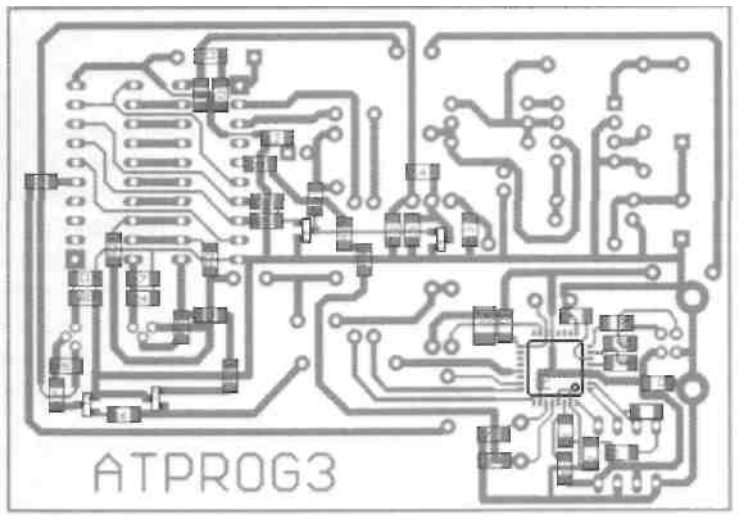
Vývod **XTAL1** programovaného mikrokontroléru je ovládán linkou **RTS#.**

Obr. 6.4 uvádí výkres desky plošných spojů. V zájmu snížení ceny se jedná o jednostrannou desku. Součástky jsou však osazeny z obou stran.

Situaci pro SMD uvádí *obr. 6.5. Obr. 6.6* ukazuje osazovací plánek ze strany součástek.



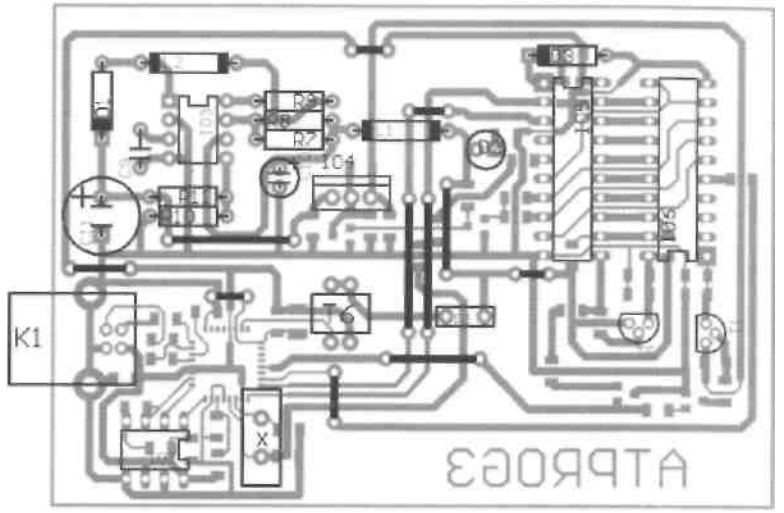
Obr. 6.4 Výkres desky plošných spojů ATPR0G3 (BEN 0154)



Obr. 6.5 Osazovacíplánek (strana spojů)

Seznam součástek pro ATPR0G3 (cena asi 350 Kč):

C, až C_3, C_{12} až C_{17}	CK+100N X7R	9 ks
C_4	CK+33N X7R	1 ks
C_5	CK+10NX7R	1 ks
C_6	CTS6M8/10VB	1 ks
C_7, C_8	CK+27P NPO	2 ks

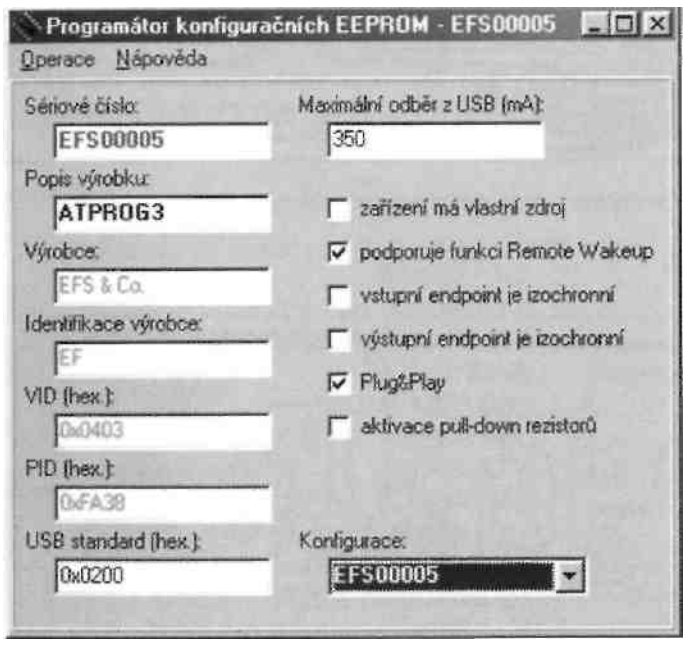


Obr. 6.6 Osazovací plánek (strana součástek)

C_9	CK 1N5 X7R	1 ks
C_{10}	E100M/10V	1 ks
C„	E470M/16V	1 ks
D_1	1N5819	1 ks
D_2	LED5MM 1MCD	1 ks
D_3	1N4148	1 ks
IO,	FT232BM	1 ks
10_2	93LC46B	1 ks
10_3	MC34063AP1	1 ks
10_4	7805	1 ks
10_5	89C2051-24PI (ATPROG3.BIN)	1 ks
10_6	programovaný mikrokontrolér	1 ks
K,	USB1X90B PCB	1 ks
L,	TL.33uH	1 ks
L_2	TL.IOOuH	1 ks
P,	RXE025	1 ks
R_1	RR+2K2 SMD	1 ks
R_2	RR+10KSMD	1 ks
R_3. R13, R_{14}	RR+470R SMD	3 ks
R_4, R_5	RR+27R SMD	2 ks
R_6	RR+1K5SMD	1 ks
R_7, R_8	RR 1R2	2 ks
R_9	RR180R	1 ks
R_{10}	RR 2K	1 ks
Rn	RR 22K	1 ks
Ris. R20. R_{25}. R_{26}. R32	RR+100K SMD	5 ks
R12. R16. R17, R23	RR+1K SMD	4 ks

Ris. R19. ^{R}2i. ^{R}22. R_{24}	RR+10K SMD	5 ks
R_{27} až R_{31}	RR+OR SMD	5 ks
T_v T_5	BC558	2 ks
T_2'až T_4, T_7	BC848	4 ks
T_6	IRFD9120	1 ks
X	QM 6.000MHZ	1 ks

Obsah paměti **93LC46B** je uveden formou *obr. 6.7.* Zařízení je pojmenováno jako **ATPROG3, VID = 0x0403, PID = 0xFA38,** maximální odběr je **350** mA (programátor má odběr 250 mA plus rezerva 100 mA).



Obr. 6.7 Konfigurace E^2PROM provedená programem EFSProg

6.3.1 Program řídicího mikrokontroléru

Program řídicího mikrokontroléru **AT89C2051** je poměrně jednoduchý. Po obvyklé konfiguraci sériového kanálu (je použita nestandardní přenosová rychlost **6250 Bd,** která se snadno odvodí z krystalu 6 MHz) a uvedení linek portů P1 a P3 do neaktivního stavu, přechází program do čekací smyčky.

Příjem dat sériovou linkou totiž není realizován přes přerušení (z principu by to bylo však možné), ale tak, že se testuje bit RI. Pokud je RI nastaven, znamená to, že byl přijat znak sériovou linkou.

Obsluha příjmu znaku je jednoduchá. Nejdříve se vynulují bity RI a TI (tím se provede příprava na další příjem) a poté se program větví podle toho, zda byl přijat v pořadí lichý nebo sudý bajt. Pokud je uživatelský příznak F0 vynulován, chápe se přijatá hodnota jako stav portu P3. Pokud je F0 nastaven, chápe se přijatá

hodnota jako stav portu P1. Příznak F0 se totiž neguje po příjmu každého bajtu, takže se mění mezi 0 a 1 podle toho, který bajt bude následně přijat.

Při příjmu hodnoty pro port P3 se údaj pouze přenese na linky tohoto portu. Uvědomme si však, že mezi linky portu P3 patří i vývody RxD a TxD, které programátor používá pro komunikaci. Protože není operace časově náročná, jsou stavy linek kopírovány z akumulátoru po jednom bitu (tak se snadno zabrání nevhodné změně stavu linek RxD a TxD). Pro kontrolu komunikace je přijatá hodnota odeslána zpět do počítače.

U portu **P1** je možno přijatou hodnotu vystavit přímo na port P1. Vzhledem k tomu, že linky portu P1 jsou mezi programovacím a programovaným mikrokontrolérem přetočeny, musí se provést programová reverze bitů. Aby byl program co nejjednodušší a přitom univerzální, provede se po krátké prodlevě zpětné čtení stavu portu P1 a odeslání této hodnoty sériovou linkou do počítače. Pokud na port P1 vyšleme hodnotu FFh, lze takto číst stav linek (log. 1 na vývodu programovacího mikrokontroléru je poměrně „slabá" - proud asi 100 uA- takže ji log. 0 vyvolaná čtením z programovaného mikrokontroléru bez problémů „přetáhne").

ATPROG3.ASM:

```
                $MODxx51
BAUD    EQU   251                ;6250 Bd (SMOD=l)

START:  CLR   F0                 ;nuluj počitadlo
        MOV   TH1,#BAUD          ;nastav
                                  ;přenos, rychlost
        MOV   TMOD,#00100000B    ;č/č 1, 8b reload
        MOV   SCON,#01010000B    ; 8b async. přenos
        MOV   PCON,#10000000B    ;SMOD=l
        SETB  TRI                ;zapni č/č 1
        MOV   P3,#80H            ;Ucc=0V,Upp=0V
        SETB  P3.0               /RxD a TxD
        SETB  P3.1               ;zatim v log. 1
        MOV   P1,#00H            ;P1 v log.O

                ;přijem:
TEST:   JNB   RI,TEST            ;čekej dokud RI=1
        CLR   RI                 ;vynuluj RI
        CLR   TI                 ;vynuluj TI
                ;přijat znak:
        MOV   A,SBUF             ;načti znak do A
        JB    F0,DATAP1          /větveni

                ;přijat 1. bajt:
DATAP3: MOV   C,ACC2             /předej hodnoty
        MOV   P3.2,C             /na P3.2
```

```
                MOV  C,ACC3
                MOV  P3.3,C              ;P3.3
                MOV  CACC.4
                MOV  P3.4,C              ;P3.4
                MOV  C,ACC5
                MOV  P3.5,C              ;P3.5
                MOV  CACC.7
                MOV  P3.7,C              ;P3.7
                MOV SBUF,ACC             ;pošli zpět
                AJMP FINÁL               ;a konec

                ;přijat 2. bajt:
DATAP1:         MOV  CACC.O              ; reverze bitů
                MOV  P1.7,C              ;před odesláním
                MOV  CACC.l              ;na port PÍ
                MOV  P1.6,C
                MOV  C/ACC.2
                MOV  P1.5,C
                MOV  CACC.3
                MOV  P1.4,C
                MOV  CACC.4
                MOV  P1.3,C
                MOV  C,ACC5
                MOV  P1.2,C
                MOV  C,ACC6
                MOV  P1.1,C
                MOV  C,ACC7
                MOV  P1.0,C
                MOV  A,#5                ;počkej
                DJNZ ACC,$               ;cca 20 ps
                MOV  CP1.0               ; reverze stavu PÍ
                MOV  ACC.7,C             ;před odesláním
                MOV  C,P1.1              ; sériovou linkou
                MOV  ACC.6,C
                MOV  C,P1.2
                MOV  ACC.5,C
                MOV  CPI. 3
                MOV  ACC.4,C
                MOV  C,P1.4
                MOV  ACC.3,C
                MOV  CPI. 5
                MOV  ACC.2,C
```



```
                MOV  C,P1.6
                MOV  ACC.1,C
                MOV  C,P1.7
                MOV  ACC.0,C
                MOV SBUF,ACC             ;pošle stav PÍ

FINÁL:          CPL F0                   ;zvyš počitadlo
                AJMP TEST                ;a znovu
                END
```

6.3.2 Popis tvorby ovládacího programu

Z hlediska použití obvodu **FT232BM** je pochopitelně nejdůležitější popis programové komunikace mezi počítačem a programátorem. Proto je vypsán pouze obsah modulu **HWInterface,** který tyto operace zajišťuje.

HWInterface.h:

```
#ifndef  HWInterfaceH
#define  HWInterfaceH
// ---------------------------------------------------------------------------
#include  <vcl\vcl.h>
#include  "Ftd2xx.h"
// ---------------------------------------------------------------------------
typedef unsigned char Bajt;
typedef unsigned short Slovo;
enum TAtmelTypí                  //definuje signatury
    tal051=0xl21e,               //jednotlivých mikrokontrolérů
    ta2051=0x211e,               //jako symbolické konstanty
    ta4051=0x411e,               //výčtového typu TAtmelTyp
    taChyba=0};
// ---------------------------------------------------------------------------
//typy události pro komunikaci s hlavnim programem:
typedef void _____ fastcall (______closure *TNextEvent)
    (int Step);
typedef void ______ fastcall (______closure *TSizeEvent)
    (int Size);
typedef void _____ fastcall (______closure *TReadEvent)
    (Bajt Data, bool Eof);
// ---------------------------------------------------------------------------
#pragma pack(push)
#pragma pack(l)
```

```
class TATProg:public TObject{ private:
        TNextEvent FOnNext;
        TSizeEvent FOnSize;
        TReadEvent FOnRead;
        //pošle hodnoty na porty P3 a PÍ
        Byte __fastcall Send(Bajt P3, Bajt PÍ);
        //čekací rutina s mikrosekundovým rozlišením:
        void __fastcall SDKSleep(DWORD Delay);
    public:
         _fastcall TATProg();
         _fastcall -TATProg();
        //zjisti signaturu:
        TAtmelTyp____fastcall CtiSignaturu();
        //smaže Flash:
        void ___fastcall SmazObvod();
        //zapiše zvolený bit zámku:
        void ___fastcall ZapisZamek(bool F);
        //zápis do Flash:
        void __fastcall ZapišZeStreamu(
            TMemoryStream *ms, bool Verifikace);
        //čteni z Flash:
        void ___fastcall PrectiDoStreamu(
            TMemoryStream *ms, int MaxDelka=0);
        //události:
         _ property TNextEvent OnNext=
        {read=FOnNext,write=FOnNext};
         _ property TSizeEvent OnSize=
        {read=FOnSize,write=FOnSize};
         _ property TReadEvent OnRead=
        {read=FOnRead,write=FOnRead} ;
        //Handle pro FT232BM zařízeni:
        FT_HANDLE ftHandle;
};
#pragma pack(pop)
// ---------------------- _ ------------------------------------------------------------------------
#endif
```

HWInterface.cpp:

```
#include <vcl\vcl.h> #pragma
hdrstop
```



```
#include "HWInterface.h"
#include "ATPMain.h"
//---------------------------------------------------------------------------------------------
 _fastcall TATProg::TATProg()
    :T0bject()
{
    FT_STATUS ftStatus;
    //otevřeni programátoru:
    ftStatus=FT_OpenEx(
    "ATPR0G3",FT_OPEN_BY_DESCRIPTION,&ftHandle);
    if( ftStatus!=FT_0K)
    throw Exception("Programátor ATPR0G3 neni připojen");

    //nastaveni přenosové rychlosti:
    FT_SetDivisor(ftHandle,OxOleO) ;          //6250 Bd
    FT_SetTimeouts(ftHandle,100,100);

    //inicializace patice:
    FT_SetDtr(ftHandle);                      //Upp=0V
    FT_SetRts(ftHandle) ;                     //XTAL1=0
    Send(0x80,0) ;                            //Ucc=0, linky v log.0
}
//---------------------------------------------------------------------------------------------
 _fastcall TATProg::-TATProg()
{
    //inicializace patice:
    FT_SetDtr(ftHandle);
    if(FT_OK==FT_SetRts(ftHandle)){
        Send(0x80,0);
        FT_Close(ftHandle);                   //odevzdáni zařízeni
    } else{
     Application->MessageBox( "Programátor byl
     odpojen?", Application->Title.c_str(),
     MB_ICONHAND); }
}
// ---------------------------------------------------------------------------------------------
//čeká stanovený počet mikrosekund:
void _fastcall TATProg::SDKSleep(DWORD Delay)
{
```

```
    LARGE_INTEGER f,casl,cas2,delta;
    QueryPerformanceFrequency(&f);
    QueryPerformanceCounter(&casl);
    //mikrosekundy:
    deltll.QuadPart=(Delay*f.QuadPart)/1000000.0;

    while(QueryPerformanceCounter(&cas2),
        cas2.QuadPart-casl.QuadPart<=deltll.QuadPart);
}
// ------------------------------------------------------------------------------
//pošle hodnoty na porty P3 a PÍ, vraci stav PÍ:
Byte ___fastcall TATProg::Send(Bajt P3,Bajt PÍ)
{
    Bajt Read;
    DWORD p;
    DWORD Ok;

    FT_Write(ftHandle,&P3,l,&p);
    FT_Read(ftHandle,SRead,1,&p);
    FT_Write(ftHandle,&P1,1,&p);
    FT_Read(ftHandle,&Read,1,&p); if (p=0)
        throw Exception("Chyba komunikace");
    return Read;
}
// ------------------------------------------------------------------------------
//Zapiše zamykaci bity:
//F=0 pro bit 1,
//F=l pro bit 0
void ___fastcall TATProg: : ZapisZamek (bool F)
{
    Bajt P3=0x38+F*4;
    Send(P3,0xFF);                  //Upp=12V,P3.2=1
    Send(P3&0xDF,0xFF);             //Upp=12V,P3.2=0
    SDKSleep(lO);                   //počkej
    Send(P3,0xFF);                  //Upp=12V,P3.2=1
}
//------------------------------------------------------------------------------
//Smaže Flash:
void ___fastcall TATProg::SmazObvod()
{
```



```
    FT_SetDtr(ftHandle);            //Upp=0V
    FT_SetRts(ftHandle);            //XTAL1=0
    Send(0x20,0xFF);                //Ucc=5V,P3.2=l
    FT_ClrDtr(ftHandle);            //Upp=5V
    Send(0x30,0xFF);                //Upp=12V,P3.2=l,P3.3=1
    Send(0xl0,OxFF);                //Upp=12V,P3.2=0,P3.3=1
    Sleep(IO);                      //počkej
    Send(0x30,0xFF);                //Upp=12V,P3.2=1,P3.3=1
    FT_SetDtr(ftHandle);            //Upp=0V
    Send(0x80,0);                   //Ucc=0V
}
// ------------------------------------------------------------------------------
//Vrátí signaturu obvodu:
TAtmelTyp _fastcall TATProg::CtiSignaturu()
¡
    Slovo Výsledek; Bajt
    bl,b2;

    FT_SetDtr(ftHandle);            //Upp=0V
    FT_SetRts(ftHandle);            //XTAL1=0
    Send(0x20,OxFF);                //Ucc=5V,P3.2=l
    FT_ClrDtr(ftHandle);            //Upp=5V
    bl=Send(0x20,OxFF);             //bl je dolní bajt signatury
    FT_ClrRts(ftHandle);            //XTAL1=1
    SDKSleep(lO);                   //počkej
    FT_SetRts(ftHandle);            //XTAL1=0
    b2=Send(0x20,OxFF);             //b2 je horní bajt signatury
    FT_SetDtr(ftHandle);            //Upp=0V
    FT_SetRts(ftHandle);            //XTAL1=0
    Send(0x80,0);                   //Ucc=0V

    Vysledek=b2*256+bl;             //sestav signaturu

    //zjištění typu procesoru
    switch(Výsledek){
        čase tal051:
        čase ta2051:
        čase ta4051:
            return((TAtmelTyp)Výsledek);
    }
    return(taChyba);
}
//------------------------------------------------------------------------------
```

```
//Zapisuje do Flash, včetně verifikace a zámků
void __fastcall TATProg::ZapisZeStreamu(
    TMemoryStream *ms, bool Verifikace)
{
    Bajt write,read; int
    pocitadlo=0; int i;

    if(FOnSize)
        FOnSize(ms->Size);

    FT_SetDtr(ftHandle);                    //Upp=0V
    FT_SetRts(ftHandle);                    //XTAL1=0

    ms->Position=0;
    while(ms->Position<ms->Size){
    ms->ReadBuffer(swrite, 1) ;
    Send(0x20,write) ;                      //Ucc=5V,P3.2=l
        FT_ClrDtr(ftHandle) ;               //Upp=5V
        Send(0x2C,write);                   //Upp=12V,P3.2=1,P3.4 = 1,P3.5=1
        Send(OxOC,write);                   //Upp=12V,P3.2=0,P3.4=1,P3.5=1
        SDKSleep(DobaZapisu);       //počkej
        Send(0x2C,write);                   //Upp=12V,P3.2=1,P3.4=1,P3.5=1
        //verifikace, pokud je požadována: if(Verifikace){
            read=Send(0x24,OxFF);
            if(write!=read){
            //při chybě inicializuj patici:
            FT_SetDtr(ftHandle);//Upp=0V
            Send(0x80,0);                   //Ucc=0V
              if(FOnNext)
       FOnNext(ms->Size); throw Exception ("Chyba
zápisu"); } }
FT_ClrRts(ftHandle);                        //XTAL1=1
SDKSleep(lO);                               //počkej
FT_SetRts(ftHandle);                        //XTAL1=0
pocitadlo++;
if(FOnNext)
FOnNext(počitadlo); if(Application->Terminated||HlForm->Prerusit)
break;
```



```
    }

    //zápis zamykacich bitů:
    switch(HlForm->Zamek){ čase 0:
            break; čase
        1:
        ZapisZamek(1); break;
        čase 2 :
     ZapisZamek(1);
     ZapisZamek (0); break; }

    if(FOnNext)
        FOnNext(ms->Position);

    //inicializace patice:
    FT_SetDtr(ftHandle);                    //Upp=0V
    Send(0x80,0);                           //Ucc=0V
}
// ---------------------------------------------------------------------------
//Čte obsah Flash:
void __fastcall TATProg::PřectiDoStreamu(
    TMemoryStream *ms, int MaxDelka)
{
    Bajt Read;
    int pocitadlo=0;
    int Délka;

    switch (CtiSignaturu ()) {
        čase tal051:Delka=1024;break;
        čase ta2051:Delka=2048;break;
       default:Delka=40 96;break; }
    if(MaxDelka&&Delka>MaxDelka)
        Delka=MaxDelka;
    if(FOnSize)
        FOnSize(Délka);

    FT_SetDtr(ftHandle);                    //Upp=0V
    FT_SetRts(ftHandle);                    //XTAL1=0
```

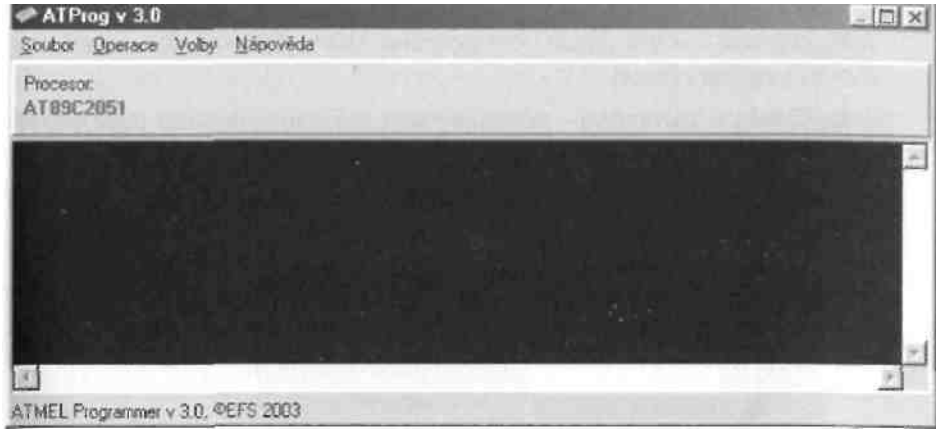
```
while(1){
    Send(0x20,OxFF)  ;                  //Ucc=5V,P3.2=1
    FT_ClrDtr(ftHandle);                //Upp=5V
    Read=Send(0x24 , OxFF) ;      //Read je přečtená hodnota
    FT_ClrRts(ftHandle);                //XTAL1=1
    SDKSleep(lO);                       //počkej
    FT_SetRts(ftHandle);                //XTAL1=0
    pocitadlo++;
    if(FOnNext)
        FOnNext(počitadlo);
    if(FOnRead)
        FOnRead(Read,falše); ms-
    >WriteBuffer(SRead,1);
    if(pocitadlo==Delka)
break; if(Application->Terminated||HlForm->Prerusit) break; };
if(FOnRead)
        FOnRead(Read,true); if
(FOnNext)
    FOnNext(Délka);

ms->Size=pocitadlo;
FT_SetDtr(ftHandle);                //Upp=0V
    Send(0x80,0);                   //Ucc=0V
}
```

Vysvětlení podané komentáři se mi zdá jako postačující. Větší pozornost věnuji použití funkce FT_SetDivisor. Jedná se o možnost, jak nastavit nestandardní přenosovou rychlost na 6250 Bd. Další informace viz kapitolu 8.1.

Kmitočet 3 **MHz** je dělen číslem, které odpovídá spodním čtrnácti bitům dělitele, horní dva bity určují jemné ladění. V použitém děliteli 0x01e0, jsou horní dva bity nulové (jemné ladění není použito) a spodních čtrnáct bitů má hodnotu **480.** Přenosová rychlost je tedy skutečně 6250 Bd.

6.4 STRUČNÁ UŽIVATELSKÁ PŘÍRUČKA

Při spuštění ovládacího programu se pokusí navázat komunikace se zařízením ATPROG3. Pokud je komunikace úspěšná, je zjištěna signatura mikrokontroléru vloženého do programovací patice. Typ mikrokontroléru je zobrazen v jednom z titulků (viz *obr. 6.8)*.



Obr. 6.8 Aplikace po spuštění

Menu Soubor:

- Soubor|Otevřít - otevře binární soubor pro programování Flash (soubor s příponou BIN lze přetáhnou technikou drag&drop ze souborového manažeru přímo do okna programátoru),
- SouborJUložit - uloží obsah okna pod zvoleným názvem do souboru,
- Soubor|Konec - konec aplikace.

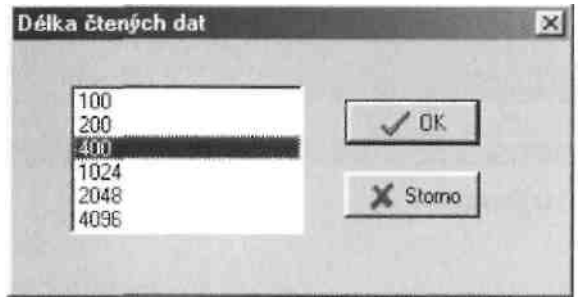
Menu Operace:

- Operace|Smaž procesor - smaže Flash,
- Operace|Zapiš data do procesoru - zapíše do Flash obsah okna (viz *obr. 6.*9),
- Operace|Čti data z procesoru - čte obsah Flash a zobrazuje jej v okně,
- Operace|Zjištění typu procesoru - zjistí typ mikrokontroléru čtením jeho signatury.

Obr. 6.9 Průběh čtení dat z Flash

Menu Volby:

- Volby|Zámek - volba zámku pro ochranu Flash (žádný; zábrana zápisu; zábrana zápisu i čtení),
- Volby|Zápis s kontrolou - povoluje nebo zakazuje verifikaci (test úspěšného zápisu),
- Volby|Délka čtených dat - zobrazí dialog pro volbu počtu bajtů čtených operací Operace|Čti data z procesoru.



Obr. 6.10 Dialog pro volbu délky čtených dat

Menu Nápověda:

- Nápověda|0 aplikaci -zobrazí klasický dialog o aplikaci, včetně odkazu na WWW stránky autora.

Nejpodstatnější novinky ATPROG 3.0

- Operace čtení a zápisu lze přerušit - v průběhu čtení nebo zápisu je zobrazeno tlačítko Přerušit operaci, které umožňuje přerušení zápisu či čtení Flash (viz *obr. 6.9),*
- Podpora zamykacích bitů - lze používat zamykací bity (viz výše),
- Podpora přetahování souborů mezi souborovým manažerem a programátorem - soubory s příponou BIN lze nyní přetahovat ze souborového manažeru přímo do programátoru.

Inicializační soubor ATPROG.INI

Inicializační soubor ATPROG.INI musí být umístěn ve stejném adresáři jako ovládací program. Tento soubor obsahuje volby vztahující se jednak k rozměrům a umístění okna aplikace při posledním běhu a dále volby umístěné v sekci VOLBY:

- Cesta - cesta k posledně otevřenému souboru,
- Zámek - zámek pro Flash (0 až 2),
- DelkaCteni - posledně zvolený počet bajtů pro čtení Flash,
- DobaZapisu - čekací doba po zápisu v mikrosekundách (nominálně 2000).

Příklad obsahu ATPROG.INI:

[OKNO]

X=7 9
Y=84
W=596
H=2 7 7

[VOLBY]

Cesta=C:\Matousek\FTDI\PROGRAMY\KAP6\ATMEL
Zamek=0
DelkaCteni=100
DobaZapisu=1200

6.5 AUTORSKÁ PRÁVA

Na doprovodném CD-ROM naleznete pouze cílový soubor programátoru ATPROG3.EXE a soubory pro programování programovacího mikrokontroléru ATPROG3.ASM a ATPROG3.BIN.

Myslím, že vytvořený programátor je na poměrně vysoké úrovni a proto si zdrojový text chráním. Nebráním však nikomu v tom, aby program bezúplatně používal!

6.6 LADICÍ PROGRAM ATP3DBG.EXE

Podobně jako u předchozí verze ATPROG 2.1, poskytuji pro amatérskou výrobu programátoru ladicí program. Ten umožňuje provést zápis na porty P1 a P3 a ovládat linky modemu. Takže lze pohodlně diagnostikovat funkci programátoru.

Před spuštěním ladicího programu nesmí být programovaný mikrokontrolér vložen do patice (mohlo by dojít k jeho poškození).

Okamžitě po svém spuštění se ladicí program pokusí navázat komunikaci s programátorem. Pokud se to nezdaří, je to možná z jedné z těchto příčin:

- součástky jsou špatně připájeny,
- obsah E^2PROM je chybný,
- programovací mikrokontrolér není naprogramován.

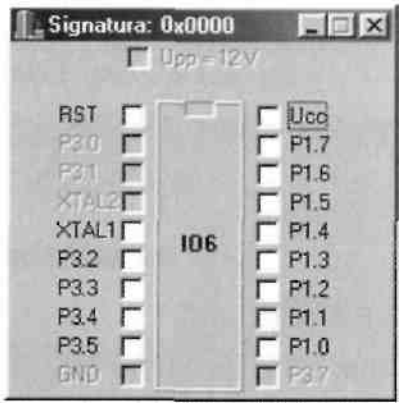
Poté je možno vyslat zvolenou kombinaci na porty P1 a P3 a voltmetrem provést kontrolní měření. Velmi důležité je také sledovat stav vývodů U_{cc}, RST (zejména hodnoty 5 a 12 V) a XTAL1 na programovací patici.

Linky P3.5 a P3.7 jsou spojeny, proto se stav P3.5 přenáší na P3.7. Napětí na vývodu RST je blokováno v okamžiku, kdy je U_{cc} = 0 V. Je-li U_{cc} = 5 V, lze pomocí check boxu RST zapínat a vypínat napětí vývodu RST. Hodnota napětí je buď 5 V (P3.3 i P3.4 jsou neaktivní) nebo 12 V (P3.3 nebo P3.4 je aktivní).

Pokud jsou výše popsané testy úspěšné, ukončete program a vložte programovaný mikrokontrolér do patice. Po novém spuštění programu ATP3DBG by se měl mikrokontrolér přihlásit se signaturou dle *tab. 6.3.* Pokud je signatura platná, jsou

check boxy odstaveny, takže nemůže dojit k poškození mikrokontroléru nevhodnou manipulací s nimi.

V této chvíli lze program ukončit a provést první test naprogramování pomocí programu **ATPROG3.EXE.**



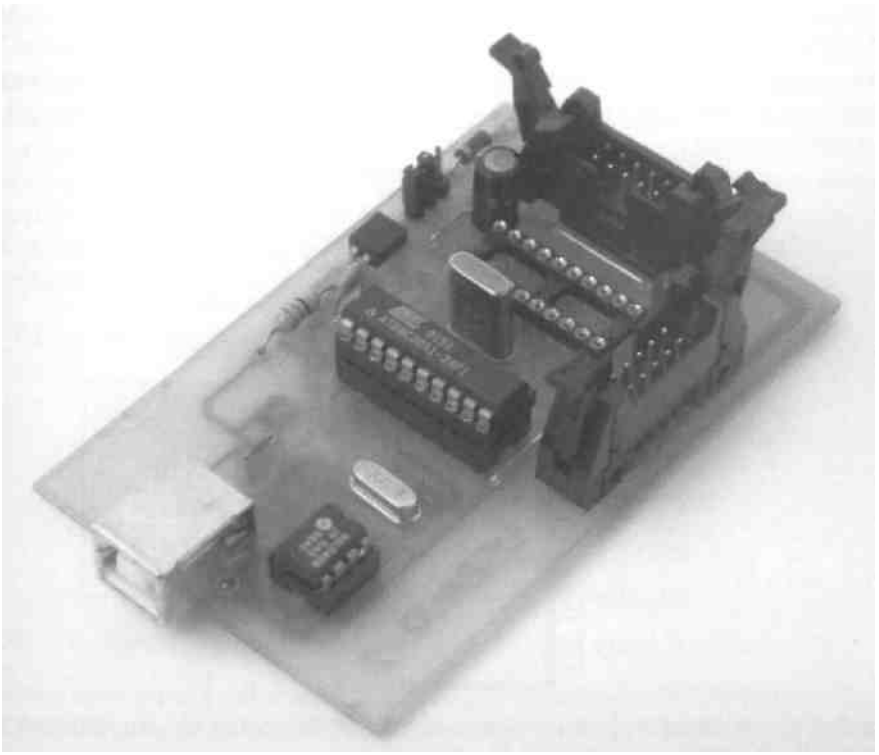
Obr. 6.11 ATP3DBGvakci

6.7 PŘÍKLADY POUŽITÍ

Příklady použití mikrokontroléru **AT89C2051** ve spojení s konvertorem **FT232BM** naleznete v kapitole 8.2.

7

USBAVR – VÝVOJOVÉ KITY PRO MIKROKONTROLÉR AT90S2313



V této kapitole jsou uvedeny dvě konstrukce vývojových kitů pro mikrokontrolér AT90S2313. Tento mikrokontrolér disponuje možností sériového downloadu. To dává možnost rychlého přechodu mezi návrhem programu a jeho vyzkoušením, protože pro připojování periferií se použije stejná deska.

7.1 POPIS MIKROKONTROLÉRU AT90S2313 A SÉRIOVÉHO DOWNLOADU

V této kapitole se zaměřím pouze na uvedení základních vlastností mikrokontroléru **AT90S2313** a stručný popis **sériového downloadu,** který platí pro všechny mikrokontroléry ATMEL AVR. Další informace lze získat zejména z [3].

7.1.1 Základní vlastnosti mikrokontroléru AT90S2313

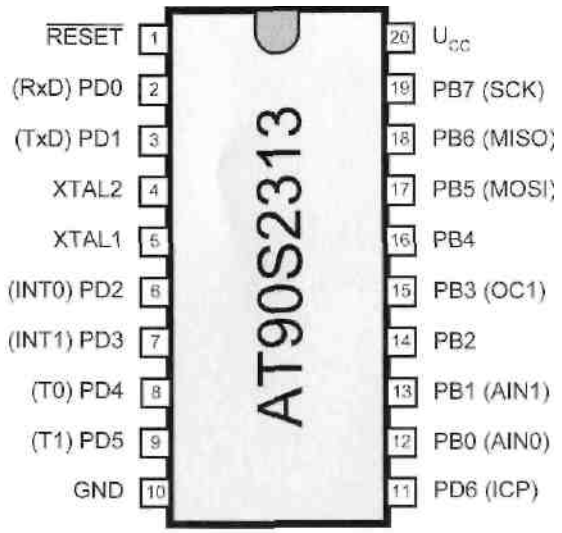
Mikrokontrolér **AT90S2313** je nízkopříkonový 8bitový mikrokontrolér založený **na** architektuře **AVR RISC.** Tím, že provádí výkonné instrukce v jediném hodinovém cyklu, dosahuje výpočetního výkonu **1 MIPS na 1 MHz.**

Instrukční soubor obsahuje 118 instrukcí. Mikrokontrolér má k dispozici 32 registrů délky 8 bitů.

Paměť programu je tvořena zabudovanou **Flash,** jejíž kapacita je 2 KB. Flash je programovatelná buď klasickým paralelním programátorem nebo přímo v systému pomocí rozhraní SPI (počet přeprogramování je 1000 cyklů).

Datová paměť je typu **RAM** (128 B) a **E^{2}PROM** (128 B; počet přeprogramování je 100 000 cyklů). E^{2}PROMje programovatelná přímo v systému pomocí rozhraní SPI.

Mikrokontrolér má zabudován poměrně jednoduchý **8bitový čítač/časovač,** k dispozici je i dokonalejší **16bitový čítač/časovač** (režimy Output Compare, In-put Capture, PWM).



Obr. 7.1 Rozložení vývodů mikrokontroléru AT90S2313 na pouzdru DIP 20

Dále je k dispozici obvod **WDT** (Watchdog Timer; hlídač správného běhu programu), analogový komparátor, zabudovaný asynchronní sériový kanál **(UART).**

7.1.2 Paměťové zámky

Mikrokontroléry AVR poskytují dva zámky pro programovou (Flash) a datovou (E^{2}PROM) paměť. Nenaprogramovaný bit zámku má hodnotu 1, naprogramovaný 0. Význam je uveden formou *tab. 7.1.*

Tab. 7.1 Paměťové zámky

Paměťové zámky			Typ ochrany
Režim	**LB1**	LB2	
1	1	1	není aktivována žádná ochrana
2	0	1	zákaz programování Flash a E^{2}PROM
3	0	0	zákaz programování Flash a E^{2}PROM, odstavena verifikace

Z tab. 7.1 je zřejmé, že při běžném provozu se používá režim 1 (ochrana není aktivována). Režimy 2 a 3 zakazují programování obou pamětí (je to jistá ochrana proti nevhodné modifikaci). Režim 3 odstavuje verifikaci (zpětné čtení obsahu), brání tedy neoprávněnému kopírování navrženého programu.

7.1.3 Signatura

Mikrokontroléry AVR obsahují 3 bajty signatury. Jedná se o bajty, které zapsal výrobce. Tyto bajty určují typ programovaného mikrokontroléru. Tímto způsobem tedy programátor snadno rozezná, jaký mikrokontrolér je programován.

První bajt má vždy hodnotu $1E (indikuje výrobce = ATMEL). Druhý bajt indikuje velikost Flash ($90 - 1 KB, $91 - 2 KB, $92 - 4 KB, $93 - 8 KB). Poslední bajt indikuje konkrétní typ mikrokontroléru. Viz *tab. 7.2.*

Signaturu nelze číst v režimu zámku 3 (viz kapitolu 7.1.2).

Tab. 7.2 Mikrokontroléry a jejich signatury

Typ mikrořadiče	**$0000**	$0001	$0002
AT90S2313	$1E	$91	$01
AT90S2343	$1E	$91	$03
AT90S4433	$1E	$92	$03
AT90S8515	$1E	$93	$01
AT90S8535	$1E	$93	$03

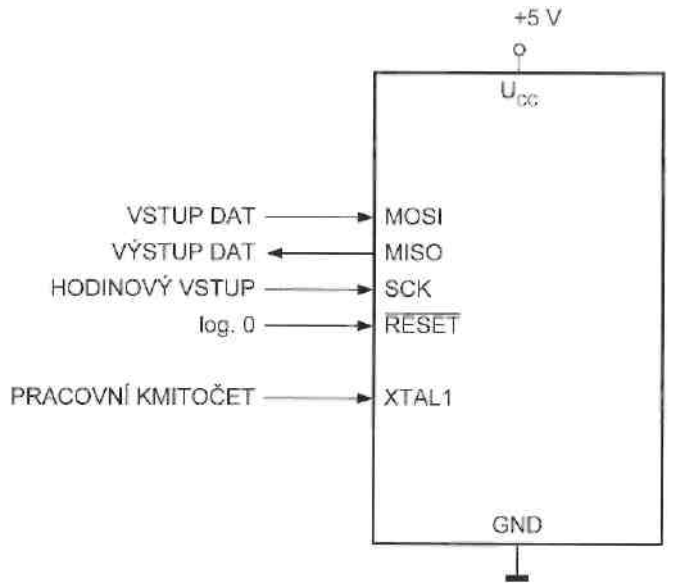
7.1.4 Sériový download

Sériový download je velmi pokroková programovací metoda. Umožňuje programovat mikrokontrolér přímo v aplikaci, čímž se celý vývoj výrazně urychlí (odpadá

složité vkládání/vyjímání programovaného obvodu do/z programátoru resp. aplikační desky).

Sériovým downloadem disponují všechny míkrokontroléry AVR (narozdíl od řady AT89, u které je tento způsob programování spíše výjimečný), což je jistě velmi potěšující skutečnost.

Jak programová, tak i datová paměť může být programována použitím SPI sběrnice v okamžiku, kdy je RESET = 0. Sériové rozhraní obsahuje vývody: SCK (hodiny), MOSI (vstup) a MÍSO (výstup). Viz *obr. 7.2.*



Obr. 7.2 Ideové zapojení pro sériový download

Po aplikaci RESET = 0 , musí být jako první vyslána instrukce Povolení programování (Programming Enable Instruction).

Instrukce Smazání čipu (Chip Erase) uvede obsah všech paměťových buněk programové i datové paměti do stavu $FF.

Datová E^2PROM má zabudovaný nulovací cyklus, takže ji není třeba před zápisem smazat pomocí instrukce Smazání čipu (Chip Erase).

Programová (Flash) i datová (E^2PROM) paměť mají při programování oddělené adresní prostory. Maximální adresa je dána velikostí paměti. Ta závisí na typu použitého mikrokontroléru.

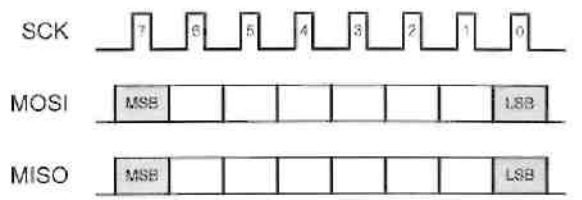
Při programování musí být mezi vývody XTAL1, XTAL2 připojen krystal (případně lze připojit zdroj vnějšího kmitočtu na vývod XTAL1). Kladný a záporný pulz SCK musí být delší než perioda nejvyššího přípustného pracovního kmitočtu.

Algoritmus sériového programování

Zapisovaná data jsou vzorkována náběžnou hranou SCK. Čtená data se objevují souběžně se sestupnou hranou SCK. Viz *obr. 7.3.*

Tab. 7.3 Instrukční soubor sériového downloadu

Instrukce	Formát				Stručný popis
	bajt 1	bajt 2	bajt 3	bajt 4	
Povolení programování	10101100	01010011	xxxxxxxx	xxxxxxxx	aktivuje programovací rozhraní
Smazání čipu	10101100	100xxxxx	xxxxxxxx	xxxxxxxx	smaže Flash a E²PROM
Čtení kodu	0010H000	aaaaaaaa	bbbbbbb	cccccccc	čtení instrukce z Flash
Zápis kodu	0100H000	aaaaaaaa	bbbbbbb	iiiiiii	zápis instrukce do Flash
Čtení dat	10100000	aaaaaaaa	bbbbbbb	cccccccc	čtení dat z E²PROM
Zápis dat	11000000	aaaaaaaa	bbbbbbb	iiiiiii	zápis dat do E²PROM
Zápis zámku	10101100	11111AB1	xxxxxxxx	xxxxxxxx	zápis zamykacích bitů
Čtení signatury	00110000	xxxxxxxx	xxxxxxbb	cccccccc	čtení signatury obvodu



Obr. 7.3 Průběhy signálů při sériovém downloadu

7.2 DVĚ KONSTRUKCE USBAVR

Návrh vývojového kitu probíhal ve dvou etapách:

Původně jsem chtěl použít režim Bit Bang. Takže programování probíhalo v paralelním režimu linek FT232BM. Testováním jsem však zjistil, že programování je poměrně pomalé. Naproti tomu se objevila jedna výhoda: Po přepnutí do normálního režimu se dal vývojový kit použít i pro komunikaci mikrokontroléru se sběrnicí USB bez nutnosti použití dalších součástek.

Druhá varianta sledovala zvýšení rychlosti programování. Což se mi povedlo. Nebylo však již možno využívat obvod FT232BM zabudovaný v kitu pro komunikaci s naprogramovaným mikrokontrolérem. Naproti tomu lze volit, zda je kit napájen z USB (tedy vlastně z počítače) nebo z vnějšího zdroje.

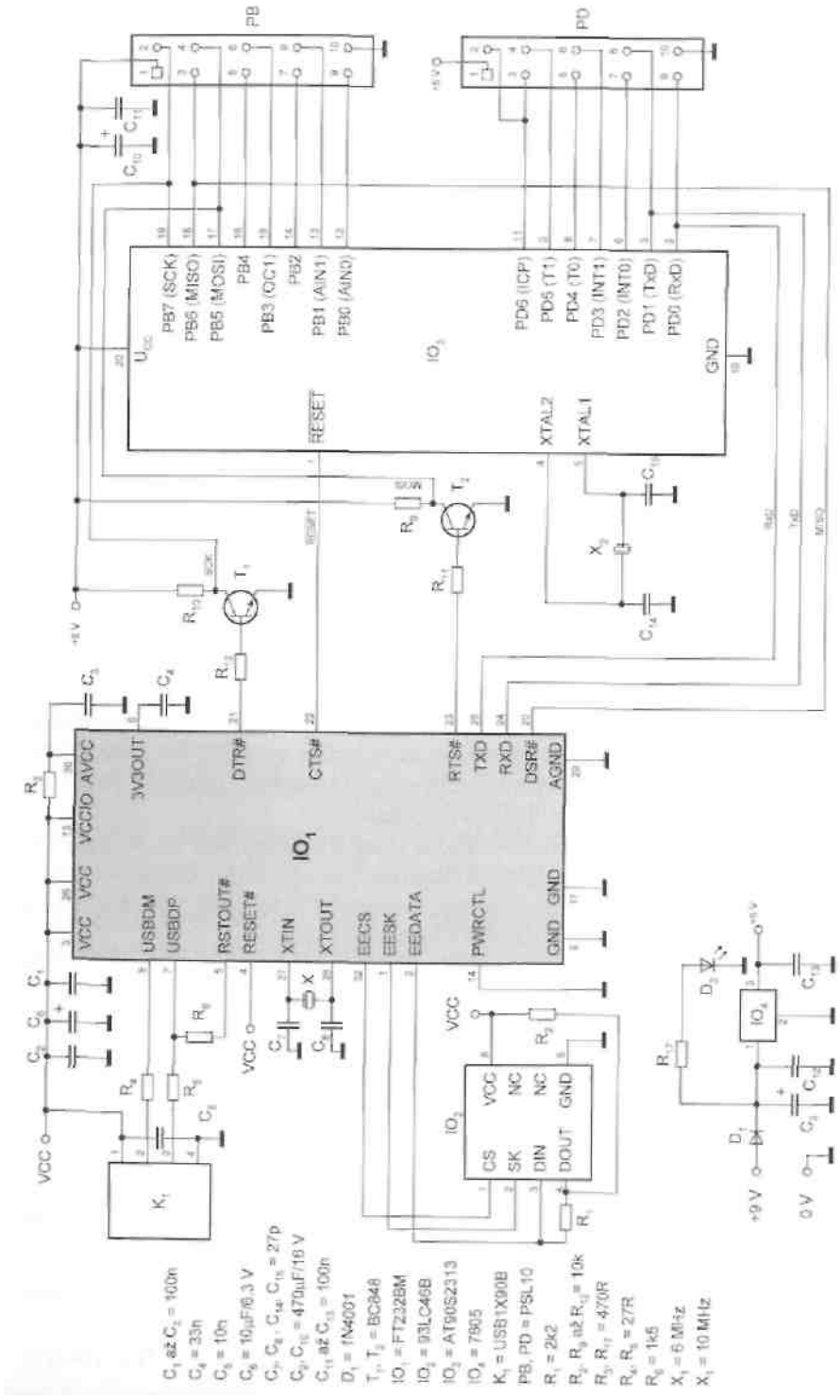
Zvláštností konstrukce je jednotný ovládací program. Po svém spuštění si program zjistí, který z programátorů je připojen. Pokud jsou připojeny programátory obou verzí, vybere se automaticky programátor verze 2.0.

7.3 USBAVR VERZE 1.0

Tato verze vývojového kitu má níže uvedené vlastnosti:

- napájení z vnějšího zdroje 9 až 12 V (odběr z USB pouze 100 mA),
- možnost používat kit i pro testy komunikace přes USB (použití linek **RXD, TXD** obvodu FT232BM a UART jednotky mikrokontroléru AT90S2313),
- relativně nízká rychlost programování (propustnost sběrnice v režimu Bit Bang je poměrně nízká).

Schéma zapojení USBAVR v 1.0 je uvedeno na *obr. 7.4.*



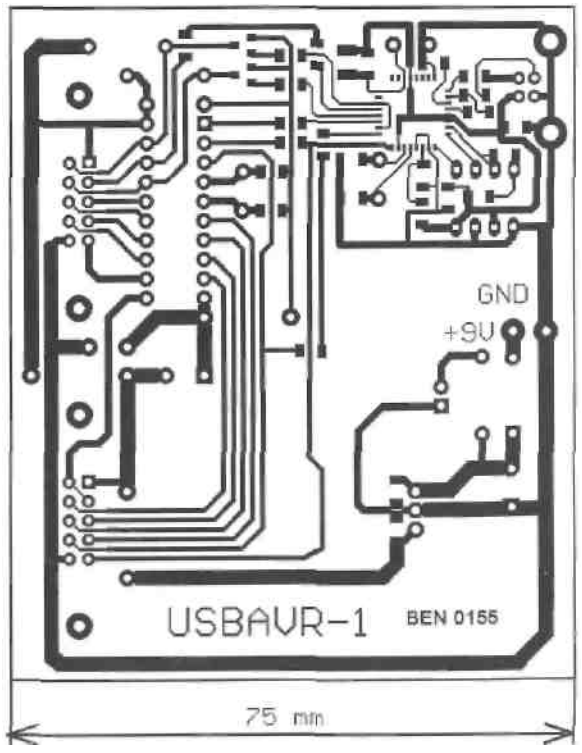
Obr. 7.4 *Schéma zapojení USBAVR v 1.0*

Programovací linky jsou řízeny přímo (při programování je používán režim Bit Bang). Jak je zřejmé, vstupní linky **SCK** a **MOSI** jsou připojeny přes tranzistory T_1 a T_2 (takže jsou negovány). Důvod tohoto řešení spočívá ve snaze zabránit poškození obvodu **FT232BM** (IO_1) při připojení nějakého přípravku k portu PB v době programování (V nejhorším případě se může poškodit některý z tranzistorů. Protože se obvykle používají výstupy opatřené směrem k log. 1 rezistorem pull-up, je toto riziko nepatrné). Při kolizi může dojít ke stažení programovací linky na log. 0, což vede pouze k chybě programování (musíme jej opakovat znovu při odpojeném přípravku).

Nulovací signál programovaného mikrokontroléru je ovládán linkou **CTS#.** Linky **TXD** a **RXD** obvodu IO_1 jsou připojeny na linky **RxD** a **TxD** programovaného mikrokontroléru IO_3. To dává možnost používat vývojový kit i pro testy komunikace pomocí sériového kanálu.

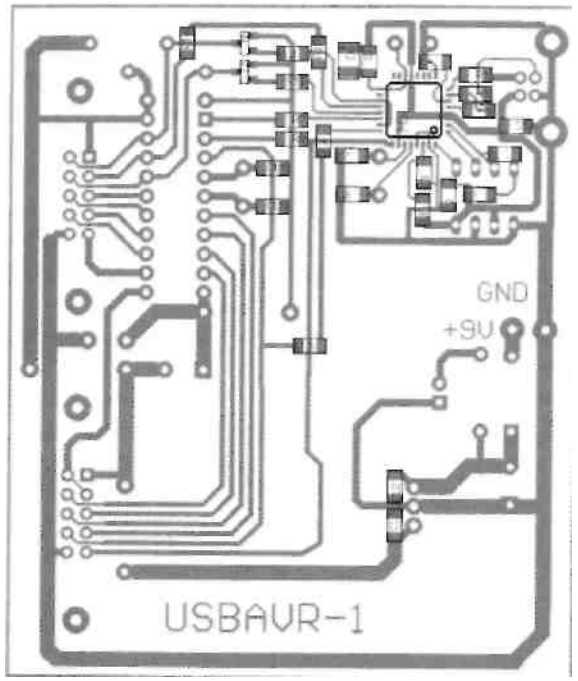
Mikrokontrolér používá vnější napájecí zdroj (není napájen ze sběrnice USB) čímž se snižuje riziko poškození portu nevhodnou manipulací (toto řešení je tedy vhodnější pro amatérskou výrobu). Na druhé straně musíme použít vnější zdroj (stabilizátor je však součástí vývojového kitu), takže konstrukce není tak elegantní jak by mohla být (jiné řešení uvádí kapitola 7.4). Mikrokontrolér dále používá vlastní krystal X_2, což dává možnost upravit si pracovní kmitočet podle vlastní potřeby (v rozsahu 1 až 10 MHz).

Výkres desky plošných spojuje uveden na *obr. 7.5.* Osazovací plánky jsou uvedeny na

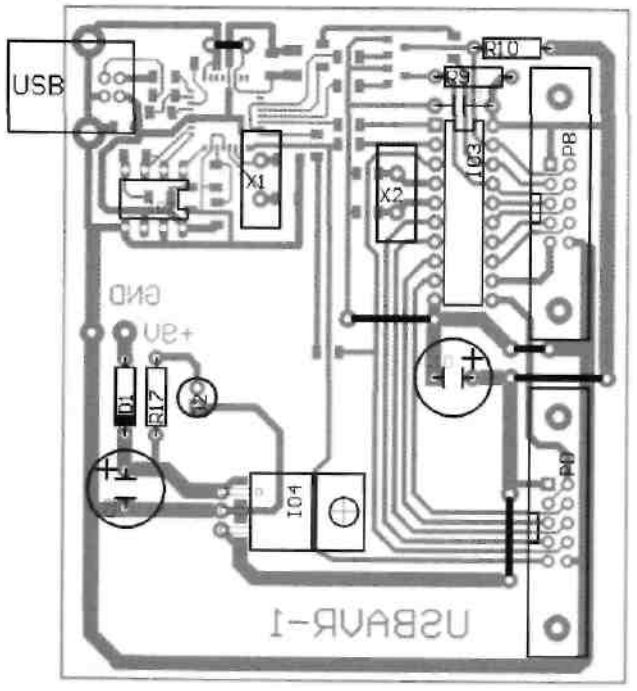


Obr. 7.5 Výkres desky plošných spojů USBAVR-1 (BEN 0155)

obr. 7.6 a *obr. 7.7.*



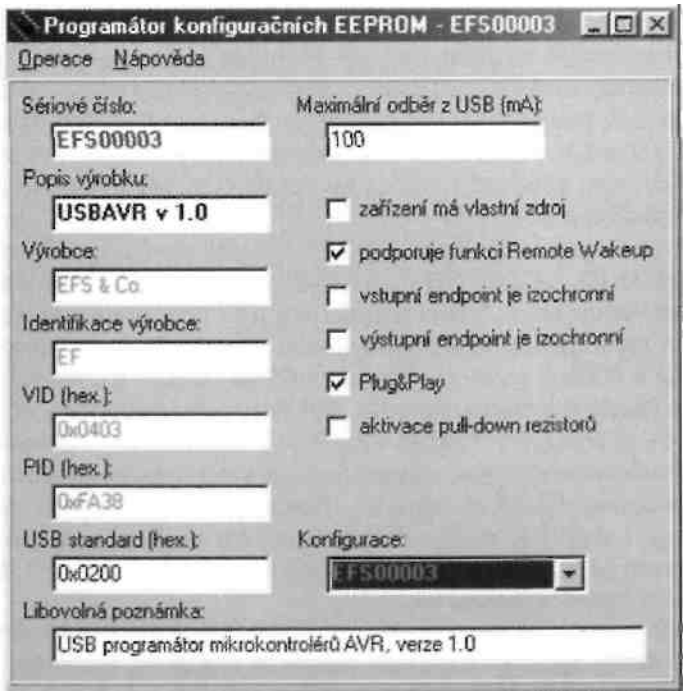
Obr. 7.6 Osazovací plánek (strana spojů)



Obr. 7.7 Osazovací plánek (strana součástek)

Seznam součástek pro USBAVR v 1.0 (cena asi 350 Kč):

d až C_3, C_{12}, C_{13}	CK+100NX7R	5 ks
C_4	CK+33N X7R	1 ks
C_5	CK+10NX7R	1 ks
C_6	CTS6M8/10VB	1 ks
C_7, C_8, C_{14}, C_{15}	CK+27P NPO	4 ks
C_9, C_{10}	E470M/16V	2 ks
Cn	CK100NX7R	1 ks
D,	1N4001	1 ks
D_2	LED 5MM 200MCD	1 ks
IO_1	FT232BM	1 ks
IO_2	93LC46B	1 ks
IO_3	90S2313-10PI	1 ks
IO_4	7805	1 ks
PB, PD	PSL10	2 ks
R_1	RR+2K2 SMD	1 ks
R_2	RR+10KSMD	1 ks
R_3	RR+470R SMD	1 ks
R_{4l} R_5	RR+27R SMD	2 ks
R_6	RR+1K5SMD	1 ks
R_7, R_8, R_{13} až R_{16}	RR+0R SMD	6 ks



Obr. 7.8 Konfigurace E^2PROM provedená programem EFSProg

R_9, R_{10}	RR10K	2 ks
R_{11}, R_{12}	RR+10KSMD	2 ks
R_{17}	RR 470R	1 ks
T_1, T_2	BC848	2 ks
USB	USB1X90B PCB	1 ks
X_1	QM 6.000MHZ	1 ks
X_2	QM 10.000MHZ	1 ks

Konfigurace zařízení je zřejmá z *obr. 7.8.*

7.4 USBAVR VERZE 2.0

Tato verze vývojového kitu má níže uvedené vlastnosti:

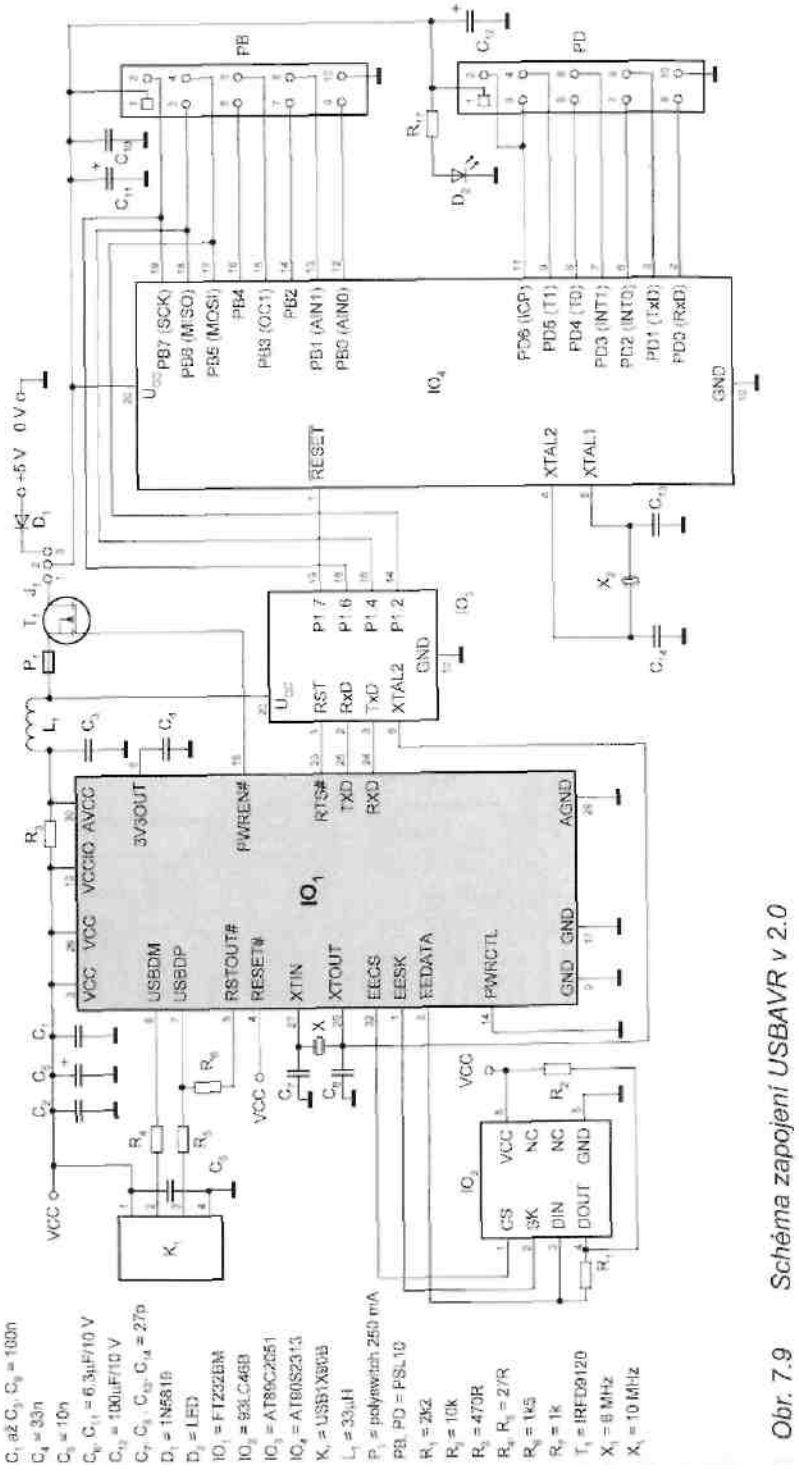
- napájení ze sběrnice USB (předvolený odběr 300 mA, lze pochopitelně zvýšit) nebo z vnějšího zdroje 5 V (volba se provede jumperem). Pokud je využita možnost napájení přímo z USB, nepotřebuje uživatel vnější zdroj,
- kit není možno používat pro testy komunikace pomocí USB (na port PD lze však připojit přípravek **FT232TST,** který je ovládán dalším portem USB),
- relativně vysoká rychlost programování.

Schéma zapojení **USBAVR v 2.0** je uvedeno na *obr. 7.9.*

Základní část zapojení je stejná jako v předchozích příkladech. Následuje tlumivka L_1 a polyswitch P_1 spolu s tranzistorem T_1,pomocí vývodu **PWREN#** je pak získáno napájecí napětí ze sběrnice USB. Pokud je jumper J, v pozici 1-2, je programovaný mikrokontrolér IO_4 připojen k takto získanému napájení. Pokud je jumper J., v pozici 2-3, musí se přivést napájení z vnějšího zdroje 5 V, který připojíme mezi svorky 0 V a 5 V (dioda D_1 brání možnému přepólování). V zájmu snížení rozměrů desky není součástí konstrukce stabilizátor, předpokládá se, že bude umístěn do vnějšího zdroje.

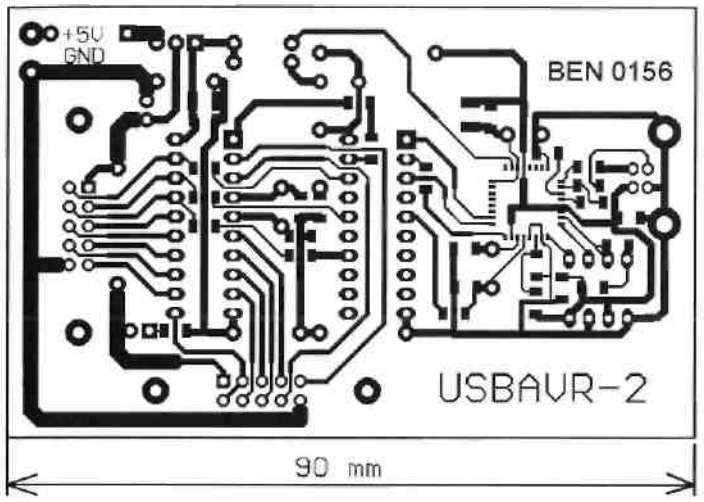
Vlastní programování mikrokontroléru IO_4 je zajištěno programem uloženým v mikrokontroléru IO_3 **(AT89C2051),** program je sice jednoduchý, ale umožňuje zvýšit rychlost komunikace. Mikrokontrolér IO_3 totiž používá asynchronní sériový přenos a sám zajišťuje serializaci přijatých dat na linky **SCK, MÍSO** a **MOSI.** Kromě linek **TXD** a **RXD** je ještě použita linka **RTS#,** kterou lze mikrokontrolér IO_3 resetovat. Po resetu mikrokontroléru IO_3 (RST# přejde nejdříve do log. 1 a pak se vrátí do log. 0), je linkou P1.7 držen v resetu i programovaný mikrokontrolér IO_4. Potom se již vyčkává na příjem programovacích dat a provádí se jejich serializace do programovaného mikrokontroléru IO_4. Pokud je RST# = 1, jsou všechny linky portu P1 v log. 1 (log. 1 je realizována proudovým zdrojem cca 100 uA). Takže uložený program se bez komplikací rozběhne, linky P1.6, P1.4 a P1.2 navíc neovlivňují chování mikrokontroléru IO_4.

Zajímavostí je získání hodinového kmitočtu pro mikrokontrolér IO_3 přímo z krystalu X,. Mikrokontrolér IO_4 může používat krystal X_2 prakticky libovolného kmitočtu v rozsahu cca 1 až 10 MHz.

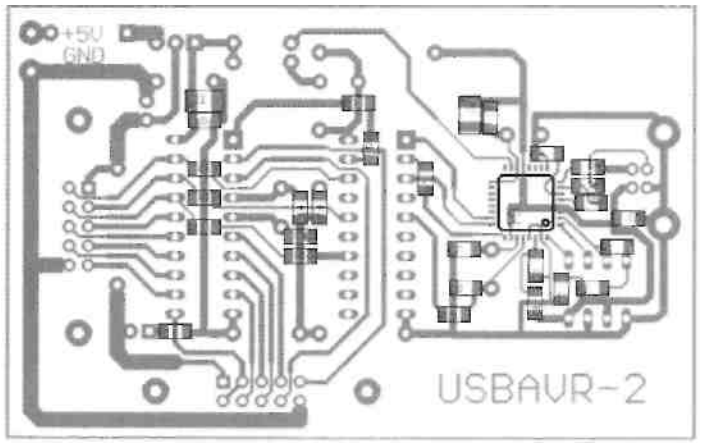


Obr. 7.9 Schéma zapojení USBAVR v 2.0

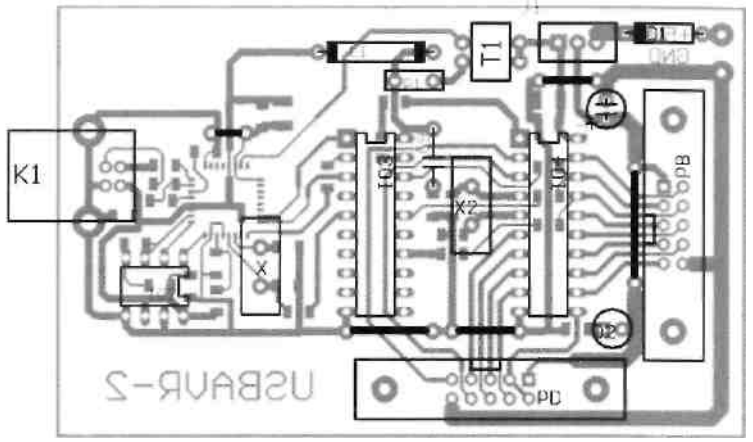
Výkres desky plošných spojů je uveden na *obr. 7.10.* Osazovací plánky jsou uvedeny na *obr. 7.11* a *obr. 7.12.*



Obr. 7.10 Výkres desky plošných spojů US BA VR-2 (BEN 0156)



Obr. 7.11 Osazovací plánek (strana spojů)

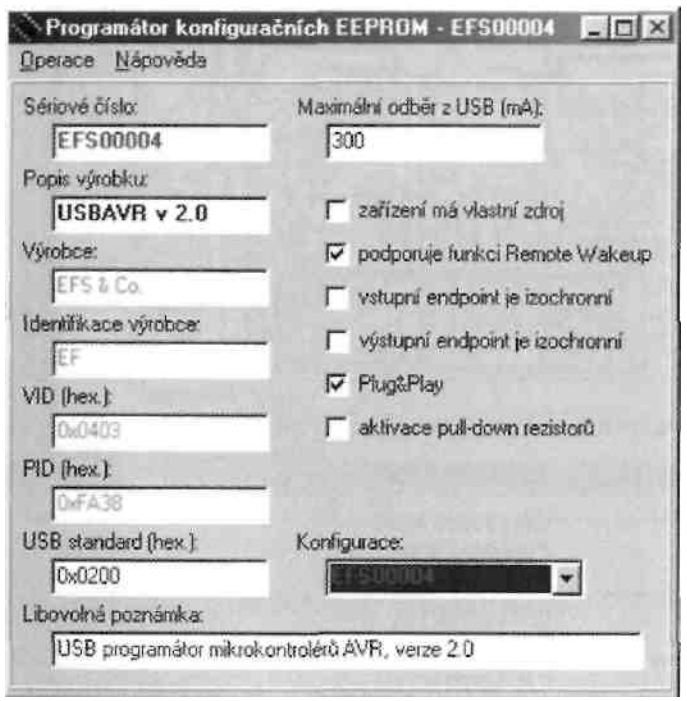


Obr. 7.12 Osazovací plánek (strana součástek) ***Seznam součástek***

pro USBAVR v 2.0 (cena asi 420 Kč):

C^ažCs	CK+100NX7R	3 ks
C_4	CK+33N X7R	1 ks
C_5	• CK+10NX7R	1 ks
C_6, C_{t1}	CTS6M8/10VB	2 ks
C_7, C_8, C_{13}, C_{14}	CK+27P NPO	4 ks
C_9	CK100NX7R	2 ks
C_{t2}	E100M/10V	1 ks
D,	1N5819	1 ks
D_2	LED 5MM 200MCD	1 ks
IO_1	FT232BM	1 ks
IO_2	94LC46B	1 ks
IO_3	89C2051-24PI	1 ks
IO_4	90S2313-10PI	1 ks
JT	3kolíky z S1G20	1 ks
K,	USB1X90B PCB	1 ks
L,	TL. 33uH	1 ks
P,	RXE250	1 ks
P_B, P_D	PSL10	2 ks
R,	RR+2K2 SMD	1 ks
R_2	RR+10KSMD	1 ks
R_3	RR+470R SMD	1 ks
R_4, R_5	RR+27R SMD	2 ks
R_6	RR+1K5SMD	1 ks
R_7 až R_{15}	RR+OR	9 ks
R_{17}	RR+1KSMD	1 ks
T,	IRFD9120	1 ks
X,	QM 6.000MHZ	1 ks
X_2	QM 10.000MHZ	1 ks

Konfigurace zařízení je zřejmá z *obr. 7.13.*



Obr. 7.13 Konfigurace E²PROM provedená programem EFSProg

Popis programu pro mikrokontrolér $I0_3$ (AT89C2051)

Jak bylo naznačeno výše, je v konstrukci **USBAVR v 2.0** použit mikrokontrolér **AT89C2051**, který zajišťuje potřebnou sériovou komunikaci sám o sobě a nezatěžuje tak sběrnici USB ani počítač serializací. Komunikace probíhá na přenosové rychlosti **31250 Bd.**

Program se rozbíhá tak, že se programovaný mikrokontrolér $I0_4$ resetuje. Nejdříve je na resetovací vstup přivedena log. 1 (program uložený v $I0_4$ se na chvíli rozběhne) a pak přejde resetovací vstup do log. 0 (mikrokontrolér $I0_4$ je držen v resetu). Před touto akcí však musí být linka **SCK** vynulována, jinak by se sériový download nezdařil.

Po konfiguraci sériového kanálu (UART) program vyčkává na příchod dat. Čekání je řešeno nekonečnou smyčkou (instrukcí **SJMP** $). Ze smyčky se vystoupí příjmem znaku sériovým kanálem (aktivací přerušení).

Po příjmu znaku se přejde na návěští **SERIÁL.** Přijatý bajt se vysílá po jednom bitu na linku **MOSI** a současně se přijímají bity z linky **MÍSO.** Takto přijatá hodnota je pak odeslána zpět do ovládacího programu (podle okolností se například může jednat o data přečtená z Flash nebo E²PROM či bajt signatury).

USBAVR2.ASM:

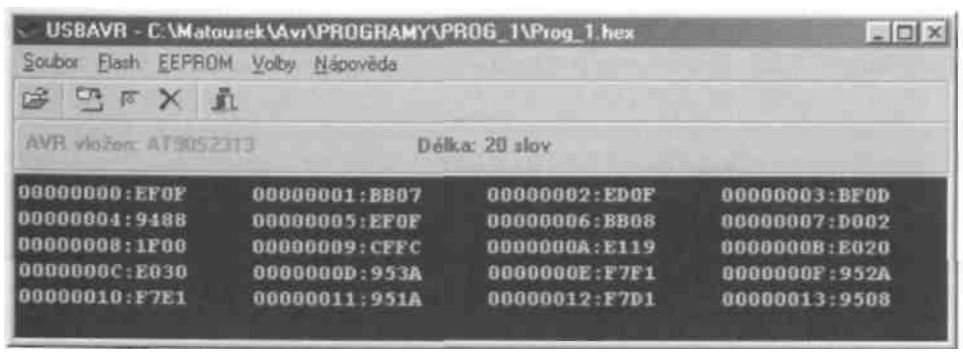
```
                $MODxx51
                DSEG  AT  30H
DATIN:          DS  1                   ;přečtená data
DATOUT:         DS  1                   ;zapisovaná data
MOSI            EQU  P1.2               ;signály
MÍSO            EQU  P1.4               ;AT90S2313
SCK             EQU  P1.6
RST             EQU  P1.7
BAUD            EQU  255                ;31250 Bd (SMOD=I)

                CSEG  AT  0000H
                AJMP  RESET             ;reset
                CSEG  AT  0023H
                AJMP  SERIÁL            ;obsluha UART
                ;reset:
RESET:          CLR  SCK                ;SCK=0
                SETB  RST               ;rozběh (RST=1)
                ACALL  CEKEJ            ;počkej
                CLR  RST                ;reset (RST=0)
                ACALL  CEKEJ            ;počkej
                MOV  TH1,#BAUD          ;přenos, rychlost
                MOV  TMOD,#00100000B    ;8bitový č/č 1
                MOV  SCON,#01010000B    ;8b async. přenos
                MOV  PCON,#10000000B    ;SM0D=1
                SETB  TR1               ;spusť č/č 1
                SETB  EA                ;povol
                SETB  ES                ;přerušeni
                SJMP  $                 ;čekej na data

                ;obsluha UART:
SERIÁL:         CLR  TI                 ;nuluj TI
                JBC  RI,SERIAP          ;test přijmu
                RETI                    ;konec při odvysíláni
                ;přijem dat:
SERIAP:         MOV  DATOUT,SBUF        ;ulož data
                MOV  DATIN,#0
                MOV  B,#8               ;8bitový přenos
SERIAS:         MOV  A,DATIN            ; čti
                MOV  C,MISO             ;bit
                RLC  A                  ; z MISO
```

```
        MOV   DATIN,A
        MOV   A,DATOUT          ;vysuň
        RLC   A                 ;nejvyšší
        MOV   MOSI,C            ;bit na MOSI
        SETB  SCK               ;SCK=1
        MOV   DATOUT,A
        CLR   SCK               ;SCK=0
        DJNZ  B,SERIAS          ;všechny bity?
        MOV   SBUF,DATIN        ;pošli přečtený bajt
SERIAK: RETI                    ;konec obsluhy

        ;čekací rutina:
CEKEJ:  MOV  A,#0
CEKSM:  MOV  B, #0
        DJNZ  B,$
        DJNZ  ACC,CEKSM
        RET
        END
```

7.5 OVLÁDACÍ PROGRAM PRO OBĚ VERZE USBAVR

Ovládací program **USBAVR** umožňuje ovládat programovou i datovou paměť mikrokontrolérů ATMELAVR, hlavní znaky:

- podporuje mikrokontroléry ATMELAVR typů: **AT90S2313, AT90S2343, AT90S4433, AT90S8515** a **AT90S8535** (je tedy možno vytvořit vývojové kity i pro další typy mikrokontrolérů),
- možnost nastavení zámků chránících obsah mikrokontrolérů proti zpětnému čtení,
- možnost smazat Flash a E^{2}PROM a provést reset mikrokontrolérů.



Obr. 7.14 Vývojové prostředí SDKAVR

Požadavky kladené na počítač:

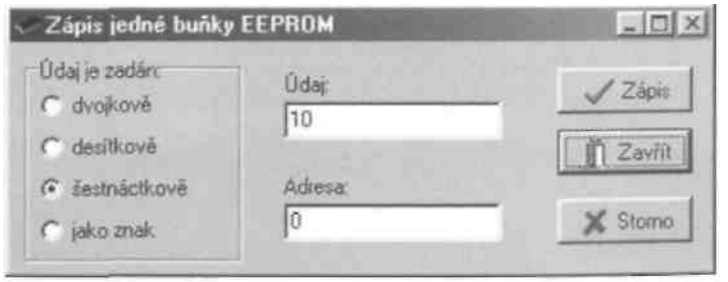
- operační systém Windows 98 a vyšší,
- volný USB port pro komunikaci s vývojovým kitem,
- instalace zabere méně než 1 MB prostoru na pevném disku.

Práce se soubory:

- **Soubor|Otevřít HEX-**otevře HEX soubor obsahující překlad programu (pozor ovládací program USBAVR vyžaduje generický formát HEX souboru),
- **Soubor|Uložit HEX** - uloží obsah okna do podoby HEX souboru,
- **Soubor|Čti Signaturu** - čte signaturu mikrokontrolérů a tím zjišťuje jeho typ,
- **Soubor|Konec** - ukončí vývojové prostředí.

Ovládání programové paměti (Flash):

- **Flash|Programovat Flash** - naprogramuje Flash obsahem dříve načteného HEX souboru,
- **Flash|Číst Flash** - čte obsah Flash a zobrazí jej v okně (lze uložit položkou menu **Soubor|Uložit HEX** do souboru),
- **Flash|Reset** - resetuje mikrokontrolér,
- **Flash|Smazat** - smaže obsah Flash i E^{2}PROM (nutné například před novým programováním po aplikaci zámků úrovně 2 nebo 3).



Obr. 7.15 Dialog pro editaci zvolené buňky E^{2}PROM

Ovládání datové paměti (E^{2}PROM):

- **EEPROM|Editace** - umožní editaci zvolené buňky E^{2}PROM jako dvojkové, desítkové nebo šestnáctkové číslo nebo znak (viz *obr. 7.15),*
- **EEPROM|Dump** - zobrazí výpis zvolené části E^{2}PROM (viz *obr. 7.16),*
- **EEPROM|Programovat EEP souborem** - naprogramuje E^{2}PROM souborem s příponou EEP, který je výsledkem překladu (podobně jako HEX).

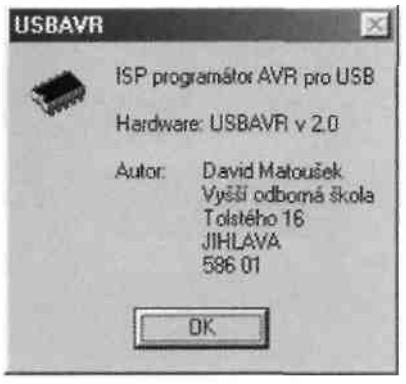
Obr. 7.16 Dialog pro zobrazení obsahu E^2PROM

Zámky:

- **Volby|Zámek** - volí zámek, který použijeme pro ochranu návrhu. Možnosti: **Nepoužit, Odstavení zápisu, Odstavení zápisu a verifikace.**

Nápověda:

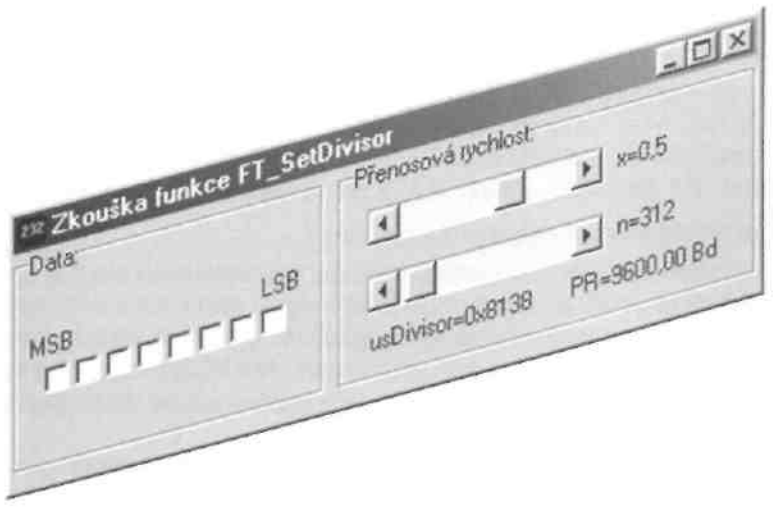
- **Nápověda|0 aplikaci** - podá krátkou informaci o programu. Kromě informací o autorovi se zobrazí název připojeného kitu (položka Hardware), viz *obr. 7.17.*



Obr. 7.17 Dialog o aplikaci USBAVR

8

PŘÍKLADY KOMUNIKACE VYBRANÝCH MIKROKONTROLÉRŮ S OBVODEM FT232BM



*V předešlých kapitolách jsme se seznámili s obvodem **FT232BM** a jeho ovládáním. Dále byly uvedeny konstrukce programátorů pro mikrokontroléry **AT89C2051** a **AT90S2313.** Je tedy na čase ukázat na jednodušších příkladech, jak s těmito mikrokontroléry pracovat ve spojení s konvertorem **FT232BM.***

8.1 NASTAVENÍ PŘENOSOVÉ RYCHLOSTI

Je jasné, že při použití mikrokontrolérů bude vhodné používat asynchronní kanál dostupný na linkách TXD a **RXD.** Signály pro řízení modemu můžeme z komunikace vypustit.

Patrně nejdůležitější je umět nastavit přenosovou rychlost. Připomeňme, že obvod FT232BM je schopen používat standardní přenosové rychlosti v rozsahu 300 Bd až 3 **MBd.** Dále je možné nastavovat i další rychlosti odvozené ze základního kmitočtu 3 MHz.

Nastavení standardní přenosové rychlosti

Nastavení standardní přenosové rychlosti je velmi jednoduché. Použijeme funkci FT_SetBaudRate případně FT_W32_SetCommState. Připomeňme hlavičku jednodušší z obou funkcí:

```
FT_STATUS FT_SetBaudRate(
    FT_HANDLE ftHandle,  ; handle zařízeni DWORD
    dwBaudRate   ; přenosová rychlost v baudech
);
```

*Obr. 8.1 Funkce **FT_SetBaudRate***

Přenosová rychlost (tedy 32bitový parametr dwBaudRate) se udává přímo v baudech. Například hodnota dwBaudRate = 9600 odpovídá 9600 Bd. Je-li uvedena jiná než standardní přenosová rychlost, je nastavení neúspěšné a funkce vrátí hodnotu FT_INVALID_BAUD_RATE (viz dále v textu!!!).

Nastavení nestandardní přenosové rychlosti

Použití nestandardní přenosové rychlosti je velmi důležité hlavně ve spojení s jednoduššími typy mikrokontrolérů. Ty totiž obvykle mají velmi jednoduchý generátor přenosových rychlostí a tak je nutné používat specifické hodnoty oscilačních kmitočtů. Příkladem může být mikrokontrolér AT89C2051, který ve spojení s asynchronním sériovým přenosem obvykle potřebuje použít krystal hodnoty 11,059 MHz.

Možnost nastavit nestandardní přenosovou rychlost však dovoluje vypořádat se s touto obtíží a mikrokontrolér může používat prakticky libovolnou hodnotu oscilačniho kmitočtu volenou například s ohledem na jednoduché časování doby provedení určitého sledu instrukcí (u mikrokontrolérů AT89C2051 trvá při použití krystalu 12 MHz jeden instrukční cyklus právě 1 us).

Nastavení nestandardní přenosové rychlosti probíhá pomocí funkce FT_SetDivisor. Její hlavička je velmi podobná výše popsané funkci FT_SetBaudRate.


```
FT_STATUS FT_SetDivisor(
    FT_HANDLE ftHandle,  ; handle zařízeni
    USHORT USDiVÍSOR     ; nastaveni děličky
);
```

*Obr. 8.2 Funkce **FT_SetDivisor***

Dělička (tedy 16bitový parametr **usDivisor)** udává požadované dělení základního kmitočtu 3 MHz. Dělitel sestává z celé částí (označujeme jako n) a desetinné části (označujeme jako x). Takže výsledná přenosová rychlost je:

$$PR = \frac{3\ MHz}{n + x}$$

Obvody řady BM (tedy i FT232BM) mohou pracovat s desetinným dělitelem těchto hodnot: 0; 0,125; 0,25; 0,375; 0,5; 0,625; 0,75; 0,875. Takže možné hodnoty dělitele lze vyjádřit *tab. 8.1.*

Tab. 8.1 Možné hodnoty dělitelů

Dělitelé			
n+0	n+0,25	n+0,5	n+0,75
n+0,125	n+0,375	n+0,625	n+0,875

Fakt, že obě části dělitele se zadávají jako jediné 16bitové číslo poněkud komplikuje celé nastavení. Výrobce totiž v původní verzi FT232AM vyhradil pro desetinnou část dělitele pouze 2 bity. To znamená, že funkce FT_SetDivisor není schopna nastavit jiné dělitele než tyto: n + 0; n + 0,5; n + 0,25; n + 0,125.

*Obr. 8.3 Jednotlivé bity **usDivisor***

Horní 2 bity parametru usDivisor odpovídají desetinnému děliteli a spodních 14 bitů pak celočíselnému děliteli. Bitové vyjádření jednotlivých hodnot desetinného dělitele je uvedeno formou *tab. 8.2.* Vzhledem k tomu, že desetinný dělitel je 3b;tové číslo, lze funkcí FT_SetDivisor nastavit jen dělitele uvedené v levé části této tabulky (ty mají nejvyšší bit nulový). Bohužel jiná funkce pro nastavení přenosové rychlosti není v manuálu výrobce uvedena.

Tab. 8.2 Desetinný dělitel a jeho kódování

kód	x	kód	x
000	0	100	0,375
001	0,5	101	0,625
010	0,25	110	0,75
011	0,125	111	0,875

Příklad č. 1: Určete hodnotu **usDivisor** pro přenosovou rychlost **6250 Bd.**

Nejdříve musíme stanovit hodnotu dělitele:

$$n + x = \frac{3\ \text{MHz}}{PR} = \frac{3\,000\,000}{6250} = 480\,,$$

je tedy zřejmé: **n = 480, x = 0.** Výsledek: **usDivisor = 480** (hexadecimálně 0x01E0).

Příklad č. 2: Určete hodnotu **usDivisor** pro přenos rychlost **6243,5 Bd.**

Stanovíme hodnotu dělitele:

$$n + x = \frac{3\ \text{MHz}}{PR} = \frac{3\,000\,000}{6243{,}5} = 480{,}49972\,,$$

přibližně tedy: **n = 480,** x = 0,5. Výsledek hexadecimálně 0x41 E0: **usDivisor = 16864.**

Příklad č. 3: Určete, jaká přenosová rychlost odpovídá nastavení ***usDivisor = 32924.***

Nejdříve převedeme hodnotu dělitele do dvojkové soustavy: 1000 0000 1001 1100. Tedy: n = 156, x = 0,25. Výsledná přenosová rychlost:

$$PR = \frac{3\,000\,000}{156{,}25} = \mathbf{19\,200\ Bd}\,.$$

Poznámka k funkci FT_SetBaudRate:

*V popisu funkce **FT_SetBaudRate** se dříve uvádělo, zeje schopna nastavovat pouze standardní přenosové rychlosti.*

*Po uvolnění plné verze ovladače však došlo k tomu, že nestandardní přenosovou rychlost lze nastavit nejen funkcí **FT_SetDivisor,** ale i funkcí **FT_SetBaudRate.** Pokud hodnota uvedená parametrem funkce **FT_SetBaudRate** neodpovídá přesně děliteli, najde se takový dělitel, který poskytne rychlost nejbližší požadované hodnotě.*

*Funkce **FT_SetBaudRateje** tedy schopna využívat i jemné dělení zavedené obvodem **FT232BM,** což funkce **FT_SetDivisor** nedovoluje. Sám aplikační inženýr týmu FTDI **Keith Dingwall** uznává, že funkce **FTJSetDivisor** je nyní považována za zastaralou!*

Program pro testování přenosové rychlosti

Pro účely praktického ověření funkce **FT_SetDivisor** byl vytvořen testovací program. Najdete jej na doprovodném CD-ROM v adresáři **PROGRAMY\KAP_08\Divisor.**

Program je velmi jednoduchý, bajt nastavený pomocí checkboxů v panelu **Data** je neustále vysílán zvolenou přenosovou rychlostí. Hodnoty obou částí dělitele se zadávají v panelu **Přenosová rychlost.** Program je určen pro testovací přípravek **FT232TST.**

UNIT1.CPP:

```
// ---------------------------------------------------------------------------
#include   <vcl.h> #include
"Ftd2xx.h" #pragraa   hdrstop
#include   "Unitl.h"
// ---------------------------------------------------------------------------
#pragma   package(smart_init) #pragma
resource   "*.dfm" TForml    *Forml;
// ---------------------------------------------------------------------------
   fastcall   TForml:  :TForml (TComponent*   Owner)
    :TForm(Owner)
{
    ftStatus=FT_OpenEx("Test",
        FT_OPEN_BY_DESCRIPTION,SftHandle); if
    (ftStatus!=FT_OK)
        throw   Exception("Zařízení   test   není   připojeno!");

    int   i;
    sbXScroll(NULL,scEndScroll, i) ;
}
1/ ---------------------------------------------------------------------------
 _fastcall   TForml::-TForm1()
{ FT_Close(ftHandle);
}
//---------------------------------------------------------------------------
//obsluha   obou   scrollbarů:
void    __fastcall   TForml::sbXScroll(
    TObject   *Sender,   TScrollCode   ScrollCode,
    int   SScrollPos)
{
    USHORT   usDivisor;
    int   n,x;
    //hodnoty   dle   tab.   8.2:
    int   konv[4]={0,3,2,1};

    //přestaveni:
    if (ScrollCode==scEndScroll)  { //uloženi
        pozic   scrollbarů:
```

```
n=sbN->Position;
x=sbX->Position;

//výpis hodnot n a x:
lbN->Caption=AnsiString("n=")+n; lbX-
>Caption=AnsiString("x=")+x/4.0;

//sestaveni usDivisor: usDivisor= (
(konv[x] )«14) | (n) ;

//výpis usDivisor hexadecimálně:
lbDivisor->Caption=
    "usDivisor=Ox"+IntToHex(usDivisor, 4) ;

//výpis přenosové rychlosti:
lbPR->Caption=FormatFloat("PR=0.00 Bd",
    3000000.0/(n+x/4.0));

//nastaveni přenosové rychlosti:
ftStatus=FT_SetDivisor(ftHandle,usDivisor);
    //test chyby:
    if(ftStatus!=FT_OK)
      MessageBox(Handle,
                 "Chyba při voláni funkce FT_SetDivisor",
                 Application->Title.c_str(),
                 MB_ICONHAND);
    }
}
//---------------------------------------------------------------------------

//vysílá data při neaktivitě aplikace:
void ___fastcall TForml: :ApplicationEventsIldle(
    TObject *Sender, bool &Done)
{
    Done=false; BYTE
    b; DWORD d;

    //sestavení bajtu pro odesláni: b=cbd7-
    >Checked*12 8
        +cbd6->Checked*64
        +cbd5->Checked*32
        +cbd4->Checked*16
        +cbd3->Checked*8
        +cbd2->Checked*4
```



```
        +cbdl->Checked*2
        +cbd0->Checked;

    //odeslání:
    FT_Write(ftHandle,&b,1,&d);

//pauza cca 1 ms:
Sleep(l); }
```

Obr. 8.4 Testovací program v akci

8.2 PŘÍKLADY POUŽITÍ MIKROKONTROLÉRŮ AT89C2051 A AT90S2313

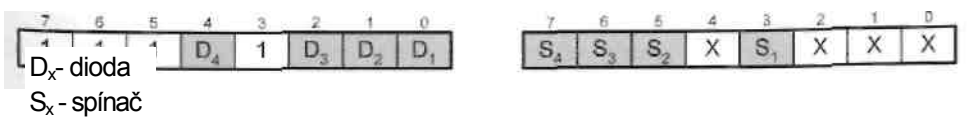
Oba příklady jsou sestaveny stejným způsobem: K portu P3 (AT89C2051) resp. **PD** (AT90S2313) je připojen přípravek **FT232TST** (jumper musí být vyjmut), na port **P1** (AT89C2051) resp. PB (AT90S2313) je připojen přípravek **FTDITEST (z** kapitoly **5.6.1).**

Formát přenosu

Formát přenosu dat mezi počítačem a mikrokontrolérem je velmi jednoduchý: Počítač odešle bajt definující, které LED na přípravku **FTDITEST** mají svítit. Ostatní bity jsou nastaveny na log. 1 (způsobí aplikaci pull-up rezistorů u mikrokontroléru **AT89C2051).**

Mikrokontrolér posílá zpět sejmutý stav portu **P1/PB.** Tento údaj pak určuje, které spínače na přípravku **FTDITEST** jsou sepnuty.

z PC do mikrokontroléru **z mikrokontroléru do PC**



Obr. 8.5 Formát přenosu

Testovací aplikace pro Windows

Testovací aplikace pro počítač je sestavena tak, jak určuje předchozí popis formátu přenosu. Časovač zajišťuje periodické čtení checkboxů pro aktivaci diod a odesílá příslušnou hodnotu do mikrokontroléru. Zároveň zobrazuje aktuální stav spínačů načtený z mikrokontroléru.

Přenosová rychlost byla zvolena jako **12 500 Bd.**

Příklad naleznete v adresáři: **CD-ROM\PROGRAMY\KAP_08\WINAPP.**

WINAPPHF.CPP:

```
//---------------------------------------------------------------------------
#include   <vcl.h>
#include   "Ftd2xx.h"
#pragma   hdrstop
#include   "WinAppHF.h"
// ---------------------------------------------------------------------------
#pragma   package(smart_init)
#pragma   resource   "*.dfm"
TForml   *Forml;
// ---------------------------------------------------------------------------
__fastcall   TForml::TForml(TComponent*   Owner)
  :TForm(Owner) {
    FT_STATUS   ftStatus;
    //otevře   zařízení   Test:
    ftStatus=FT_OpenEx("Test",
        FT_OPEN_BY_DESCRIPTION,&ftHandle); if
    (ftStatus!=FT_OK)
        throw Exception("Zařízení test není připojeno!");
    //nastavení přenosové rychlosti 12500 Bd:
    FT_SetDivisor(ftHandle,OxOOfO);
    FT_SetTimeouts(ftHandle,100,100);
}
//---------------------------------------------------------------------------
__fastcall   TForml::-TForml()
{
    FT_Close(ftHandle);
>
//---------------------------------------------------------------------------
//obsluha   časovače:
void   ____fastcall   TForml::AktivaceCasovace(
    TObject   *Sender)
{
    BYTE   OutData,InData; DWORD   p;
    //sestavení   dat   pro   odeslání: OutData=
        D4->Checked*16
        +D2->Checked*2
        +D3->Checked*4
        +Dl->Checked
        +8+32+64+128;
    //odešle   data   do mikrokontroléru:
    FT_Write(ftHandle,&OutData,1,&p);
    //přijme   data   z mikrokontroléru:
    FT_Read(ftHandle,SlnData,1,&p);
    //rozloží přijaté bity:
SW4->Checked=InData&0x80;          SW3-
>Checked=InData&0x40;              SW2-
>Checked=InData&0x20;               SWI-
>Checked=InData&0x08; }
```

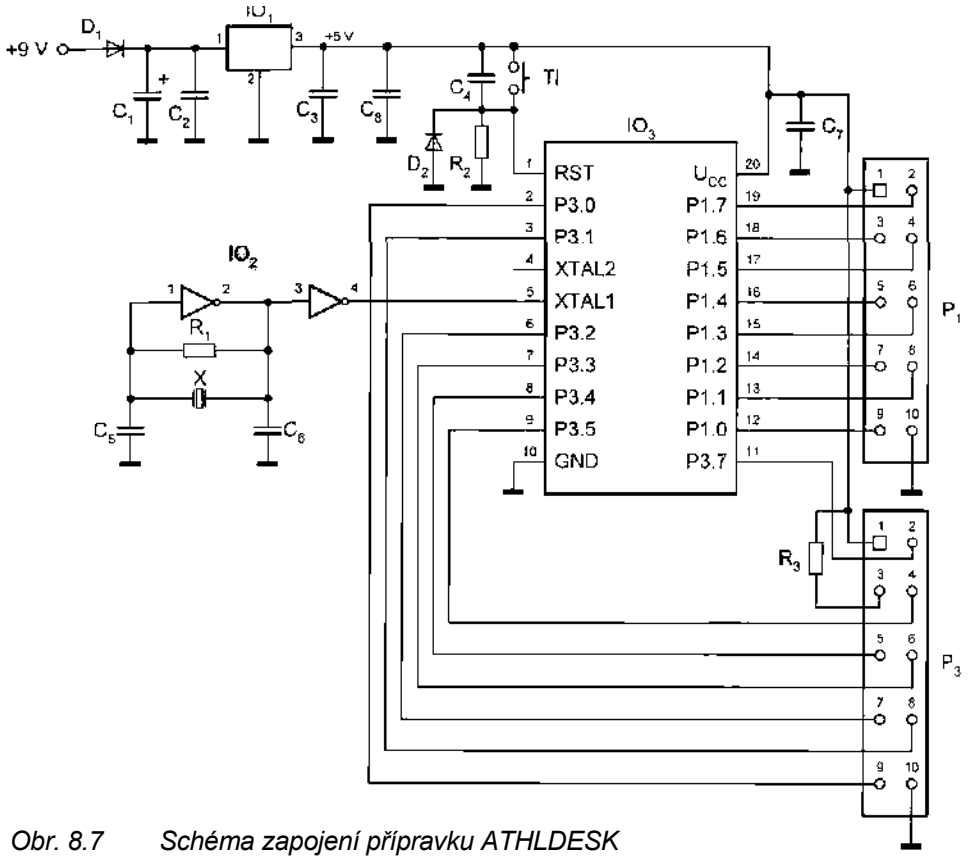
Obr. 8.6 Aplikace WINAPP.EXE v akci

Test mikrokontroléru AT89C2051

Při testování programu pro mikrokontrolér **AT89C2051** připojíme testovací přípravek **FT232BM** na port **P3,** vývody TXD a RXD odpovídají přímo linkám RxD (P3.0) a TxD (P3.1). Na port **P1** bude připojen přípravek **FTDITEST.**

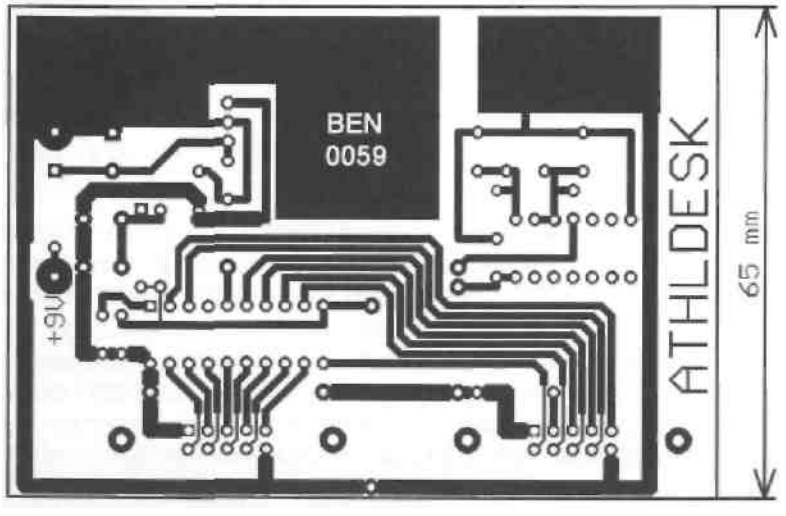
Jedno zmožných řešení testovacího přípravku bylo publikováno již v [1] pod označením **ATHLDESK.** Pro informaci je schéma zapojení uvedeno na *obr. 8.7.*

Mikrokontrolér musíme nejdříve naprogramovat souborem **TEST2051.BIN,** zdrojový soubor **TEST2051.ASM** je vypsán níže. Obojí najdete na doprovodném CD-ROM v adresáři **PROGRAMY\KAP_08\TEST2051**. Naprogramování lze provést libovolným programátorem, například lze použít **ATPROG 3.0** popsaný v kapitole 6.



Obr. 8.7 Schéma zapojení přípravku ATHLDESK

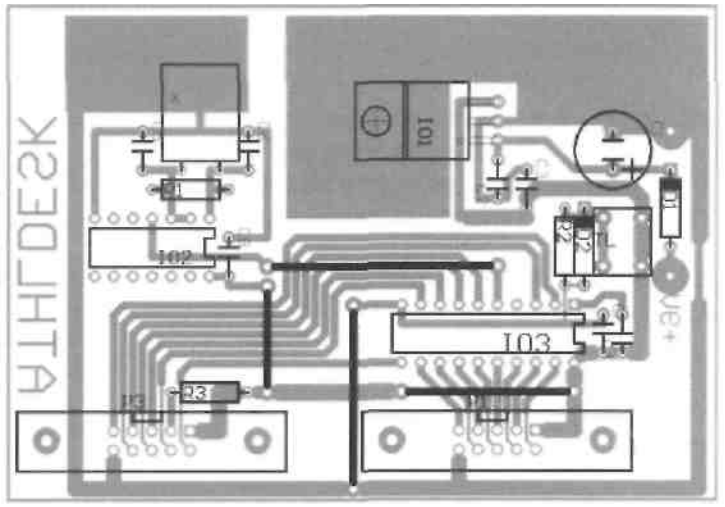
Výkres desky plošných spojů testovacího přípravku **ATHLDESK** uvádí *obr. 8.8.*



Obr. 8.8 Výkres desky plošných spojů přípravku ATHLDESK (BEN 0059)

Osazovací plánek je uveden na *obr. 8.9.* Na přípravku je několik drátových propojek. Přípravek musí být napájen z vnějšího zdroje 9 V.

UPOZORNĚNÍ: *Před připojením přípravku* ***FT232TST*** *k portu P3 nezapomeňte vyjmout jumper!*



Obr. 8.9 Osazovací plánek přípravku ATHLDESK

Rozpis součástek pro vývojový kit ATHLDESK (cena asi 70 Kč):

IO_1	7805	1 ks
IO_2	74HCT04 (74HC04)	1 ks
IO_3	precizní patice DIP20	1 ks
D1 1N4001 až1N4007	1	ks
D_2 1N4148	1	ks
X Q 24.000MHZ	1	ks
C, E470M/16V	1	ks
C_2ažC_4, C_7,C_8 100n	5	ks
C_5, C_6 10p	2	ks
R, 1M	1	ks
R_2, R_3 10k	2	ks
P_{1t} P_3 PSL10	2	ks
TI	tlačítko P-B1720	1 ks

Na celém programu pro mikrokontrolér je patrně nejdůležitější nastavení přenosové rychlosti 12 **500** Bd. Připomeňme, že přenosová rychlost se definuje nejčastěji pomocí čítače/časovače 1 podle vzorce:

$$TH1 = 256 - \frac{2^{SMOD} \cdot f_0}{32 \cdot 12 \cdot PR}$$

kde: **TH1** je obsah čítače/časovače **1,**
SMOD je bit ovlivňující přenosovou rychlost (násobení 1 x nebo 2*),
f_0 pracovní kmitočet mikrokontroléru (24 MHz),
PR požadovaná přenosová rychlost. Pro náš případ dostaneme (uvažujeme nastavení SMOD = 1):

$$TH1 = 256 - \frac{2^{SMOD} \cdot f_0}{32 \cdot 12 \cdot PR} = 256 - \frac{2 \cdot 24\,000\,000}{32 \cdot 12 \cdot 12\,500} = 256 - 10 = 246$$

Další zápisy patří již do kategorie programů řízených sériovým kanálem. Nejdříve musíme nastavit registr **TH1** a konfigurovat sériový kanál pomocí registrů **SCON** a **PCON.** Dále musí být nastaven režim čítače/časovače 1 pomocí registru **TMOD.** Následuje povolení přerušení, poté program „bloudí" v nekonečné smyčce.

Příjem nebo odvysílání znaku je doprovázeno aktivací přerušení, jehož obsluha je uvedena od návěští **SERIÁL.** Pomocí bitů **RI** a **TI** se testuje příčina přerušení.

TEST2051.ASM:

```
            $MODxx51
BAUD        EQU  246                ;12500 Bd
            SJMP  RESET             ;reset
            ORG  0023H
            SJMP  SERIÁL            ;sériový kanál
;inicializace:
RESET:      MOV  TH1,#BAUD          ;přenos, rychlost
            MOV  TMOD,#00100000B    ;č/č 1, 8b reload
            MOV  SCON,#01010000B    ;8b async. přenos
            MOV  PCON,#10000000B    ;SMOD=l
            SETB  TRI               ;spusť č/č 1
            SETB  EA                ;povol přesušení
            SETB  ES                ;od UART
            SJMP  $                 /nekonečná smyčka

/obsluha UART:
SERIÁL:     JB  RI,SERIAR           /test RI/TI
            /TI nastaven->odvysílání
SERIAT:     CLR  TI                 /vynuluj TI
            RÉTI                    ;a konec
            /RI nastaven->příjem
SERIAR:     MOV  A,SBUF             /čti přijatá data
            MOV  PÍ,A               /vystav na port
            MOV  A,PÍ               /čti stav portu
            MOV  SBUF,A             /a odešli zpět
            CLR  RI                 /vynuluj RI
```



```
            RETI                    /a konec
            END
```

UPOZORNĚNÍ: *Vzhledem k vnitřní konstrukci mikrokontroléru* ***AT89C2051*** *(log. 1 je vytvářena přes pull-up) nebudou patrně diody D_1 až D_4 svítit dostatečně silně (proud je zhruba 100 uA).*

Test mikrokontroléru AT90S2313

Při testování programu pro mikrokontrolér **AT90S2313** připojíme testovací přípravek **FT232BM** na port **PD,** vývody TXD a RXD odpovídají přímo linkám RxD (PDO) a TxD (PD1). Na port **PB** bude připojen přípravek **FTDITEST.**

Osobně doporučuji použít vývojový kit **USVAVR 2.0,** který dovoluje mikrokontrolér jak naprogramovat, tak i vyzkoušet (viz kapitolu 7.4).

UPOZORNĚNÍ: *Před připojením přípravku* ***FT232TSTk*** *portu* ***PD*** *nezapomeňte vyjmout jumper!*

Nastavení přenosové rychlosti provádíme pomocí registru **UBRR:**

$$UBRR = \frac{f_0}{16 \cdot PR} - 1$$

kde: **UBRR** je obsah čítače/časovače 1,
f_0 pracovní kmitočet mikrokontroléru (uvažuji 10 MHz), **PR** požadovaná přenosová rychlost. Pro náš případ dostaneme:

$$UBRR = \frac{f_0}{PR} - 1 = \frac{10\,000\,000}{16 \cdot 12\,500} - 1 = 50 - 1 = 49$$

V inicializační fázi je nutno stanovit vývody, které mají funkci výstupu pomocí registru **DDRB.** Dále se nastaví přenosová rychlost pomocí registru **UBRR** a nakonec se sériový kanál konfiguruje jako 8bitový UART s povolením příjmu i vysílání. Povolena je pouze aktivace přerušení při příjmu znaku.

Vektor přerušení se používá pouze pro příjem, při příjmu znaku se totiž hodnota sejmutá z portu PB odesílá zpět do počítače.

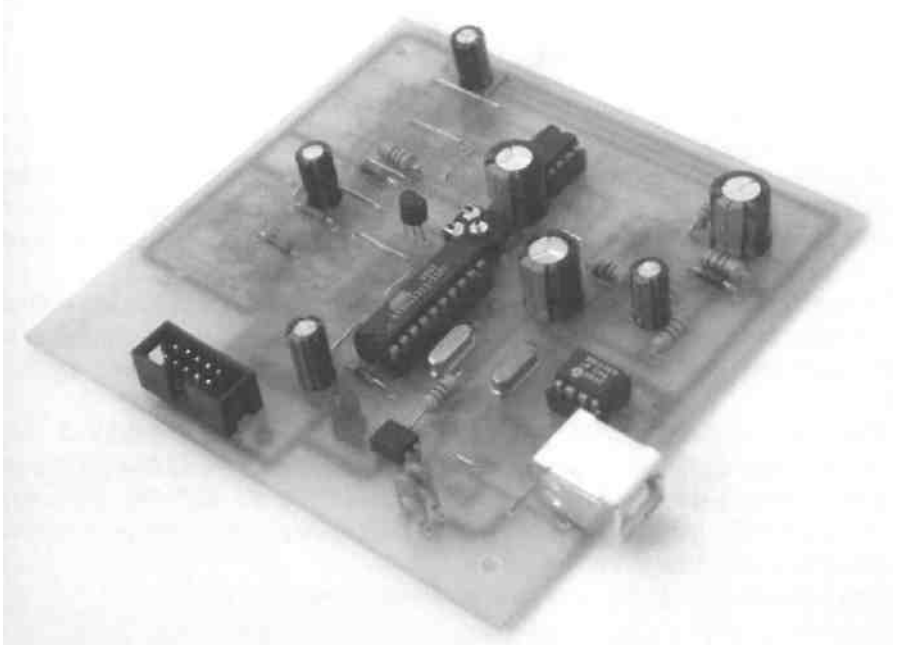
TEST2313.ASM:

```
            .NOLIST
            .INCLUDE  "2313def.inc"
            .LIST
            .CSEG                   /kódový segment
            .DEF  REG=R16           /pracovní registr
            RJMP  RESET
            .ORG  $0007             /vektor UART RX
            RJMP  RX
```

```
/inicializace:
RESET:      LDI   REG,RAMEND
            OUT   SPL,REG                 /nastavení   SP
            LDI   REG,0b00010111
            OUT   DDRB,REG                /aktivace   výstupů
            LDI   REG,4 9
            OUT   UBRR,REG                /12 50 0   Bd
            LDI   REG,0bl0011000
            OUT   UCR,REG                 /povol   přijem/vysiláni
            SEI                           /povol   UART   RX
            RJMP  PC                      /smyčka

/obsluha  UART   RX:
RX:         IN   REG,UDR                  /čti   data
            OUT   PORTB,REG               /vystav   data   na   PB
            IN   REG,PINB                 /čti   stav   PB
            OUT   UDR,REG                 /pošli   zpět
            RETI                          /konec   obsluhy
```

10

USBMC – UNIVERZÁLNÍ MĚŘICÍ DESKA



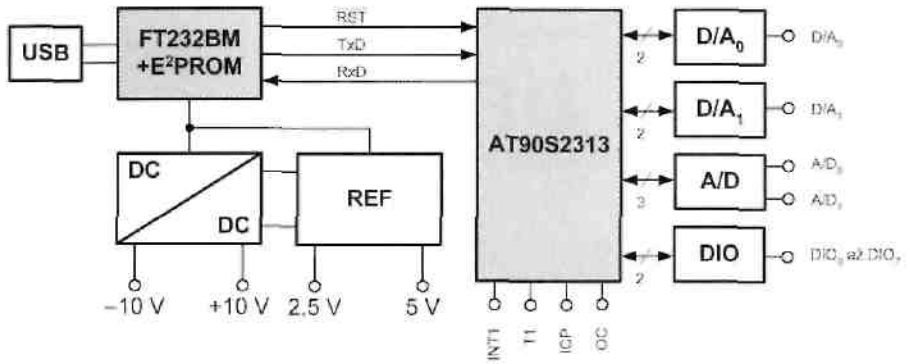
Poslední konstrukcí je univerzální měřicí deska napájená přímo ze sběrnice USB (nepotřebuje vnější zdroj), která obsahuje: 8 digitálních vstupů/výstupů, 2kanálový 8bitový D/A převodník pracující v rozsahu -5 V až +5 V a 2kanálový 10bitovýA/D převodník pracující v rozsahu -5 V až +5 V s diferenčními vstupy.

*Dále jsou k dispozici vývody řídicího mikrokontroléru poskytující jednak vstup přerušení, hodinový vstup 16bitového čítače/časovače, vstup jednotky Input Cap-**ture a** výstup jednotky Output Compare.*

*Jako řídicí mikrokontrolér byl použit obvod **AT90S2313,** který disponuje řadou kvalitních zabudovaných periferií. K dispozici jsou zdrojové texty ovládacích rutin na úrovni **C++** Builderu (rutiny pro **Delphi** najdete v [20]).*

10.1 SCHÉMA ZAPOJENÍ

Blokové schéma zapojení měřicí desky je uvedeno na *obr. 10.1.*



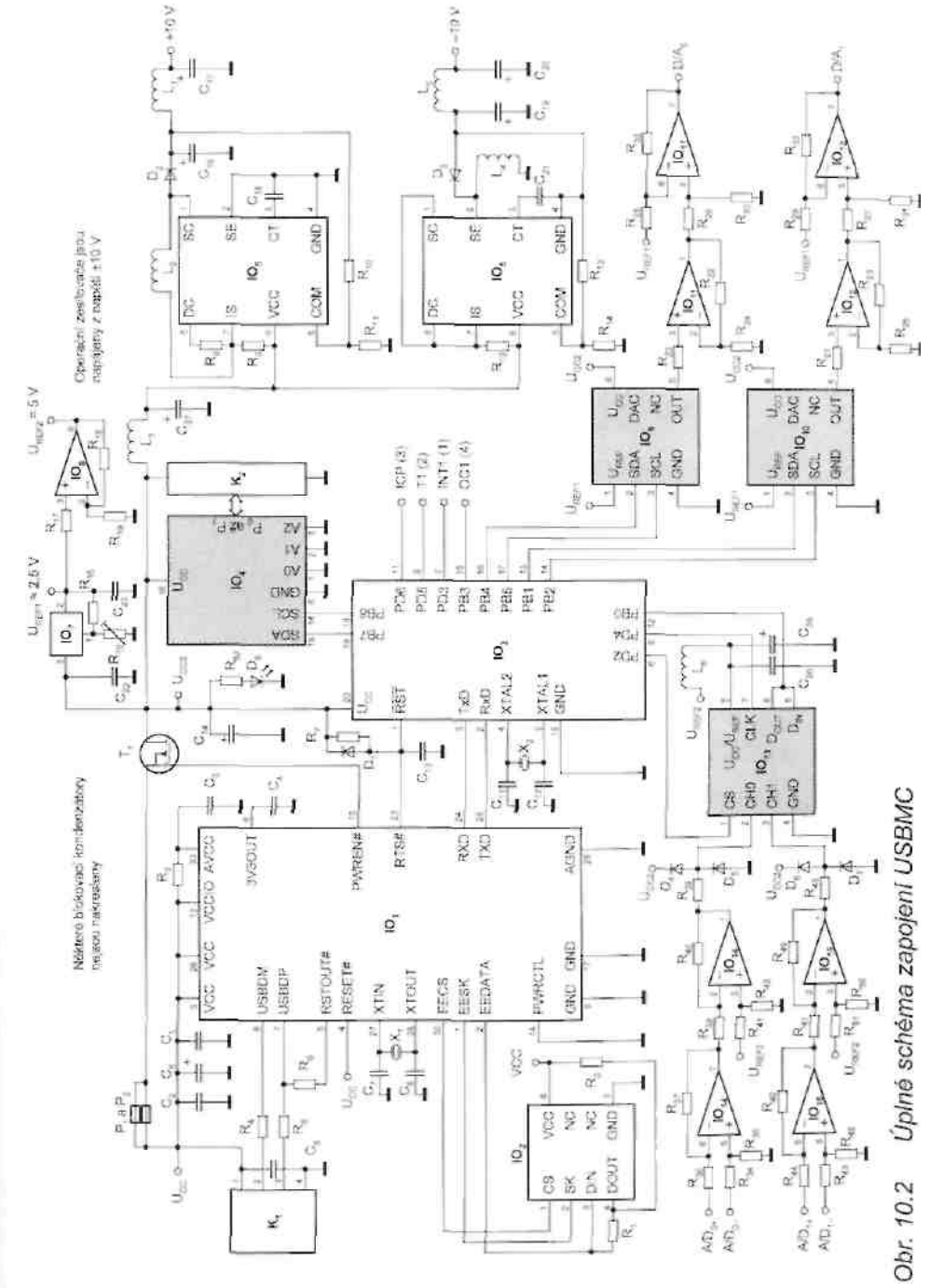
Obr. 10.1 Blokové schéma USBMC

Jak vidíme, je napájení pro analogové obvody získáno pomocí DC/DC konvertorů. Jsou použity obvody MC34063, které v daném zapojení poskytují výstupní napětí přibližné hodnoty +10 V a-10 V. Dále jsou vytvořena referenční napětí hodnot +2,5 V a +5 V.

Pro realizaci D/A převodníků byly použity levné obvody TC1320. Jako A/D převodník jsem použil obvod MCP3002, který disponuje dvěma kanály. V zájmu snížení rušení mají oba A/D kanály diferenční vstupy realizované pomocí operační sítě.

Vzhledem k relativně malému počtu vývodů mikrokontroléru AT90S2313, musel být pro realizaci digitálních vstupů a výstupů použit obvod PCF8574A. Dále jsou k dispozici vývody mikrokontroléru označené jako INT1, T1, ICP, OC, které dovolují používat vnější vstup přerušení a 16bitový čítač/časovač včetně jeho jednotek Input Capture a Output Compare.

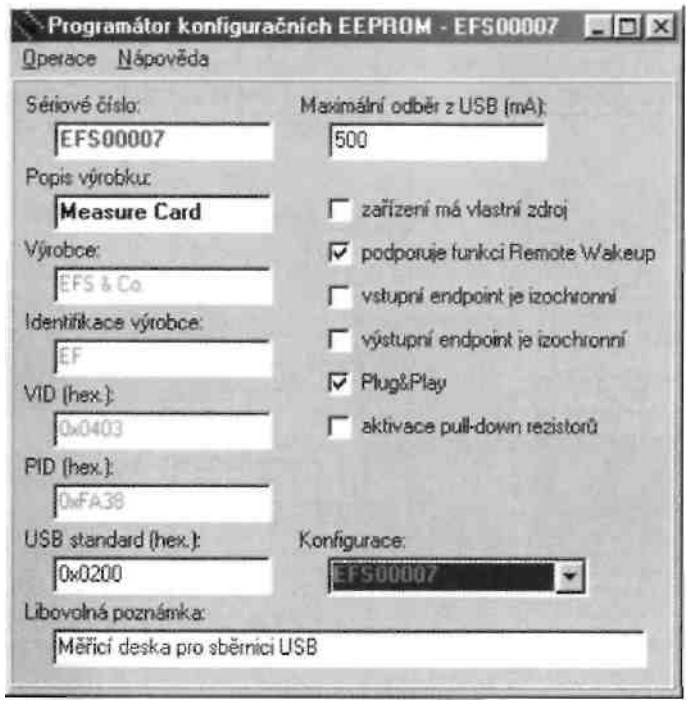
Úplné schéma zapojení měřicí karty je uvedeno na *obr. 10.2.*



Obr. 10.2 Úplné schéma zapojení USBMC

Komunikační část

Komunikační část je tvořena obvodem **FT232BM** (10,) spolu s konfigurační E^2PROM ($I0_2$), konfigurace je zřejmá z *obr. 10.3.* Z konvertoru FT232BM se používají vývody **RTS#, TXD** a **RXD** pro ovládání resetu mikrokontroléru a komunikačních linek **RxD** a **TxD.**



Obr. 10.3 Konfigurace E^2PROM provedená programem EFSProg

Připojení napájení

Odběr ze sběrnice USB je značný a proto je omezen dvěma paralelně řazenými součástkami typu polyswitch, které jsou označeny jako P, a P_2. Obě mají hodnotu 250 mA (také lze použít jediný polyswitch hodnoty 500 mA). Připojení desky k napájení zajišťuje tranzistor T, **(IRFD9120),** indikaci připojení pak zajišťuje dioda D_8.

Napěťové konvertory

Napětí nominální hodnoty +5 V získané z USB sběrnice je pro snížení rušení produkovaného do počítače odděleno tlumivkou L,. Pomocí konvertorů $I0_5$ a $I0_6$ **(MC34063)** je pak převedeno na hodnoty zhruba +10 V a -10 V. Z těchto napětí jsou napájeny všechny operační zesilovače typů **TL071** a **TL072** ($I0_8$, **$I0_{11f}$** $I0_{12}$, IO,4. $I0_{15}$).

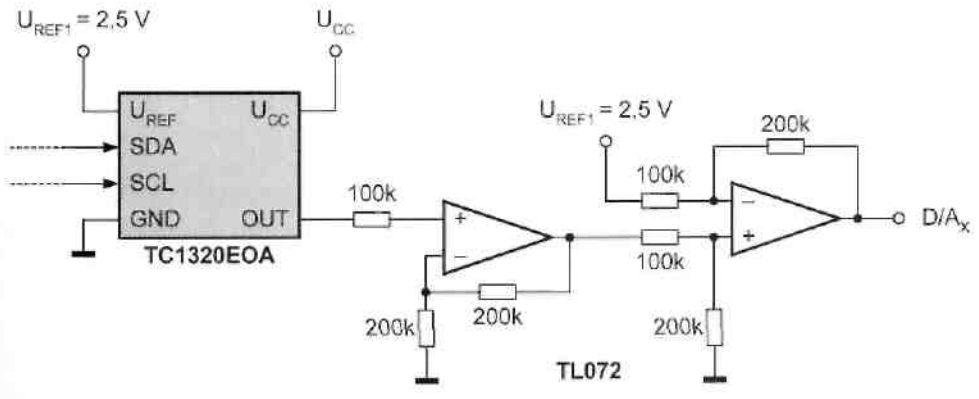
Referenční napětí

Pro A/D a D/A převodníky je také důležitý zdroj referenčního napětí. Jako napěťová reference je použit obvod **LM317L** ($I0_7$), pomocí trimru R_{16} se nastaví hodnota vývodu 2 proti zemi přesně na U_{REF1} = 2,5 V. Jako druhé referenční napětí se

používá U_{REF2} = 5 V. Toto napětí je získáno operačním zesilovačem **TL071** ($I0_8$) prostým vynásobením 2*. Pro omezení rušení je vložen filtrační článek L_6 a C_{36} (zjistil jsem, že tyto součástky prakticky nejsou nutné: cívku lze nahradit propojkou, kondenzátor nemusíme osazovat).

DIA převodníky

Oba D/A převodníky jsou řešeny samostatně. Obvod **TC1320** (podrobný popis byl proveden v kapitole 5.8.1) má totiž svoji adresu stanovenu napevno již z výroby a tak není možno na jednu I^2C sběrnici připojit více než jeden tento obvod. Jelikož je referenční napětí obvodů **TC1320** ($I0_9$, IO_{10}) hodnoty 2,5 V, musí být na požadovaný výstupní rozsah -5 V až +5 V převedeno operační sítí obvodů **TL072** ($I0_{11f}$ $I0_{12}$). Výstupní napětí z D/A převodníků je nejdříve 2x zesíleno (má rozsah 0 až 5 V) a poté je od něj odečtena hodnota U_{REF1} = 2,5 V. Vše se pak ještě 2* zesílí (tak dostaneme požadovaný rozsah výstupu -5 V až +5 V). Vstupnímu číslu 0 odpovídá výstupní napětí -5 V, vstupnímu číslu 255 odpovídá výstupní napětí +5 V. Pro lepší představu je operační síť zachycena na *obr. 10.4.*



Obr. 10.4 Operační síť D/A převodníku

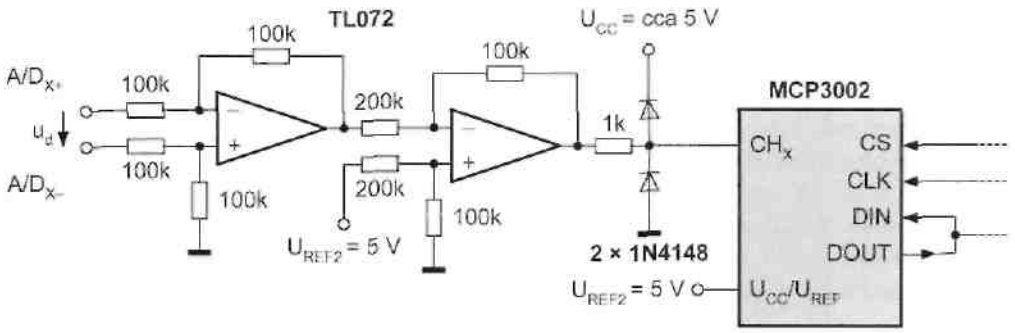
A/D převodníky

A/D převodníky byly realizovány 2kanálovým 10bitovým A/D převodníkem **MCP3002** (podrobný popis lze najít například v [16], originální dokumentace je také k dispozici na doprovodném CD-ROM). Programově se zpracovává pouze horních 8 bitů, takže rozsah je 8 bitů.

Vstupy A/D převodníků jsou diferenční, výhodou tohoto řešení je především snížení vstupního rušení (rušivé signály na diferenčních vstupech se vzájemně odečítají). Dále je možno měřit napětí v libovolném místě obvodu, nemusí být vztaženo proti zemi. To dává možnost jednoduché realizace měřicích úloh (například lze snadno měřit proud použitím snímacího rezistoru, který nemusí být uzemněn; viz kapitolu 10.8). Operační síť je pro lepší představu uvedena na *obr. 10.5.*

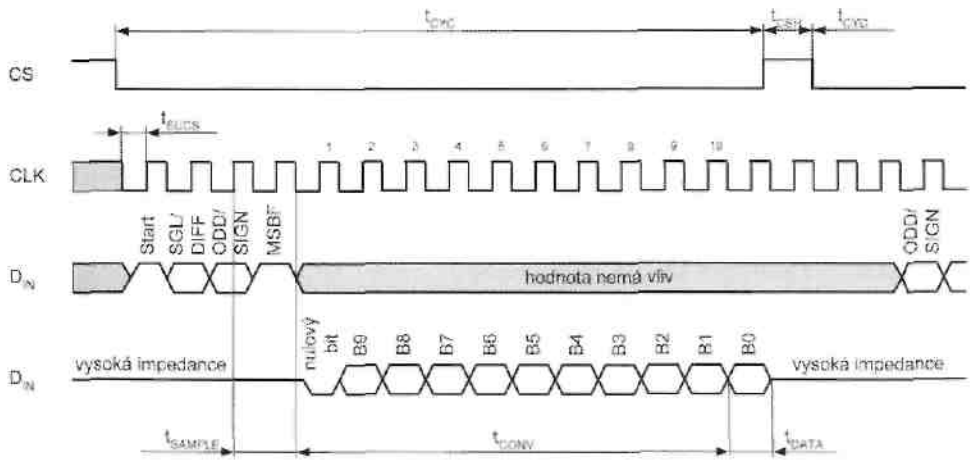
Rozdíl napětí (u_d) přivedený mezi vstupy A/D_{x+} a A/D_x_je nejdříve převeden proti zemi v diferenčním zesilovači se ziskem 1. Dále se tato hodnota dělí dvěma a odečítá od poloviny U_{REF2} = 5 V. Je-li u_d v rozsahu -5 V až +5 V, dostaneme na

výstupu druhého operačního zesilovače napětí v rozsahu 0 až 5 V. Referenční napětí hodnoty 5 V bylo nutné z hlediska vyvážení odporů operační sítě (je-li každý ze vstupů operačního zesilovače zakončen stejným odporem, výrazně se zmenší vliv proudové nesymetrie vstupů operačního zesilovače). Diody typu **1N4148** spolu s rezistory hodnoty 1 kíž chrání vstupy A/D převodníku proti přepětí.



Obr. 10.5 Operační síť A/D převodníku

Stručné zobrazení komunikačního protokolu obvodu **MCP3002** je uvedeno na *obr. 10.6.*

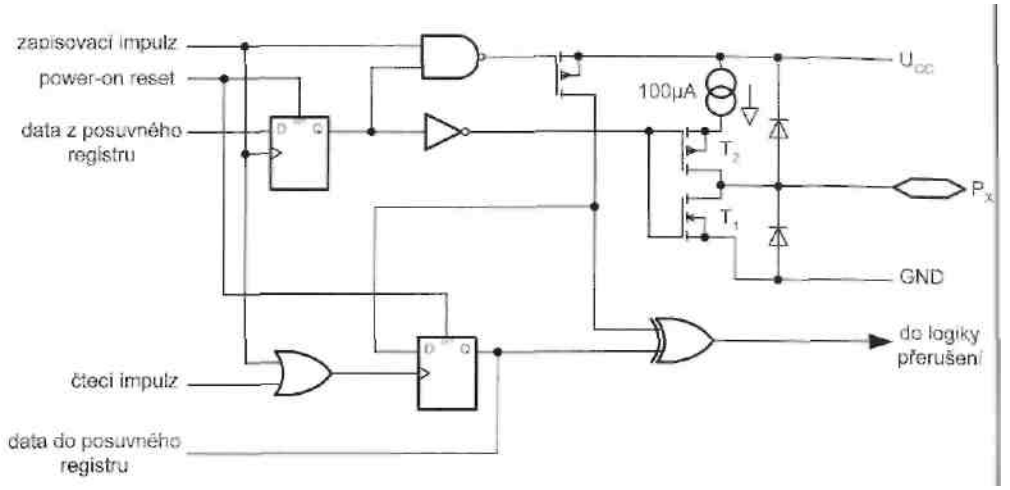


Obr. 10.6 Komunikace s MCP3002

Digitální vstupy/výstupy

Poslední jednotkou jsou digitální vstupy/výstupy realizované obvodem **PCF8574A** (musel být použit další integrovaný obvod, protože mikrokontrolér neměl dostatečný počet volných vývodů).

Výhodou obvodu **PCF8574A** je skutečnost, že jeho vývody jsou kvaziobousměrné (podobně jako u mikrokontroléru **AT89C2051).** Každý vývod lze tedy konfigurovat jako vstup nebo výstup. Viz *obr. 10.7.*



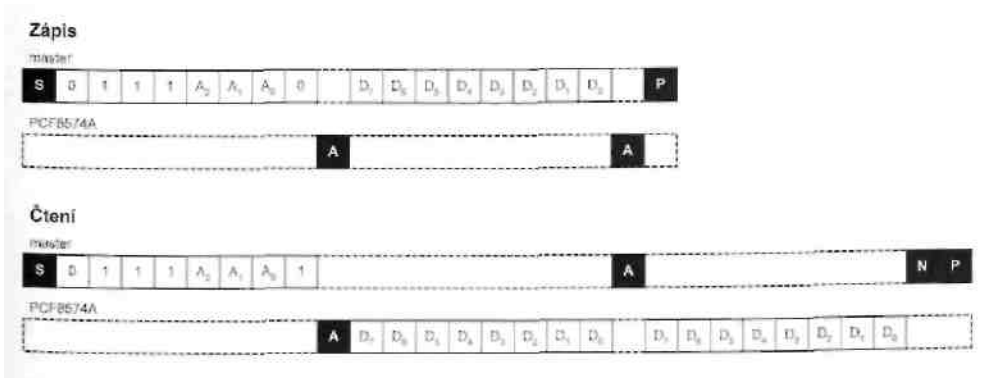
Obr. 10.7 Výklad funkce kvaziobousměrného vývodu

Je-li uložena hodnota log. 1 (tranzistor T, rozpojen a tranzistor T_2 sepnut), bude k vývodu P_x připojen zdroj proudu 100 uA. Chování pak odpovídá vytažení linky směrem k log. 1. Odvedeme-li tento poměrně malý proud například spínačem zapojeným mezi vývod P_x na zem, bude zpětně čtena hodnota log. 0. Pokud bude spínač rozpojen, přečte se zpětně hodnota log. 1.

Pokud je uložena hodnota log. 0 (tranzistor T, sepnut a tranzistor T_2 rozpojen), je vývod P_x poměrně silně stažen k log. 0.

Závěr: Při zápisu log. 1 se vývod P_x chová jako vstup (nebo jej lze chápat jako výstup uvedený do log. 1). Při zápisu log. 0 se jedná o výstup v log. 0.

Podrobný popis komunikace je uveden v [15], originální dokumentace je také umístěna na doprovodném CD-ROM. Stručně je komunikace naznačena formou *obr. 10.8.*



Obr. 10.8 Komunikace s obvodem PCF8574A

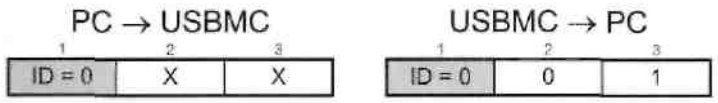
Pozor na mutace PCF8574A a PCF8574. Tyto obvody mají stejnou funkci, liší se však pevnou částí adresy, kterou do nich vložil výrobce. Adresu však lze volit pomocí souboru USBMC.INI. Sekce [DIO] obsahuje řetězcový klíč Verze. Možné hodnoty jsou PCF8574A nebo PCF8574.

10.2 ŘÍDICÍ PŘÍKAZY

Komunikace mezi počítačem a měřicí kartou probíhá pomocí šesti příkazů. Ve všech případech se jedná o 3bajtové skupiny dat obsahující jednak identifikační číslo příkazu (vždy první bajt skupiny) a dále data pro jeho provedení.

Zjištění verze firmwaru (ID = 0)

Vzhledem k tomu, že předpokládám další rozšiřování konstrukce, rozhodl jsem se použít první řídicí příkaz k tomu, aby se dala zjistit verze připojené desky. Tato verze nemusí být závislá jen na vlastním hardwaru, ale také na firmwaru (obslužném programu uloženém v mikrokontroléru).



Obr. 10.9 Příkaz ID = 0

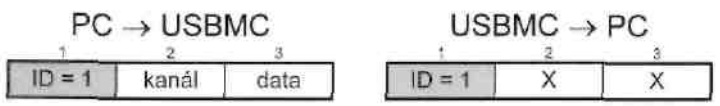
Počítač odešle 3 bajty. První musí mít hodnotu 0, která definuje požadovaný příkaz. Další dva bajty mohou být libovolné hodnoty (nejsou použity).

Mikrokontrolér vrací zpět 3 bajty. První je opět 0. Další dva bajty mají význam nižšího a vyššího čísla verze firmwaru. V našem případě bude vráceno 0 a 1, což značí verzi 1.0.

Ovládací program podle verze firmwaru dokáže zjistit, které příkazy může provádět.

Zápis do D/A převodníků (ID = 1)

Tento příkaz slouží pro nastavení dat pro D/A převodníky. Měřicí karta obsahuje dva D/A převodníky a tak je součástí příkazu i číslo kanálu.

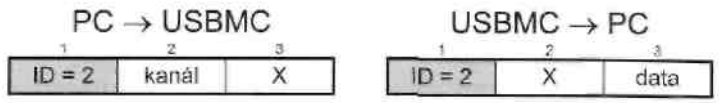


Obr. 10.10 Příkaz ID = 1

Počítač tedy nejdříve odešle číslo příkazu (1), poté číslo kanálu (0 nebo 1) a nakonec data. Mikrokontrolér vyšle zpět číslo příkazu, obsah dalších dvou bajtů není definován.

Čtení stavu D/A převodníků (ID = 2)

Zpětné čtení stavu D/A převodníků je užitečné pro případ, že chceme po spuštění ovládacího programu zjistit hodnoty posledně uložených údajů v D/A převodnících.



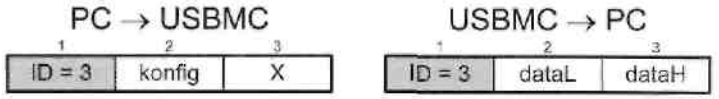
Obr. 10.11 Příkaz ID = 2

Počítač nejdříve odešle číslo příkazu (2) následované číslem kanálu (0 nebo 1). Poslední bajt není použit a může tedy mít libovolnou hodnotu.

Mikrokontrolér vrací zpět číslo příkazu, obsah druhého bajtu není definován a třetí bajt obsahuje data přečtená z určeného D/A převodníku.

Čtení dat z A/D převodníku (ID = 3)

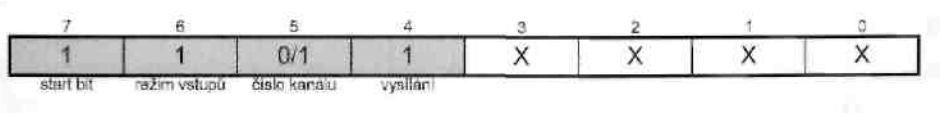
Tento příkaz slouží pro spuštění A/D převodu a příjmu takto získaných dat.



Obr. 10.12 Příkaz ID = 3

Nejdříve se vyšle číslo příkazu (3), druhý bajt obsahuje konfigurační bity dle *obr. 10.13*. Pomocí konfiguračních bitů se určuje kanál, který se snímá a režim vstupů. Poslední bajt má libovolnou hodnotu, protože není používán.

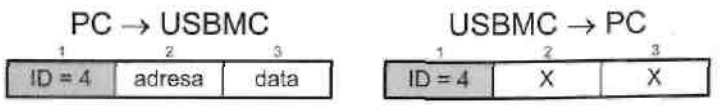
Mikrokontrolér vrací zpět číslo příkazu, které je následováno dolním a horním bajtem převodu. Převod je totiž 10bitový. Obslužná rutina popsaná dále však pracuje s přesností pouze 8 bitů (uvažují se tedy dolní 2 bity horního bajtu a horních 6 bitů dolního bajtu).



Obr. 10.13 Konfigurační bity vysílané do A/D převodníku

Zápis na digitální vstupy/výstupy (ID = 4)

Tímto příkazem zapíšeme data na obvod digitálních vstupů/výstupů.

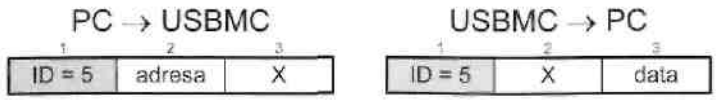


Obr. 10.14** Příkaz **ID = 4

Nejdříve je vyslán kód příkazu (4) následovaný zapisovací adresou obvodu (0x70 pro PCF8574A nebo 0x40 pro PCF8574) a datový údaj. Mikrokontrolér opět vrací kód příkazu, další dva bajty nemají definovaný obsah.

Čtení z digitálních vstupů/výstupů (ID = 5)

Tento příkaz sejme stav vývodů obvodu digitálních vstupů/vstupů.



Obr. 10.15 Příkaz ID = 5

Nejdříve se vyšle kód příkazu (5) následovaný čtecí adresou **(0x71** pro **PCF8574A** nebo **0x41** pro **PCF8574).** Poslední bajt je zase „jen do počtu".

Mikrokontrolér vrací kód příkazu, nedefinovaný bajt a nakonec hodnotu přečtenou z obvodu.

10.3 OVLÁDACÍ ROZHRANÍ

Po popisu řídicích příkazů budeme pokračovat výkladem funkcí zapsaných v jazyce C++, které dovolují ovládat měřicí kartu z počítače.

Komunikační rychlost jsem stanovil na 62 **500** Bd (dobře se nastaví ze strany mikrokontroléru a je to dostatečně vysoká hodnota).

Tyto funkce byly pro snazší ovládání zahrnuty jako metody do třídy **TUSBMC.** Pro praktické použití je tedy nutno vytvořit instanci třídy **TUSBMC.** Parametr kon-struktoru určuje, zda používáme obvod **PCF8574A** (výchozí, pak není třeba nic zadávat) nebo **PCF8574.**

Metody

- **_fastcall TUSBMC(unsigned char DIOAddr = tdioPCF8574A)** - kon-struktor; parametr **DIOAddr** určuje adresu pro práci s obvodem DIO, možnosti jsou buď **tdioPCF8574A** (což je výchozí volba a odpovídá obvodu **PCF8574A)** nebo **tdioPCF8574** (odpovídá obvodu **PCF8574),**
- **_fastcall ~TUSBMC()** - destruktor,
- **unsigned short ___ fastcall GetVersionQ** - zjistí verzi firmwaru; horní bajt výsledku obsahuje vyšší číslo verze, dolní bajt pak nižší číslo verze (například 0x0100 odpovídá verzi 1.0),
- **void __fastcall WriteDAC(TDAChannel Channel, unsigned char Data)** - pošle data určená hodnotou **Data** na D/A převodník určený **Channel** (tdacDAC0=0, tdacDAC1=1),
- **unsigned char ___ fastcall ReadDAC(TDAChannel Channel)** - vrací data dříve uložená do D/A převodníku určeného parametrem **Channel** (tdac-DAC0=0, tdacDAC1=1),
- **unsigned char _____fastcall ReadADC(TADChannel Channel)** - vrací výsledek převodu kanálu A/D převodníku určeného parametrem **Channel** (tadcADC0=0, tadcADC1=1),
- **void __fastcall WriteDIO(unsigned char Data)** - pošle **Data** na obvod di-gitálního vstupu/výstupu,
- **unsigned char ___ fastcall ReadDIO()** - vrací stav vývodů obvodu digitálního vstupu/výstupu,

- **double___fastcall ByteToVolt(unsigned char Data)** - převádí 8bitový údaj **Data** v rozsahu 0 až 255 na hodnotu napětí v rozsahu -5 V až +5 V,
- **unsigned char ___fastcall VoltToByte(double Volt)** - převádí hodnotu na pětí **Volt** v rozsahu -5 V až +5 V na 8bítový údaj v rozsahu 0 až 255.

Implementace

Pro lepší představu a vlastní experimenty jsou vypsány zdrojové soubory třídy **TUSBMC.**

USBMCINF.H:

```
#ifndef   USBMCINFH
#define   USBMCINFH
//---------------------------------------------------------------------------
#include    "ftd2xx.h"
//---------------------------------------------------------------------------
enum   TDAChannel{tdacDAC0=0,tdacDACl=l} ;
enum   TADChannel{tadcADC0=0,tadcADCl=l};
enum   TDIOAddr{tdioPCF8574=0x40,tdioPCF8574A=0x70};
//---------------------------------------------------------------------------
class   TUSBMC:public   TObject{
private:
FT_HANDLE   ft; unsigned   char
DIOAddr; public:
    _fastcall   TUSBMC(
        unsigned   char   DIOAddr=tdioPCF8574A) ;
    __fastcall   -TUSBMC( ) ;
        unsigned   short _______fastcall   GetVersion() ;
    void   ___fastcall   WriteDAC(
        TDAChannel   Channel,   unsigned   char   Data);
    unsigned   char   ____fastcall   ReadDAC(
        TDAChannel   Channel);
    unsigned   char   _____fastcall   ReadADC(
        TADChannel   Channel);
    void   ___fastcall   WriteDIO(
        unsigned   char   Data);
    unsigned   char   ____fastcall   ReadDIO() ;
    double   ___ fastcall   ByteToVolt(
        unsigned   char   Data);
    unsigned   char   ____fastcall   VoltToByte(
        double   Volt); };
```

```
//---------------------------------------------------------------------------
#endif
```

USBMCINF.CPP:

```
#include <vcl.h> #pragma
hdrstop #include
"USBMCINF.h"

// ---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
_fastcall TUSBMC::TUSBMC(unsigned char DIOAddr)
    :DIOAddr(DIOAddr)
{
    //otevře zařízení:
    DWORD ftStatus=FT_OpenEx(
        "Measure Card",FT_OPEN_BY_DESCRIPTION, &ft);

    if(ftStatus!=FT_OK)
        throw Exception("Měřicí karta není připojena!");

    //ustálení:
    Sleep(IOO);

    //přenosová rychlost 62500 Bd:
    FT_SetBaudRate(ft,62500) ;

    //timeouty 100 ms:
    FT_SetTimeouts(ft,100,100);
}
//---------------------------------------------------------------------------
__fastcall TUSBMC::-TUSBMC()
{
    //zavře zařízení:
    FT_Close(ft) ;
}
//---------------------------------------------------------------------------
unsigned short _____fastcall TUSBMC::GetVersion()
{
    DWORD d;
    BYTE ID=0;
    BYTE RID;
    BYTE DataL,DataH;
```



```
    //pošle příkaz, ID=0:
    FT_Write(ft,&ID,1,&d); //pošle data
    (nedefinováno): FT_Write(ft,&DataL,1,&d);
    //pošle data (nedefinováno):
    FT_Write(ft,&DataH,1,&d);

    //čte ID:
    FT_Read(ft,&RID,1,&d);
    if(ID!=RID)
        throw Exception("Chyba komunikace!"); //čtení
    bajtů verze: FT_Read(ft,SDataL, 1,&d) ;
    FT_Read(ft,&DataH,1,&d);

    return (DataH<<8)|DataL;
}
//---------------------------------------------------------------------------
void ___fastcall TUSBMC::WriteDAC(
    TDAChannel Channel, unsigned char Data)
{
    DWORD d;
    BYTE ID=1;
    BYTE RID;

    //pošle příkaz, ID=1:
    FT_Write(ft,&ID,1, &d); //pošle číslo
    kanálu: FT_Write(ft,&Channel,1,&d) ;
    //pošle data: FT_Write(ft,SData,1,&d);

    //čte ID:
    FT_Read(ft,&RID,1,&d)?
    if(ID!=RID)
        throw Exception("Chyba komunikace!");
    //čtení dalších bajtů (nepoužito):
    FT_Read(ft,&RID,1,&d);
    FT_Read(ft,&RID, 1, &d) ;
}
//---------------------------------------------------------------------------
unsigned char_____ fastcall TUSBMC::ReadDAC(
    TDAChanne1 Channe1)
```

```
(
    DWORD  d;
    unsigned      char     ID=2;
    unsigned      char     RID;
    unsigned      char     Data;

    //pošle   příkaz,    ID=2:
    FT_Write(ft,&ID,1,&d)  ; //pošle   číslo
    kanálu: FT_Write(ft,schannel,1,&d);
    //pošle   data    (nedefinováno):
    FT_Write(ft,SData, 1, &d) ;

    //čte   ID:
    FT_Read(ft,&RID, 1, &d) ;
    if(ID!=RID)
        throw   Exception("Chyba   komunikace!"); //čtení
    dalšího   bajtu    (nepoužit): FT_Read(ft,&RID,1, &d) ;
    //čtení   dat: FT_Read(ft,&Data, 1, &d) ;

    return   Data;
}
// ---------------------------------------------------------------------------------------
unsigned  char _____  fastcall  TUSBMC::ReadADC(
    TADChannel   Channel)
{
    DWORD  d;
    unsigned  char   ID=3;
    unsigned   char   RID;
    unsigned   char   Data,DataL,DataH;

    switch(Channel){
        //konfigurační   bity:
        čase   tadcADCO:Data=Oxd0;break;//CH0   proti   zemi
        čase   tadcADCI:Data=0xf0;break;//CHI   proti   zemi
    }

    //pošle  příkaz,    ID=3: FT_Write(ft,&ID,
    1,&d)  ; //pošle   konfig.   bity:
    FT_Write(ft,SData,1,&d); //pošle   data
    (nedefinováno):
```



```
    FT_Write(ft,&Data,1,&d) ;

    //čte   ID:
    FT_Read(ft,&RID,1,&d);
    if(ID!=RID)
        throw   Exception("Chyba   komunikace!"); //čtení
    dolního  bajtu: FT_Read(ft,SDataL, 1,&d) ; //čtení   horního
    bajtu: FT_Read(ft,&DataH, 1, &d);

    d=(DataH«8)  | DataL;
    return   d>>2;
}
// ---------------------------------------------------------------------------------------
void ____ fastcall   TUSBMC:  :WriteDIO (
    unsigned   char  Data)
{
    DWORD  d;
    BYTE   ID=4;
    BYTE   RID;
    unsigned   char   Addr=DIOAddr+0;

    //pošle  příkaz,    ID=4:
    FT_Write(ft,&ID,1,&d); //pošle
    adresu: FT_Write(ft,&Addr,1,&d);
    //pošle   data: FT_Write(ft,&Data,
    1,&d) ;

    //čte   ID:
    FT_Read(ft,&RID,1, &d);
    if(ID!=RID)
        throw   Exception("Chyba   komunikace!");
    // čtení   dalších  bajtů    (nepoužito):
    FT_Read(ft,&RID,1, &d)  ;
    FT_Read(ft,&RID,1,&d);
}
// ---------------------------------------------------------------------------------------
unsigned   char _____ fastcall   TUSBMC::ReadD10()
{
    DWORD  d;
    unsigned   char   ID=5;
    unsigned   char   RID;
```

```
    unsigned  char  Data;
    unsigned  char  Addr=DIOAddr+I;

    //pošle  příkaž,   ID=5:
    FT_Write(ft,&ID,1,&d); //pošle  adresu:
    FT_Write(ft,&Addr, 1, &d) ; //pošle  data
    (nedefinováno): FT_Write(ft,&Data,1,&d);

    //čte  ID:
    FT_Read(ft,&RID, 1, &d) ;
    if(ID!=RID)
        throw  Exception("Chyba  komunikace!"); //čteni  dalšího
    bajtu  (nepoužito): FT_Read(ft,&RID, 1, &d) ; //čteni  dat:
    FT_Read(ft,SData, 1, &d) ;

    return  Data;
}
//---------------------------------------------------------------------------
double ____ fastcall  TUSBMC:  :ByteToVolt(
    unsigned  char  Data)
{
    return   (Data-128)/128.0*5;
}
//---------------------------------------------------------------------------
unsigned  char  ____fastcall  TUSBMC::VoltToByte(
    double  Volt)
{
    return  Volt/5*128+128; }
```

10.4 FIRMWARE - PROGRAM PRO MIKROKONTROLÉR AT90S2313

Pojmem firmware většinou označujeme program vložený do mikrokontroléru. Pro jednoznačnost budu toto označení používat také.

Je jasné, že firmware musí zajistit reakci na příkazy vysílané z počítače. Jak jsou příkazy implementovány není důležité. Ve své realizaci jsem se snažil o co nejúspornější zápisy.

Firmware je verze 1.0, předpokládám pozdější rozšiřování (viz kapitolu 10.9). Vlastní realizaci lze rozložit do 4 částí:

- inicializace uvedená od návěští **RESET,**
- obsluha sériového kanálu (návěští RX a TX),
- rutiny pro práci se sběrnicí I^2C **(SCLSET, SCLCLR, SDASET, SDACLR, I2CSTA, I2CSTP, I2CPSL, I2CCTI),**
- rutiny pro obsluhu jednotlivých příkazů **(GVER, WDAC, RDAC, RADC, WDIO, RDIO)**

Přestože toto není kniha o programování mikrokontroléru, zastavíme se u několika nejzajímavějších programových zápisů.

Jednou z věcí, kterou jsem potřeboval vyřešit jednoduše, byl „rozskok" podle požadovaného příkazu. Pokud máme větší počet řídicích příkazů, jistě není vhodné testovat několikanásobným porovnáváním číslo příkazu a podle toho provést příslušný skok. Mnohem jednodušší je nějakým způsobem adresu skoku vypočítat. Za tímto účelem jsem vybudoval tabulku adres, kde jsou uloženy relativní skoky na jednotlivé obslužné rutiny příkazů. Tato tabulka je označena návěštím **IDTAB.** Každá položka tabulky je vlastně jedna instrukce **RJMP.** Všechny položky této tabulky jsou tedy stejně dlouhé a zabírají jedno slovo. Takže adresa příslušného skoku se získá přičtením prvního bajtu příkazu (ID) k adrese tabulky **(IDTAB).** Vlastní skok pak realizuje instrukce **ICALL** (absolutní volání podprogramu podle obsahu registru Z; volá se jako podprogram proto, aby se po provedení akce vrátil zpět). Tento zápis naleznete v rutině RX.

Další komplikace byly spojeny s používáním několika obvodů vybavených sběrnicí I^2C. Problém byl v tom, že jsem z konstrukčního důvodu musel, nebo i chtěl (jednak se zjednodušil plošný spoj a dále to dává možnost pracovat s obvody současně), řídit každý z obvodů nezávisle. Každý I^2C obvod má tedy zvláštní vodiče **SDA** a **SCL.** Ve [3] jsem řízení I^2C sběrnice popsal s tím, že bylo uvažováno řízení obvodů na sdílené sběrnici. V nové realizaci tedy bylo nutno provést jisté úpravy. Protože instrukční soubor mikrokontroléru AVR není vybaven instrukcemi pro nulování nebo nastavování určitých bitů vstupně/vstupních registrů, musel jsem „šáhnout" k instrukcím **AND** a **OR.** Ty totiž umožňují nulování a nastavování bitů. Jedinou podmínkou bylo pro každý obvod nadefinovat speciální masky obsahující jedničku jen na tom bitu, který chceme ovládat. Tyto masky byly zavedeny jako symboly **SCLDAO, SDADAO, SCLDA1, SDADA1, SCLDIO** a **SDADIO.** Konkrétní řešení problému nastavení a nulování linek **SCL** a **SDA** je zřejmé ze zápisu rutin **SCLSET, SCLCLR, SDASET** a **SDACLR.** Každé použití těchto rutin však vyžaduje předchozí nastavení speciálních registrů **I2CSCL, I2CSCI, I2CSDA** a **I2CSDI** maskami pro bity **SCL** a **SDA** případně jejich negacemi **(I2CSCI** je negací **I2CSCL, I2CSDI** je negací **I2CSDA).**

USBMC.ASM:

```
            .NOLIST
            .INCLUDE   "2313def.inc"
            .LIST
```

```
;masky portu pro I2C obvody:
            .EQU  I2CP0=P0RTB       ;je použit
            .EQU  I2CDD=DDRB        ;port B
            .EQU  I2CPI=PINB

;masky bitů pro I2C obvody:
            .EQU  SCLDAO=0b00100000 ;maska bitů pro
            .EQU  SDADAO=0b00010000 ; IO9 (D/A0)
            .EQU  SCLDAI=0b00000100 ;maska bitů pro
            .EQU  SDADAI=Ob00000010 ; IO10 (D/A-:)
            .EQU  SCLDIO=0b01000000 ;maska bitů pro
            .EQU  SDADIO=0bI0000000 ; IO, (DIO)

;čisla bitů pro MCP3002:
            .EQU  CS=2              ;PD2
            .EQU  CLK=4             ;PD4
            .EQU  D=0               ;PB0
            .DSEG                   /datový segment
            .DEF  REG=R16           ;pracovni registr
            .DEF  CNT=R17           ;počitadlo
            .ORG  19                ;adresa 19=R19
BUFFER:     .BYTE(3)                ;3bajtový buffer
                                    ;v adresnim prostoru /registrů
                                    R19 až R21
            .EQU  BUFKON=BUFFER+2   /konec bufferu
            .DEF  ID=R19            /ID přikážu
            .DEF  LBYTE=R20         /dolni bajt dat
            .DEF  HBYTE=R21         ;horní bajt dat
            .DEF  I2CSCL=R22        /pracovní
            .DEF  I2CSCI=R2 3       /registry
            .DEF  I2CSDA=R24        /pro podproru
            .DEF  I2CSDI=R2 5       /podprogramů
            .DEF  I2CACC=R2 6       /řídicích
            .DEF  I2CCT1=R2 7       /obvody
            .DEF  I2CCT2=R28        /se sběrnicí I2C
            .DEF  BUFPOS=R30        /ukazovátko bufferu

            .CSEG                   /kódový segment
            RJMP  RESET
            .ORG  URXCaddr          /vektor UART RX
            RJMP  RX
            .ORG  UTXCaddr          /vektor UART TX
            RJMP  TX
```



```
/inicializace:
RESET:      LDI   REG,RAMEND
            OUT   SPL,REG           ;nastaveni SP
            SBI   DDRD,CS           ;CS a CLK
            SBI   DDRD,CLK          ;jsou výstupy
            CLR   ZH                ;BUFPOS(Z)
                                      ;Ukazuje
            LDI   BUFPOS,BUFFER     ;na začátek bufferu
            LDI   REG,10            ;konfigurace UART:
            OUT   UBRR,REG          ; 62500 Bd pro 10 MHz
            LDI   REG,ObIOOIIOOO
            OUT   UCR,REG           ;povol UART RX/TX
            SEI                     ;I=1
            RJMP  PC                /čeká na přerušení

/obsluha UART RX:
RX:         IN  REG,UDR             ;čti data
            ST  Z+,REG              ;ulož do bufferu
            CPI  BUFPOS,BUFKON+I    ;konec?
            BRNĚ  RXKON             ;ne->skoč na RXKON
            /přijato vše:
            LDI  R30,LOW(IDTAB)     ;Z obsahuje
            LDI  R31,HIGH(IDTAB)    ;adresu IDTAB
            ADD  R30,ID             ;přičti ID
            CLR  REG                ;k obsahu
            ADC  R31,REG            ;registru Z
            ICALL                   ;proveď akci
            CLR  ZH                 ;přestav Z
            LDI  REG,ObOIOIIOOO     ;povol vysílání
            OUT  UCR,REG
            LDI  BUFPOS,BUFFER      ;ukazovátko
                                      /na začátek
            LD  REG,Z+              ;čti první bajt
            OUT  UDR,REG            ;a pošli jej
RXKON:      RETI

/obsluha UART TX:
TX:         LD  REG,Z+              ; čti bajt
            OUT  UDR,REG            ;a pošli jej
            CPI  BUFPOS,BUFKON+1    /konec?
            BRNE  TXKON             /ne->skoč na TXKON
            /posláno vše:
```

```
                LDI  REG,ObIOOIIOOO     ;povol přijem
                OUT  UCR,REG
                LDI  BUFPOS,BUFFER      /ukazovátko
TXKON:          RÉTI                    ;na začátek

;přikaz 0:  zjištěni verze firmwaru:
GVER:           LDI  HBYTE,1            ;1
                LDI  LBYTE,0            ;0
                RET                     ;tedy 1.0

;přikaz 1:  zápis dat do D/A:
WDAC:           CPI  LBYTE,1            ;test čísla
                BREQ WDAC1              ;D/A
                /nastaveni pro D/A0:
WDACO:          LDI  I2CSCL, SCLDAO     ;maska pro SCL
                LDI  I2CSCI,SCLDAO      ; a jej i negace
                COM  I2CSCI             ;v I2CSCL, I2CSCI
                LDI  I2CSDA,SDADAO      ;maska pro SDA
                LDI  I2CSDI,SDADAO      ;a jej i negace
                COM  I2CSDI             ;v I2CSDA, I2CSDI
                RJMP WDACS
                ;nastaveni pro D/Ax:
WDAC1:          LDI  I2CSCL,SCLDA1      ;maska pro SCL
                LDI  I2CSCI,SCLDA1      ;a jej i negace
                COM  I2CSCI             ;v I2CSCL, I2CSCI
                LDI  I2CSDA,SDADA1      /maska pro SDA
                LDI  I2CSDI,SDADA1      ;a jej i negace
                COM  I2CSDI             ;v I2CSDA, I2CSDI
                ;vlastni akce:
WDACS:          RCALL I2CSTA            ;start
                LDI  I2CACC,0bl0010000
                RCALL I2CPSL            ;adresa
                LDI  I2CACC/0
                RCALL I2CPSL            ;přikaz 0
                MOV  I2CACC,HBYTE
                RCALL I2CPSL            ;data z HBYTE
                RCALL I2CSTP            ;stop
                RET

;přikaž 2:  čteni dat z D/A:
RDAC:           CPI  LBYTE,1            ;test čisla
                BREQ RDAC1              ;D/A
```



```
                /nastaveni pro D/A^:
RDACO:          LDI  I2CSCL,SCLDAO      /maska pro SCL
                LDI  I2CSCI,SCLDAO      ;a jej i negace
                COM  I2CSCI             ;v I2CSCL, I2CSCI
                LDI  I2CSDA, SDADAO     /maska pro SDA
                LDI  I2CSDI,SDADAO      /a jej i negace
                COM  I2CSDI             ;v I2CSDA, I2CSDI
                RJMP RDACS
                /nastaveni pro D/A^.
RDAC1:          LDI  I2CSCL,SCLDA1      /maska pro SCL
                LDI  I2CSCI,SCLDA1      ;a jej i negace
                COM  I2CSCI             ;v I2CSCL, I2CSCI
                LDI  I2CSDA,SDADA1      /maska pro SDA
                LDI  I2CSDI,SDADA1      /a její negace
                COM  I2CSDI             ;v I2CSDA, I2CSDI
                /vlastní akce:
RDACS:          RCALL I2CSTA            /start
                LDI  I2CACC,Obl0010000
                RCALL I2CPSL            /adresa
                LDI  I2CACC,0
                RCALL I2CPSL            /příkaz 0
                RCALL I2CSTA            /znovu start
                LDI  I2CACC,0bl0010001
                RCALL I2CPSL            /adresa pro čtení
                RCALL I2CCTI            /čtení
                MOV  HBYTE,I2CACC       /uložení dat
                RCALL I2CSTP            /stop
                RET

/příkaz 3:  čtení dat z A/D:
RADC:           SBI  DDRB,D             ;D jako DIN
                CBI  PORTD,CS           ;CS=0 (aktivace)
                LDI  CNT,4              /počet konf. bitů
                MOV  REG,LBYTE          /konfigurace
                /pošle konf. bity:
VYSSM:          CBI  PORTD,CLK          /CLK=0
                ROL  REG                /bit do C
                BRCS VYS1
VYSO:           CBI  PORTB,D            /pro C=0
                RJMP VYS
VYS1:           SBI  PORTB,D            /pro C=l
VYS:            NOP                     /ustálení
```

```
          SBI   PORTD,CLK        ;CLK=1
          DEC   CNT              ;sniž počitadlo
          BRNĚ  VYSSM            ;konec?
          ;přijímá bity převodu:
          CLR   LBYTE            ;nulování
          CLR   HBYTE            ;přijimacích reg.
          LDI   CNT,11           ;11 bitů
          CBI   DDRB,D           ;D jako DOUT
PRIJSM:   CBI   PORTD,CLK        ;CLK=0
          NOP                    /ustálení
          SEC                    ;C=1
          SBIS  PINB,D           ;pro DIN=1 přeskoč
          CLC                    ;C=0
          SBI   PORTD,CLK        ;CLK=1
          ROL   LBYTE            ;vsuň C
          ROL   HBYTE            ;do výsledku
          DEC   CNT              ;sniž počítadlo
          BRNĚ  PRIJSM           ;konec?
          SBI   PORTD,CS         ;CS=1 (deaktivace)
          RET

/příkaz 4: zápis dat na DIO:
WDIO:     LDI   I2CSCL,SCLDI0    ;maska pro SCL
          LDI   I2CSCI,SCLDI0    ;a její negace
          COM   I2CSCI           ;v I2CSCL, I2CSCI
          LDI   I2CSDA,SDADI0    ;maska pro SDA
          LDI   I2CSDI,SDADI0    ;a její negace
          COM   I2CSDI           ;v I2CSDA, I2CSDI
          ;vlastni akce:
          RCALL I2CSTA           ;start
          MOV   I2CACC,LBYTE     ;čtení adresy
          RCALL I2CPSL           ;adresa
          MOV   I2CACC,HBYTE
          RCALL I2CPSL           ;data z HBYTE
          RCALL I2CSTP           ; stop
          RET

;příkaz 5: čteni dat z DIO:
RDIO:     LDI   I2CSCL,SCLDIO    ; maska pro SCL
          LDI   I2CSCI,SCLDIO    ;a její negace
          COM   I2CSCI           ;v I2CSCL, I2CSCI
          LDI   I2CSDA,SDADIO    ;maska pro SDA
```



```
          LDI   I2CSDI,SDADIO    ;a její negace
          COM   I2CSDI           ;v I2CSDA, I2CSDI
          ;vlastni akce:
          RCALL I2CSTA           ;start
          MOV   I2CACC,LBYTE     ;čtení adresy
          RCALL I2CPSL           /adresa pro čtení
          RCALL I2CCTI           ;čteni
          MOV   HBYTE,I2CACC     /uložení dat
          RCALL I2CSTP           /stop
          RET
/tabulka skoků:
IDTAB:    RJMP  GVER             ; ID=0
          RJMP  WDAC             /ID=1
          RJMP  RDAC             ;ID=2
          RJMP  RADC             ;ID=3
          RJMP  WDIO             /ID=4
          RJMP  RDIO             /ID=5

/nastaví bit určený maskou I2CSCL (ISCL=1):
SCLSET:   IN    I2CCT1,I2CP0     ; čte stav portu
          OR    I2CCT1,I2CSCL    /přidá bit z I2CSCL
          OUT   I2CP0,I2CCT1     /vystaví na port
          IN    I2CCT1,I2CDD     ; čte směr
          AND   I2CCT1,I2CSCI    /odebere bit I2CSCL
          OUT   I2CDD,I2CCT1     /zapíše směr
          RET

/nuluje bit určený maskou I2CSCL (SCL=0):
SCLCLR:   IN    I2CCT1,I2CP0     ;čte stav portu
          AND   I2CCT1,I2CSCI    /odebere bit I2CSCL
          OUT   I2CP0,I2CCT1     /vystaví na port
          IN    I2CCT1,I2CDD     ;čte směr
          OR    I2CCT1,I2CSCL    /přidá bit z I2CSCL
          OUT   I2CDD,I2CCT1     /zapíše směr
          RET

/nastaví bit určený maskou I2CSDA (SDA=1):
SDASET:   IN    I2CCT1,I2CP0     ;čte stav portu
          OR    I2CCT1,I2CSDA    /přidá bit z I2CSDA
          OUT   I2CP0,I2CCT1     /vystaví na port
          IN    I2CCT1,I2CDD     ;čte směr
          AND   I2CCT1,I2CSDI    /odebere bit I2CSDA
```

```
                OUT   I2CDD,I2CCT1          ;zapiše směr
                RET

;nuluje bit určený maskou I2CSDA (SDA=0):
SDACLR:         IN    I2CCT1,I2CP0          ;čte stav portu
                AND   I2CCT1,I2CSDI         /odebere bit I2CSDA
                OUT   I2CPO,I2CCTI          /vystaví na port
                IN    I2CCT1,I2CDD          ;čte směr
                OR    I2CCT1,I2CSDA         /přidá bit z I2CSDA
                OUT   I2CDD,I2CCT1          /zapíše směr
                RET

/vyšle START:
I2CSTA:         RCALL SDASET                ;SDA=1
                RCALL SCLSET                /SCL=1
                RCALL I2CCEK                /ustálení
                RCALL SDACLR                /SCL=1,  SDA=0
                RCALL I2CCEK                /ustálení
                RCALL SCLCLR                /SCL=0,  SDA=0
                RCALL I2CCEK                /ustálení
                RET

/vyšle STOP:
I2CSTP:         RCALL I2CCEK                /ustálení
                RCALL SDACLR                /SDA=0
                RCALL I2CCEK                /ustálení
                RCALL SCLSET                ;SCL=1,SDA=0
                RCALL I2CCEK                /ustálení
                RCALL SDASET                ;SCL=1,5DA=1
                RET

/pošle obsah I2CACC
/vstup: I2CACC-hodnota pro vyslání
/výstup: C-stav ACK
/poznámka: I2CACC se po volání nezmění
I2CPSL:         LDI   I2CCT2,8              ;8 bitů
I2CPSM:         ROL   I2CACC                /horní bit do C
                BRCC  I2CPS0
                RCALL SDASET                ; SDA=C pro OI
                RJMP  I2CPS1
I2CPS0:         RCALL SDACLR                /SDA=C pro C=0
I2CPS1:         RCALL I2CCEK                /ustálení
                RCALL SCLSET                ;SCL=1
```



```
                RCALL I2CCEK                /ustálení
                RCALL SCLCLR                ;SCL=0
                DEC   I2CCT2                /sniž počítadlo
                BRNĚ  I2CPSM                /konec?
                RCALL I2CCEK                /ustálení
                RCALL SDASET                /příprava na ACK
                RCALL I2CCEK                /ustálení
                RCALL SCLSET                ;SCL=1
                RCALL I2CCEK                /ustálení
                SEC                         /C=I
                IN    I2CCT1,I2CPI          /čte SDA
                AND   I2CCT1,I2CSDA         /(testuje ACK)
                BRNĚ  I2CPS2
                CLC                         ;C=Q
I2CPS2:         RCALL SCLCLR                /SCL=0
                ROL   I2CACC                /obnov I2CACC
                RET

/čeká, aby se zajistilo ustálení linek
I2CCEK:         LDI   I2CCT1,50             /asi 15 ys
I2CCCK:         DEC   I2CCT1
                BRNĚ  I2CCCK
                RET

/čte do I2CACC
/vstup: není
/výstup: I2CACC-přečtená hodnota
I2CCTI:         RCALL SDASET                /SDA plovoucí
                LDI   I2CCT2,8              ;8 bitů
I2CCTS:         RCALL I2CCEK                /ustálení
                RCALL SCLSET                ;SCL=1
                RCALL I2CCEK                /ustálení
                SEC                         ;01
                IN    I2CCT1,I2CPI          /čte SDA
                AND   I2CCT1,I2CSDA         /(čte přijatý bit)
                BRNĚ  I2CCT2
                CLC                         ;O0
I2CCT2:         ROL   I2CACC                /vsuň do I2CACC
                RCALL SCLCLR                /SCL=0
                RCALL I2CCEK                /ustálení
                DEC   I2CCT2                /sniž počítadlo
                BRNĚ  I2CCTS                /konec?
                RCALL SDASET                ;SDA=1
```

```
RCALL   I2CCEK          /ustáleni
RCALL   SCLSET          ;SCL=1
RCALL   I2CCEK          /ustáleni
RCALL   SCLCLR          ;SCL=0
RET
```

10.5 KOMPLEXNÍ TESTOVACÍ APLIKACE

Testovací aplikace vznikla především z požadavku snadného oživení karty. Dále nám tato aplikace předvede, co vlastně karta umí.

Aplikace obsahuje tři skupiny odpovídající digitálním vstupům/výstupům, 2 kanálům A/D převodníku a 2 kanálům D/A převodníku. Ovládání je ryze intuitivní.

Zajímavostí je náhrada checkboxů pomocí jakýchsi spínačů, komponentu jsem pojmenovat LogBox. Nejedná se však o klasickou komponentu (není ji třeba instalovat), ale vlastně jen o zdrojové soubory LOGBOX.H/.CPP obsahující definice a deklarace třídy TLogBox odvozené z TCheckBox. Tím je dáno, že komponenty musí být generovány programově, má to však výhodu v možnosti uložení jejich ukazatelů do pole (pak jde logboxy procházet v cyklu).

USBMCF.H:

```
#ifndef   USBMCFH
#define   USBMCFH
//---------------------------------------------------------------------------
#include      <vcl\Classes.hpp>
#include      <vcl\Controls.hpp>
#include      <vcl\StdCtrls.hpp>
finclude      <vcl\Forms.hpp>
frinclude     <vcl\ExtCtrls.hpp>
#include   <Graphics.hpp>
//---------------------------------------------------------------------------
#include   "USBMCINF.h"
#include   "LogBox.h"
#include   <Menus.hpp>
#include   <AppEvnts.hpp>
//---------------------------------------------------------------------------
class   TTestForm   :   public   TForm
{
_ published:                    //   IDE-managed  Components
    TTimer   *Casovac;
    TMainMenu   *MainMenu;
    TMenuItem   *MenuSoubor;
```



```
    TMenuItem   *MenuSouborKonec;
    TMenuItem   *MenuNapoveda;
    TMenuItem   *MenuNapovedaOaplikaci;
    TPanel   *HlPanel;
    TGroupBox   *DIOPanel;
    TGroupBox   *ADCPanel;
    TLabel   *lbADC0;
    TLabel   *lbADCl;
    TLabel   *Labell;
    TLabel   *Label2;
    TBevel   *Bevell;
    TBevel   *Bevel2;
    TGroupBox   *DACPanel;
    TLabel   *lbDA0;
    TLabel   *lbDAl;
    TBevel   *Bevel3;
    TBevel   *Bevel4;
    TLabel   *Label3;
    TLabel   *Label4;
    TScrollBar   *sbDA0;
    TScrollBar   *sbDAl;
    void   ___ fastcall   sbDAScroll(TObject   *Sender,
        TScrollCode   ScrollCode,   int   SScrollPos);
    void   _fastcall   cbDIOClick(TObject   *Sender);
    void   ___ fastcall   OnCasovac(TObject   *Sender);
    void   __ fastcall   MenuSouborKonecClick(
        TObject   *Sender);
    void   __ fastcall   MenuNapovedaOaplikaciClick(
        TObject   *Sender); private:  //
User   declarations
    //ukazatel na ovládači rozhráni:
    TUSBMC  *USBMC;
    //pole ukazatelů na logboxy:
    TLogBox  *log[8];
    //verze firmwaru:
    unsigned   short  Verze;
    //typ DIO obvodu:
    TDIOAddr  Dio;
public:                                 //  User   declarations
     _ fastcall   TTestForm(TComponent*   Owner);
     _fastcall   -TTestForm();
```

```
};
//---------------------------------------------------------------------------
extern TTestForm *TestForm;
//---------------------------------------------------------------------------
#endif
```

USBMCF.CPP:

```
#include <vcl\vcl.h>
#include <inifiles.hpp>
#include <stdio.h>
#pragma hdrstop
#include "USBMCF.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TTestForm *TestForm;
//---------------------------------------------------------------------------
__fastcall TTestForm::TTestForm(TComponent* Owner)
    : TForm(Owner)
{
  TIniFile *ini=new TIniFile(GetCurrentDir()
     +"\\USBMC.INI");
  //zjisti posledni umistěni okna: Left=ini-
  >ReadInteger("DESKTOP","X",0); Top=ini-
  >ReadInteger("DESKTOP","Y", 0) ; //zjisti typ DIO
  obvodu: Dio=
    (ini->ReadString("DIO","Verze","PCF8574A")
    =="PCF8574A")
    ?tdioPCF8574A
    :tdioPCF8574;
  delete ini;

  //vytvoři instanci ovládaciho rozhráni: USBMC=new
  TUSBMC(Dio);

  //čteni verze firmware: Verze=USBMC-
  >GetVersion();

  //čteni stavu obou D/A převodniků: sbDAO-
  >Position=USBMC->ReadDAC(tdacDACO); sbDAI-
  >Position=USBMC->ReadDAC(tdacDACI);

  //aktualizace scrollbarů pro D/A převodniky: int x=0;
```



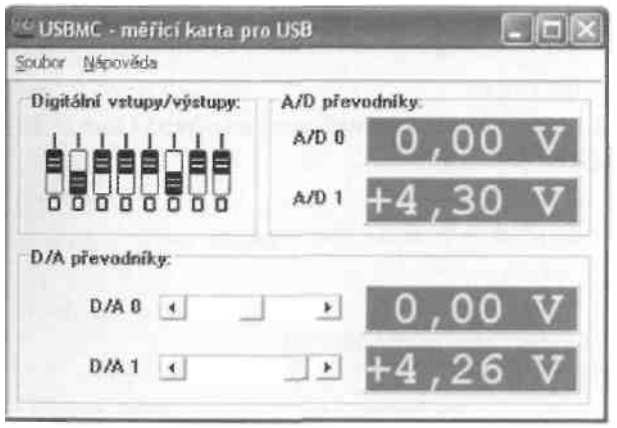
```
  sbDAScroll(NULL,scEndScroll,x);

  //vytvořeni logboxů: for(int
  i=0;i<8;i++){
    log[i]=new TLogBox(this);
    log[i]->Parent=DIOPanel;
    log[i]->Left=16+i*16;
    log[i]->Top=26;
    log[i]->OnClick=cbDIOClick;
  }
  //úprava výšky okna:
  ClientWidth=HlPanel->Width;
  ClientHeight=HlPanel->Height;
}
//---------------------------------------------------------------------------
__fastcall TTestForm::~TTestForm()
{
  //rušeni logboxů: for(int
  i=0;i<8;i++) delete log [i];

  //rušeni ovládaciho rozhráni:
  delete USBMC;
  //uloženi posledni umistěni okna:
  TIniFile *ini=new TIniFile(GetCurrentDir()
     +"\\USBMC.INI");
  ini->WriteInteger("DESKTOP","X",Left);
  ini->WriteInteger("DESKTOP","Y",Top);
  delete ini;
}
//---------------------------------------------------------------------------
//obsluha obou scrollbarů pro D/A převodniky:
void__ __fastcall TTestForm::sbDAScroll(TObject *Sender,
  TScrollCode ScrollCode, int &ScrollPos) {
  //zobrazeni hodnoty D/A převodníku čislo 0: lbDAO-
  >Caption=
  FormatFloat("+0.00 V;-0.00 V; 0.00 V", USBMC-
  >ByteToVolt(sbDAO->Position)); //zobrazení hodnoty D/A
  převodníku číslo 1: lbDAI->Caption=
    FormatFloat("+0.00 V;-0.00 V; 0.00 V",
```

```
            USBMC->ByteToVolt(sbDAI->Position));
    //získání ukazatele na scrollbar:
    TScrollBar   *sb=dynamic_cast<TScrollBar*>(Sender);

    //vyslání dat:
    if(ScrollCode==scEndScroll&&sb){
        USBMC->WriteDAC((TDAChannel)sb->Tag,
        sb->Position); }
}
// ---------------------------------------------------------------------------
//reakce na změnu logboxů uživatelem:
void  _fastcall  TTestForm::cbDIOClick(TObject  *Sender)
{
    unsigned char  Data=0;

    //sestavení bajtu: for(int
    i=0;i<8;i++)
        Data= (Data«I)  I log [i] ->Checked; //vyslání
    dat: USBMC->WriteDIO(Data);
}
// ---------------------------------------------------------------------------
//periodické čtení stavu DIO a A/D převodníků přes časovač:
void  ___fastcall  TTestForm::OnCasovac(TObject  *Sender)
{
    Časovač->Enabled=false;

    //čtení a zobrazení stavu A/D převodníku číslo 0: lbADCO-
    >Caption=
        FormatFloat("+0.00 V;-0.00 V; 0.00 V",
            USBMC->ByteToVolt(USBMC->ReadADC(tadcADCO))); //čtení a
    zobrazení stavu A/D převodníku číslo 1: lbADCI->Caption=
        FormatFloat("+0.00 V;-0.00 V; 0.00 V",
            USBMC->ByteToVolt(USBMC->ReadADC(tadcADCI)));

    //čtení stavu DIO:
    unsigned char Data=USBMC->ReadDIO() ;
    unsigned char Maska=0x80;
    //odpojení logboxů od události DIOClick:
    for(int i=0;i<8;i++)
```



```
        log[i]->OnClick=NULL; //dekódování jednotlivých
    bitů DIO: for (int i=0;i<8;i++) {
        log[i]->Checked=Data&Maska;
        Maska=Maska»I ; }
    //nové napojení logboxů na událost DIOClick: for (int i =
    0;i<8;i++)
        log[i]->OnClick=cbDIOClick;

    Casovac->Enabled=true;
}
// ---------------------------------------------------------------------------
//zavření aplikace:
void ___ fastcall TTestForm::MenuSouborKonecClick(
    TObject *Sender)
{
    CloseO ;
}
// ---------------------------------------------------------------------------
//informace o aplikaci:
void ___ fastcall TTestForm::MenuNapovedaOaplikaciClick(
    TObject *Sender)
{
//řetězec obsahující verzi firmwaru: AnsiString
sVerze=
    "v "
    +AnsiString ( (Verze&OxFFOO) »8)
    +AnsiString(" . ")
    +AnsíString(Verze&0x00FF) ; //řetězec obsahující
typ DIO obvodu: AnsiString sDIO=
    (Dio==tdioPCF857 4A)
    ?"PCF8574A"
    :"PCF8574";

//zobrazí messagebox s informacemi:
MSGBOXPARAMS mbp;
mbp.cbSize=sizeof(mbp);
mbp.hwndOwner=Handle;
mbp.hInstance=HInstance;
char str[1024]="Měřicí karta pro USB\n\n"
```

```
        "Firmware:    %s\n"
        "DIO:    %s\n\n"
        "Autor:\tDavid   MatoušekAn"
        "XtVyšši   odborná   škola\n"
        "XtTolstého   16\n"
        "\tJIHLAVA\n"
        "\t586   01"; char
    text[1024] ;
    sprintf(text,str,sVerze.c_str(),sDIO.c_str()); mbp.lpszText=text;
    mbp.lpszCaption="USBMC"; mbp.dwStyle=MB_USERICON;
    mbp.lpszIcon="MAINICON";
//vlastni   zobrazeni:
MessageBoxIndirect(&mbp); }
```



Obr. 10.16 Aplikace v akci pod Windows XP

Důležitý je také inicializační soubor **USBMC.INI:**

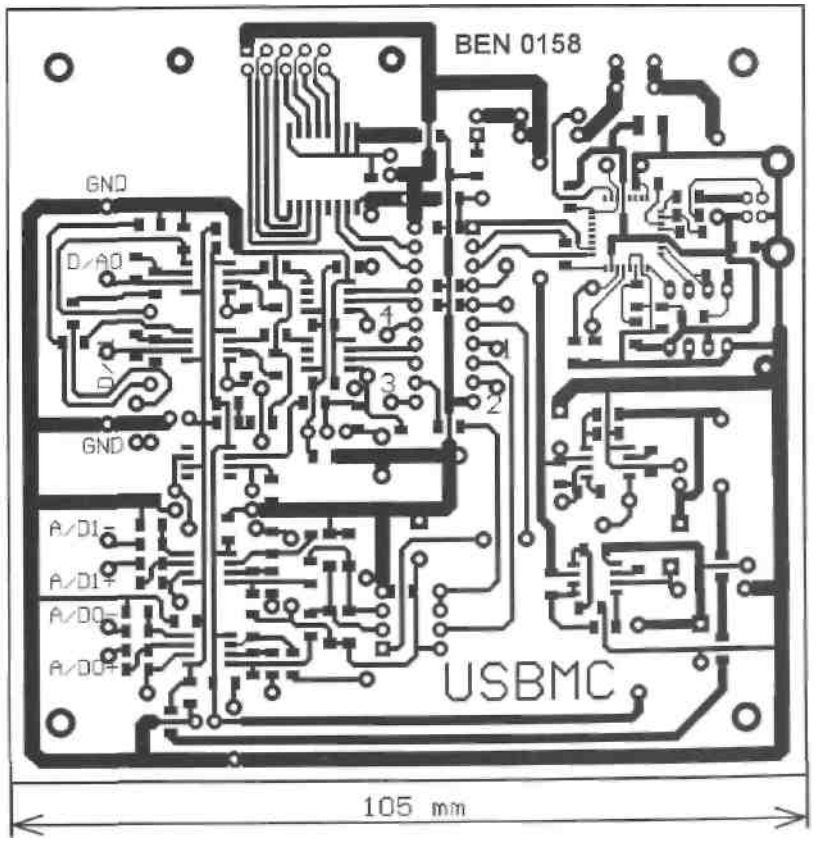
USBMC.INI:

```
[DESKTOP]              ;posledni   umístěni   okna
X=351
Y=185

[DIO]                  /parametry   obvodu   DIO
Verze=PCF8574A         /indikuje   obvod   PCF8574A
```


Obr. 10.17 Dialog o aplikaci informuje o verzi firmware a použitém obvodu DIO

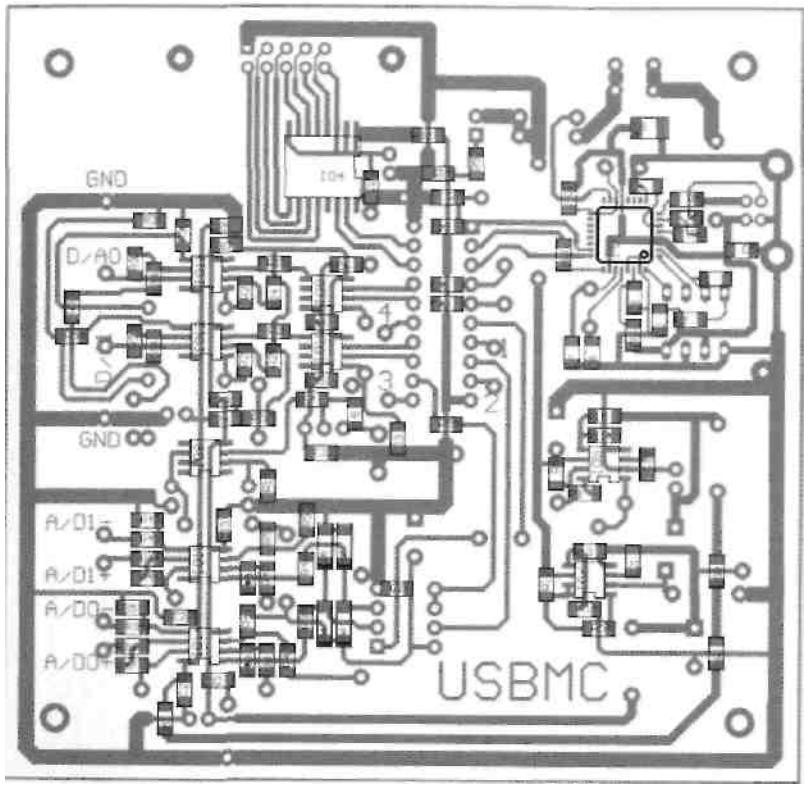
10.6 PLOŠNÉ SPOJE

Výkres plošných spojů pro přípravek **USBMC** je uveden na *obr. 10.18*. V zájmu snížení rozměrů bylo použito značné množství SMD.



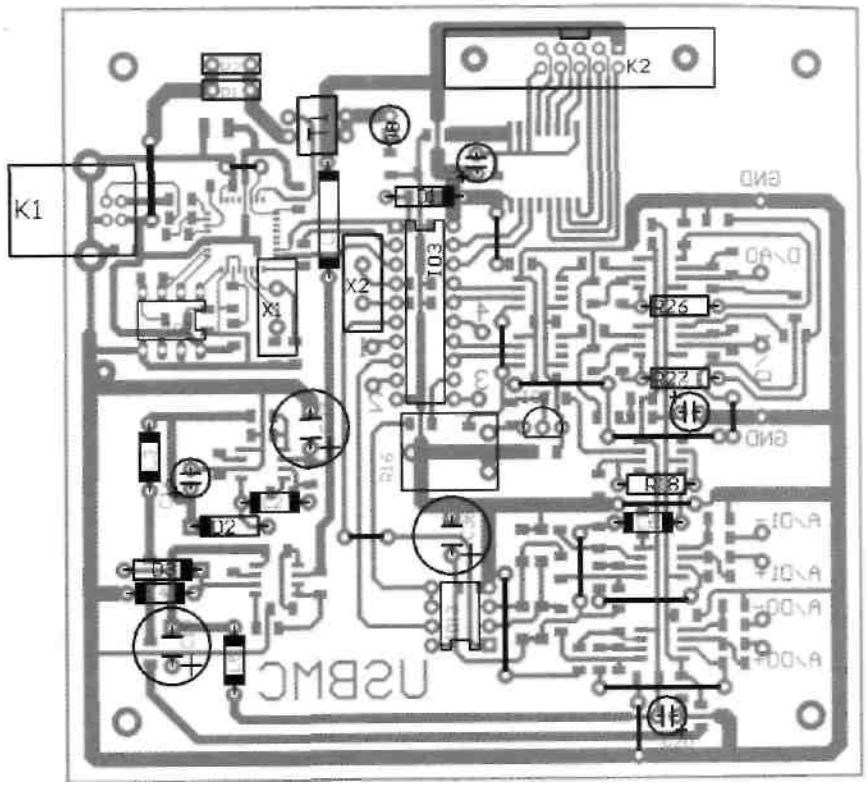
Obr. 10.18 Výkres desky plošných spojů USBMC (BEN 0158)

Osazovací plánek ze strany spojů (pro SMD) je uveden na *obr. 10.19.*



Obr. 10.19 Osazovací plánek ze strany spojů

Osazovací plánek ze strany součástek je uveden na *obr. 10.20.*



Obr. 10.20 Osazovací plánek ze strany součástek

Seznam součástek pro USBMC (cena asi 700 Kč):

C1 ažC3	CK+100NX7R	3 ks
C4	CK+33N X7R	1 ks
C_5	CK+10NX7R	1 ks
C_6	CTS6M8/10VB	1 ks
$C_{7)}$ C_8, C_{11f} C_{12}	CK+27P NPO	4 ks
C_{13}ažC_{l5}	CK+100NX7R	3 ks
C10, C_{16}, C_{17}, C_{20}	E100M/10V	4 ks
C_{18}, C_{21}	CK+1N5X7R	2 ks
C_{19}, C_{36}, C_{37}	E470M/16V	3 ks
C_{22}ažC_{30}	CK+100NX7R	9 ks
C_{32}ažC_{35}	CK+100NX7R	4 ks
D_1	1N4148	1 ks
D_2, D_3	1N5819	2 ks
D_4ažD_7	1N4148SMD	4 ks
D_8	LED 5MM 200 MCD	1 ks
IO_1	FT232BM	1 ks
IO_2	93LC46B	1 ks
IO_3	90S2313-10PI	1 ks
IO_4	PCF8574AT SMD	1 ks
IO_5, IO_6	MC34063AD	2 ks
IO_7	LM317L	1 ks
IO_8		

	TL071 SMD	1 ks
IO_9, IO_{10}	TC1320EOA	2 ks
IOn, IO_{12}, IO_{14}, IO_{15}	TL072 SMD	4 ks
$I0_{13}$	MCP3002-I/P	1 ks
K,	USB1X90B PCB	1 ks
K_2	PSL10	1 ks
UW.UL.	TL.100MH	4 ks
L_3) L_5	TL.33MH	2 ks
p_1(P_2	RXE025 (nebo 1 ks RXE050)	2 ks

R_1	RR+2K2 SMD	1 ks
R_2, R_{10}, R_{14}	RR+10K SMD	3 ks
R_3	RR+470R SMD	1 ks
R_4, R_5	RR+27R SMD	2 ks
Re.Rn.Ris	RR+1K5SMD	3 ks
R_7	RR+100KSMD	1 ks
R_8, R_{12}	RR+1RSMD	2 ks
R_9	RR+180RSMD	1 ks
R_{15}, R_{39} R_{48}, R_{52}	RR+1K SMD	4 ks
R_{16}	PT10VK002.5(trimr2k5)	1 ks
RIT. R20. R21	RR+100K SMD	3 ks
R_{18}	RR 200K	1 ks
R_{19}, R^ až R_{25}	RR+200K SMD	5 ks
R_{26}. R_{27}	RR100K	2 ks
R_{28}. R_{29}	RR+100KSMD	2 ks
R_{30} až R_{33}	RR+200K SMD	4 ks
R_{34} až R_{37}	RR+100K SMD	4 ks
R_{38}. R41. R47. R_5i	RR+ 200K SMD	4 ks
R_{40}, R_{42} až R_{46}, R_{49}, R_{50}	RR+100K SMD	8 ks
R_{53} až R_{63}	RR+OR SMD	11 ks
T,	IRFD9120	1 ks
X,	QM 6.000 MHZ	1 ks
X_2	QM 10.000 MHZ	1 ks

10.7 OŽIVENÍ

Přes komplikovanost konstrukce (velký počet použitých SMD) je oživení relativně jednoduché. Po rutinní kontrole správnosti osazení (především polarity diod a elektrolytických kondenzátorů) můžeme ožívovací akce provést takto:

1. Nejdříve neosazujte obvody IO_3 (mikrokontrolér AT90S2313) a IO_{13} (A/D převodník MCP3002), doporučuji použít patice. Jezdec trimru R_{16} otočte šrou-bováčkem tak, aby měl tento trimr co nejmenší odpor.
2. Naprogramujte paměť IO_2 a vložte ji do karty. Připojte kartu USB kabelem k počítači. Mělo by dojít k úspěšné enumeraci (vyžádá se nebo se vyhledá ovladač).
3. Nyní zkontrolujte napájecí napětí pro operační zesilovače. Například lze měřit napětí vývodu č. 8 obvodu IO_{14} proti zemi (cca +9,5 V) a napětí vývodu č. 4 obvodu IO_{14} proti zemi (cca -9,5 V). Připojte voltmetr mezi prostřední vývod obvodu IO_7 a zem. Změnou polohy jezdce trimru R_{16} nastavte napětí na 2,5 V (přesně). Zkontrolujte napětí mezi vývody 4 a 8 patice pro obvod IO_{13}. Zde musí být dvojnásobek předešlé hodnoty (tedy 5 V). Číslicové obvody (kolektor tranzistoru T, proti zemi) by měly mít napětí minimálně 4,5 V.

. 4. Odpojte kartu od počítače. Vložte do patice pro obvod IO_3 naprogramovaný mikrokontrolér AT90S2313 (souborem USBMC.HEX), dále osaďte patici obvodu IO_{13} převodníkem MCP3002.

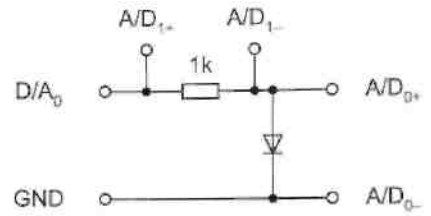
5. Připojte kartu k počítači. Spusťte program USBMC.EXE. Pokud je spuštění bezproblémové, můžete pokračovat testem D/A převodníků (voltmetrem změříte generované napětí).
6. Funkci A/D převodníků lze nejsnáze ověřit tak, že na vstup A/D^ připojíme výstup D/A_0 a na vstup A/D_{0+} připojíme výstup D/A,. Pokud na D/A_0 nastavíme napětí 0 V, musí hodnota nastavená na D/A, přesně odpovídat napětí, které změří A/D_0. Podobně lze otestovat funkci druhého kanálu.
7. Před testem funkce DIO doporučuji ověřit variantu obvodu, který byl zapájen (PCF8574nebo PCF8574A) v souboru **USBMC.INI.** Poté již stačí měnit stav logboxů a například voltmetrem sledovat změny na konektoru K_2. Vzhledem k tomu, že stav vývodů je periodicky čten zpět do programu, musí se po každé změně pozice logboxu stav zachovat. Pokud uživatel mění stav logboxů, ale ty stále zobrazují samé 1, značí to neúspěšnou komunikaci s obvodem PCF8574A (resp. PCF8574). Závada může být ve špatné komunikační adrese.

10.8 MĚŘENÍ V-A CHARAKTERISTIKY DIODY - PŘÍKLAD KONKRÉTNÍ APLIKACE

Abych ukázal praktické použití měřicí karty, ukážeme si jej na příkladu měření V-A charakteristiky diody.

Měřená dioda bude připojena dle *obr. 10.21.* Rezistor hodnoty 1 kQ pracuje jako převodník proudu procházejícího diodou na napětí, které lze snímat. Je jasné, že jeden kanál A/D převodníku měří napětí diody. Druhý kanál pak snímá napětí na pomocném rezistoru. Potože je hodnota odporu právě 1 kO, je změřené napětí vlastně proud tekoucí diodou v miliampérech.

Program musí zajistit generování napětí na výstupu prvního kanálu D/A převodníku a snímání napětí z obou kanálů A/D převodníku. Údaje A/D převodníku pak představují napětí a proud diody.



Obr. 10.21 Schéma zapojení měřicí úlohy

Snímací program je relativně jednoduchý a je vystavěn na podobném základu jako samotná testovací aplikace. Vlastní snímání dat probíhá v metodě Mereni. Uživatel může pomocí nabídky Počet bodů nastavit počet měřicích bodů (tedy vlastně přesnost měření) na 16, 64 nebo 256. Výsledek je zřejmý z *obr. 10.22.*

DIODAF.H:

```
#ifndef DiodaFH
#define DiodaFH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------------
#include "USBMCINF.h"
#include "Mapper.h"
#include <ExtCtrls.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
//maximální počet měřicích kroků: const int
POCET=256;

//aktuální hodnota měřícího kroku:
int KR0K=16;
//---------------------------------------------------------------------------
class TDiodaForm : public TForm
{
_ published:    // IDE-managed Components
    TMainMenu *MainMenu;
    TMenuItem *MenuSoubor;
    TMenuItem *MenuSouborKonec;
    TMenuItem *MenuPocetBodu;
    TMenuItem *MenuPocetBodul6;
    TMenuItem *MenuPocetBodu64;
    TMenuItem *MenuPocetBodu256;
    void ___ fastcall FormActivate(TObject *Sender);
    void ___ fastcall FormPaint(TObject *Sender);
    void __ fastcall FormResize(TObject *Sender);
    void __ fastcall MenuSouborKonecClick(TObject *Sender);
    void ___ fastcall MenuPocetBodu256Click(TObject *Sender);
private: // User declarations

    //ukazatel na instanci rozhraní: TUSBMC
    *USBMC;

    //pole změřených hodnot:
```



```
    double U[POČET],I[POČET];

    //pole bodů pro zobrazení: TPoint
    body[POČET];

//příznak pro OnPaint: bool Připraven;
public:  // User declarations
     _fastcall TDiodaForm(TComponent* Owner);
     _fastcall -TDiodaForm(); void
    fastcall Mereni();
};
//---------------------------------------------------------------------------
extern PACKAGE TDiodaForm *DiodaForm;
// ---------------------------------------------------------------------------
#endif
```

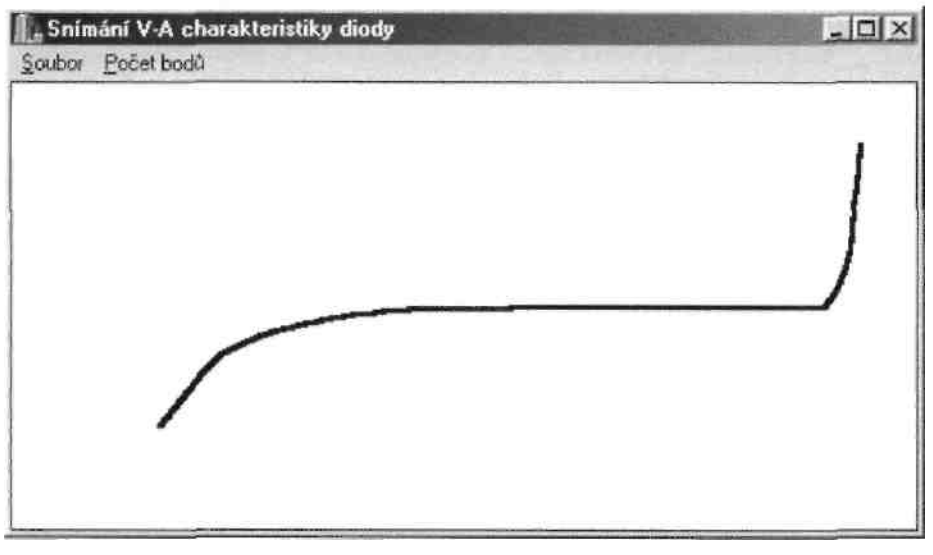
DIODAF.CPP:

```
finclude <vcl.h>
#pragma hdrstop
#include "DiodaF.h"
// ---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TDiodaForm *DiodaForm;
// ---------------------------------------------------------------------------
   fastcall TDiodaForm::TDiodaForm(TComponent* Owner) :
            TForm(Owner)
{
    //vytvoří instanci ovládacího rozhraní:
    USBMC=new TUSBMC();

    Pripraven=false;
    Canvas->Brush->Style=bsClear;
}
// ---------------------------------------------------------------------------
 _fastcall TDiodaForm::-TDiodaForm()
{
    delete USBMC;
}
// ---------------------------------------------------------------------------
//sejmutí dat do polí U a I:
void     fastcall TDiodaForm::Mereni()
```

```
{
    for(int    i=0;i<POCET/KROK;i++)
    {
        Application->ProcessMessages() ;
        USBMC->WriteDAC(tdacDACO,i*KROK);
        U[i]=USBMC->ByteToVolt(USBMC->ReadADC(tadcADCO));
        I[i]=USBMC->ByteToVolt(USBMC->ReadADC(tadcADCI));
    }
}
//---------------------------------------------------------------------------
//obnovováni   obrazu:
void   ___fastcall   TDiodaForm::FormPaint(TObject   *Sender)
{
    //nakreslí   obrysový   obdélník: Canvas->Pen-
    >Width=l; Canvas->Rectangle(ClientRect);

    //kreslení   V-A   charakteristiky: Canvas->Pen-
    >Width=2;

    //namapování   souřadnic   na: //osa   x    (U):
    -3   až   +1   V, //osa   y    (I):   -5   až   +5
    mA: TMapper   *m=new  TMapper(Canvas,
        -3,+1,-5,+5,
        100,100,ClientWidth,ClientHeight); //transformace
    bodů: for(int   k=0;k<POCET/KROK;k++){
        body[k]  .x=m->TransfX(U[k]);
        body[k]  .y=m->TransfY(I[k] )  ; }

    //zobrazení:
    if (Připraven) {
        Canvas->MoveTo(body[0].x,body[0].y)/
        for(int   k=l;k<P0CET/KR0K;k+=3)
            //prokládá   Bézierovou   křivkou:
            Canvas->PolyBezierTo(&body[k],2);
    }
    delete(m);
}
// ---------------------------------------------------------------------------
```



```
//reaguje   na   změnu   velikosti   okna:
void   ____fastcall   TDiodaForm:  :FormResize(TObject   *Sender)
{
    //překreslení   obrázku: Invalidate();
}
// ---------------------------------------------------------------------------
//aktivace   formuláře-provede   první   odměr:
void    __fastcall   TDiodaForm::FormActivate(TObject   *Sender)
{
    //přesýpací   hodiny:
    Screen->Cursor=crHourGlass;
    //měření:
    Mereni();
    //vrátí   výchozí   kurzor:
    Screen->Cursor^crDefault ;
    //povolí   překreslení:
    Pripraven=true ;
}
//---------------------------------------------------------------------------
void    __fastcall   TDiodaForm::MenuSouborKonecClick(
    TObject   *Sender)
{
    CloseO ;
}
//---------------------------------------------------------------------------
//reaguje   na   změnu   počtu   měřicích   kroků:
void   ___fastcall   TDiodaForm::MenuPocetBodu256Click(
    TObject   *Sender)
{
    TMenuItem   *mi=dynamic_cast<TMenuItem*>(Sender);
    if(mi){
        mi->Checked=true;
        KROK=POCET/mi->Tag;
        FormActivate(this) ;
    Invalidate(); } }
```



Obr. 10.22 Aplikace v akci (snímání na 16 bodů pro Zenerovu diodu 3V6)

POZNÁMKA: Pro proložení sejmutých bodů křivkou je použita metoda TCanvas::PolyBezierTo. Mapování souřadnic okna na rozsah nasnímaných hodnot zajišťuje speciální třída TMapper, která byla podrobně popsána v [11 j.

10.9 NÁVRHY NA DALŠÍ VYLEPŠENÍ

Výhledově počítám s mírnými úpravami především na programové úrovni:

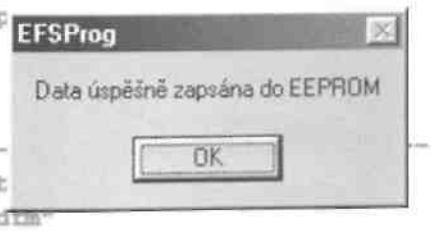
- Předně bych mohl upravit verzi firmware na 2.0. Tato verze by již zpřístupnila čítač/časovač 1. Takže by se pak daly používat vývody vstupu hodinového kmitočtu a jednotky Input Capture a Output Compare. Tato úprava by například dovolila generovat výstupní signál určitého kmitočtu či měřit kmitočet nebo periodu vstupního signálu. Též by šlo používat zabudovaný PWM generátor.
- Dále uvažuji o úpravách ovládacího rozhraní tak, aby jej šlo využívat například v Delphi nebo ve Visual Basicu. Má to své nesporné výhody hlavně s ohledem na skutečnost, že řada začátečníků volí spíše programování v pascalu nebo v basicu a ne v C++. Začátečníkům se sice programování v C++ zdá náročné, ale já si tento programovací jazyk oblíbil (mám s ním lepší zkušenosti, i když ovládám i další programovací jazyky).
- Zajímavou možností je také použít přerušovací vstup pro obsluhu nějaké kritické situace, kterou lze takto sledovat. Například by šlo propojit přerušovací výstup obvodu PCF8574A s přerušovacím vstupem mikrokontroléru a tak zajistit předností reakci na změnu stavu vývodů konfigurovaných jako vstupy.

11

EFSPROG – PROGRAMÁTOR KONFIGURAČNÍCH E²PROM

EFSPM.CPP:

```
//---------------------------------------------------------------------
#include <vcl.h>
#include <inifiles.hp
#include "Ftd2xx.h"
#pragma hdrstop
#include "EFSPM.h"
//---------------------
#pragma package(smart
#pragma resource "*.dfm"
TProgForm *ProgForm;
//---------------------------------------------------------------------
__fastcall TProgForm::TProgForm(
  TComponent* Owner)
  :TForm(Owner)
```



*Přestože firma **FTDI Chip** dodává ke svým obvodům i programátor konfiguračních E²PROM, narazil jsem při jeho použití na četné problémy. Předně ovládání se mi jevilo velmi komplikované.*

*Proto jsem vytvořil vlastní programátor nazvaný **EFSProg,** možná i vám výrazně pomůže při vývoji vlastních aplikací. Já si na něj nemohu stěžovat. Vyjma první konstrukce (FT232TST), kterou jsem naprogramoval programátorem od výrobce, jsem jej používal pro všechny přípravky.*

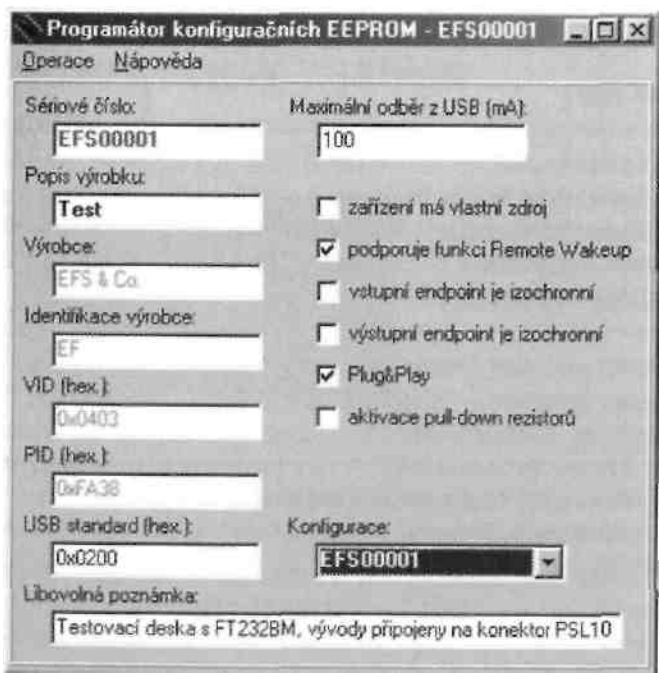
Mimo jiné naleznete v této kapitole výpis programátoru, jedná se tedy také o ukázku použití funkcí uvedených v kapitole 3.3.

11.1 UŽIVATELSKÁ PŘÍRUČKA PROGRAMU EFSPROG

Programátor má název EFSPROG.EXE (je umístěn na doprovodném CD-ROM v adresáři FTDI\EFSProg). Po spuštění jsou jednotlivá pole nastavena podle výchozích hodnot (jedná se pouze o náznaky, jak pole vyplnit). Tyto výchozí hodnoty lze měnit (popis naleznete v kapitole 11.1.3).

Příklad správného vyplnění ukazuje *obr. 11.1* (jedná se o konfiguraci přípravku FT232TST z kapitoly 5):

- Sériové číslo je libovolný řetězec maximálně 8 písmen. Všechny výrobky v této knize jsou registrovány předponou **EFS.** Proto jsem volil kombinaci této předpony a číselného označení (tak mohu vytvořit až 99 999 výrobků, které se vzájemně liší sériovým číslem),



Obr. 11.1 EFSProg

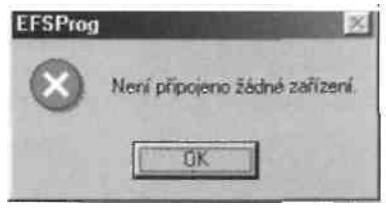
- **Popis výrobku** je libovolný řetězec, měl by být jedinečný (jako sériové číslo). Zařízení se totiž obvykle otevírají pomocí jména (což je popis výrobku),
- **Výrobce** je řetězec popisující výrobce, jedná se o libovolný text. **Identifikace výrobce** je dvouznakový řetězec (obvykle se jedná o zkratku sestavenou z prvních dvou znaků výrobce),
- **VID** a **PID** jsou hexadecimální čísla odpovídající položkám Vendor ID a Product ID. Zápis hexadecimálních hodnoty musí být uvozen znaky **Ox.** Pro výrobky EFS platí, že **VID = 0x0403 a PID = 0xFA38** (originální hodnoty pro FTDI Chip jsou VID = 0x0403 a PID = 0x0601),
- **USB Standard** je opět hexadecimální hodnota, představuje verzi USB pro kterou je zapsán deskriptor (0x0200 značí **USB** 2.0, 0x0101 značí **USB** 1.1),
- **Maximální odběr z USB** odpovídá maximálnímu předpokládanému odběru, který bude zařízení mít. Údaj je desítková hodnota braná v miliampérech,
- následují zaškrtávací pole, význam je zřejmý z titulků,
- **Libovolná poznámka** představuje volitelný text, který se ukládá spolu s ostatními údaji (není však programován do E²PROM). Tento údaj má přispět k lepší orientaci.

11.1.1 Stručný popis položek menu

- **Operace|Ulož data z formuláře** - uloží zadané hodnoty do inicializačního souboru **EFSProg.INI,** takto uložené hodnoty lze vybírat pomocí seznamu **Konfigurace** (viz *obr. **11.1),***
- **Operace|Zapiš do EEPROM** - zapíše aktuální hodnoty z formuláře do konfigurační E²PROM (aby se data uvážila, je nutno zařízení odpojit a poté znovu připojit),
- **Operace|Čti z EEPROM** - čte data z konfigurační E²PROM do formuláře (takto lze restaurovat data přímo ze zařízení),
- **Operace|Konec** - konec programu,
- **Nápověda|0 aplikaci** - klasické informace o aplikaci.

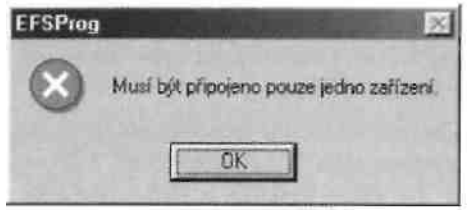
11.1.2 Možné chyby při manipulaci s programem

Před aktivací zápisu dat do E²PROM je nutné, aby bylo připojeno programované zařízení.



Obr. 11.2 Nelze programovat, pokud není připojeno žádné zařízení

Pro jednoznačnost trvá programátor na tom, aby v dané chvíli bylo připojeno jediné zařízení. Viz *obr. 11.2* a *obr. 11.3.*

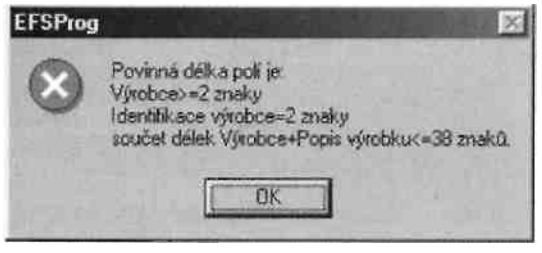


Obr. 11.3 Nelze programovat, pokud je připojeno více zařízení než jedno

Některá pole nejsou textová. Například hodnota VID a PID musí být zadána jako 16bitové číslo. Pokud se překročí rozsah 16 bitů, je to ohlášeno a akce (uložení/ zápis) nebude provedena, viz *obr. 11.4.*

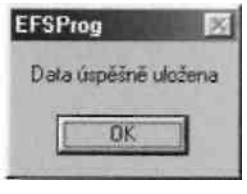
Obr. 11.4 VID a PID musí být 16bitová čísla

Kontrole také podléhají některá textová pole. Délky některých polí musí být přesné, u jiných polí je třeba dodržet minimální či maximální délku (nebo součet délek více polí). Viz *obr. 11.5.*



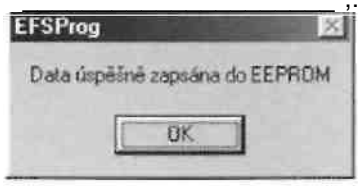
Obr. 11.5 Omezení délek některých polí

Pokud jsou data správně zadána, informuje program po aktivaci položky menu OperacejUlož data z formuláře o úspěšném zápisu do inicializačního souboru. Viz *obr. 11.6.*



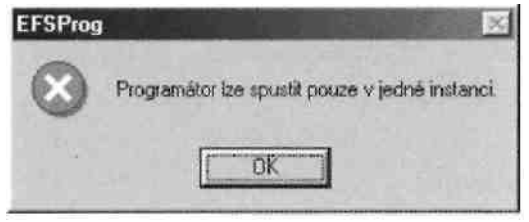
Obr. 11.6 Informace o úspěšném uložení dat do inicializačního souboru

Podobně, pokud je úspěšný zápis dat (vyvolaný položkou menu Operace|Zapiš do EEPROM) do konfigurační E^2PROM, je o tom uživatel rovněž informován. Viz *obr. 11.7.*



Obr. 11.7 Informace o úspěšném zápisu dat do konfigurační E^2PROM

V zájmu vyloučení nejednoznačností je programátor sestaven tak, aby jej bylo možno spustit pouze jednou. Pokud je programátor spuštěn a budeme jej chtít spustit znovu, dostaneme chybové hlášení dle *obr. 11.8.*



Obr. 11.8 EFSProg lze spustit pouze jednou!

11.1.3 Inicializační soubor EFSProg.INI

Programátor používá pro ukládání konfiguračních informací inicializační soubor **EFSProg.INI.** Tento soubor se musí nacházet ve stejném adresáři jako samotný programátor EFSPROG.EXE (musí se jednat o adresář, do kterého lze zapisovat). Každý výrobek je zapsán ve zvláštní sekci, označení sekce odpovídá sériovému číslu. Níže je uveden výpis pro sekci EFS00001 odpovídající *obr. 11.1.*

Pro pokročilé uživatele je také k dispozici sekce DEFAULT. Ta obsahuje informace, které se automaticky zobrazí po startu programu. Je zde výchozí hodnota výrobce, VID a PID. Pokud se rozhodnete vyrábět nové výrobky, patrně budete používat vlastní PID (získáte jej od FTDI Chip). Zrovna tak výrobce bude jiný. Potom stačí změnit záznamy v sekci **DEFAULT** a po novém spuštění programu již uvidíte takto zadané hodnoty jako výchozí.

EFSPROG.INI (část):

```
[DEFAULT]
Manufacturer="EFS & Co."
ManufacturerID="EF"
VendorID=0x0403
ProductID=0xFA38
```

```
[EFS00001]
Manufacturer=EFS & Co.
ManufacturerID=EF
VendorID=0x0403
ProductID=0xFA38
Description=Test
USBVersion=0x0200
MaxPower=100 ŠelfPowered=0
RemoteWakeup=l lsoln=0
IsoOut=0 PnP=l
PullDownEnable=0
Memo=Testovací deska s FT232BM, vývody připojeny na konektor PSL10

[EFS00002]
Manufacturer=EFS & Co.
```

11.1.4 Příklad naprogramování konfigurační E^2PROM

Pro lepší orientaci přikládám kratičký popis provedení konfigurace výrobku pomocí programu **EFSProg:**

- Spustíme program **EFSPROG.EXE,**
- zadáme hodnoty polí a data uložíme položkou menu **Operace|Ulož data z formuláře,**
- pokud se uložení zdaří, připojíme konfigurované zařízení k USB,
- aktivujeme položku menu **Operace|Zapiš do EEPROM,**
- ukončíme programátor,
- zařízení odpojíme a znovu připojíme ke sběrnici USB,
- pomocí programu **Seznam** (z kapitoly 5.5, je umístěn na doprovodném CD-ROM v adresáři **FTDI\PROGRAMY\KAP5\Seznam)** zkontrolujeme, zda se zařízení po novém připojení úspěšně enumerovalo.

POZNÁMKA: *Pokud má vytvářené zařízení při prvním připojení k USB prázdnou E^2PROM, bude nutné instalovat kromě* ***EFS*** *ovladačů také ovladače* ***FTDI.*** *Tyto ovladače jsou umístěny na doprovodném CD-ROM v adresáři* ***FTDI\OVLADA CE\D2XX\FTDI.***

11.2 KOMENTOVANÝ VÝPIS PROGRAMU EFSPROG

Komentovaný výpis hlavního souboru programátoru je zajímavý jako příklad použití funkcí popsaných v kapitole 3.3.

EFSPM.CPP:

```
//---------------------------------------------------------------------------
#include   <vcl.h>
#include   <inifiles.hpp>
#include   "Ftd2xx.h"
#pragma   hdrstop
#include   "EFSPM.h"
//---------------------------------------------------------------------------
#pragma   package(smart_init)
Ipragma   resource   "*.dfm"
TProgForm   *ProgForm;
//---------------------------------------------------------------------------
 _fastcall   TProgForm::TProgForm(
     TComponent*   Owner) :TForm(Owner)
{
     Application->OnException=AppException;
     //otevřeni inicializačního souboru pro načteni dat: ini=new   TIniFile
     (GetCurrentDir()+"\\EFSProg.INI");

     //čteni názvu sekci do seznamu Konfigurace: TStringList
     *list=new   TStringList; ini->ReadSections(list) ; if(list-
     >Strings[0]=="DEFAULT") list->Delete(0);
     SerNumbers->Items->Assign(list); delete   list;

     //čteni sekce DEFAULT a inicializace poli: Manufacturer-
     >Text=ini->ReadString("DEFAULT",
          "Manufacturer","Výrobce"); ManufacturerID->Text=ini-
     >ReadString("DEFAULT",
          "ManufacturerID","00"); VendorID->Text=ini-
     >ReadString("DEFAULT",
          "VendorID","0x0403"); ProductID->Text=ini-
     >ReadString("DEFAULT",
          "ProductID","0x6002");
}
```

```
// ---------------------------------------------------------------------------------------------
 _fastcall   TProgForm::-TProgForm()
{
    delete   ini;
}
// ---------------------------------------------------------------------------------------------
void   ___ fastcall   TProgForm::SerNumbersClick(
    TObject   *Sender)
{
    //reakce na výběr položky ze seznamu Konfigurace: if(SerNumbers-
    >ItemIndex!=-l){ AnsiString   section=
            SerNumbers->Items->Strings[SerNumbers->ItemIndex];

    //čtení údajů vybrané sekce:
    SerialNumber->Text=section;
    Manufacturer->Text=ini->ReadString(section,
        "Manufacturer",""); ManufacturerID->Text=ini->ReadString(section,
        "ManufacturerID","" ); VendorID->Text=ini-
    >ReadString(séction,
        "VendorID",""); ProductID->Text=ini->ReadString(section,
        "ProductID", *"") ; Description->Text=ini->ReadString(section,
        "Description",""); USBVersion->Text=ini-
    >ReadString(séction,
        "USBVersion",""); MaxPower->Text=ini-
    >ReadInteger(séction,
        "MaxPower",44) ; SelfPowered->Checked=ini-
    >ReadBool(séction,
        "ŠelfPowered",falše); RemoteWakeup->Checked=ini-
    >ReadBool(séction,
        "RemoteWakeup",true); IsoIn->Checked=ini-
    >ReadBool(séction,
        "Isoln",falše); IsoOut->Checked=ini-
    >ReadBool(séction,
        "IsoOut",falše); PnP->Checked=ini-
    >ReadBool(séction,
        "PnP",true); PullDownEnable->Checked=ini->ReadBool(section,
        "PullDownEnable",false);
```



```
    Poznamka->Text=ini->ReadString(séction,
        "Memo",""); Caption="Programátor   konfiguračních   EEPROM   -   "+section;
    }
}
//---------------------------------------------------------------------------------------------------------------
void   _fastcall   TProgForm::UlozDataZFormulareExecute( TObject   *Sender)
{
    //uložení   dat   do   inicializačního   souboru: AnsiString
    section=SerialNumber->Text;

    //různé   testy:
    if(VendorID->Text.Length()>6|IProductID->Text.Length()>6)
        throw Exception("VID a PID musí být 16bitová čísla"); if(Manufacturer-
    >Text.Length()<2
        | |ManufacturerID->Text.Length ()  !=2 |
        |Manufacturer->Text.Length ()
        +Description->Text.Length()>38) throw   Exception("Povinná délka polí je:\n"
        "Výrobce>=2   znaky\n" "Identifikace   výrobce=2   znaky\n" "součet   délek
        Výrobce+Popis   výrobku<=38   znaků");

    ini->WriteString(section,
        "Manufacturer",Manufacturer->Text) ; ini-
    >WriteString(section,
        "ManufacturerID",ManufacturerID->Text) ; ini-
    >WriteString(section,
        "VendorID",VendorID->Text); ini-
    >WriteString(section,
        "ProductID",ProductID->Text) ; ini-
    >WriteString(section,
        "Description",Description->Text); ini-
    >WriteString(section,
        "USBVersion",USBVersion->Text); try{
        ini->WriteInteger(section,
            "MaxPower",MaxPower->Text.ToInt());
        ini->WriteBool(section,
            "SelfPowered",SelfPowered->Checked);
        ini->WriteBool(section,
```

```
        "RemoteWakeup",RemoteWakeup->Checked); ini-
      >WriteBool(section,
        "IsoIn",IsoIn->Checked)  ; ini-
      >WriteBool(section,
        "IsoOut",IsoOut->Checked) ; ini-
      >WriteBool(section,
        "PnP",PnP->Checked) ; ini-
      >WriteBool(section,
        "PullDownEnable",PullDownEnable->Checked); ini-
      >WriteString(section,
        "Memo",Poznamka->Text) ;

      //aktualizace seznamu Konfigurace: TStringList
      *list=new TStringList; ini->ReadSections(list); if
      (list->Strings[0]=="DEFAULT" )
      list->Delete(0); SerNumbers->Items->Assign(list);
   delete list; } catch(...){
      throw Exception("Data jsou neúplná nebo chybná"); }
MessageBox(Handle,"Data úspěšně uložena",
"EFSProg",MB_OK); }
// ---------------------------------------------------------------------------
^oid ___fastcall TProgForm::ZapisDoEEPROMExecute(
   TObject *Sender) {
   FT_PROGRAM_DATA data;
   DWORD numdevices=0; char
   _Manufacturer[32]; char
   _ManufacturerId[16] ; char
   _Description[64] ; char
   _SerialNumber[16];

   //čteni textových poli do řetězců:
   strcpy(_Manufacturer,Manufacturer->Text.c_str());
   strcpy(_ManufacturerId,ManufacturerID->Text.c_str());
   strcpy(_Description,Description->Text.c_str() ) ;
   strcpy(_SerialNumber,SerialNumber->Text.c_str() ) ;
```



```
//test připojeni jediného zařízeni:
ftStatus=FT_ListDevices(Snumdevices,NULL,
   FT_LIST_NUMBER_ONLY); if
(ftStatus!=FT_OK)
   throw Exception("Celkové selháni D2XX");
if(numdevices==0)
   throw Exception("Neni připojeno žádné zařízení");
if(numdevices>I)
   throw Exception("Musí být připojeno pouze jedno zařízení");

//různé testy:
data.VendorId=VendorID->Text.ToInt() ;
data.ProductId=ProductID->Text.ToInt();
if(VendorID->Text.Length ()>6| IProductID->Text.Length()>6)
   throw Exception("VID a PID musí být 16bitová čísla"); if(Manufacturer-
>Text.Length()<2
    ||ManufacturerID->Text.Length()!=2 |
    |Manufacturer->Text.Length ()
    +Description->Text.Length()>38) throw Exception("Povinná délka polí je:\n"
    "Výrobce>=2 znaky\n" "Identifikace výrobce=2 znaky\n" "součet délek
    Výrobce+Popis výrobku<=38 znaků");

//sestavení dat:
datli.Manufacturer=_Manufacturer;
datli.ManufacturerId=_ManufacturerId;
datli.Description=_Description;
datli.SerialNumber=_SerialNumber;
datli.MaxPower=MaxPower->Text.ToInt();
datli.PnP=PnP->Checked;
datli.ŠelfPowered=SelfPowered->Checked;
datli.RemoteWakeup=RemoteWakeup->Checked;
datli.Rev4=true;
datli.IsoIn=IsoIn->Checked;
datli.IsoOut=IsoOut->Checked;
datli.PullDownEnable=PullDownEnable->Checked;
datli.SerNumEnable=true;
dat11.USBVersion=USBVersion->Text.ToIntO ;

//otevření jednoho zařízení:
ftStatus=FT_Open(0,SftHandle);
```

```
    if ( ftStatus!=FT_OK)
        throw Exception("Zařízeni se nepodařilo otevřít");

    //programováni:
    ftStatus=FT_EE_Program(ftHandle,&data); if
    (ftStatus!=FT_OK)
        throw Exception("Chyba zápisu");

    //zavřeni zařízeni:
    FT_Close(ftHandle);
    MessageBox(Handle,"Data úspěšně zapsána do EEPROM",
        "EFSProg",MB_OK);
}
//---------------------------------------------------------------------------
void _fastcall TProgForm::CtiZEEPROMExecute(
    TObject *Sender) {
    FT_PROGRAM_DATA data;
    DWORD numdevices;
    char _Manufacturer[32] ;
    char _ManufacturerId[16] ;
    char _Description[64] ;
    char _SerialNumber[16];
    datli.Manufacturer=_Manufacturer;
    datli.ManufacturerId=_ManufacturerId;
    datli.Description=_Description;
    datli.SerialNumber=_SerialNumber;

    //test připojeni jediného zařízeni:
    ftStatus=FT_ListDevices(Snumdevices,NULL,
        FT_LIST_NUMBER_ONLY); if
    (ftStatus!=FT_OK)
        throw Exception("Celkové selháni D2XX");
    if(numdevices==0)
        throw Exception("Neni připojeno žádné zařízeni"); if(numdevices>l)
        throw Exception("Musi být připojeno pouze jedno zařízeni");

    //otevřeni jednoho zařízeni:
    ftStatus=FT_Open(0,&ftHandle); if
    (ftStatus!=FT_OK)
        throw Exception("Zařízeni se nepodařilo otevřít");
```

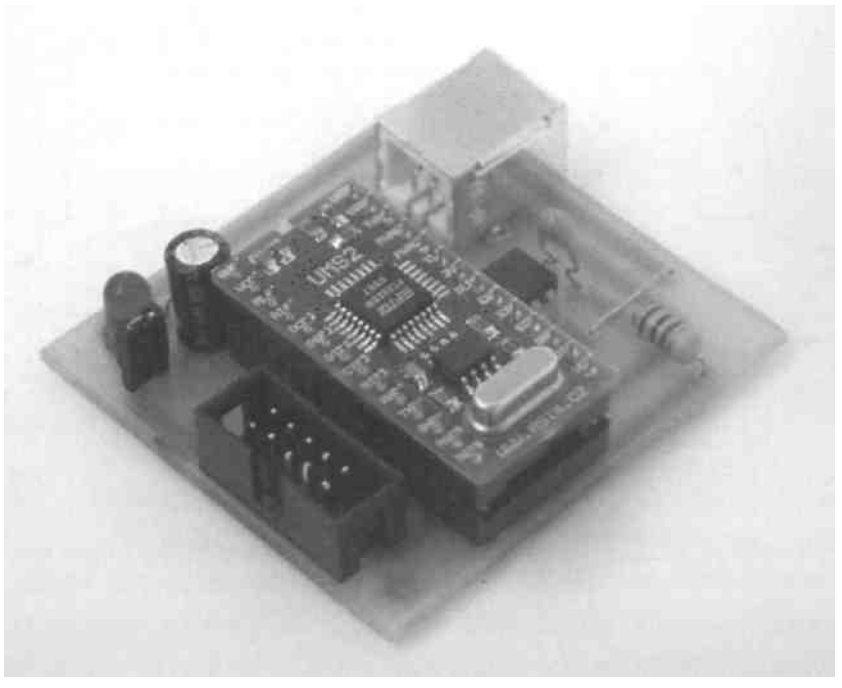


```
    //čteni:
    ftStatus=FT_EE_Read(ftHandle,&data);
    if(ftStatus!=FT_OK)
        throw Exception("Chyba čteni");

    //zavřeni zařízeni:
    FT_Close(ftHandle);

    //zobrazeni dat:
    VendorID->Text="Ox"+IntToHex(datli.VendorId,4);
    ProductID->Text="Ox" + IntToHex(datli.ProductId, 4) ;
    Manufacturer->Text=dat11.Manufacturer;
    ManufacturerID->Text=datll.ManufacturerId;
    Description->Text=datll.Description;
    SerialNumber->Text=datll.SerialNumber;
    MaxPower->Text=datll.MaxPower;
    PnP->Checked=datll.PnP;
    ŠelfPowered->Checked=datll.ŠelfPowered;
    RemoteWakeup->Checked=datll.RemoteWakeup;
    IsoIn->Checked=datll.IsoIn;
    IsoOut->Checked=datll.IsoOut;
    PullDownEnable->Checked=datll.PullDownEnable;
    USBVersion->Text="Ox" + IntToHex(datli.USBVersion, 4) ;
}
// ---------------------------------------------------------------------------
void __ fastcall TProgForm::KonecExecute(
    TObject *Sender)
{
    Close() ;
}
// ---------------------------------------------------------------------------
void _fastcall TProgForm::SerNumbersDrawItem(
    TWinControl *Control,
    int Index, TRect &Rect,
    TOwnerDrawState State)
{
    //pouze měni barvy položek Konfigurace: TColor
    colors[]={clYellow,clGreen,clRed, clBlue} ; SerNumbers->Canvas-
    >Font->Color=colors[Index%4]; SerNumbers->Canvas->FillRect(Rect);
    SerNumbers->Canvas->TextOut(
```

```
        Rect.Left,Rect.Top,SerNumbers->Items->Strings[Index]);
}
// ---------------------------------------------------------------------------
void  _fastcall  TProgForm::AppException( TObject
    *Sender, Exception  *E)
{
    //obsluha  nezachycených  výjimek:
    if(E->ClassNameIs("EConvertError")){ Applicatíon-
        >MessageBox(
            "Musí  být  zadáno  číslo","EFSProg",MB_ICONHAND);
    } else {
        Application->ShowException(E);
    }
}
// ---------------------------------------------------------------------------
void   __ fastcall  TProgForm::OaplikacilClick(
    TObject  *Sender)
{
    //nápověda  o  aplikaci:
    MSGBOXPARAMS  mbp;
    mbp.cbSize=sizeof(mbp);
    mbp.hwndOwner=Handle;
    mbp.hInstance=HInstance;
    mbp.lpszText="EFSProg  v  1.0\n\n"
        "Programátor  EEPROM pro  FTDI  aplikace\n\n"
        "Autor:\tDavid  Matoušek\n"
        "\tVyšší  odborná  škola\n"
        "NtTolstého  16\n"
        "\tJIHLAVA\n"
     "\t586  01";
mbp.lpszCaption="EFSProg" ;
mbp.dwStyle=MB_USERICON;
mbp.lpszIcon="MAINICON";
mbp.dwContextHelpId=0;
mbp.lpfnMsgBoxCallback=NULL;
mbp.dwLanguageId=0x0405;
MessageBoxIndirect(&mbp); }
```

12

KONVERTORY SBĚRNICE USB<=>RS-232C A POUŽITÍ MODULU UMS2



V této kapitole jsou uvedeny dvě konstrukce konvertorů sběrnice USB na RS-232C. Takové konvertory mají velký význam pro přenos stávajících aplikací ovládaných přes sériový port na použití sběrnice USB. Je třeba připomenout, že dnešní počítače již běžně disponují pouze jedním sériovým portem (původní dva sériové porty také pro některé náročnější uživatele nevyhovují), takže zveřejnění takovýchto konvertorů se mi jeví jako velmi zajímavé (uvážíme-li hlavně poměrně vysokou cenu komerčně prodávaných konvertorů tohoto typu).

Dále je uvedena konstrukce testovací desky pro modul UMS2 vyráběný firmou ASIX (popis viz kapitolu 4).

12.1 USB<=>RS VERZE 1.0

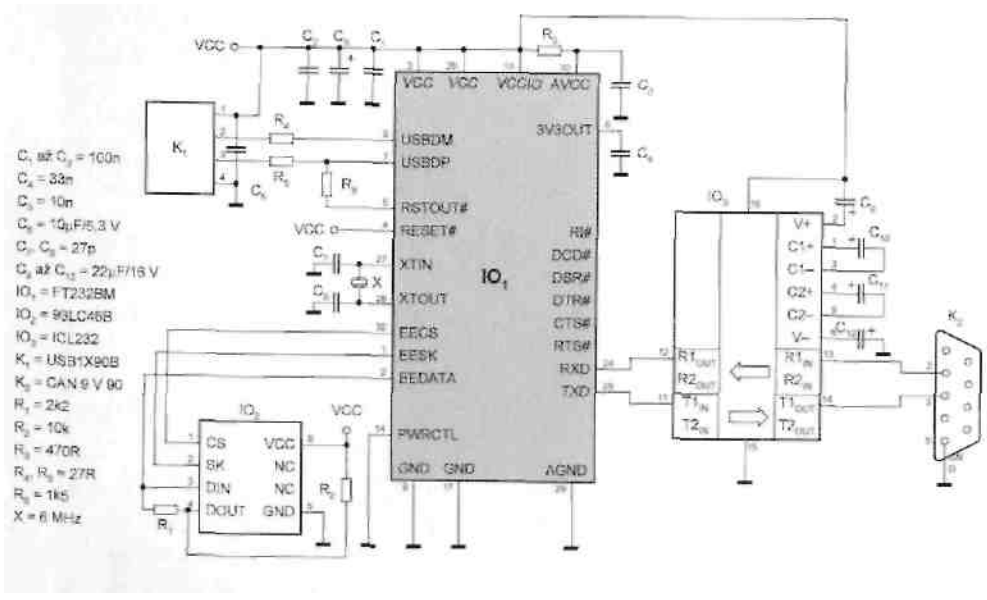
Tato verze konvertoru má níže uvedené vlastnosti:

- jednoduchá a levná konstrukce založená na obvodu **ICL232,**
- neúplný konvertor, k dispozici jsou pouze linky **RxD** a **TxD** (pro jednoduché aplikace to však postačí).

Schéma zapojení je velmi jednoduché je uvedeno na *obr. 12.1.* Jedná se vlastně o zjednodušení výchozí konstrukce publikované výrobcem (viz *obr. 2.9).*

Na místě převodníku úrovní TTL na RS-232C a obráceně je použit obvod typu MAX232 (konkrétně jeho levná varianta **ICL232).**

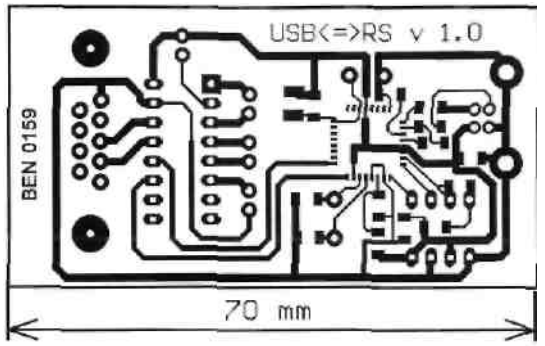
^ Na konektor sériového portu (K_2) jsou vyvedeny pouze linky **RxD** a **TxD.** Toto řešení sice nedovoluje používat linky modemu, ale mnoho jednoduchých konstrukcí to neomezuje (příklady jsou uvedeny v kapitole 12.3). ***Pozor,*** přípravek neobsahuje linky CTS a RTS, které se obvykle používají pro hw handshake. Pokud to je na



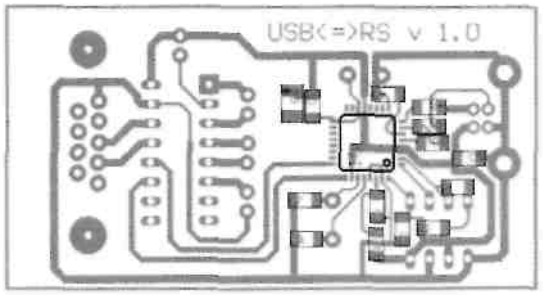
Obr. 12.1 Schéma zapojení USB<=>RS v 1.0

obtíž, můžete používat konvertor USB<=>RS v 2.0 popsaný v kapitole 12.2. Také lze zajistit programový handshake tak, že ovládané zařízení odesílá zpět potvrzovací data. Tak byly ostatně realizovány všechny konstrukce v této knize.

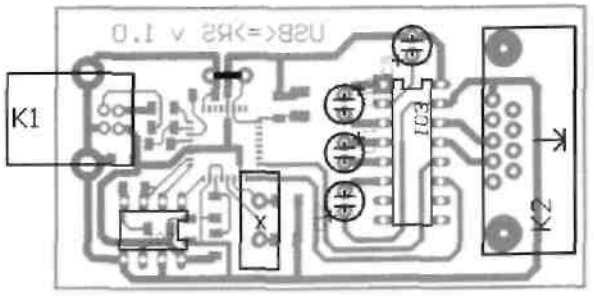
Obr. 12.2 uvádí výkres desky plošných spojů. Následují osazovací plánky ze strany spojů a ze strany součástek.



Obr. 12.2 Výkres desky plošných spojů USB <=>RS v 1.0 (BEN 0159)



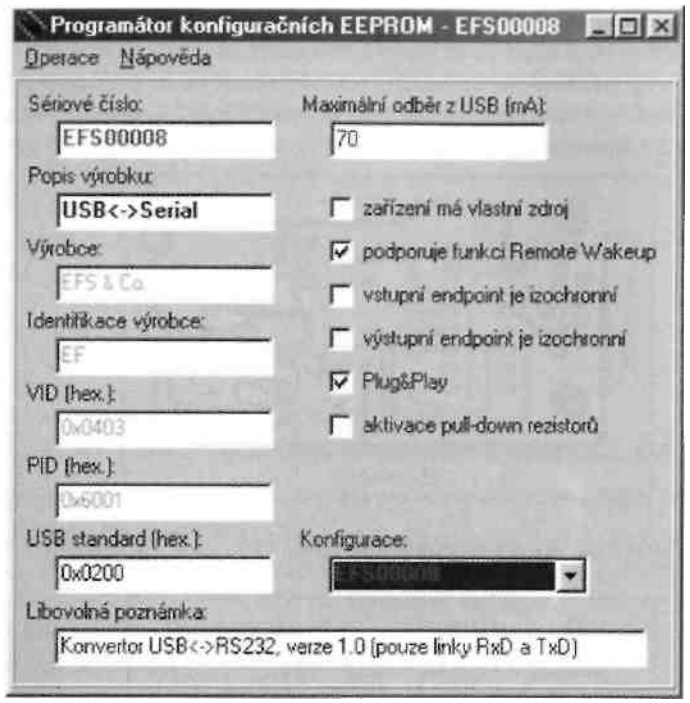
Obr. 12.3 Osazovací plánek (strana spojů)



Obr. 12.4 Osazovací plánek (strana součástek)

Formou *obr. 12.5* je uvedena konfigurace E^2PROM ($I0_2$). Pozor, nyní je **PID = 0x6001.** To je nutné, protože konstrukce předpokládá použití ovladače VCP.

POZNÁMKA: *Paměť $I0_2$ není nutno programovat (nebo osazovat), pokud je na daném počítači používán jediný konvertor.*



Obr. 12.5 Konfigurace E²PROM provedená programem EFSProg

Seznam součástek pro USB<=>RS v 1.0 (cena asi 300 Kč):

C^ažCg	CK+100NX7R	3 ks
C_4	CK+33N X7R	1 ks
C_5	CK+10NX7R	1 ks
C_6	CTS6M8/10VB	1 ks
C_7) C_8	CK+27P NPO	2 ks
C_9ažC$_{12}$	E22M/16V	4 ks
IO,	FT232BM	1 ks
10_2	94LC46B	1 ks
10_3	ICL232	1 ks
K_1	USB1X90B PCB	1 ks
K_2	CAN 9 V 90	1 ks
Ri	RR+2K2 SMD	1 ks
R_2	RR+10KSMD	1 ks
R_3	RR+470R SMD	1 ks
R_4>R_5	RR+27R SMD	2 ks
R_6	RR+1K5SMD	1 ks
X	QM 6.000MHZ	1 ks

12.2 USB<=>RS VERZE 2.0

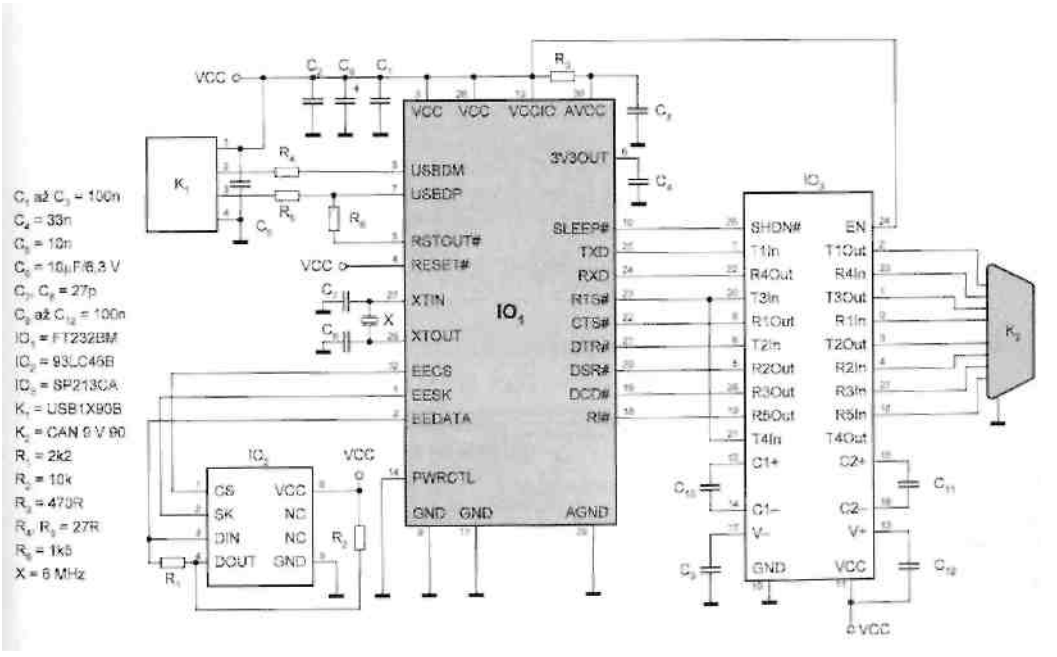
Tato verze konvertoru má níže uvedené vlastnosti:

- úplný konvertor, k dispozici jsou všechny linky sériového kanálu,
- vyšší cena než u USB<=>RS v 1.0.

Schéma zapojení je uvedeno na *obr. 12.6.* Toto zapojení je přímo publikováno výrobcem (výchozí část byla uvedena již na *obr. 2.9)*.

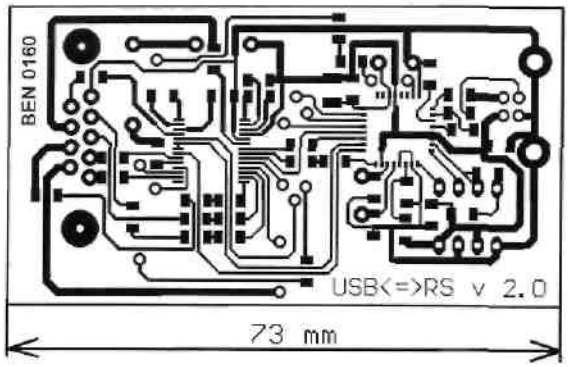
Na konektor sériového portu (K_2) jsou vyvedeny všechny linky sériového kanálu včetně linek modemu. Propojení odpovídá konektoru sériového portu na počítači, proto je možno připojovat zařízení ovládaná sériovým portem přímo k tomuto převodníku.

Propojení signálů **SLEEP# (IO_3)** a **SHDN#** ($I0_3$) umožňuje snížit spotřebu převodníku v režimu sleep pod požadovaných 500 uA. Katalogový list obvodu **SP213CA** najdete v adresáři **DATASHEET** na doprovodném CD-ROM.

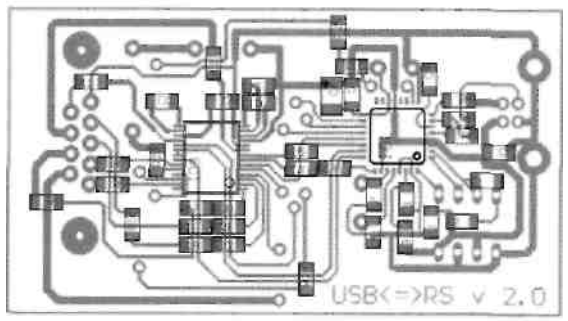


Obr. 12.6 Schéma zapojení USB<=>RS232 v 2.0

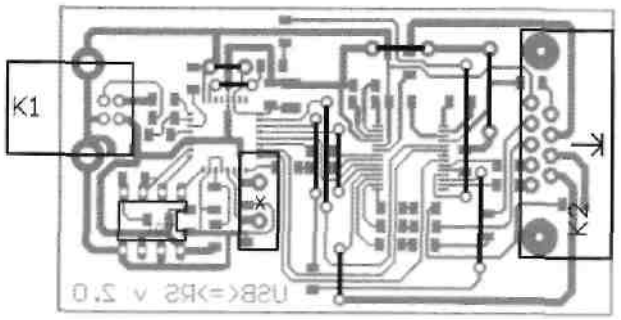
Obr. 12.7 uvádí výkres desky plošných spojů. Následují osazovací plánky ze strany spojů a ze strany součástek.



Obr. 12.7 Výkres desky plošných spojů USB <=>RS v 2.0 (BEN 0160)



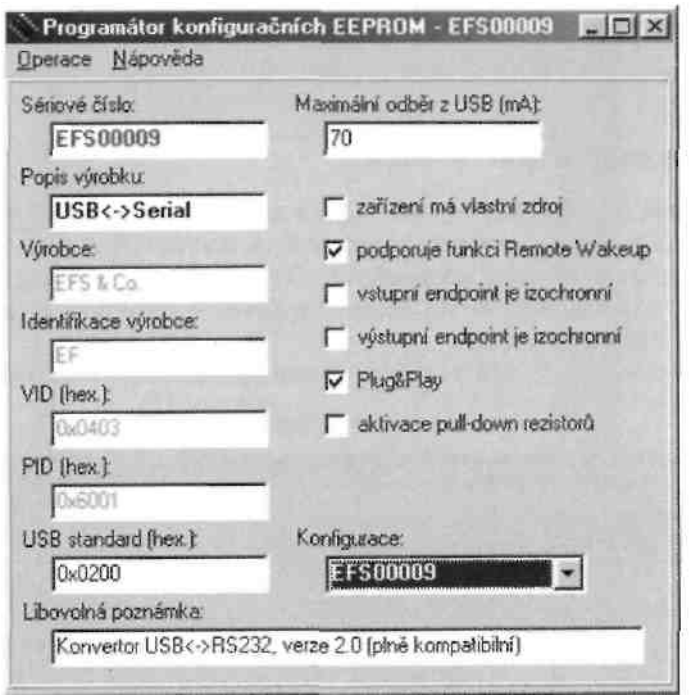
Obr. 12.8 Osazovací plánek (strana spojů)



Obr. 12.9 Osazovacíplánek (strana součástek)

Formou *obr. 12.10* je uvedena konfigurace E^2PROM ($I0_2$). **PID** má opět hodnotu **0x6001.** To je nutné, protože konstrukce předpokládá použití ovladače VCP.

POZNÁMKA: *Paměť10>$_2$ není nutno programovat (nebo osazovat), pokud je na daném počítači používán jediný konvertor.*



Obr. 12.10 Konfigurace E^2PROM provedená programem EFSProg

Seznam součástek pro USB<=>RS v 2.0 (cena asi 350 Kč):

C_1 až C_3, C_9 až C_{12}	CK+100N X7R	7 ks
C_4	CK+33N X7R	1 ks
C_5	CK+10NX7R	1 ks
C_6	CTS 6M8/10V B	1 ks
C_7, C_8	CK+27P NPO	2 ks
IO-,	FT232BM	1 ks
10_2	93LC46B	1 ks
10_3	SP213CA	1 ks
K,	USB1X90B PCB	1 ks
K_2	CAN 9 Z 90	1 ks
R,	RR+2K2 SMD	1 ks
R_2	RR+10KSMD	1 ks
R_3	RR+470R SMD	1 ks
R_4, R_5	RR+27R SMD	2 ks
R_6	RR+1K5SMD	1 ks
R_7ažR_{25}	RR+ORSMD	19 ks
X	QM 6.000MHZ	1 ks

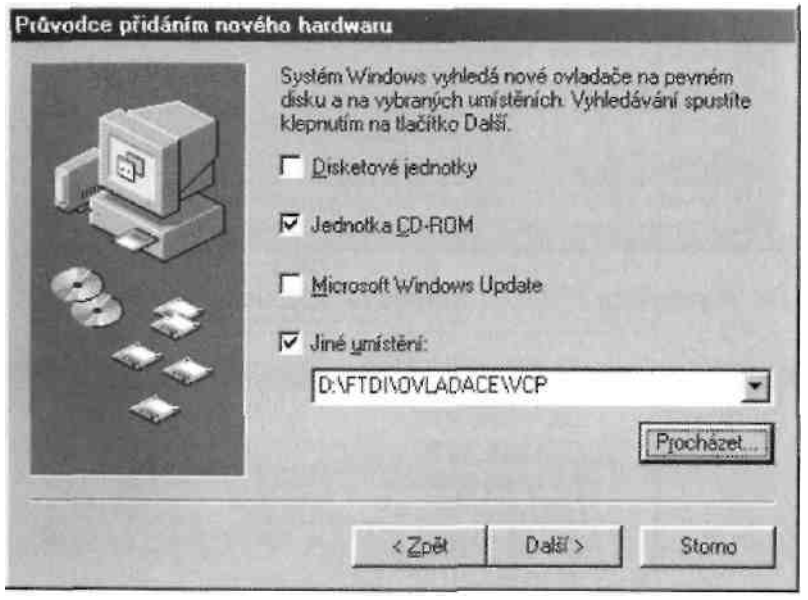
12.3 PŘÍKLADY POUŽITÍ

V této kapitole se zaměříme na ukázky použití výše publikovaných konvertorů.

12.3.1 Na řadě je VCP ovladač

Jak by mělo být zřejmé zejména z *obr. 12.5* a *obr. 12.10,* je nyní **PID** nastaveno na **0x6001.** Což je výchozí hodnota stanovená výrobcem. Používám tedy jinou hodnotu PID než v předchozích konstrukcích. Chtěl jsem, aby se mnou vytvořené konstrukce daly snadno identifikovat. Tento požadavek je velmi důležitý z hlediska enumerace a přiřazení ovladače.

Po připojení konvertoru k USB a úspěšné identifikaci je třeba zvolit ovladač VCP. Vizoor. *12.11.*



Obr. 12.11 Výběr VCP ovladače pro zařízení s VID = 0x0403 a PID = 0x6001

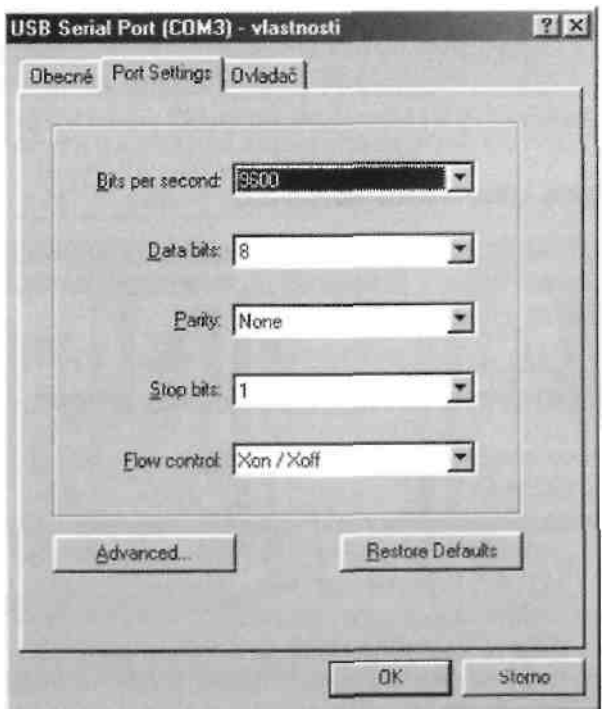
Po úspěšné instalaci lze pomocí dialogu **Systém** (Start | Nastavení | Ovládací panely) najít virtuální sériový port. Viz *obr. 12.12.*

Po stisku tlačítka **Vlastnosti** se zobrazí dialog dle *obr. 12.13.*

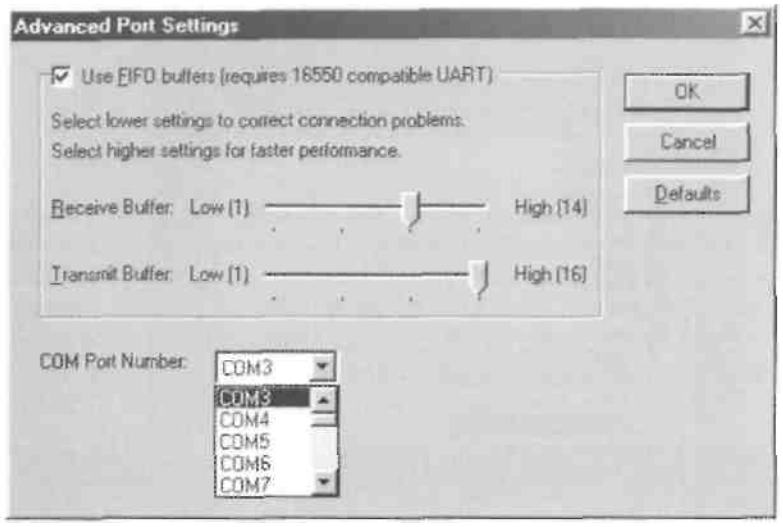
V tomto dialogu lze vybrat kartu **Port Settings** a nastavit tak parametry sériového portu. Dále lze volit i jeho číslo (po stisku tlačítka **Advanced,** viz *obr. 12.**14).***

***Pozor!** Doporučuji nespoléhat se na to, že operační systém (Windows XP) ovladače najde. Lepší je ovladač vyhledat ručně přímo na CD-ROM. Instalace je dvoufázová. Po instalaci VCP ovladače se instaluje ovladač pro nově vzniklý virtuální sériový port.*

Obr. 12.12 V systému je instalován virtuální sériový port COM3



Obr. 12.13 Dialoq vlastností sériového portu COM3, karta Port Settings



Obr. 12.14 Možnost změny čísla sériového portu v dialogu Advanced Port Settings

12.3.2 Aplikace USB<=>RS verze 1.0

Konvertor USB<=>RS v 1.0 lze použít pouze pro aplikace nevyžadující hardwarové řízení toku dat. Tedy pro ty aplikace, které nepoužívají linky modemu, ale pouze přenosové linky **TxD** a **RxD.**

Jedním z případů může být programátor mikrokontrolérů **AT89C2051** nazvaný **ATPROG2.1** (viz [1] nebo [16]) a většina zařízení publikovaných v [16].

12.3.3 Aplikace USB<=>RS verze 2.0

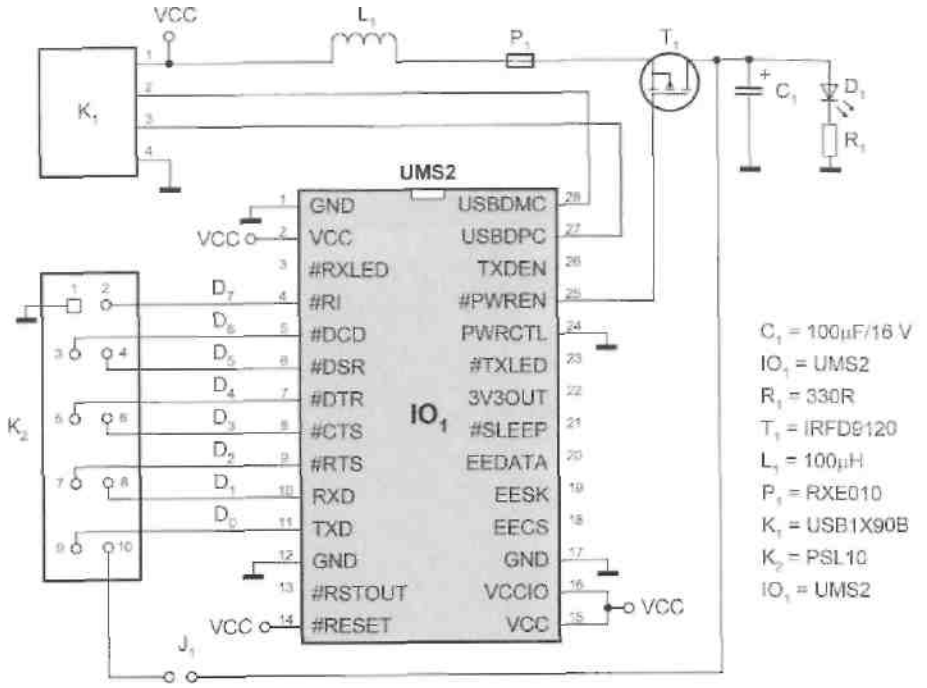
Dražší varianta konvertoru má smysl pouze pro aplikace vyžadující použití linek modemu. Jedná se například o programátory mikrokontrolérů **AT89S8252, AT90S2313, AT90S8535,** které byly uvedeny v [2] a [3].

12.4 UMS2TST - TESTOVACÍ DESKA PRO MODUL UMS2

Nyní si ukážeme jednodušší použití modulu **UMS2.** který vyrábí firma **ASIX** (popis byl proveden v kapitole 4).

Zapojení je na úrovni testovacího přípravku **FT232TST,** který byl popsán v kapitole 5. Stejně je řešen i konektor K_2. To znamená, že stejně dobře jako byl v kapitole 5 testován přípravek **FT232TST,** lze otestovat i tento přípravek nazvaný **UMS2TST.**

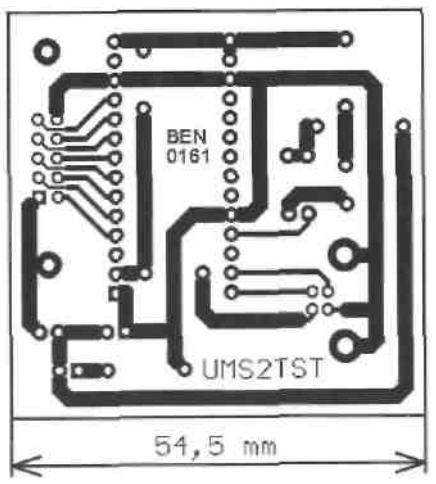
Nad rámec původního řešení je použit tranzistor T_t, který připojuje zátěž až po úspěšné enumeraci. Dále je doplněna cívka L, a kondenzátor C_r „Šikovné" je připojení LED označené D,, podle ní můžeme sledovat úspěšnou enumeraci. Opět je zařazen jumper J, pro případ, že vnější zařízení má vlastní napájecí zdroj. Po-lyswitch P, omezuje odebíratelný proud na 100 mA.



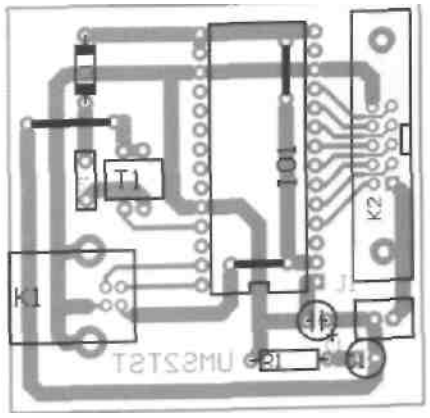
12.15 Schéma zapojení přípravku UMS2TST

***POZNÁMKA** k obr. 12.15: Vývody č. 2 a č. 15 modulu UMS2 jsou vnitřně spojeny. Takže drátová propojka spojující vývody č. 2 ač. 15 není nutná.*

Výkres desky plošných spojů a osazovací plánek přípravku **UMS2TST** jsou uvedeny na *obr. 12.16 a obr. 12.17.*

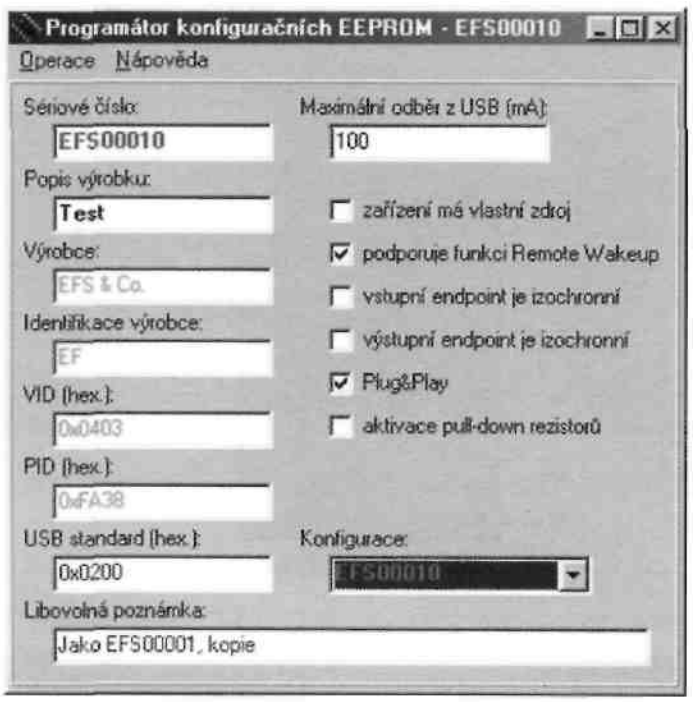


Obr. 12.16 Výkres desky plošných spojů UMS2 TS T (BEN 0161)



Obr. 12.17 Osazovací plánek UMS2 TS T

Konfigurace E^{2}PROM je zřejmá z *obr. 12.18.* Je třeba zdůraznit, že E^{2}PROM je přímou součástí modulu **UMS2**



Obr. 12.18 Konfigurace E^{2}PROM provedená programem EFSProg

Seznam součástek pro přípravek UMS2TST (cena modulu UMS2 asi 70 Kč):

K,	USB1X90B	1 ks
K_2	PSL10	1 ks
L,	TLIOOuH	1 ks
T,	IRFD9120	1 ks
P,	RXE010	1 ks
R,	330R	1 ks
C,	E100M/16V	1 ks
D,	LED 5MM 200 MCD	1 ks
Jт	jumper 1 ks	
10,	modul UMS2 v patici DIP28	1 ks

Pro test přípravku lze použít libovolný z programů, které byly popsány v kapitole 5.

13

VÝPIS SOUBORŮ EFS.INF A FTD2XXUN.INI

V této kapitole naleznete částečně komentovaný výpis instalačního souboru ***EFS.INFpro*** *zařízení třídy* ***EFS FT232BM Device.*** *Tento soubor je nutný pro identifikací zařízení s* ***VID = 0x0403, PID = 0xFA38.***

Výpis také ukazuje, jak soubor upravit pro vlastní potřebu (například pro nový PID).

EFS.INI:

```
;Installation inf for FTDI Direct Driver

/Copyright (c) 2001-2003 Future Technology ;Devices
International Ltd.

[Version]
Signature="$CHICAGO$"
Class=USB
ClassGUID={3 6fc9e60-c4 65-llcf-8056-44 4 55354 0 000}
Provider=%EFS%/poskytovatel
/CatalogFile=ftd2xx.cat
DriverVer=01/24/2003,1.05.04

[ SourceDisksNames ]
l=%DriversDisk%, , ,

[SourceDisksFiles]  /seznam souborů pro instalaci
FTD2XX.sys = 1
FTD2XX.inf = 1
FTD2XX.dll = 1
FTD2XXUN.exe = 1
FTD2XXUN.ini = 1

[Manufacturer]/výrobce
%EFS%=EFSHw

[EFSHw]/definice VID a PID
%USB\VID_0403&PID_FA38.DeviceDesc%=FTD2XX,  USB\VID_0403&PID_FA38

[ControlFlags]/seznam VID a PID
ExcludeFromSelect=USB\VID_04 03&PID_FA38

/ [PreCopySection]
;HKR,,NoSetupUI, , 1

[DestinationDirs]
DefaultDestDir = 11
FTD2XX.Files.Ext = 10,System32\Drivers
FTD2XX.Files.Dll = 11 FTD2XX.Files.Exe = 11

/For Win98 .../sekce definujici položky pro Win98/Me
[FTD2XX]
/soubory, které je nutno zkopírovat:
CopyFiles=FTD2XX.Files.Ext, FTD2XX.Filés.Dli,
    FTD2XX.Files.Exe /soubory, které se připoji do
Registru:
AddReg=FTD2XX.AddReg, FTD2XX.UnInst

[FTD2XX.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,FTD2XX.sys
HKLM,"System\Currentcontrolset\Services\FTD2XX\Parameters",
    "MaximumTransferSize",0x10001, 4096
HKLM,"System\Currentcontrolset\Services\FTD2XX\Parameters",
    "DebugLevel",0x10001,2

[FTD2XX.Files.Ext]
FTD2XX.sys

[FTD2XX.Files.Dll]
FTD2XX.dll

[FTD2XX.Files.Exe]
FTD2XXUN.exe
FTD2XXUN.ini

[FTD2XX.UnInst]
HKLM,%WINUN%,"FTD2XX"
HKLM,%WINUN%\FTD2XX , "UninstallString" , ,
    "%ll%\FTD2XXUN.exe %11%\FTD2XXUN.INI "
HKLM,%WINUN%\FTD2XX , "DisplayName",,
    "FTDI FTD2XX USB Drivers"

/ For Win2000 .../sekce definujici položky pro Win2000/XP

[FTD2XX.NT]
/soubory, které je nutno zkopírovat:
CopyFiles=FTD2XX.Files.Ext, FTD2XX.Files.Dli, FTD2XX.Files.Exe
```

;soubory, které se připoji do Registru:
AddReg=FTD2XX.AddReg, FTD2XX.UnInst

[FTD2XX.NT.Services] Addservice = FTD2XX, 0x00000002,
FTD2XX.AddService

[FTD2XX.AddService] DisplayName = %FTD2XX.SvcDesc%
ServiceType = 1 ; SERVICE_KERNEL_DRIVER StartType - 3
; SERVICE_DEMAND_START ErrorControl = 1 ;
SERVICE_ERROR_NORMAL ServiceBinary =
%10%\System32\Drivers\FTD2XX.sys LoadOrderGroup = Base

[Strings] EFS="EFS" **/definice třidy zařizeni:**
USB\VID_0403&PID_FA38.DeviceDesc="EFS FT232BM Device"
FTD2XX.SvcDesc="FTD2XX.SYS FT8U2XX device driver"
WINUN="Software\Microsoft\Windows\CurrentVersion\Uninstall" DriversDisk="FTDI
FTD2XX Drivers Disk"

Velmi důležitý je také inicializační soubor **FTD2XXUN.INI,** který je použit deinstalačním programem **FTD2XXUN.EXE** pro korektní odinstalování ovladače ze systému. Uvědomme si totiž, že každý ovladač pro USB zanechá v Registru záznamy vztahující se k VID a PID zařízení, které ovládá.

FTD2XXUN.INI:
[Uninstall]
Device=VID_04 03&PID_FA38 **;VID a PID záznamů**
Converter=FTD2XX

Na závěr se velice krátce zmíním o tom, u kterých firem lze zakoupit součástky a moduly používané v konstrukcích publikovaných v této knize nebo již hotová zařízení.

14.1 SEZNAMY SOUČÁSTEK

U každé konstrukce je zde v knize seznam potřebných součástek. Názvy (kódy) součástek jsou převzaty z aktuálního katalogu GM electronic. Podle nich je možné objednávat (nakupovat) nejen v obchodní síti GM electronic, ale i v zásilkové službě Electronic Obecnice.

14.2 GM ELECTRONIC

S touto firmou mám, přes relativně dlouhé dodací termíny, dobré zkušenosti. V současnosti je schopna dodat prakticky všechny potřebné součástky. V sortimentu nalezneme i obvody FT232BM, D/A převodníky TC1320EOA a obvod SP213CA.

Kontakt: MALOOBCHOD
ZÁSILKOVÁ SLUŽBA ČR
Sokolovská 32 186 00 Praha 8
tel.: 224 816 491
zasilkova.sluzba@gme.cz

14.3 ZÁSILKOVÁ SLUŽBA ELECTRONIC OBECNICE

Tato firma zajišťuje velmi pružné dodávky součástek v rozsahu srovnatelném s GM Electronic.

Kontakt: Zásilková služba Electronic Obecnice
Obecnice 318
262 21
tel.: 318 635 650, 318 621 963
www.elektro-obecnice.cz

14.4 ASIX

U výhradního zástupce FTDI Chip pro Českou republiku lze zakoupit zejména moduly UMS2, obvody FT232BM, paměti **94LC46,** krystaly 6 MHz a USB konektory typu B. Dále by se v sortimentu měly objevit D/A převodníky TC1320EOA.

Kontakt: ASIX s. r. o.
Staropramenná 4
150 00 Praha 5 - Smíchov
tel./fax: 257 312 378
www.asix.cz

14.5 HOTOVÉ PŘÍPRAVKY

Hotové přípravky lze objednat přímo u autora knihy.

Úplná nabídka dostupných přístrojů včetně aktuálních cen je uvedena na:

www.mujweb.cz/www/efs-prodej

Objednávku lze učinit přímo na této stránce nebo písemně na adrese:

Ing. David Matoušek
Vyšší odborná škola
Tolstého 16 Jihlava 586
01

14.6 ZÁVĚR

Nejdříve bych rád poděkoval panu Ing. Václavu Dvořáčkovi z firmy ASIX s. r. o., který mi poskytl úvodní informace o obvodech **FT232BM.**

Dík patří také panu Ing. Ludvíku Dvořákovi (rovněž z firmy ASIX) za zápůjčku modulů s obvody FTDI, které mi umožnily první testy těchto obvodů.

V dalších knihách zaměřených na USB bych se rád věnoval mikrokontrolérům, které mají sběrnici USB zabudovanou. Je pravda, že jejich cena je vyšší než cena obvodů FTDI. Také často nejsou k dispozici ovladače. Přesto však má toto řešení určité výhody - například větší variabilitu, nižší počet součástek.

Dále bych chtěl upozornit na to, že chystám knihu o ovládání zařízení v Delphi. Jistě to bude pro mnohé čtenáře zajímavější, protože jim umožní vlastní experimenty (čtenáři budou moci sami programovat, protože znalosti programování v Delphi jsou obecně rozšířenější než v C++ Builderu).

Připomínám také, že pro ty zájemce, kteří se neodváží k amatérské stavbě přípravků, je kontakt na výrobce uveden v kapitole 14.4.

Na závěr vám přeji mnoho úspěchů s obvody FTDI, které byly navrženy opravdu geniálně a tak poskytují vlastně ještě více výhod, než má samotná sběrnice USB.

PŘEHLED PŘÍPRAVKU

PLOŠNÉ SPOJE

Originální klišé pro výrobu plošných spojů jsme předali firmě **SPOJ.**
V objednávce postačí uvést označení plošného spoje, např. BEN 0151.
Pro zájemce, kteří si plošné spoje budou chtít vyrobit sami, jsou klišé otištěny
přímo v textu a na doprovodném CD-ROM jsou ve formátu TIF v adresáři SPOJ.
Další informace jsou na str. 9 (Co najdete na doprovodném CD).
Vseje pozitivní, tj.
černá je spojový obrazec, rozlišení 300 dpi v měřítku 1:1.

Kontaktní adresa:

SPOJ - výroba plošných spojů, Nosická 16, 100 00 Praha 10
tel. 274 813 823, mobil 604 853 525
e-mail: spoj@volny.cz
Internet: http://www.volny.cz/plspoj
otevírací doba: Po-Pá 8.00-11.00 a 16.00-18.00 hod.