TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

BAKALÁŘSKÁ PRÁCE

Liberec 2009 **Jan Bílek**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011 – Elektronické informační a řídící systémy

Počítačové zpracování a vyhodnocení záznamu o otáčkách motoru a rychlosti vozidla

Computer processing the record of engine rpm and vehicle velocity

Bakalářská práce

Autor: **Jan Bílek**
Vedoucí práce: doc. Ing. Petr Tůma, CSc.

V Liberci 29. 5. 2009

Originál zadání práce !!!!!!!!!!

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Petru Tůmovi, CSc za odborné vedení při zpracovávání práce.

Abstrakt

Cílem bakalářské práce je navrhnout a vytvořit počítačový program, který bude vyhodnocovat a zpracovávat naměřená data z měřicí desky. Měřicí deska snímá a zaznamenává do paměti hodnoty rychlosti a otáček mechanického zařízení.

V úvodu práce je seznámení se současnými způsoby pořízení informací o otáčkách a rychlosti, které se používají v osobních automobilech. Následuje popis rozhraní USB pro připojení měřicí desky, popis obvodu FT232RL, který je použit jako převodník USB na RS232. Možnost použití a způsob zapojení tohoto obvodu je popsáno v datasheetu. Je zde rozebráno a nastíněno jakým způsobem obvod FT232RL komunikuje s počítačem. Popsány jsou také ovladače zařízení od firmy FTDI, které zpřístupní zařízení v systému Windows a to jak ovladače VCP, které se chovají jako virtuální sériový port a využívají funkce Windows API, tak ovladače D2XX pro přímý přístup na port USB. Dále je popsána realizace aplikace Vyhodnocení dat z tachometru. Popis začíná rozborem základních požadavků aplikace a pokračuje popisem pracovního prostředí. Aplikační prostředí je popsáno po jednotlivých blocích programu. Nejprve hlavní okno programu. To slučuje okno Stahování dat a okno Rekonstrukci jízdy, což jsou okna, která se nejvíc používají. Potom jsou popsána dialogová okna pro zobrazování aktuálních hodnot. Je to okno Tachometr, které se snaží napodobit palubní desku automobilu a okno Statistika, které počítá a zobrazuje statistiky průběhu jízdy, např. ujetá dráha, max. rychlost apod. Ovládání programu je shrnuto v krátkém návodu k použití, ve kterém je popsán význam jednotlivých ovládacích prvků. V závěru bakalářské práce jsou shrnuty poznatky z používání programu, možné rozšíření a dosažené výsledky.

Klíčová slova: Delphi, program, zpracování dat, USB, FTDI.

Abstract

The aim of thesis is to design and create a computer program that will evaluate and process the data from a device which is used for measuring and recording engine rpm and vehicle velocity.

At the begin of the work is familiar with current methods of acquisition of information about engine rpm and vehicle velocity, which are used in passenger cars. Here is a description of the USB interface for connecting the measuring plate, a description of the circuit FT232RL, which is used as a USB to RS232 converter. Possibilites to use and method of engagement, which are described in the datasheet of the circuit. It discusses and outlines how FT232RL circuit communicates with the computer. Are also described device drivers from FTDI, which devices make visible in Windows and they are VCP drivers, which are treated as a virtual serial port and uses the Windows API function or drivers D2XX for direct access to the USB port. Description starts with analysis of the essential requirements and continue description of the working environment. This is described in individual program blocks. First, the main window of the program. This merges the data download window and the window for the reconstruction of the journey, which are windows that are most used. Then are described dialog boxes to which display the current values. It is a window tachometer, which seek to simulate a car dashboard and Statistics window, which counts and displays the statistics the journey, such total distance, max speed, etc.. Control Program is summarized in a short instruction manual, in which is briefly described the importance of individual controls.The thesis conclusion summarizes the lessons drawn from the use program, and the possible extension and the results.

Keywords: Delphi, program, data processing, USB, FTDI.

Obsah

Seznam zkratek

UART –Universal Assynchronous Receiver Transmitter

USB – Universal serial bus

RS232 – sériový port

I/O porty – vstupně výstupní porty

API - Application programming interface – funkce pro programování pod OS windows

Paměť RAM – Random Access memory

MCU - microcontroller unit – mikropočítač

Seznam symbolů

b – bit – jednotka informace

B – byte – jednotka informace 1B = 8b

Úvod

Bakalářská práce se zabývá zpracováním dat o otáčkách a rychlosti naměřených na vozidle pomocí měřicí desky a jejím cílem je zkonstruovat počítačový program, který zpracování a vyhodnocování bude vykonávat. Data se měří pomocí měřicí desky, která je ukládá do paměti po stejnoměrném časovém useku. V paměti se uchovává kompletní záznam celé jízdy a je v ní uložena vždy pro každý měřící krok, hodnota otáček, hodnota rychlosti. Původně bylo počítáno s využitím více měření, což bylo nakonec pro požadavky uživatele změněno na měření jedno. Měření má uloženo svůj start a konec, program si převezme celý obsah paměti z měřicí desky a roztřídí si data dokud nenarazí na konec měření. Další možností je měření v reálném čase, kdy jakmile jsou hodnoty změřeny, odesílají se ke zpracování do programu .Deska je spojena s počítačem pomocí převodníku RS232 na USB s FT232RL od firmy FTDI chip.

Vyhodnocovací program se spojí s deskou přes ovladač USB portu dodávaný s převodníkem, vyžádá si od desky počet měření, stáhne si uživatelem požadované měření a uloží si ho do pole. Pro měření v reálném čase se hodnoty ukládají do pole postupně podle toho, jak je program obdrží. Potom je možné naměřená data exportovat do formátu txt, csv nebo tisknout. Dále je možné zrekonstruovat průběh jízdy, kdy se zobrazí v grafu průběh otáček, rychlosti nebo obojího najednou. Průběh rekonstrukce se dá zpomalit i zrychlit podle potřeby. Dále se průběh dá zobrazit i na virtuální palubní desce. Jsou k dispozici i statistiky, kde je možno vidět průběžnou spotřebu, maximální otáčky a maximální rychlost a nakonec po skončení jízdy i celkovou spotřebu a ujetou dráhu. Graf je samozřejmě možné uložit do obrázku nebo ho vytisknout.

1. Teoretický rozbor

1.1 Přehled o současném stavu problematiky

V dnešní době je centrem automobilu řídicí jednotka, do které se dostávají a vyhodnocují signály ze všech čidel, které snímají různé druhy veličin a mezi nimi jsou i čidla pro snímání otáček a rychlosti. Tyto veličiny se dají snímat různými způsoby, ale v poslední době většina výrobců používá pro snímání a vyhodnocování podobné metody a postupy. Otáčky jsou snímány dvěma čidly a to indukčním čidlem a Hallovou sondou.

Induktivní snímače se skládají dle [7] ze tří hlavních součástí a to z nehybné cívky, části magneticky měkkého železa a trvale magnetické části. Změna magnetického toku je způsobována otáčením ozubeného kola. V cívce se indukuje změnou magnetického toku střídavé sinusové napětí, to je přibližně úměrné změně magnetického toku Φ. Velikost signálu je přímo úměrná otáčkám ozubeného kola. Na výstupu se napětí tvaruje Schmittovým obvodem na pravoúhlý průběh. Amplituda signálu závisí výrazně na vzduchové mezeře a velikosti zubů. Při zvětšení otáček motoru roste amplituda i frekvence signálu. Výhodou těchto snímačů jsou nízké výrobní náklady, odolnost proti rušení a velký teplotní rozsah.

Hallův snímač patří mezi magnetostatické snímače, které měří stejnosměrné magnetické pole. Oproti induktivním snímačům se dá lépe použít pro miniaturizaci a je levnější. Hallův snímač využívá Hallova jevu. Vyhodnocuje se pomocí tenkých destiček, kterými protéká elektrický proud a zároveň prochází magnetické pole. Jsou-li proud a magnetická indukce na sebe kolmé, lze příčně ke směru proudu naměřit Hallovo napětí U_H, úměrné velikosti magnetického pole. Příčinou je působení magnetického pole na elektrony jako nosiče náboje.

Diferenciální Hallovy snímače jsou dva kompletní Hallovy systémy umístěné v definované vzdálenosti na jednom čipu a vyhodnocuje se rozdíl Hallových napětí. Tyto snímače se většinou používají k měření otáček. Rotorem je ozubené kolo z kovového materiálu. Přiblížení a oddálení jednotlivých zubů způsobuje změny magnetického pole v blízkosti permanentního magnetu. Oproti induktivním snímačům mají tyto snímače výhodu v tom, že jejich výstupní napětí není závislé na rychlosti a dá se tak snadno elektronicky zpracovat.

V automobilu se dnes používají pro měření otáček dvě čidla. První je umístěno u setrvačníku se zuby a je to induktivní snímač. Měří otáčky, ale i horní úvrať

a vyhodnocuje se z něj rovnoměrnost chodu motoru. Druhé čidlo, induktivní snímač nebo Hallova sonda je umístěno u vačky ventilů. V jednotce se vyhodnocují společně a řídí se podle nich spalování motoru.

Rychlost se měří také induktivním snímačem, ten je ale umístěn v převodovce, kvůli možnosti řazení různých převodových stupňů.

Naměřené signály se dále posílají do řídicí jednotky automobilu, která je vyhodnotí a pošle signál pro zobrazení na palubní desce. Řídicí jednotka si bohužel neuchovává záznamy průběhu celých jízd, a tak se dá zjistit jen počet ujetých kilometrů, maximální otáčky a rychlost.

1.2 Periferie komunikace s měřící deskou

1.2.1 Obecný popis USB

USB se stalo v poslední době neoddělitelnou součástí všech zařízení, které se připojují k počítači a tím již skoro vytlačilo sériový port z jeho základní výbavy. Postupně se k USB musí přecházet, protože dnes už ubývá počítačů se sériovým portem, například u notebooku se dnes dá se sériovým portem setkat jen vyjímečně. Mezi základní parametry USB patří komunikační vzdálenost do 5 m, přenosová rychlost 1,5 Mbit/s až 480 Mbit/s, lze připojit až 127 zařízení na jeden konektor, obsahuje 5V napájení.

Podle [8] je USB sběrnice jen s jedním zařízením typu Master, tj. všechny aktivity vycházejí z PC. Data se vysílají v krátkých paketech o 8 B nebo v delších paketech o délce až 256 B. Přijímají se tak, že PC může požadovat data od zařízeni, naopak žádné zařízení nemůže vysílat data samo od sebe.Veškerý přenos dat se uskutečňuje v tzv. rámcích (frame) které trvají přesně 1 milisekundu. Uvnitř jednoho rámce mohou být postupně zpracovávány pakety pro několik zařízení. Přitom se mohou spolu vyskytovat pomalé (low-speed) i rychlé (full-speed) pakety. Obrací-li se PC na více zařízení, zajišťuje jejich rozdělení jako rozdělovač sběrnice (hub). Zabraňuje také, aby signály s plnou rychlostí (full-speed) byly vedeny na pomalá zařízení.

Pomalá zařízení pracují s přenosovou rychlosti 1,5 Mb/s. Rychlé přenosy pracuji s rychlostí 12 Mb/s. Časový průběh přenosu informace je předepisován výhradně masterem. Zařízení typu slave se musí synchronizovat s datovým tokem. Používá se k tomu metoda NRZI (Non Return To Zero). Nuly v datech vedou ke změně úrovně, jedničky nechávají úroveň beze změny. Kódování a dekódování signálů je čistě hardwarovou záležitostí. Přijímač musí být schopen získat signál, přijmout a dekódovat data. Speciální prostředky zajišťují, aby nedocházelo ke ztrátě synchronizace. Obsahuje-li původní datový tok šest po sobě jdoucích jedniček, přidá vysílač automaticky jednu nulu (vkládání bitů – bit-stuffing), aby se tím vynutila změna úrovně. Přijímač tuto nulu z datového toku opět odstraní.

Každý datový paket má za účelem synchronizace speciální zaváděcí bajt (00000001b). Přijímač v důsledku kódování NRZI a vsouvání bitů vidí osm střídajících se bitových stavů, na které se může zasynchronizovat. Během následujícího přenosu musí synchronizace zůstat zachována. Všechny tyto procesy se odehrávají pouze v odpovídajících hardwarových součástkách. Přijímač a vysílač jsou realizovány vždy

společně v jedné součástce. Zařízení USB obsahuje jednotku zvanou SIE Serial Interface Engine, která přebírá vlastní práci. K výměně dat mezi SIE a zbytkem zařízení slouží buffery FIFO. FIFO (First In First Out) jsou paměti, které mohou postupně přijímat a vydávat data podobně jako posuvné registry. Připojený mikrořadič pouze potřebuje jen přečíst data z FIFO a jiná data do FIFO zapsat. Všechno ostatní vyřídí SIE. Ve většině případů je SIE součástí mikrořadiče USB.

Zařízení USB má obecně několik pamětí FIFO, jejichž prostřednictvím je možno přenášet data. Například myš, která je připojena přes USB má vždy koncovou endpoint 0 a endpoint 1. Endpoint 0 se používá při inicializaci. Vlastní užitková data se z mikrořadiče v určitých časových odstupech zapisují do endpointu 1 a odtud si je vybírá PC. USB software tvoří tzv. trubice (pipes) k jednotlivým endpointům (koncovým adresám). Jedna pipe je logický kanál k jednomu endpointu v jednom zařízení. Pipe si můžeme představit jako datový kanál tvořený jediným vodičem. Ve skutečnosti však jsou data v pipe přenášena jako datové pakety v milisekundových rámcích a hardwarem rozdělována na reálné paměti podle jejich koncové (endpoint) adresy. Jedno zařízení může současně používat několik trubic (pipes), takže přenosová rychlost celkově vzroste.

1.2.2 FT232RL popis obvodu

Pro připojení měřicí desky mohlo být použito klasické připojení přes sériový port, ale bylo použito rozhraní USB, protože je linka RS232 na dnešních PC i noteboocích nahrazována linkou USB. Pro tento úkol byl použit obvod FT232RL od firmy FTDI Chip (*www.FTDIChip.com*). Obvod FT232RL je podle datasheetu [3] převodník USB na obecný protokol UART. Podporuje plně hardwarové řízení toku dat a také řízení pomocí X-ON/X-OFF. Můžeme nastavovat parametry přenosu jako 7 nebo 8 datových bitů, stop bity 1 nebo 2 a pět druhů parity (odd / even / mark / space / no parity), a také nastavovat přenosové rychlosti od 300 Baud do 1 MegaBaud u RS232. Obvod nejen umí převádět protokoly, ale obsahuje i další užitečné nástroje jako například přímé připojení led diod pro signalizaci vysílání nebo příjmu a hodinový generátor, který může sloužit jako zdroj hodinového signálu pro další obvody. Nespornou výhodou převodníku FT232RL je fakt, že celý USB protokol je vyřešen uvnitř čipu a je zaručena plná kompatibilita se všemi verzemi USB včetně USB 2.0 full speed.

V těchto obvodech je již integrována většina prvků potřebných pro připojení ke sběrnici USB jako například zakončovací rezistory, krystalový oscilátor nebo filtr napájecího napětí včetně resetovacího obvodu při připojení napájecího napětí, čímž je potřeba externích součástek minimální. Obvody FT232RL mají v sobě také integrovanou paměť EEPROM, která může být použita pro uložení sériového čísla, popisu zařízení, nastavení USB VID,PID a nastavení programovatelných pinů. Paměť EEPROM se dá přeprogramovat i v aplikaci přes USB. Obvod se dá využít s použitím konvertoru úrovní jako převodník na RS232 nebo na RS485, ale když se připojí na klasický UART procesoru není konvertor potřeba, protože obvod FT232RL dokáže na základě svého vlastního napájení výstupních obvodů dodávat rozdílné logické úrovně (5V, 3,3V, 2,8V, 1,8V), takže může být použit v kombinaci s procesory napájenými ze zdroje 5 V i 3,3 V.

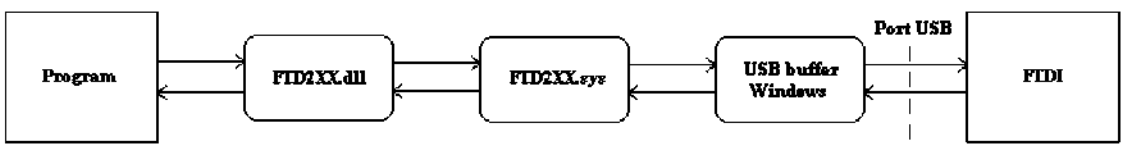
V katalogovém listu [1] obvodu FT232RL je několik možností zapojení s tímto obvodem. Pro účel aplikace jako převodníku bylo vybráno zapojení určené pro připojení k UART modulu procesoru, proto není potřeba konvertor úrovní. Napájení výstupních obvodů je v zapojení viz. [2] a je nastaveno na 5 V. Převodník je napájen přímo z portu USB počítače. Pro účel aplikace nejsou potřeba žádné programovatelné piny ani

nastavení pro hardwarové řízení toku. Jediné zapojené linky k procesoru jsou TXD, která je zapojena na RXD procesoru a RXD, která je propojena s TXD procesoru.

V obvodu FT232RL, jak je výše napsáno, je mnoho vlastností, které se musí programově nastavit. Jelikož je obvod určen k připojení na USB, dá se také přes port USB naprogramovat. K programování slouží program MPROG, který je k obvodu dodáván přímo od výrobce a je možné si ho stáhnout z výše uvedených stránek výrobce. V programu MPROG lze přes sběrnici USB naprogramovat všechny obvody od firmy FTDI Chip. Musí se v něm nastavit základní parametry, jako je použité napájení, typ obvodu nebo verzi sběrnice USB a může se nastavit ještě mnoho dalších volitelných nastavení. Potom už stačí jen nastavení potvrdit a obvod je připraven k použití.

1.2.3 Knihovna FTD2XX.dll – popis

Pro komunikaci se zařízením pomocí FT232RL můžeme použít ovladače VCP (Virtual Com Port) nebo použít ovladač D2xx. VCP je ovladač, který vytvoří virtuální sériový port a lze tak ovládat zařízení stejně, jako kdyby bylo připojeno na obyčejný fyzický sériový port. Zatímco D2xx je přímý ovladač pro Windows a umožňuje programu komunikovat s FT232RL pomocí knihovny funkcí, která se jmenuje FTD2XX.dll. Obsahuje také WDM ovladač FTD2XX.SYS, pomocí něhož komunikuje přímo s USB bufferem Windows. Komunikace probíhá tím způsobem, že program vytvořený v některém z vyšších programovacích jazyků (Delphi, C++.....), v tomto případě Delphi, přes aplikační rozhraní, komunikuje pomocí knihovny FTD2XX.dll s WDM ovladačem FTD2XX.sys, to je rozhraní ovladače, ten předává data do USB bufferu, Windows USB rozhraní a zajišťuje komunikaci se zařízením viz. Obr. 1.1.



Obr. 1.1: Struktura ovladače FTDI

Funkce ovladače D2xx jsou rozděleny do čtyř skupin:

1) Základní klasické funkce, to jsou nejzákladnější funkce pro snadný přístup na port USB.
2) Rozhraní EEPROM, které umožňuje zapisovat a číst konfigurační EEPROM, popřípadě využít místo k uložení specifických údajů pro naši aplikaci.
3) Funkce pro Bit Bang režim, to je režim, ve kterém sériový port funguje jako 8 samostatných linek.
4) FT_Win32 API, to jsou funkce odpovídající Windows API pro práci se sériovým portem.

Klasické funkce a funkce FT_Win32 API jsou plně kompatibilní a lze použít oba způsoby, výhodou API funkcí je, že se dají snadno použít pro již odladěný program pro sériový port. Musíme si však vybrat jen jeden, protože není doporučeno tyto dvě skupiny vzájemně míchat.

Každé zařízení má svůj handle, což je číslo Dword, které identifikuje dané zařízení a pomocí něj se lze na zařízení připojit a volat ho, k tomu nám slouží klasické funkce této knihovny. Pro připojení slouží funkce FT_open nebo FT_OpenEx, pro čtení a zápis dat pak FT_read a FT_write. Zařízeni se odpojí pomocí FT_close. K dispozici jsou i další funkce k ovládání jako například FT_SetTimeouts, která nastaví komunikační timeouty, FT_GetQueveStatus zjistí stav přijímací fronty a dá k dispozici počet přijatých B. Další je funkce zjišťující stav zařízení FT_GetSatus pro zjištění stavu zařízení a nebo FT_ResetDevice pro resetování USB zařízení.
Je k dispozici také funkce pro práci s vstupním a výstupním bufferem FT_Purge, která dovede vyprázdnit jeden z bufferů, podle zadaného parametru. Dalšími důležitými funkcemi jsou funkce pro nastavení parametrů přenosu. Jednou z nejdůležitějších je přenosová rychlost, kterou nastavíme pomocí FT_SetBaudRate. Můžeme si vybrat ze standardních přenosových rychlostí, které jsou definovány jako konstanty, např. FT_Baud_19200. Funkce s parametrem FT_Baud_19200 nastaví přenosovou rychlost na 19200 Baud. Pro nestandardní rychlosti slouží FT_SetDivisor. K nastavení parametrů přenášených dat použijeme FT_SetDataCharacteristics, pomocí níž se nastavuje počet datových bitů, počet stop bitů a parita. Můžeme nastavit i řízení toku dat pomocí funkce FT_FlowControl, ve které můžeme využít RTS,CTS nebo X-ON, X-OFF a DTR, DSR popřípadě řízení toku úplně vypnout. Dalším užitečným je samostatné nastavování DTR a DSR, ty se nastaví pomocí FT_SetDTR, FT_ClrDTR

a analogicky i pro DSR. Za zmínku stojí i FT_GetModemStatus, pro zjištění stavu modemu. Z výsledku lze pomocí logického součinu zjistit stav všech linek.

Definice několika důležitých funkcí:
Definice funkcí jsou součástí unity D2XXunit.pas, která využívá knihovnu D2XX.dll a je k dispozici na stránkách výrobce nebo je dostupná na přiloženém CD.

Všechny funkce vrací úspěšnost svého provedení hodnotou FT_Result. Když se funkce bez problému provede, vrátí hodnotu FT_OK, což je definovaná konstanta typu integer, která má hodnotu 0. V opačném případě vrátí jinou hodnotu a ta je posuzována s výčtem možných chyb viz [3].

function FT_GetNumDevices(pvArg1:Pointer; pvArg2:Pointer; dwFlags:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_ListDevices' ;

Po vykonání funkce se nám vrátí počet zařízení, které jsou připojeny na port USB. FT_GetNumDevices je základní funkce a bez jejího kladného výsledku se k zařízení nepřipojíme. Parametr *dwFlags:Word* může nabývat různých hodnot a tím určuje formát výsledné informace, možné hodnoty viz. [3] nebo [5].

Function FT_Open(Index:Integer; ftHandle:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_Open';
function FT_OpenEx(pvArg1:Pointer; dwFlags:Dword; ftHandle:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_OpenEx';

Funkce slouží pro otevření přístupu k zařízení, s tím že se nám vrátí handle zařízení pomocí něhož ho můžeme ovládat. Rozdíl mezi těmito funkcemi je takový, že FT_Open, neumí otevřít zařízení podle jména, naopak FT_OpenEx se proto hodí k otevření konkrétního zařízení.

function FT_Close(ftHandle:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_Close';

Slouží k uzavření zařízení pomocí jeho handle.

function FT_Read(ftHandle:Dword; FTInBuf:Pointer; BufferSize:LongInt; ResultPtr:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_Read';

Umožňuje číst data z přijímacího bufferu daného zařízení. Čeká na počet přijatých bajtů, který určíme parametrem BytestoRead nebo čeká na vypršení čtecího timeoutu, nastaveného pomocí SetTimeouts.

function FT_Write(ftHandle:Dword; FTOutBuf:Pointer; BufferSize:LongInt; ResultPtr:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_Write';

Umožňuje posílat data do vysílacího bufferru zařízení. Odesílá počet bajtů zadaných do parametru BytestoWrite nebo čeká na vypršení zapisovacího timeoutu, nastaveného pomocí SetTimeouts.

function FT_ResetDevice(ftHandle:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_ResetDevice';

Resetuje USB komunikaci zadaného zařízení.

function FT_SetBaudRate(ftHandle:Dword; BaudRate:DWord):FT_Result; stdcall; External FT_DLL_Name name 'FT_SetBaudRate';

Nastaví přenosovou rychlost daného zařízení: Vybrat můžeme z pevně daných rychlostí od 300 Bd do 921600 Bd podle [3] nebo [5].

function FT_SetDataCharacteristics(ftHandle:Dword; WordLength,StopBits,Parity:Byte):FT_Result; stdcall; External FT_DLL_Name name 'FT_SetDataCharacteristics';

Umí nastavit charakteristiky přenášených dat zařízení, délka datového slova buď 7 nebo 8 bitů, počet stop bitů 1 nebo 2 a paritu: nepoužita, lichá, sudá, značená, mezerová.

Function FT_SetDtr(ftHandle:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_SetDtr';
Function FT_ClrDtr(ftHandle:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_ClrDtr';

Nastavování linky DTR do log.1 a 0, což se dá využít např. k resetu komunikace.

function FT_GetModemStatus(ftHandle:Dword; ModemStatus:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_GetModemStatus';

Jak již bylo dříve popsáno, slouží ke zjištění stavu linek modemu z proměnné, která vrátí číslo typu Dword, můžeme zjistit pomocí logického součinu a daného symbolu viz. [4] stav linky, která nás zajímá.

function FT_Purge(ftHandle:Dword; Mask:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_Purge';

Dokáže vyprázdnit přijímací nebo vysílací buffer zařízení, závisí na parametru mask, za který se může dosadit hodnota FT_Purge_TX nebo FT_Purge_RX.

function FT_SetTimeouts(ftHandle:Dword; ReadTimeout,WriteTimeout:Dword):FT_Result; stdcall; External FT_DLL_Name name 'FT_SetTimeouts';

Nastavuje zařízení čtecí a zapisovací timeout, jakou dobu bude čekat zařízení na data. Číslo v proměnné Readtimeout a Writetimeout znamená počet milisekund, po které se čeká.

function FT_GetQueueStatus(ftHandle:Dword; RxBytes:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_GetQueueStatus';

Vrací počet bajtů uložených v přijímací frontě, tento údaj je dostupný jako FT_Qbytes a dá se použít jako parametr při čtení pomocí FT_read.

function FT_GetStatus(ftHandle:DWord; RxBytes,TxBytes,EventStatus:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_GetStatus';

Vrací aktuální stav zařízení a stav příchozí a odchozí fronty tj. RxQueve, TxQueve, EventStatus. Využívá se spolu s funkcí FTJSetEventNotification pro paměť EEPROM.

function FT_GetDeviceInfo(ftHandle:DWord; DevType,ID,SerNum,Desc,pvDummy:Pointer) : FT_Result; stdcall; External FT_DLL_Name name 'FT_GetDeviceInfo';

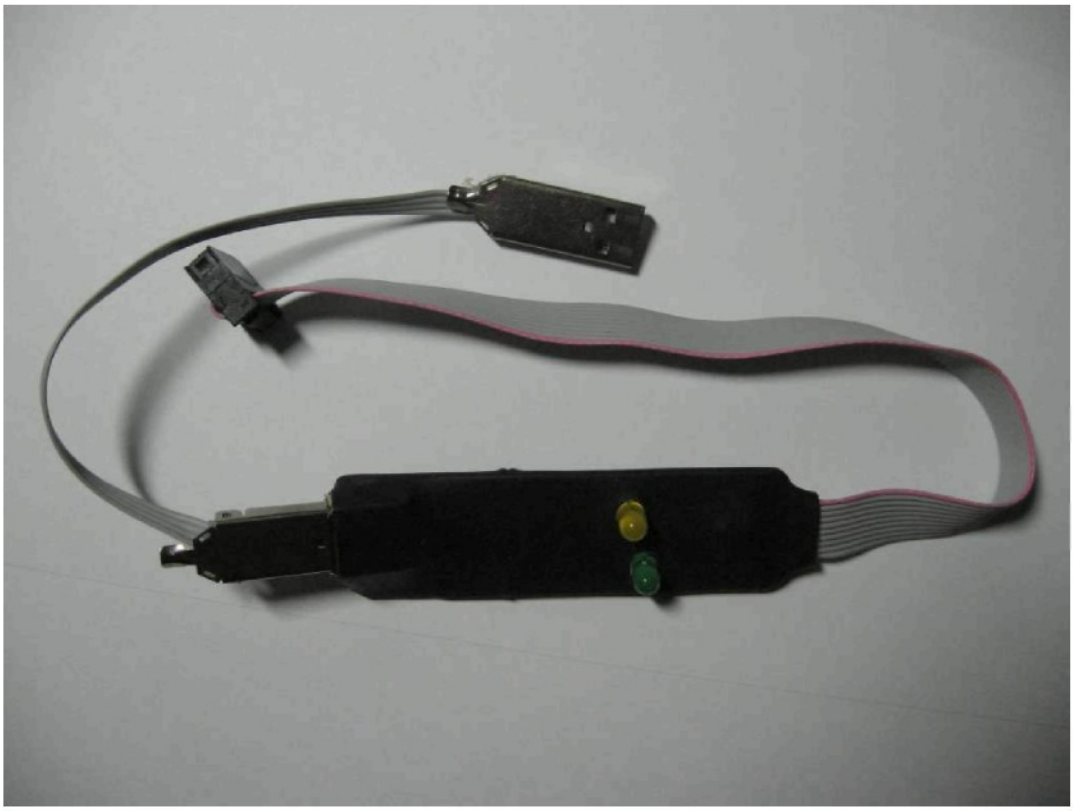
Zjistí podrobnosti o zařízení, jako je jeho název, handle, sériové číslo a jiné rozšířené informace.

function FT_GetDriverVersion(ftHandle:Dword; DrVersion:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_GetDriverVersion';
function FT_GetLibraryVersion(LbVersion:Pointer):FT_Result; stdcall; External FT_DLL_Name name 'FT_GetLibraryVersion';

Tyto dvě funkce slouží k zjištění verze ovladače zařízení a verze knihovny D2XX.dll.

1.2.4 Převodník USB⇔RS232 s FT232RL

Převodník slouží ke komunikaci s měřící deskou.



Obr. 1.2: Převodník

1.2.5 Měřící deska

Ukázka měřící desky, na které probíhalo měření.

Obr. 1.3: Měřící deska

2.Vyhodnocovací aplikace

2.1 Základní požadavky

Pracuje se zde s hodnotami naměřenými měřící deskou, které je potřeba přijmout, uložit a zpracovat. K práci s hodnotami byla vytvořena počítačová aplikace Vyhodnocení dat z tachometru. K základním funkcím aplikace patří zjistit počet zařízení, která jsou připojena k počítači a zobrazit je do seznamu, po kliknutí na zvolené zařízení je umožněno nastavit přenosové parametry.

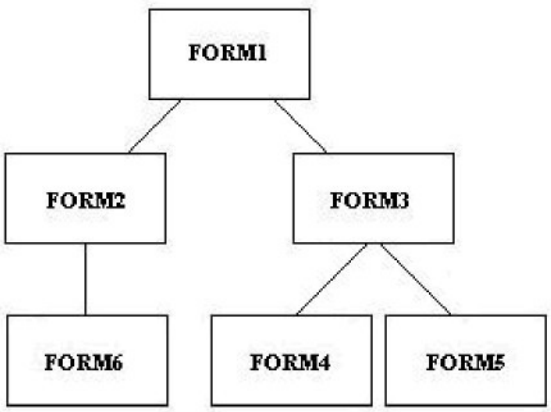
Vytvořená aplikace je určena pro používání ve Windows XP a je kompletně naprogramovaná v programu Delphi 7. Měřící deska se připojuje k počítači přes sběrnici USB, což umožňuje obvod FT232RL. Obvod slouží měřící desce jako rozhraní mezi počítačem a deskou. Umožní z pohledu počítače převést USB port na virtuální sériový port, se kterým se pracuje jako s obyčejným sériovým portem. V tomto případě jsou používány knihovny Windows API, jsou dostupné i knihovny pro obsluhu sériového portu, ale použitím API funkcí navržených přímo pro Windows se stane program spolehlivějším a robustnějším.V konečné verzi vyhodnocovací aplikace je však využit přímý přístup na port USB, pomocí ovladačů dodávaných od výrobce FT232 RL. V nich se využívá vstupní a výstupní fronta naměřených dat na portu USB a předává data do virtuálních bufferů v paměti. Z nich si program vyzvedne přesný počet údajů, které byly změřeny a ukládá si je do dynamického pole jako položky typu Tvzorek, které obsahují hodnoty rychlosti, otáček a času měření. Struktura dat je jednoduchá a využívá ukládání do souboru(*.csv), který je navržen pro ukládání naměřených hodnot z měření. Tento typ souboru se skládá ze seznamu hodnot, které jsou odděleny středníkem. Popřípadě je druhá možnost, kdy se data dají ukládat jako textový soubor (*.txt), ve kterém jsou naměřené hodnoty odděleny mezerou. Kromě editace ve vyhodnocovací aplikaci, lze také soubory otevřít v programu pro úpravu textových souborů, např. Notepad. Aplikace pracuje jen s dvěma typy souborů (*.txt, *.csv). Na ně je nastaven filtr pro otevírané i ukládané soubory. Jako další výstup je k dispozici možnost tisku kompletního přehledu zaznamenaných hodnot, a to jak načtených ze souboru, tak stažených z měřící desky. Pro grafické vyhodnocení aplikace zobrazuje graf charakteristiky rychlosti a otáček v čase, které se mění od nuly do maxima hodnot. Jeho zobrazení v čase se dá zrychlit nebo zpomalit až 4x. Pro průběh ze zaznamenaných dat měření v reálném čase se postupně zobrazují body podle toho, v jakém okamžiku ze zařízení dorazí. Samozřejmě se dá graf vytisknout nebo uložit

do souboru (*bmp, *.jpg). Jako další dobrá pomůcka je k dispozici zoomování u grafu, které přiblíží požadovanou oblast grafu v 16ti krocích.

2.2 Pracovní prostředí

Program byl navrhován tak, aby byl co možná nejjednodušší na ovládání. Dělí se na dvě hlavní části. V první části se nastavuje komunikace a vše ostatní kolem přenosu dat. V druhé části se provádí rekonstrukce jízdy. Celá aplikace obsahuje šest formulářů, které jsou rozděleny tímto způsobem:

1) Formulář Form2 slouží pro nastavení komunikace a stahování dat.
2) Form3 je pro zrekonstruování jízdy a nastavení s tím související.
3) Form4 obsahuje komponenty tachometr a otáčkoměr, které dokáží zobrazovat aktuální hodnoty rychlosti a otáček a slouží jako imitace palubní desky.
4) Form5 byl vytvořen jako panel pro statistické výpočty a jejich zobrazení.
5) Form6 byl vytvořen kvůli potřebě tisku a jeho formátování, proto je využit jako náhled tisku hodnot.
6) Form1 je hlavní formulář, který do sebe slučuje Form2 a Form3 a obsahuje hlavní nabídku programu.



Obr. 2.1: Kostra programu

Všechny zdrojové kódy, z kterých se aplikace skládá jsou komentovány a nacházejí se na přiloženém CD.

2.2.1 Hlavní formulář aplikace

Hlavní formulář zaštiťuje a spojuje dva důležité formuláře a to Form2 a 3, jak je uvedeno na předchozí stránce. Formulář Form1 obsahuje komponenty hlavní nabídka MainMenu1, komponentu PageControl1, která se chová jako záložky a tím umožní přepínání mezi Form2 a 3. Nakonec na něm ještě najdeme dialogy pro otevření, ukládání, tisk a jako poslední komponentu Statusbar.

Po spuštění, se nejdříve vytvoří všechny formuláře. Havní formulář má za úkol nastavit svoji počáteční pozici na ploše a přizpůsobit se podle velikosti formuláře, který se bude zobrazovat na zvolené stránce PageControl1 a vždy, když se aktuální záložka změní, procedura se znovu provede a rozměry se přepočítají (viz Unit1.pas-příloha).

Další funkce jsou ukryty v hlavní nabídce. V menu Soubor můžeme najít položky Otevřít, která nám dovolí otevřít soubor s dříve uloženými hodnotami. Ty jsou ve formátu txt nebo csv, u kterých aplikace rozpozná typ a podle něho je načte do pole, čímž jsou data připravena na rekonstrukci jízdy. Otevření je indikováno v titulku okna zobrazením názvu souboru. Další položky jsou Uložit a Uložit jako, které uloží provedené změny v souboru, popřípadě uloží soubor jako nový soubor s jiným názvem. Poslední položkou je Zavřít, ta nám umožní korektně zavřít soubor. Když je již soubor otevřen, není možno otevřít jiný soubor nebo začít stahovat data. Je proto potřeba soubor vždy před novým měřením nebo otevřením jiného souboru zavřít. Poznáme to tak, že zmizí název souboru z titulku okna a vyprázdní se pole dat. Poslední je položka Konec, která ukončí celý program.

Obr. 2.2: Menu Soubor

Další položkou hlavní nabídky je Tisk data, který se skládá ze dvou možností. Buď tisk dat ze souboru nebo tisk dat z měřící desky. Řešení je zvoleno z toho důvodu, protože pro tisk dat z měřící desky musí být zařízení připojeno k počítači.

Obr. 2.3: Menu Tisk data

Jako předposlední je Export data, ve kterém je možno vybrat způsob exportu dat do souboru typu *.txt nebo *.csv. Rozdíl je v tom, že soubor txt má jednotlivá data na řádku oddělená mezerou a soubor csv má data na řádku oddělená středíkem.

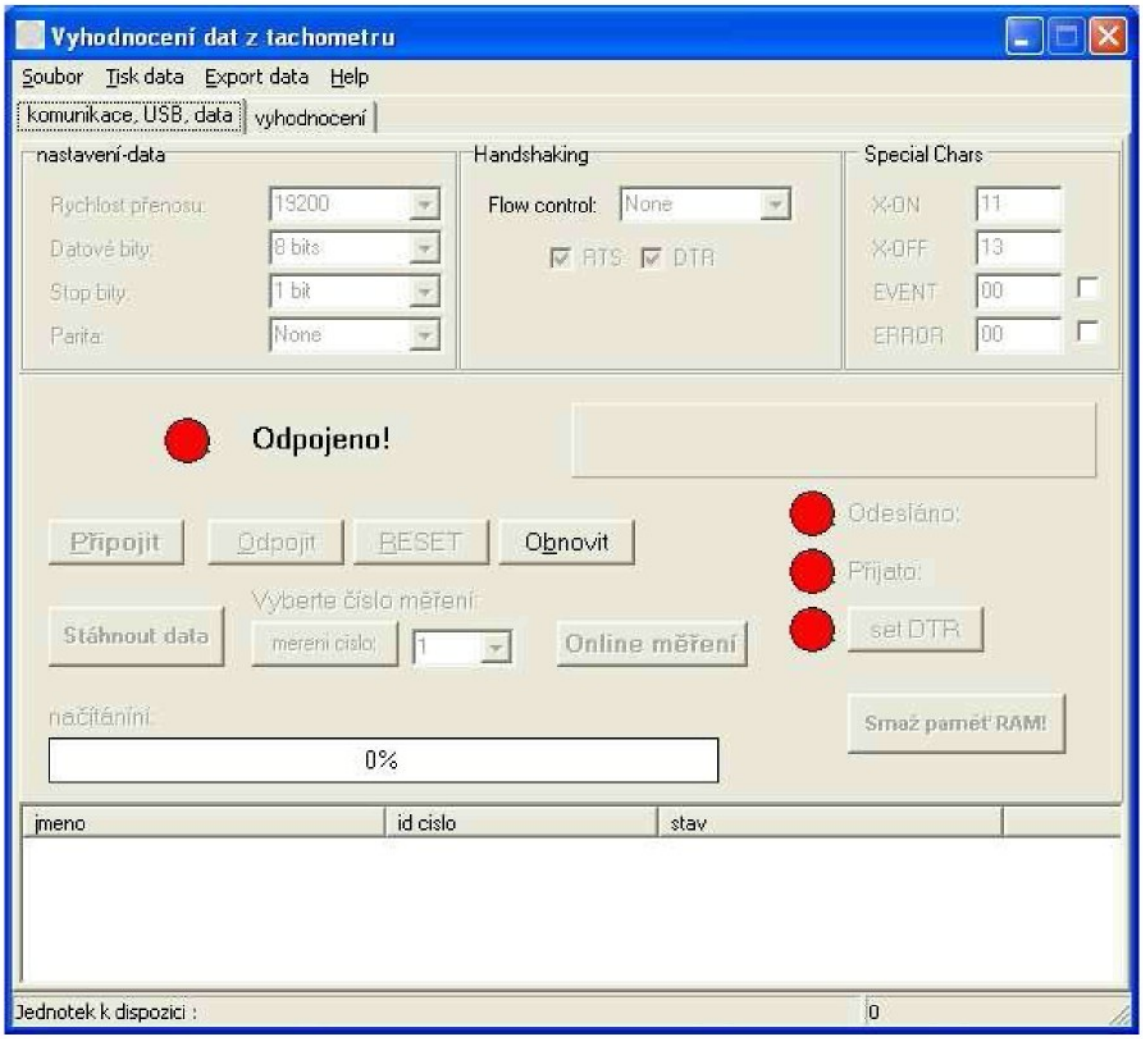
Obr. 2.4: Menu Export data

Poslední položka Help obsahuje informace o aplikaci, k čemu je aplikace určena a její verzi.

Obr. 2.5: Menu Help

2.2.2 Okno komunikace a stahování dat

Okno komunikace a stahování dat se zabývá nastavením komunikace, samotným připojením k měřící desce a stažením dat nebo zapnutí online měření. Po spuštění programu vypadá jeho okno jako na obrázku Obr.2.6.



Obr. 2.6: Okno komunikace

Okno komunikace a stahování dat je rozděleno do několika částí, které se liší svým účelem. V horní části je roletové menu, ve kterém jsou funkce již dříve popsané v hlavním okně aplikace. V dolní části okna na Statusbaru vidíme kolik je aktuálně připojených jednotek k portům USB v počítači, toto se zajišťuje v programu následujícím způsobem.

Ukázka zjištění počtu připojených jednotek (viz. Uni2.pas):

```
GetFTDeviceCount;                    //knihovní funkce FTDI
poc_jedn:=inttostr(FT_Device_Count); //převedení tohoto čísla na text
Form1.StatusBar1.Panels[0].Text:='Jednotek k dispozici :';
Form1.StatusBar1.Panels[1].Text:=poc_jedn; //zobrazení v panelu
```

Když je připojena alespoň jedna měřící deska, nastává další krok, který zobrazí postupně všechna zařízení:

Ukázka zobrazení připojených jednotek (viz Unit2.pas):

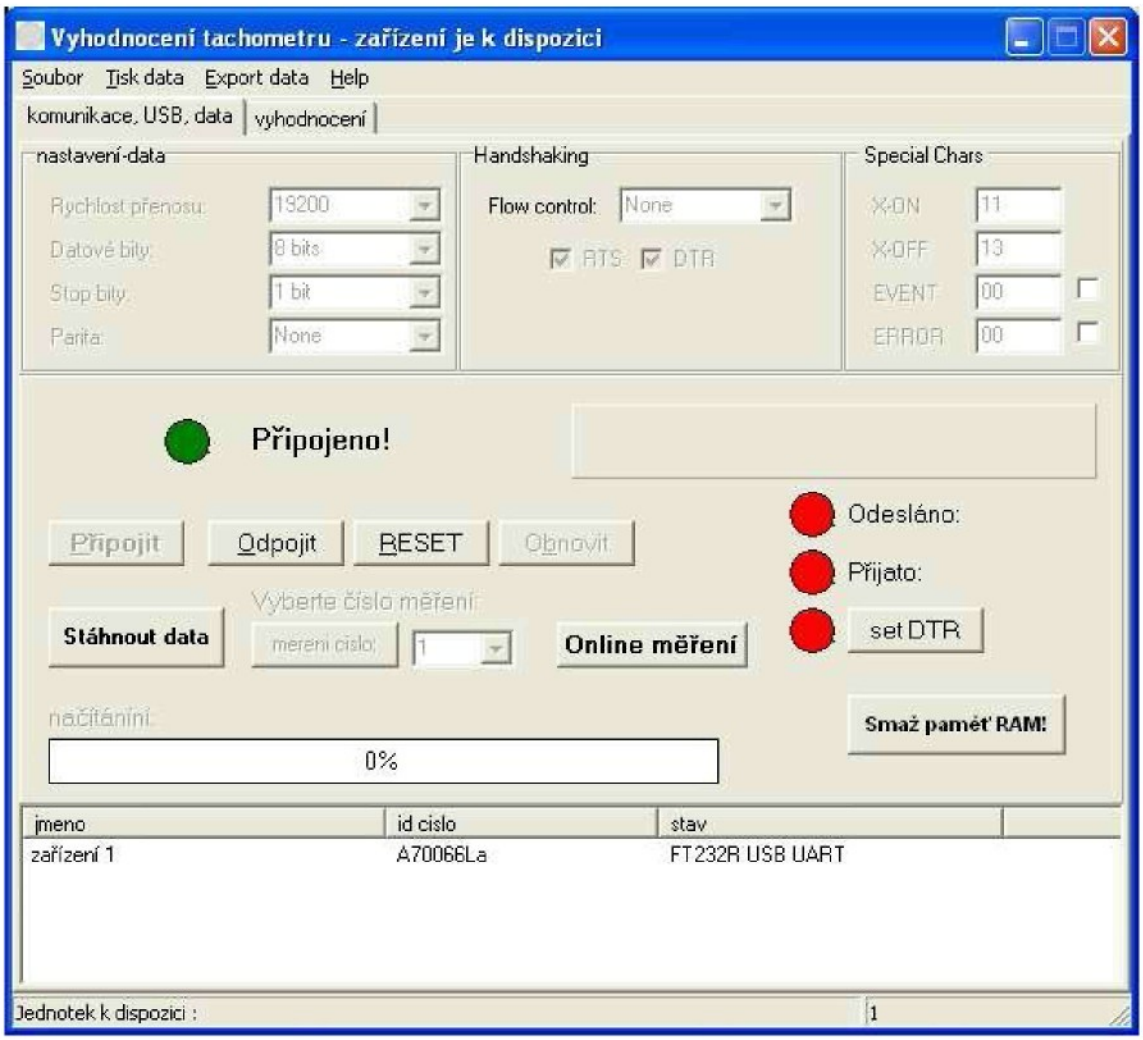
```
index_jedn:=0;
 if FT_Device_Count > 0 then
 begin
  for i:=1 to FT_Device_Count do
  begin
  LV:=LV_zarizeni.Items.Add;
  LV.Caption:='zařízení '+inttostr(i);
  GetFTDeviceSerialNo( index_jedn ); //zjistí sériové číslo zařízení
  LV.SubItems.Add(FT_Device_String);
  GetFTDeviceDescription( index_jedn );      //zjistí popis zařízení
  LV.SubItems.Add(FT_Device_String);
  index_jedn:=index_jedn + 1;
  end;
 end;
```

Zdrojový kód používá komponentu Listview, do které postupně přidává zařízení a pomocí knihovny unity D2XXUnit.pas si zjišťuje jejich sériová čísla a popis. Tento údaj je velice důležitý pro další postup v připojení. Když není k dispozici žádné zařízení nebo se nedají zjistit rozšířené údaje o zařízení, není umožněno se k němu připojit.

Pokud je zařízení připojeno do portu USB, a přesto se v nabídce nezobrazí, je třeba kliknout na tlačítko Obnovit, které provede opětovné zjištění počtu jednotek a jejich výpis. Jestliže je v seznamu zobrazeno alespoň jedno zařízení, lze se na něj připojit kliknutím na jeho název, to povolí tlačítko Připojit a také nastavení parametrů přenosu dat. Lze nastavit parametry podle potřeby, jako základní jsou nastaveny takto: přenosová rychlost 19200 Baud, počet datových bitů 8, počet stop bitů 1, parita none.

Původní hodnoty parametrů se obnoví při resetování komunikace tlačítkem Reset. Handshaking není povolen. Pokud je ale vyžadován, není problém nastavit i jeho parametry a provozovat přenos dat včetně něho. Hodnoty, které lze nastavovat jsou v rozsahu klasických parametrů sériového portu, přenosové rychlosti od 1200 do 921600 Baud, datové bity 8 nebo 9, stop bity 1 nebo 2 a parita none, odd, even, mark a space. Jestliže jsou všechny parametry nastaveny podle potřeby, je možno kliknout na tlačítko Připojit.

Po kliknutí na tlačítko Připojit si nejprve otestuje, jestli je vybraná jednotka k dispozici. Pokud je jednotka k dispozici, uplatňují se postupně jednotlivá již dříve popsaná nastavení přenosu dat a povolí se ostatní tlačítka, přičemž se tlačítko Připojit zakáže, nastaví se stav komponentě Tkontrol na zelenou barvu a v labelu se změní nápis z „Odojeno!" na „Připojeno!", jak je možné vidět na Obr. 2.7.



Obr. 2.7: Okno komunikace – připojeno

V tomto okamžiku je možno stahovat data tlačítkem Stáhnout data, které si od desky vyžádá počet měření. Počet měření se nastaví do comboboxu pro číslo měření Obr. 2.8. Po tom, co vybereme jedno z měření, vyšle program požadavek s tímto číslem do desky a začnou se přijímat data. Průběh operace je zobrazen pomocí processbaru a doprovázen komentářem na panelu. Po přijmutí se data roztřídí a uloží do pole hodnot, kde jsou připravena pro další využití. Pokud deska neobsahuje žádné měření, nastaví se do comboboxu číslo 0 a tlačítko Měření číslo zůstane zakázáno.

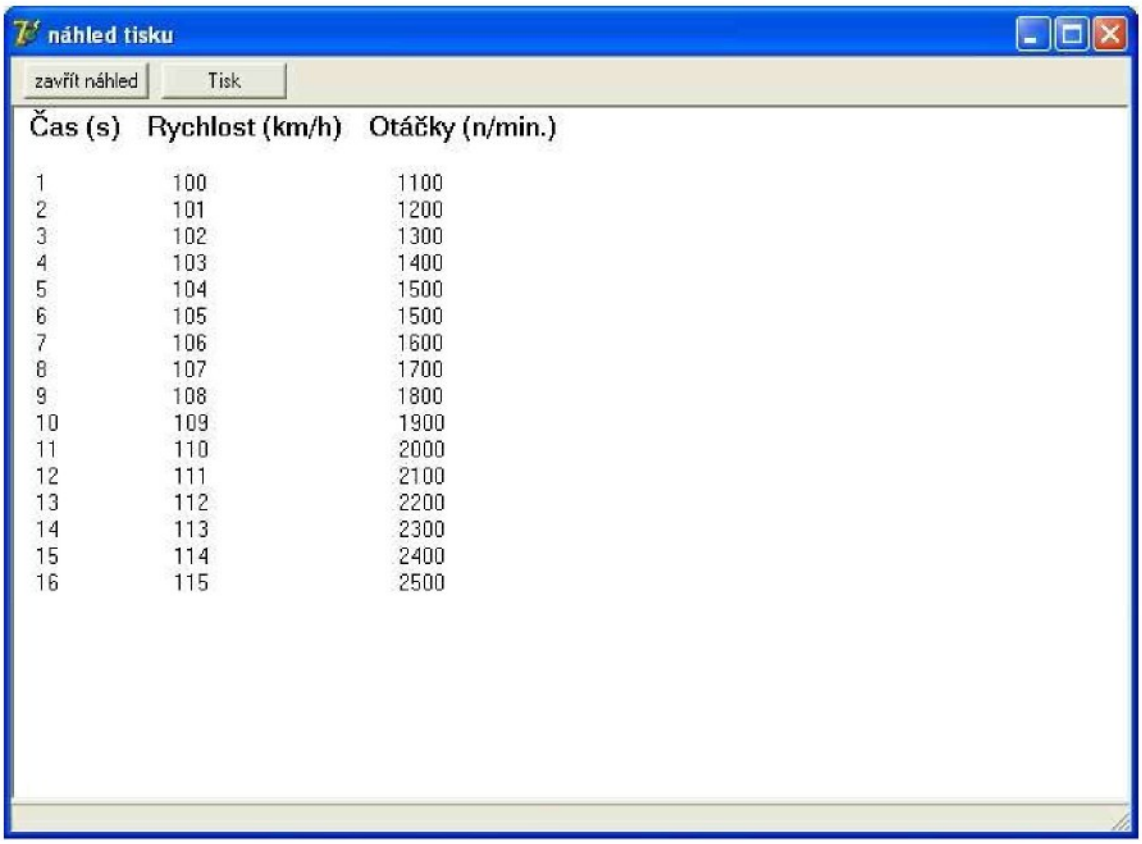
Obr. 2.8: Stahování dat

Další možností je měření v reálném čase. Měření se spustí stisknutím tlačítka Online měření, které vždy čeká na změřené hodnoty od desky. Změřené hodnoty si ukládá opět do pole a tyto hodnoty jsou ihned k dispozici k zobrazení. Online měření využívá Timer2, který provádí vždy v pevně nastaveném intervalu kontrolu vstupního bufferu na portu USB a čeká, dokud nedostane od zařízení aktuální hodnotu, kterou si uloží. Tuto činnost provádí, dokud není ukončen tlačítkem Odpojit nebo Reset.

Všechny přenosy dat lze zkontrolovat na ukazatelích Obr. 2.9, které signalizují správnost odeslaných a přijatých dat a jejich počet, lze též nastavit nebo vynulovat linku DTR, která provede reset komunikace s procesorem. Jako poslední je možnost smazat uložená měření z desky. Smazání všech uložených měření z desky se provede stiskem tlačítka Smaž paměť RAM, které pošle desce příkaz ke smazání všech měření a nastaví počet měření na 0.

Obr. 2.9: Kontrola přenosu dat

Data je možno tisknout buď po stažení nebo po otevření souboru, když jsou uložena v poli hodnot programu. K tisku slouží tlačítko Tisk data v hlavním menu programu. Zde je možno vybrat si, jak již bylo napsáno, příslušnou možnost, přičemž se vytvoří z příslušných dat náhled tisku viz Obr. 2.10, kde si je možno prohlédnout celou výstupní sestavu a pokud je vyhovující, tak tisk potvrdit nebo ho stornovat.



Obr. 2.10: Náhled tisku

Pro kontrolu, zda je zařízení připojeno k aplikaci, slouží Timer1 a jeho procedura onTimer, kde se každých 500 ms kontroluje stav zařízení. Když je zařízení připojeno a odpovídá, nastaví procedura onTimer do titulku hlavního formuláře kromě názvu aplikace ještě dodatek „ zařízení je k dispozici ". Když zařízení neodpovídá nebo je odpojeno, provede se automatické odpojení a program se nastaví do počátečního stavu. Procedura onTimer však probíhá jen pokud se zařízení nepoužívá ke stahování dat nebo online měření. Pokud jsou například stahována data, není procedura onTimer vůbec povolena a spouští se až po vykonání celé operace a v průběhu si tuto činnost hlídá samotná knihovna FTDI pomocí chybových hlášení.

K ukončení práce se zařízením slouží tlačítka Odpojit a Reset. Tlačítko Odpojit pouze provede samotné odpojení zařízení a vypne Timer1 pro ověřování jeho dostupnosti. Tlačítko Reset provede nejen samotné odpojení zařízení a vypne Timer1 pro ověřování jeho dostupnosti, ale ještě navíc nastaví vlastnosti komunikace do výchozího stavu. Je možné komunikaci ukončit také odpojením zařízení z konektoru USB, program pak zařízení odpojí automaticky.

2.2.3 Komunikace aplikace s deskou

Pro komunikaci programu s měřící deskou slouží jednoduchý model. Jestliže chce program kontaktovat měřící desku a zjistit, zda jsou k dispozici data, odešle do výstupního bufferu USB znak „A“, což odpovídá v ascii hodnotě 65. V programu úkol řeší tlačítko Stáhnout data. Na výzvu odpoví deska tím, že odešle hlášení, které obsahuje informace o tom, zda jsou nějaká naměřená data k dispozici. Když data k dispozici jsou, obsahuje zpráva ještě požadavek jestli si uživatel přeje data stáhnout. Požadavek uživatel potvrdí tím, že odešle znak „B“, odpovídá hodnotě 66 a v programu ho provádí tlačítko Měření číslo. Po stisknutí tlačítka Měření číslo deska začne posílat data, dokud neodvysílá celý obsah své paměti, v případě této desky 16 kB dat. Smazání všech měření se provádí pomocí tlačítka Smazat paměť RAM, které odesílá znak „C“, odpovídá hodnotě 67.

Pro uložení dat v paměti slouží dynamické pole Poledat, které je použito kvůli rozdílnému počtu naměřených hodnot při měřeních. Jeho prvky jsou typu Tvzorek, což je třída, která je vytvořena pro uložení stažených hodnot a má následující strukturu.

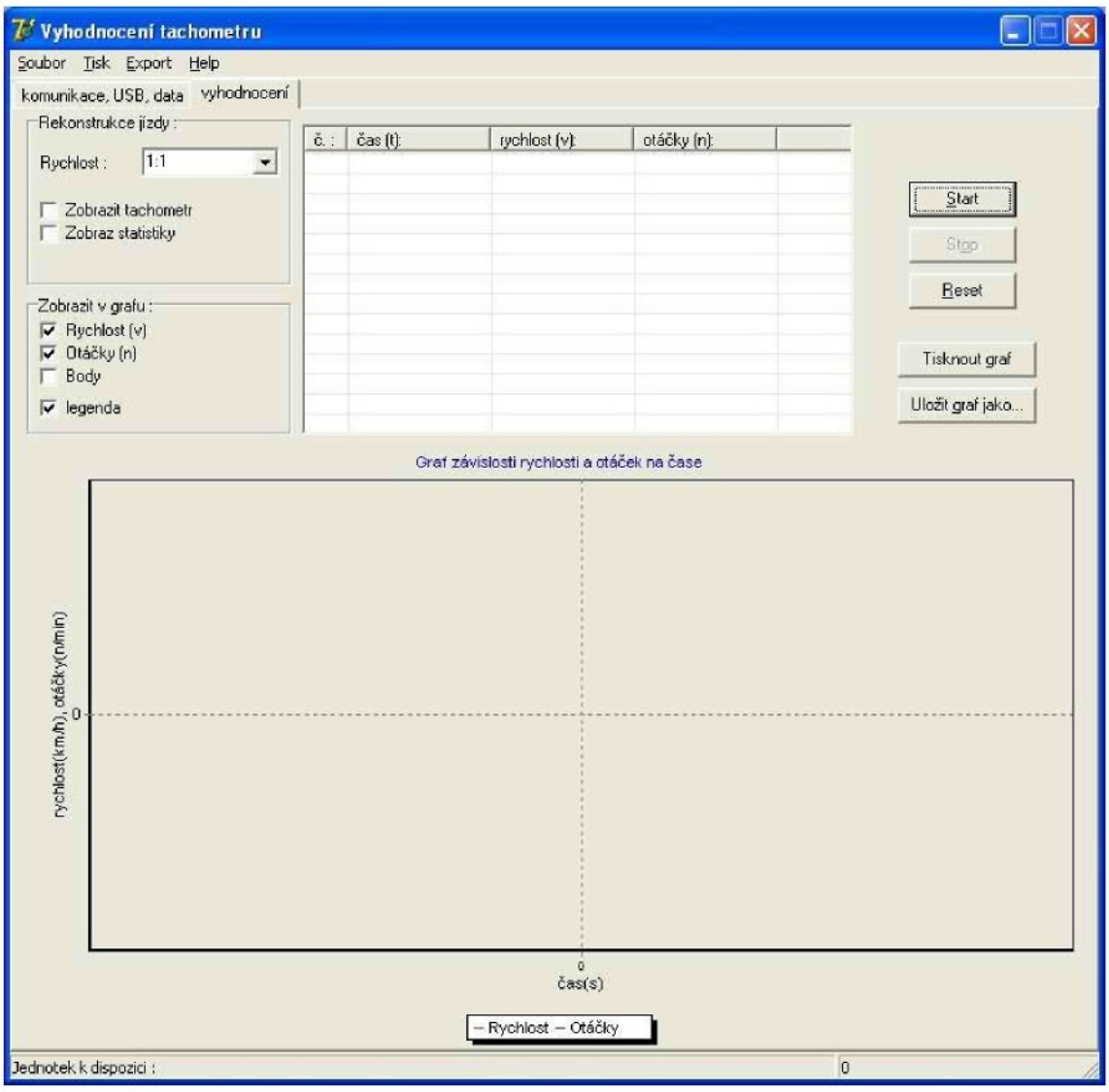
Ukázka definice třídy Tvzorek (Unit3.pas):

```
Tvzorek = class
  rychlost:real;
  otacky:real;
  cas:real;
  constructor Init(rychlost,otacky:integer;cas:longint);
  end;
Tdata = array of Tvzorek; //dynamické pole vzorků
```

Do dynamického pole jsou hodnoty tříděny ze vstupního bufferu USB, vždy první hodnota je celá část hodnoty otáček, za ní následuje desetinná část hodnoty otáček, pak celá část rychlosti a desetinná část rychlosti. Čtveřice hodnot pokračují v tomto pořadí až do konce měření. Začátek a konec měření jsou signalizovány hodnotou 255.

2.2.4 Okno rekonstrukce jízdy

V okně rekonstrukce jízdy se nastavují parametry rekonstrukce jízdy automobilu a probíhá zde grafické zobrazení hodnot uložených v poli naměřených hodnot aplikace. Z okna rekonstrukce jízdy se dále dají zapnout nebo vypnout další rozšiřující ukazatele, jako například okno tachometru, které imituje palubní desku automobilu nebo okno statistik.



Obr. 2.11: Okno rekonstrukce jízdy

Jak můžeme vidět na obrázku Obr. 2.11, skládá se okno rekonstrukce jízdy z několika částí. Vlevo nahoře je panel Rekonstrukce jízdy, na němž lze nastavit rychlost s jakou bude rekonstrukce probíhat v reálném čase. Tento parametr je možno nastavit v rozmezí 1:4 až 4:1, přičemž defaultně je nastavena rychlost 1:1. Lze ho ale nastavit jen v případě, že neprobíhá online měření. Dále je zde možnost nastavit ještě viditelnost okna tachometru a okna statistik. Pod panelem Rekonstrukce jízdy se nachází panel pro nastavení vlastností grafu, povoluje se zde zobrazení průběhů otáček a rychlosti, které mohou být zobrazeny současně nebo každá zvlášť. Dále je možno povolit nebo zakázat zobrazení bodů na průběhu a jako poslední je zobrazení legendy, která je zobrazena dole pod grafem. Vedle panelu Zobrazit v grafu jsou zobrazována data. K tomuto účelu se používá komponenta Listwiev, ve které jsou postupně zobrazeny sloupce dat.
Pořadí sloupců je následující:

a) je zobrazeno vždy číslo měření
b) čas (t) v sekundách
c) rychlost (v) v kilometrech za hodinu
d) otáčky (n) v jednotkách otáčky za minutu

Úplně vpravo jsou tlačítka, kterými se ovládá spouštění a zastavení a tlačítko Reset, které nejen že zastaví rekonstrukci, ale uvede i všechny změněné komponenty do původního stavu, jako je na obrázku Obr. 2.11.

Pod tlačítky se ještě nachází Tisknout graf a Uložit graf jako. Jak již název napovídá slouží první tlačítko k tisku grafu. Graf se vytiskne podle nastavení na celou stránku s 5px odsazením od všech okrajů a orientace je nastavena na hodnotu Landscape (horizontálně). Druhé tlačítko umožní uložit graf jako bitmapu, která je nastavena jako první volba v nabídce Savedialogu, pomocí funkce grafu SaveToBitmapFile nebo do jiného formátu pomocí SaveToMetaFile.

Ukázka zdrojového kódu uložení grafu do obrázku (viz. Unit3.pas):

```
//uloží graf jako obrázek u nebmp se musí zadat přípona
if SaveDialog1.Execute then
 begin
 if SaveDialog1.FilterIndex = 1 then
  begin
  jmn:=SaveDialog1.FileName+'.bmp';
  Chart1.SaveToBitmapFile(jmn);
  end else
  Chart1.SaveToMetafile(SaveDialog1.FileName);
 end;
```

Další užitečnou funkcí, kterou lze v grafu použít je zoom, kterým se dá průběh přiblížit podle potřeby a přiblížení probíhá podle nastavení v 16ti krocích animovaně. Samozřejmě se dá vytisknout nebo uložit i detail grafu, protože se pracuje s tím, co je vidět právě na plátně.

Celé zobrazování grafu probíhá pomocí komponenty Timer. V její proceduře onTimer je umístěn následující zdrojový kód, který řídí zobrazování průběhů a vkládání dat do komponenty ListView.

Zdrojový kód zajistí, že se zobrazí v grafu Chart1 a v ListViev lv_hodnoty všechny hodnoty, které jsou uloženy v poli s názvem Polehodnot s tím, že se bude zobrazovat popis na časové ose každých deset sekund. Zároveň se hodnoty budou zobrazovat na komponentách Totackomer1 a Ttacho1 jako aktuální hodnoty.

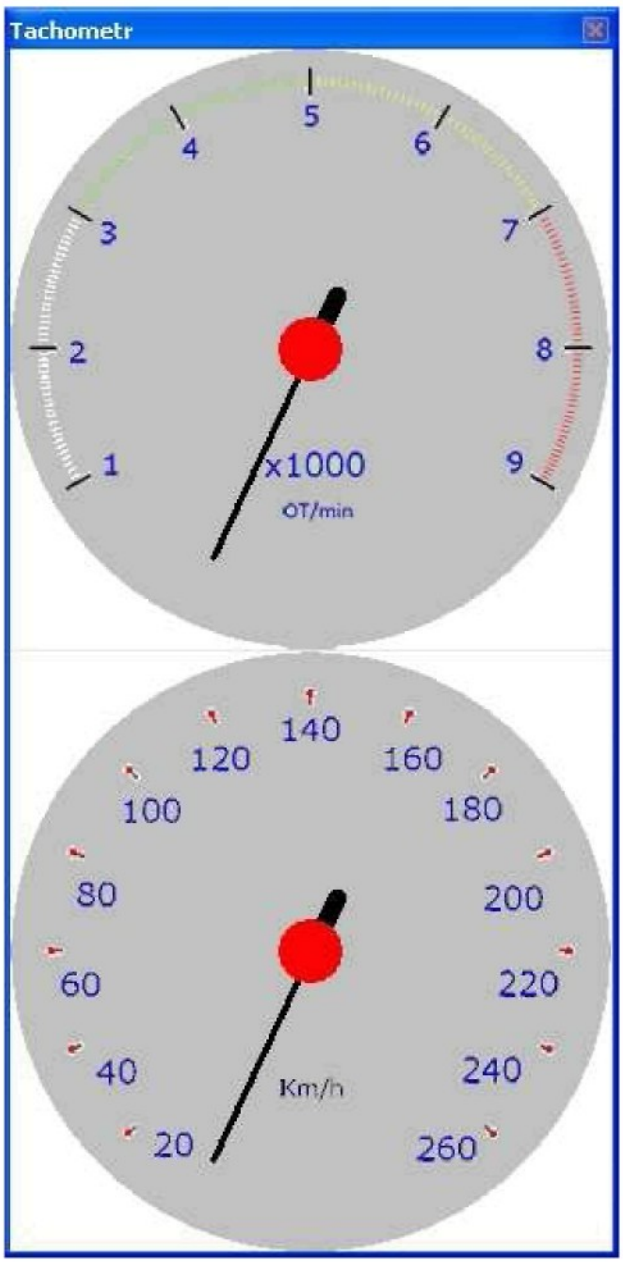
Rychlost rekonstrukce je možné měnit v jakémkoli okamžiku, když se změní hodnota rychlosti na panelu rekonstrukce, proběhne procedura onChange této komponenty a změní se tak i interval časovače.

Ukázka zdrojového kódu zobrazení hodnot do grafu (viz. Unit3.pas):

```
if citac<high(polehodnot)+1 then
 begin
  f_rych:=StrToFloat(IntToStr(polehodnot[citac].rychlost));
  s_cas:=inttostr(polehodnot[citac].cas);
  f_ot:=StrToFloat(IntToStr(polehodnot[citac].otacky));
  if citac mod 10 = 0  then
  begin
   with Series_rychlost do Add(f_rych,s_cas,clred);
   with Series_otacky do Add(f_ot,s_cas,clblue);
  end else
  begin
   with Series_rychlost do Add(f_rych,'',clred);
   with Series_otacky do Add(f_ot,'',clblue);
  end;
  Form4.totackomer1.aktual_otacky:=f_ot;
  Form4.tacho1.aktual_rychlost:=f_rych;
  lv_vys:=lv_hodnoty.Items.Add;
  lv_vys.Caption:=inttostr(citac);
  lv_vys.SubItems.Add(s_cas);
  lv_vys.SubItems.Add(floattostr(f_rych));
  lv_vys.SubItems.Add(floattostr(f_ot));
  citac:=citac+1;//posun na dalsi polozku
  Chart1.Repaint;
 end;
```

2.2.5 Okno tachometru

Okno tachometru slouží k zobrazování aktuálních hodnot rychlosti a otáček. Imituje tím palubní desku automobilu. Okno tachometru (viz. Obr. 2.12) je tvořeno formulářem Form4, na kterém jsou umístěny komponenty Ttacho a Totackomer. Komponenty Ttacho a Totackomer byly vytvořeny pro potřebu aplikace a podrobně jsou popsány v následující kapitole.



Obr. 2.12: Okno tachometru

2.2.6 Komponenty Ttacho a Totackomer

Komponenty Ttacho a Totackomer, jsou komponenty vytvořené pro účel aplikace. Obě komponenty jsou téměř stejné, liší se pouze v malých detailech, jako je například obrázek na pozadí. Jejich základem je komponenta TCustomPanel. Na panelu je vytvořena komponenta typu TImage, na jejíž plátno se vykresluje zadané pozadí a ukazatel „ručička". Ručička je vlastně pole bodů, které jsou vždy přepočítány pomocí transformačních matic a pospojovány pomocí funkce LineTo. Aktuální hodnota se posílá komponentám pomocí parametru, z kterého se spočítá rozdíl mezi minulou a aktuální hodnotou a výsledek se použije jako hodnota o kolik stupňů se má ručička otočit. Otočení se provádí pomocí funkcí rotace a transformace. Tyto funkce jsou definovány v unitu Unit_transf.pas, který je na přiloženém cd. Nachází se ve složce komp, kde jsou i celé zdrojové kódy komponent včetně souboru zdrojů a obrázků pozadí.

Ukázka nastavení polohy ručičky komponenty Ttacho (viz. Tacho.pas):

```
procedure Ttacho.vstup(const Value: real);
var
rozdil:real;//rozdil mezi minulou a aktualni hodnotou na tachu
begin
rozdil:=25;
aktualni:=1.13*(value);
rozdil:=aktualni-minula;
transformace(rotace(rozdil));
minula:=aktualni;
end;
```

Vstupní procedura, která obdrží aktuální hodnotu od programu a postará se o otočení ručičky o hodnotu proměnné Rozdil.

Zdrojový kód unity pro transformaci, ta je pro komponenty Ttacho i Totackomer společná (viz. Unit_transf.pas):

```
unit unit_transf;
interface
type Matice = array [1..3,1..3] of Real;
    Vektor = array [1..3] of Real;

Function JednotMat:Matice;
Function Rotace (Uhel: Real):Matice;
Function Posun (dx,dy: Real):Matice;
Function Nasob (Vek: Vektor; Mat: Matice):Vektor;

implementation

Function Nasob (Vek: Vektor; Mat: Matice):Vektor;
begin
 Result[1] := Vek[1]*Mat[1,1] + Vek[2]*Mat[2,1] + Vek[3]*Mat[3,1];
 Result[2] := Vek[1]*Mat[1,2] + Vek[2]*Mat[2,2] + Vek[3]*Mat[3,2];
 Result[3] := Vek[1]*Mat[1,3] + Vek[2]*Mat[2,3] + Vek[3]*Mat[3,3];
end;

Function JednotMat:Matice;
var I,J : Integer;
begin
 for I:=1 to 3 do
  for J:=1 to 3 do
   if J=I then
    Result[I,J] := 1
   else
    Result[I,J] := 0;
end;

Function Rotace (Uhel: Real):Matice;
var P : Real;
begin
  P:=Pi*Uhel/180;
  Result := JednotMat;
  Result[1,1] := cos(P);
```

```
  Result[1,2] := sin(P);
  Result[2,1] := -sin(P);
  Result[2,2] := cos(P);
end;
Function Posun (dx,dy: Real):Matice;
begin
  Result := JednotMat;
  Result[3,1] := dx;
  Result[3,2] := dy;
end;
end.
```

2.2.7 Okno statistiky

Okno statistik v této aplikaci slouží k zobrazení ukazatelů, jako jsou aktuální spotřeba, ujetá dráha a maximální hodnoty rychlosti a otáček. Tyto údaje jsou průběžně aktualizovány a přepočítávány. Po skončení rekonstrukce jízdy se doplní i celkové ukazatele, což je celková ujetá dráha a celková spotřeba.

Obr. 2.13: Okno statistiky

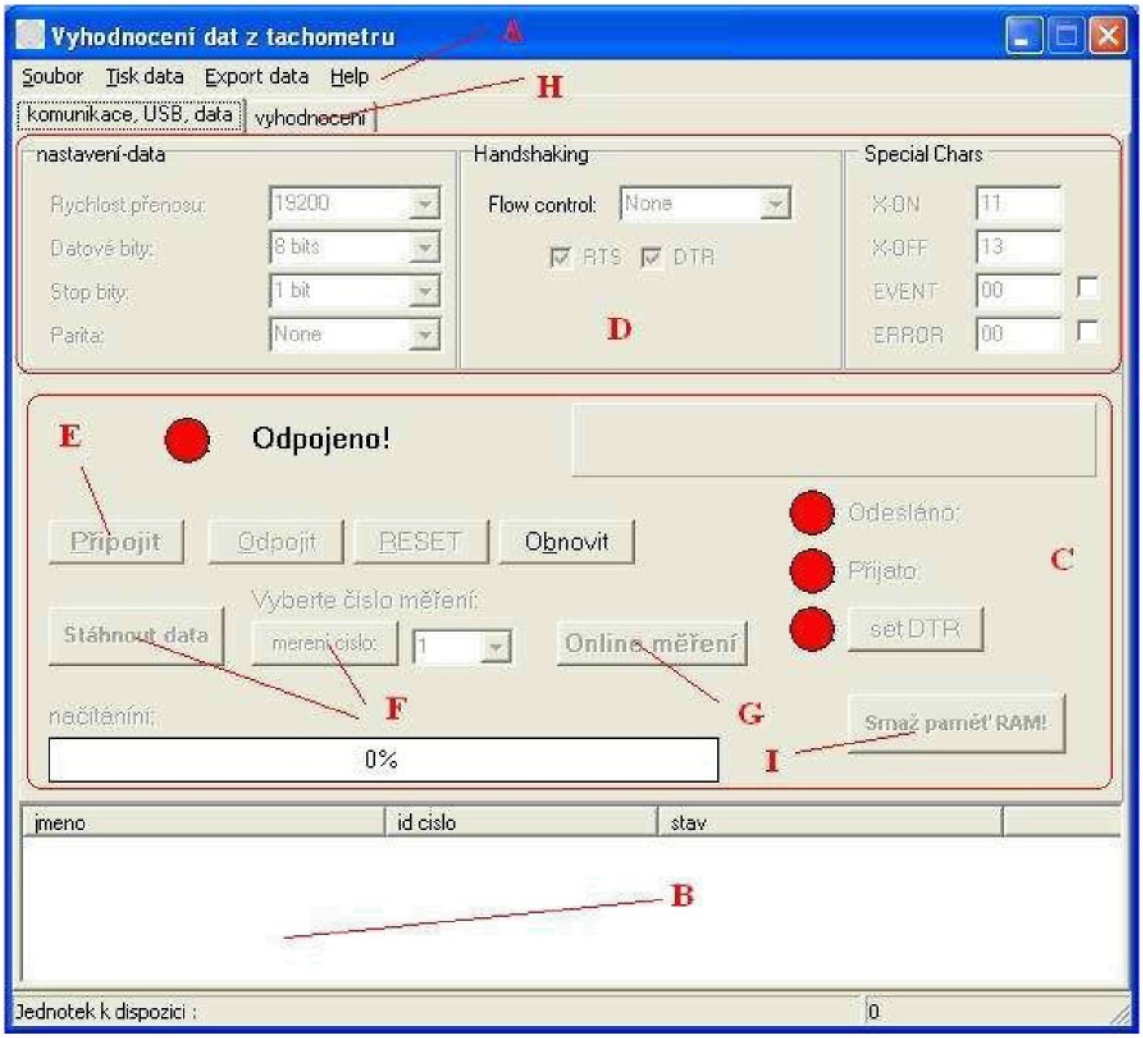
K zobrazování na tomto formuláři slouží šest komponent label. Postupně v každém kroku se počítají hodnoty všech ukazatelů, které jsou vždy před každým dalším krokem výpočtu aktualizovány a jejich nová hodnota je vložena do labelů na formuláři. Všechny hodnoty jsou počítány v rámci tohoto formuláře. Podrobný zdrojový kód formuláře je vložen na přiloženém CD.

2.3 Manuál programu

Program Vyhodnocení dat z tachometru slouží pro stažení a vyhodnocení dat, které naměřila měřící deska. Pracovní prostředí programu se skládá z jednoho hlavního okna, na kterém jsou dvě záložky a tří pomocných oken.

Hlavní okno programu:

Stahování dat

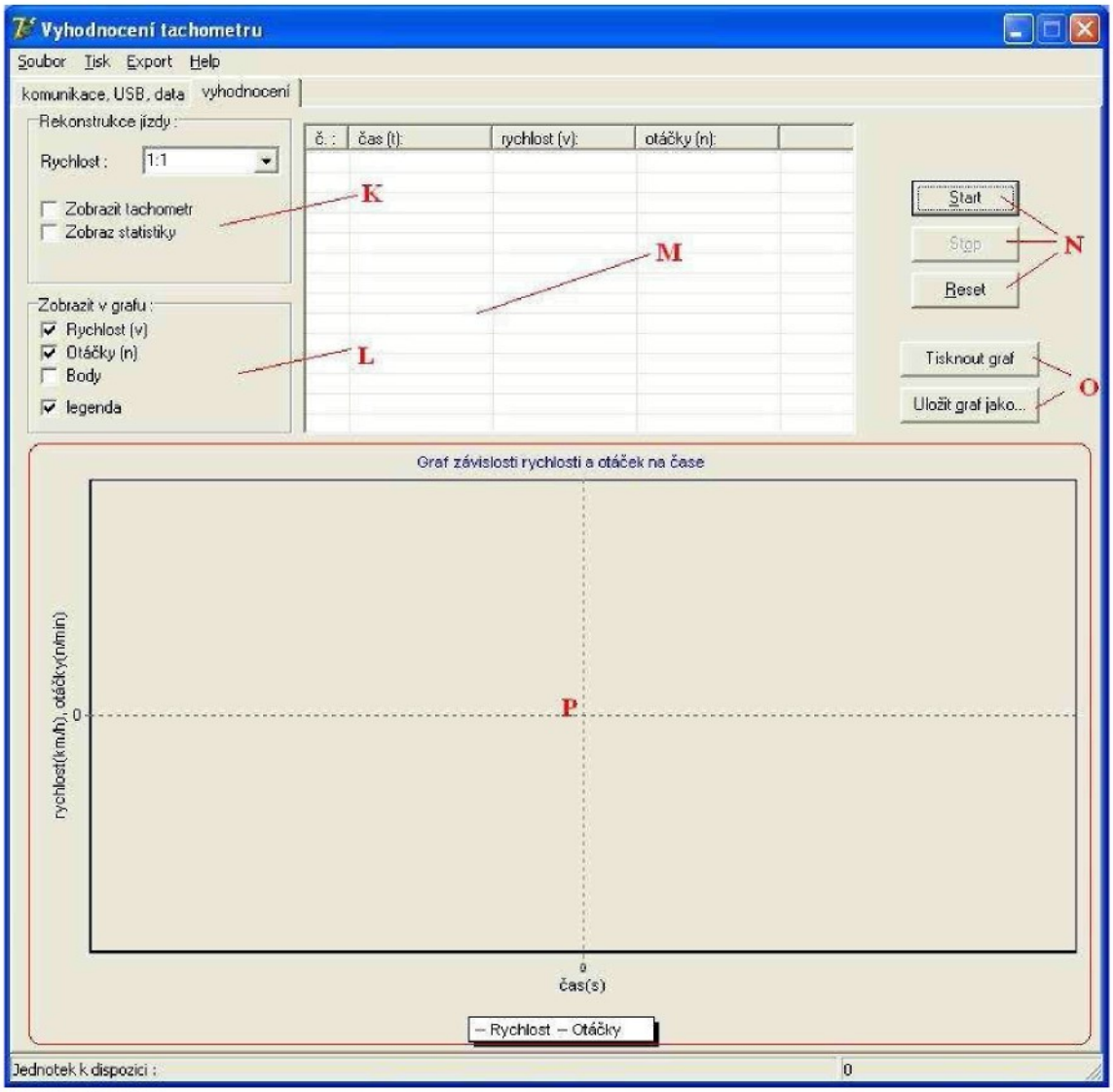


Obr. 2.14: Hlavní okno programu – stahování dat

Hlavní okno programu je zobrazeno na Obr. 2.14. Většinu funkcí poskytuje horní roletové menu (A), které je podobné jako u většiny programů pro operační systém Windows. V nabídce připojených jednotek (B) jsou zobrazeny aktuálně připojené jednotky, ke kterým se je možno připojit. Kliknutím na položku v nabídce se zpřístupní nabídka pro obsluhu připojování a stahování dat (C), zároveň se zpřístupní i nabídka pro

nastavení vlastností komunikace (D) počítače s měřící deskou. Připojit k desce je možné přes tlačítko Připojit. Ke stahování dat slouží tlačítka Stáhnout data a měření číslo (F). Pro spuštění měření v reálném čase slouží tlačítko Online měření (G). Smazat všechna uložené měření lze tlačítkem Smazat paměť RAM (I). Pro přepnutí okna do režimu vyhodnocení a rekonstrukce jízdy slouží záložka vyhodnocení (H).

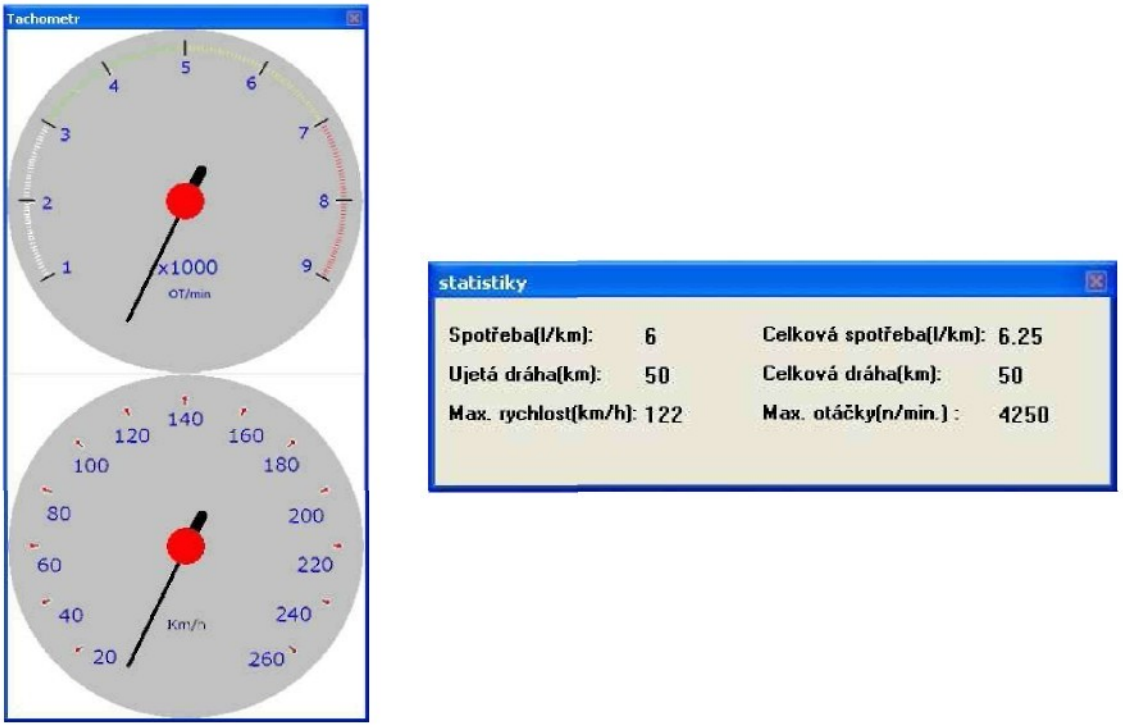
Vyhodnocení a rekonstrukce jízdy



Obr. 2.15: Hlavní okno programu – vyhodnocení a rekonstrukce jízdy

Záložka vyhodnocení je zobrazena na Obr. 2.15. Rekonstrukce se ovládá pomocí tlačítek Start, Stop a Reset (N). Parametry rekonstrukce jízdy se nastavují na panelu (K), kde je možné nastavit rychlost s jakou se bude průběh jízdy zobrazovat. Na tomto panelu je také možnost zapnutí rozšiřujících ukazatelů, okna tachometru a okna

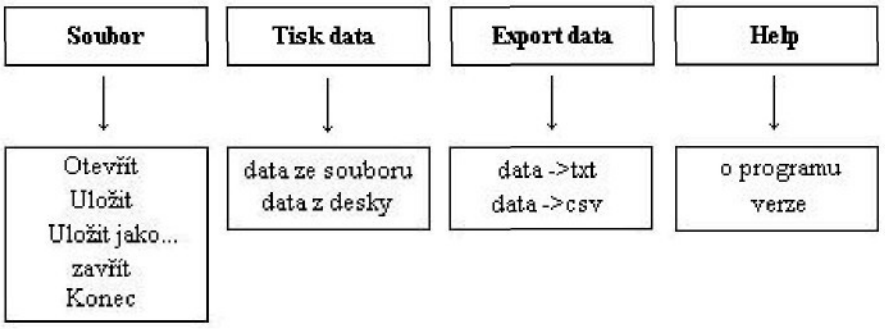
statistiky (viz. Obr. 2.16). Na panelu (L) lze nastavovat, co bude zobrazeno v grafu. Graf je na Obr. 2.15 zobrazen jako (P). Pomocí tlačítek (O) lze graf tisknout nebo ho uložit jako obrázek.



Obr. 2.16: Okna Tachometr a statistiky

Hlavní roletové menu

Většina důležitých funkcí je přístupných z hlavního roletového menu programu, všechny jeho položky je možné vidět na Obr. 2.17.



Obr. 2.17: Hlavní menu Vyhodnocení dat z tachometru

Před prvním použitím je třeba nainstalovat ovladače zařízení FTDI, dostupné na stránkách výrobce (*www.FTDIChip.com*) nebo na přiloženém CD. Pro správnou

manipulaci s měřící deskou je doporučeno prostudovat si manuál k měřící desce, vypracovaný Janem Korejtkem [4].

3. Zhodnocení dosažených výsledků

Programem bylo provedeno několik vyhodnocování dat z měřící desky, které byly většinou pořízeny v laboratorních podmínkách na generátoru funkcí. Vyhodnocení probíhalo bez větších potíží. Bylo provedeno i měření na na zkušebním stavu brzd pro kolejová vozidla firmy DAKO a.s.. Toto měření se však nepovedlo. Příčinou bylo silně elektromagnetické rušení, které se nepodařilo odstranit. Původně bylo zamýšleno realizovat měření na motocyklu, což se však nakonec nepovedlo uskutečnit.

Obr. 3.1: Graf naměřených dat

V průběhu vývoje aplikace byly oproti původním předpokladům provedeny určité změny, které byly vynuceny především změnami na měřící desce. Některé části programu byly již v teoretickém rozboru navrženy nevyhovujícím způsobem. Toto se týkalo hlavně oblasti komunikace, kde nebylo možno vyzkoušet program dříve než byl vytvořen komunikační přípravek pro měřící desku.

Aplikace splňuje všechny body zadání, ale pro větší komfort měření a pro dlouhodobější možnosti statistických výpočtů by se dala rozšířit o databázi již dříve stažených a uložených jízd. Pro komfortnější ovládání a komplexnější používaní při

testech by se daly obě části ještě rozšířit. Popřípadě ještě přidat porovnání více jízd najednou, například pro porovnání různých projetí stejného úseku.

4. Závěr

Výsledný program je schopen provádět vyhodnocování dat z měřící desky, která je předmětem realizace v bakalářské práci Jana Korejtka [4]. Aplikace byla postupně přizpůsobovaná a modifikovaná podle potřeby měřící desky a podle uživatele. V průběhu testování programu se podařilo odladit jeho chod tak, aby byl schopen bezproblémově komunikovat s měřící deskou a vyhodnotit data, která z ní obdrží. Pro lepší ověření by bylo potřeba tuto aplikaci zkoušet v delším časovém úseku, aby se poznalo co by bylo ještě dobré vylepšit nebo doplnit. Na základě dosažených výsledků lze považovat tuto práci za splněnou.

Seznam literatury

[1] Future Technology Devices International Ltd: *FT232R USB UART IC Datasheet*, version 2.02, 2009.

[2] Future Technology Devices International Ltd: *FT232R USB UART IC Datasheet*, version 2.02, 2009, s. 22-31.

[3] Future Technology Devices International Ltd: *D2XX Programmer's Guide*, version 2.02, 2009, s. 4-34.

[4] KOREJTKO, Jan. *Nezávislé pořízení a záznam informace o otáčkách motoru a rychlosti vozidla.* Liberec, 2009. 51 s. TECHNICKÁ UNIVERZITA V LIBERCI. Vedoucí bakalářské práce doc. Ing. Petr Tůma, CSc.

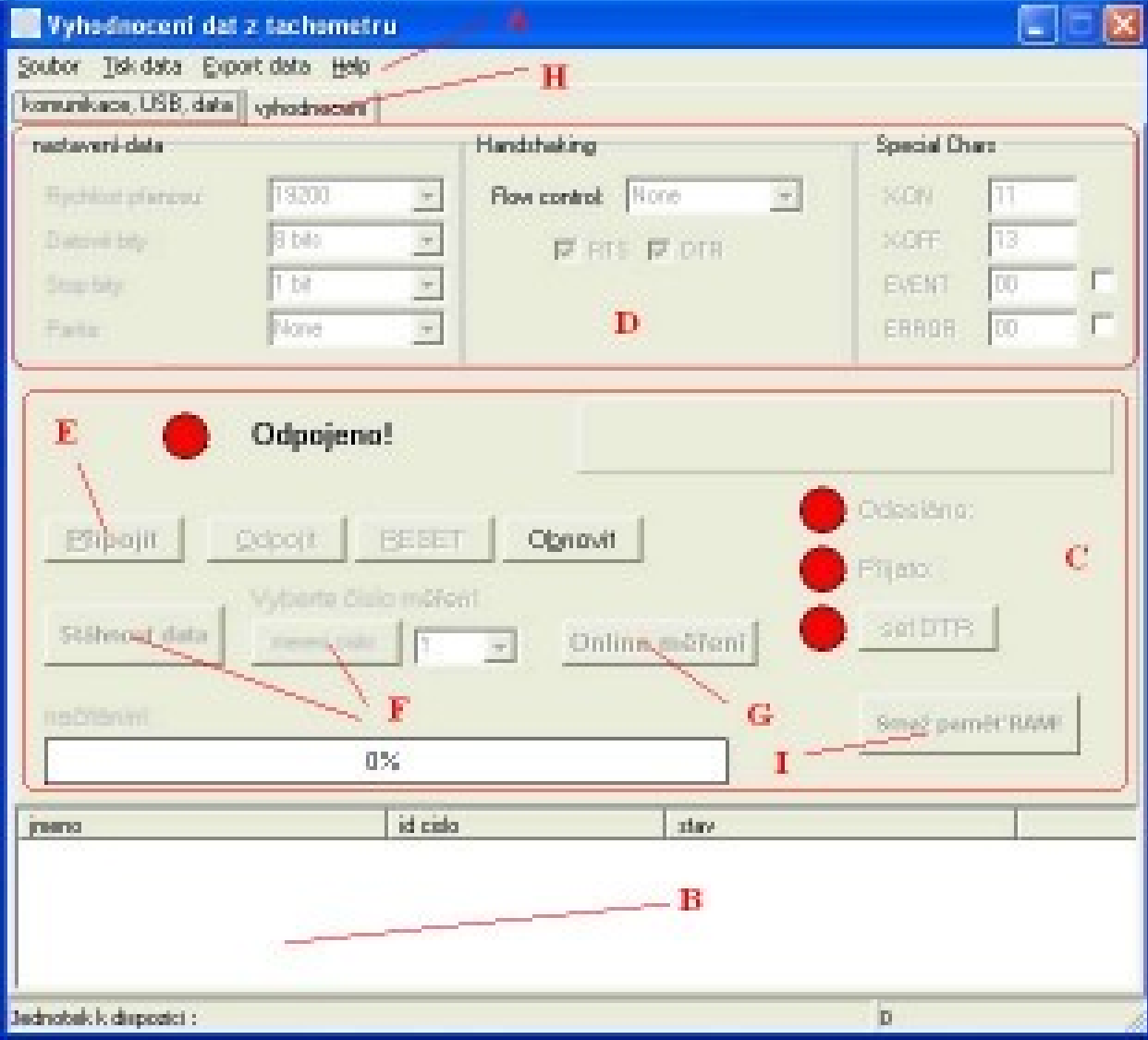
[5] MATOUŠEK, David. *USB prakticky s obvody FTDI - 1. díl : měření, řízení a regulace pomocí několika jednoduchých přípravků.* vyd. Praha : BEN - technická literatura, 2003. 272 s., CD-ROM, s. 42-69. ISBN 80-7300-103-9.

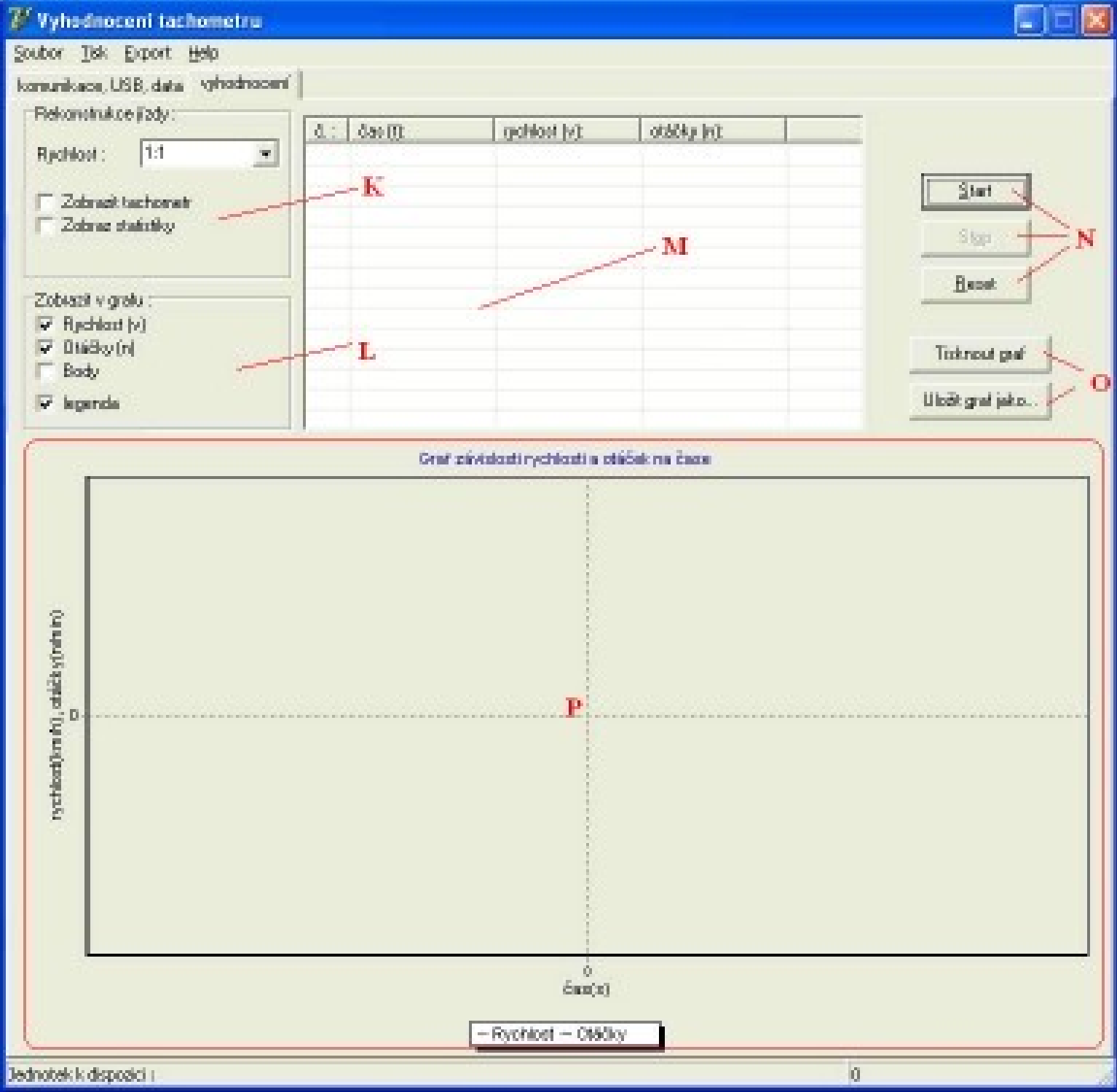
[7] Robert Bosch GmbH: *Snímače v motorových vozidlech.* Stuttgart, 2001, ISBN: 80-903132-5-6.

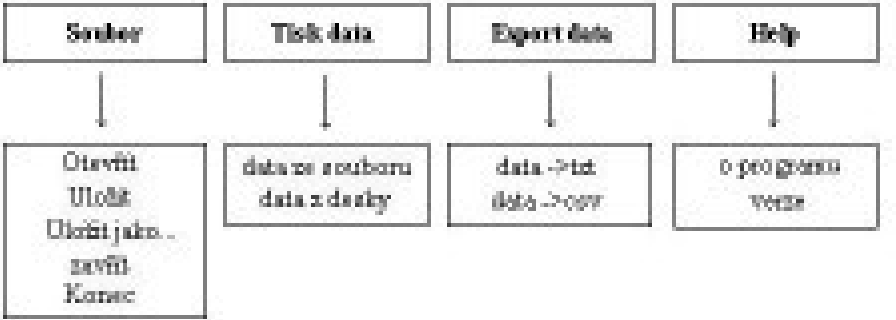
[8] *Wikipedia* [online]. 2001 , 28. 5. 2009 [cit. 2009-05-28]. Dostupný z WWW: < http://cs.wikipedia.org/wiki/USB >


TITAN

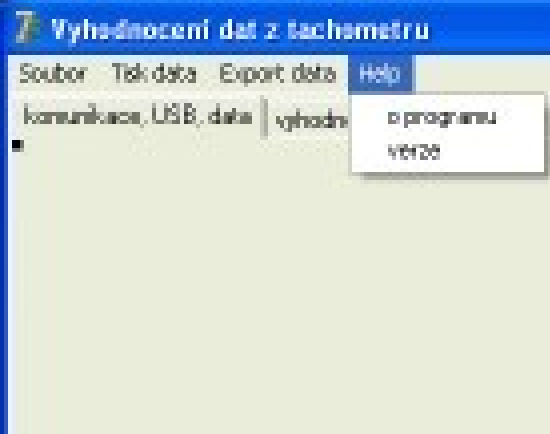

Graf závislosti rychlosti a otáček na čase
1 700
1 600
1 500
1 400
1 300
1 200
1 100
1 000
900
800
700
600
500
400
300
200
100
0
0
8
Čas(s)
Rychlost
Otáčky

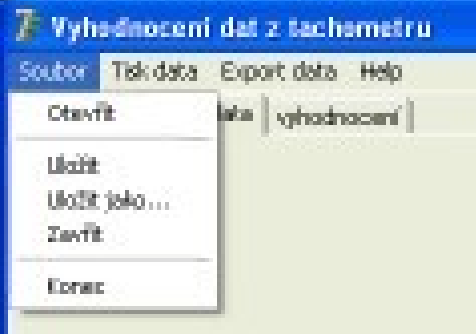

Vyhodnocení dat z tachometru
Soubor
Tisk data
Export data
Help
komunikace, USB, data
vyhodnocení
nastavení data
Handshaking
Flow control
None
RTS
DTR
Special Chars
XON
XOFF
EVENT
ERROR
D
E
Odpojeno!
Připojit
Odpojit
RESET
Obnovit
Odesláno:
Přijato:
set DTR
C
Vyberte číslo měření
Stáhnout data
Online měření
F
G
I
0%
Smaž paměť RAM
jméno
id číslo
stav
B
Jednotlivá k dispozici :
0

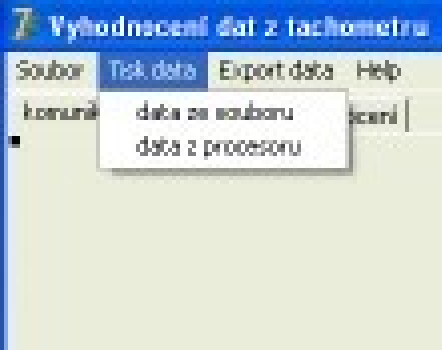

Vyhodnocení tachometru
Soubor Tisk Export Help
komunikace, USB, data
vyhodnocení
Rekonstrukce jízdy :
Rychlost :
1:1
Zobrazit tachonast
Zobraz statistiky
Zobrazit v grafu :
Rychlost (v)
Otáčky (n)
Body
legenda
Δ :
Čas (t)
rychlost (v)
otáčky (n)
K
M
L
Start
Stop
Reset
N
Tisknout graf
Uložit graf jako...
O
Graf závislosti rychlosti a otáček na čase
rychlost(km/h), otáčky(ot/min)
0
P
0
čas(s)
Rychlost
Otáčky
Jednotek k dispozici :
0

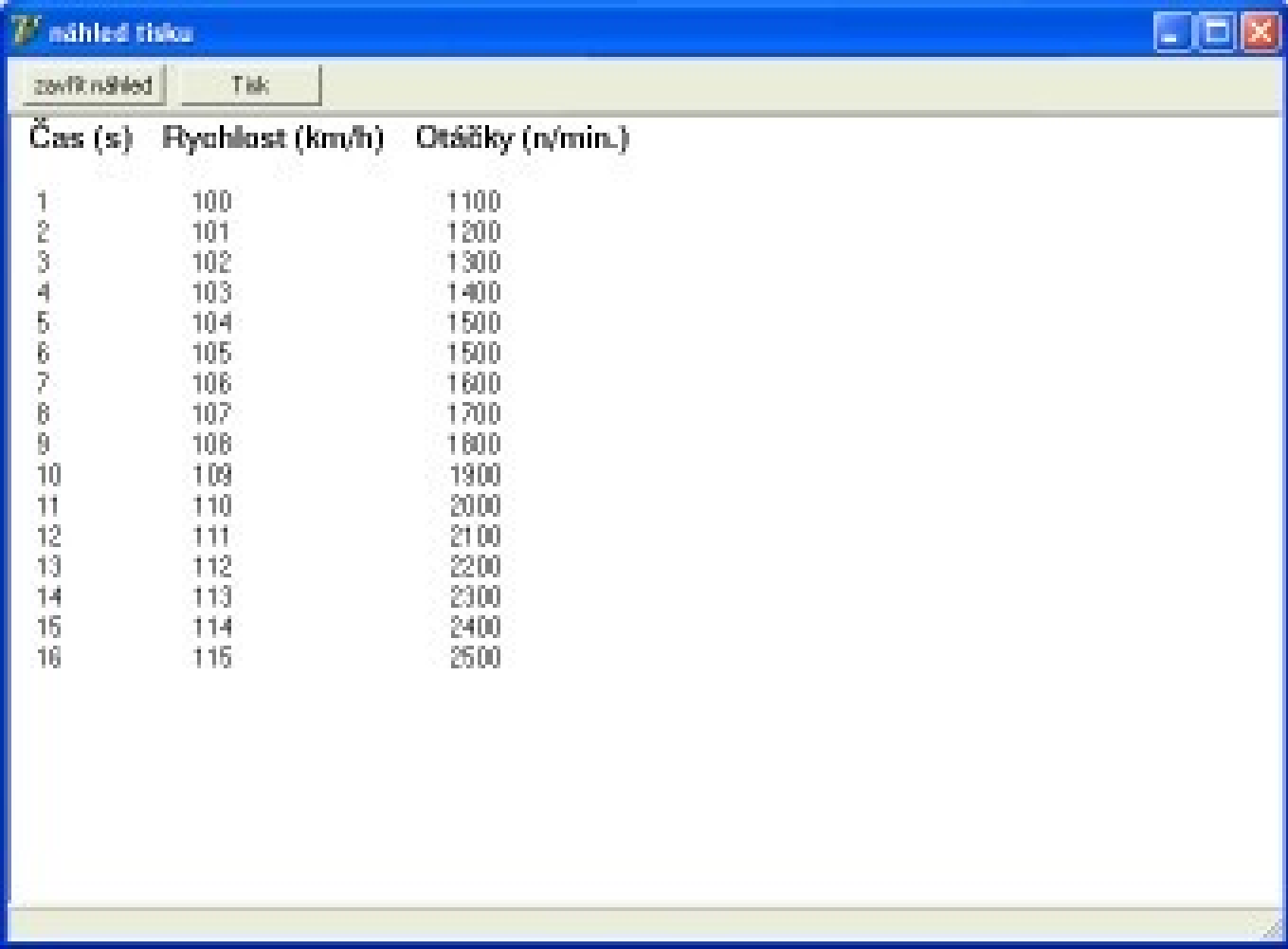

Soubor
Tisk dat
Export dat
Help
Otevřít
Uložit
Uložit jako..
zavřít
Konec
data ze souboru
data z desky
data->txt
data->csv
o programu
verze

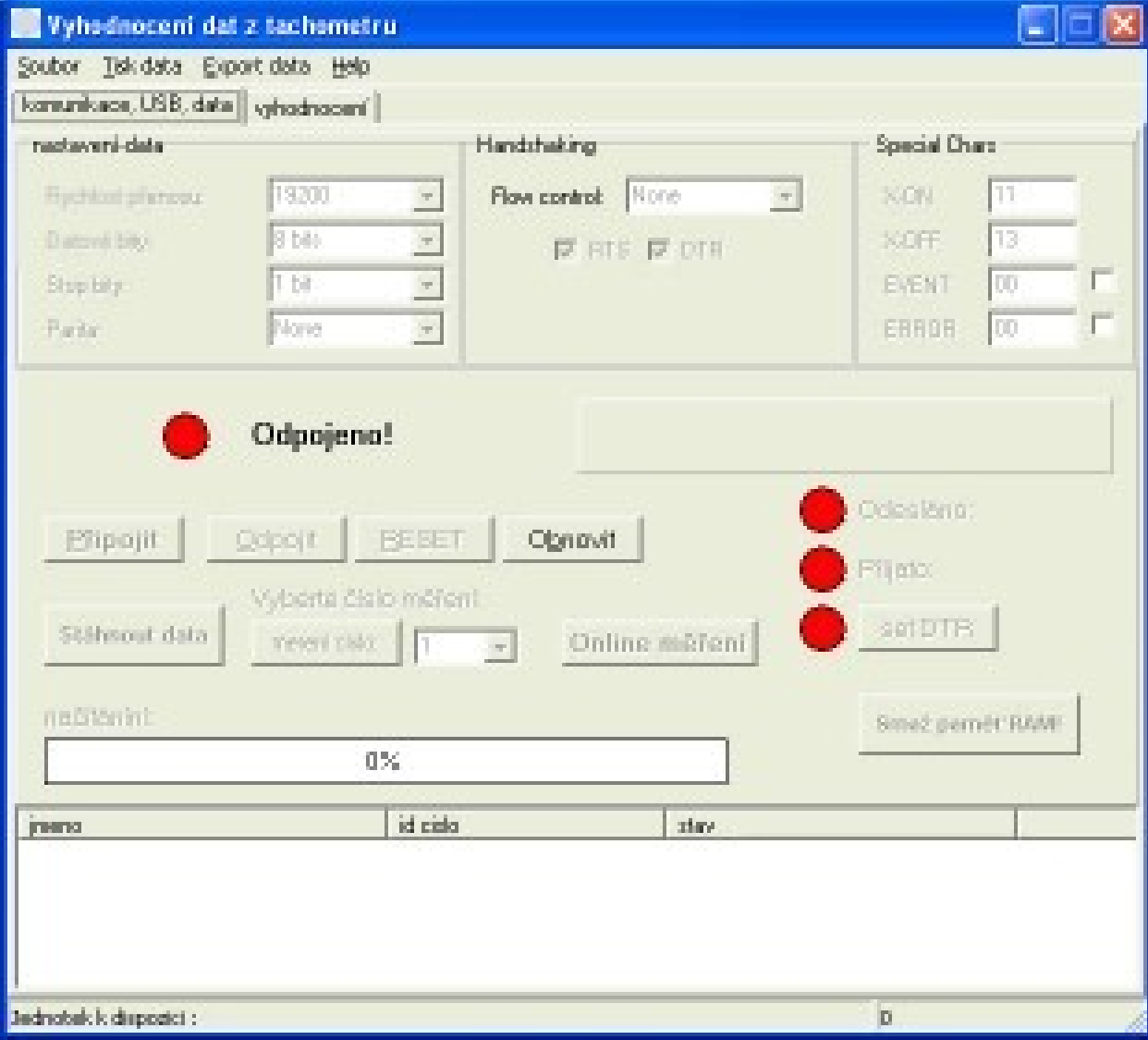

Vyhodnocení dat z tachometru
Soubor
Tisk dat
Export dat
Help
komunikace, USB,
data ->txt
data ->csv

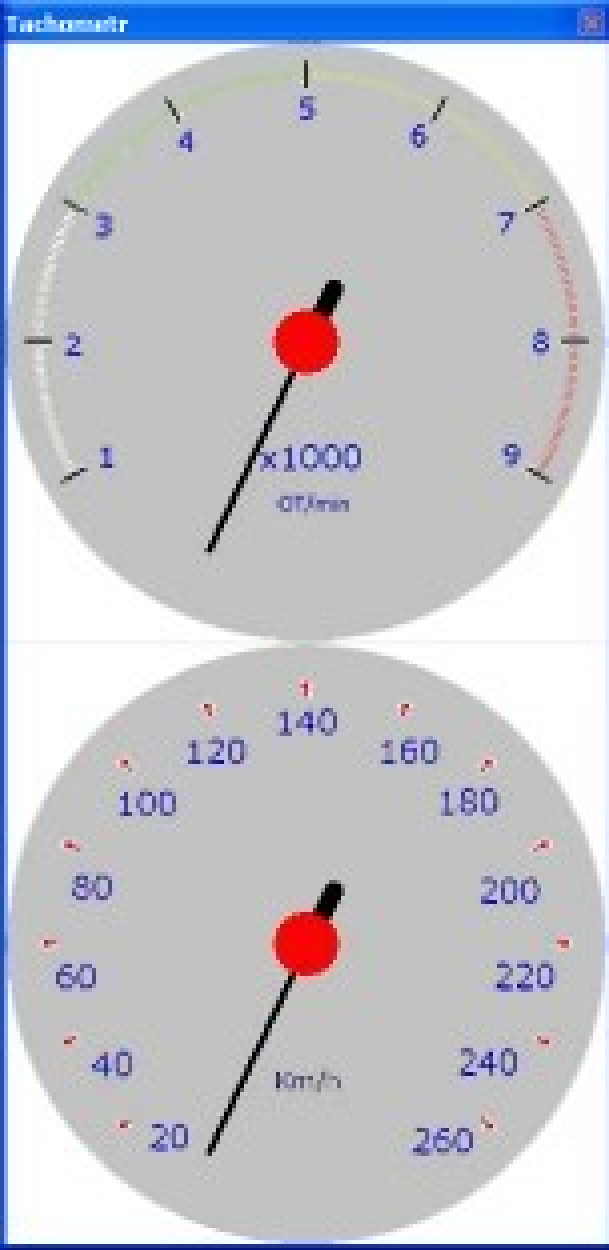

Vyhodnocení dat z tachometru
Soubor
Tisk data
Export data
Help
komunikace, USB, data
vyhodno
o programu
verze


Vyhodnocení dat z tachometru
Soubor
Tisk dat
Export dat
Help
vyhodnocení
Otevřít
Uložit
Uložit jako...
Zavřít
Konec


Vyhodnocení dat z tachometru
Soubor
Tisk dat
Export dat
Help
data ze souboru
data z procesoru


náhled tisku
zavřít náhled
Tisk
Čas (s) Rychlost (km/h) Otáčky (n/min.)
1 100 1100
2 101 1200
3 102 1300
4 103 1400
5 104 1500
6 105 1500
7 106 1600
8 107 1700
9 108 1800
10 109 1900
11 110 2000
12 111 2100
13 112 2200
14 113 2300
15 114 2400
16 115 2500

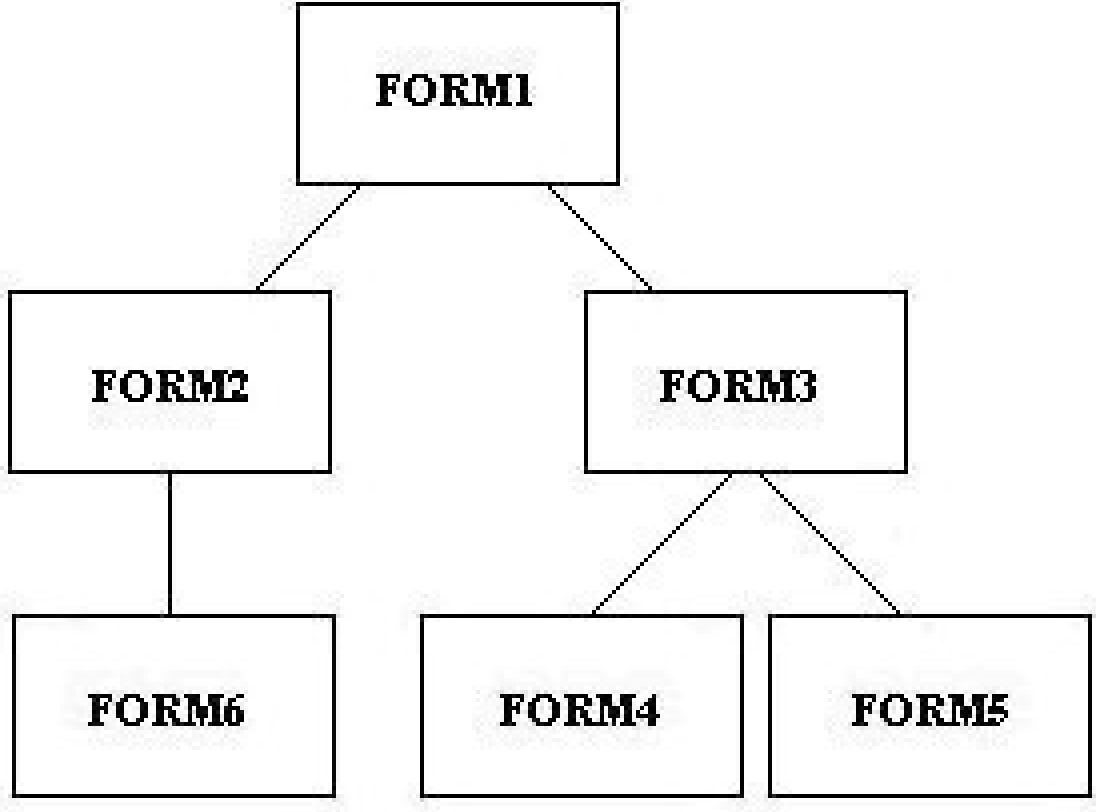

Vyhodnocení dat z tachometru
Soubor
Tisk data
Export data
Help
komunikace, USB, data
vyhodnocení
nastavení dat
Rychlost přenosu:
19200
Datové bity:
8 bit
Stop bity:
1 bit
Parita:
None
Handshaking
Flow control:
None
RTS
DTR
Special Chars
XON
17
XOFF
19
EVENT
00
ERROR
00
Odpojeno!
Připojit
Odpojit
BEEP
Obnovit
Odesláno:
Přijato:
set DTR
Vyberte číslo měření:
Stáhnout data
Vymaž data
1
Online měření
načítání:
0%
Smaž paměť RAM
jméno
id číslo
stav
Jednotek k dispozici :
0

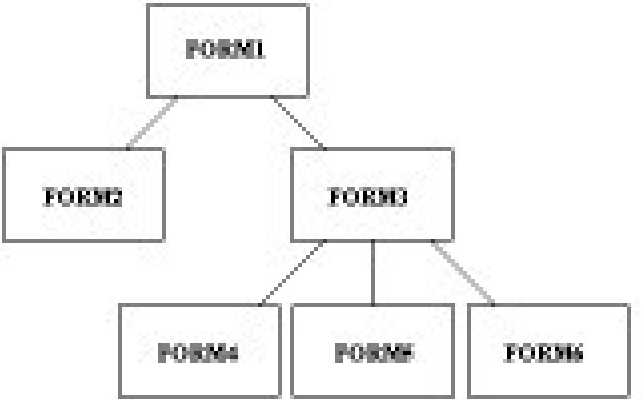

Tachometr
1
2
3
4
5
6
7
8
9
x1000
OT/min
20
40
60
80
100
120
140
160
180
200
220
240
260
Km/h


Program
FTD2XX.dll
FTDIXX.sys
USB bus for Windows
Port USB
FTDI


Program
FTD2XX.dll
FTDI2XX.sys
USB buffer
Windows
Port USB
FTDI

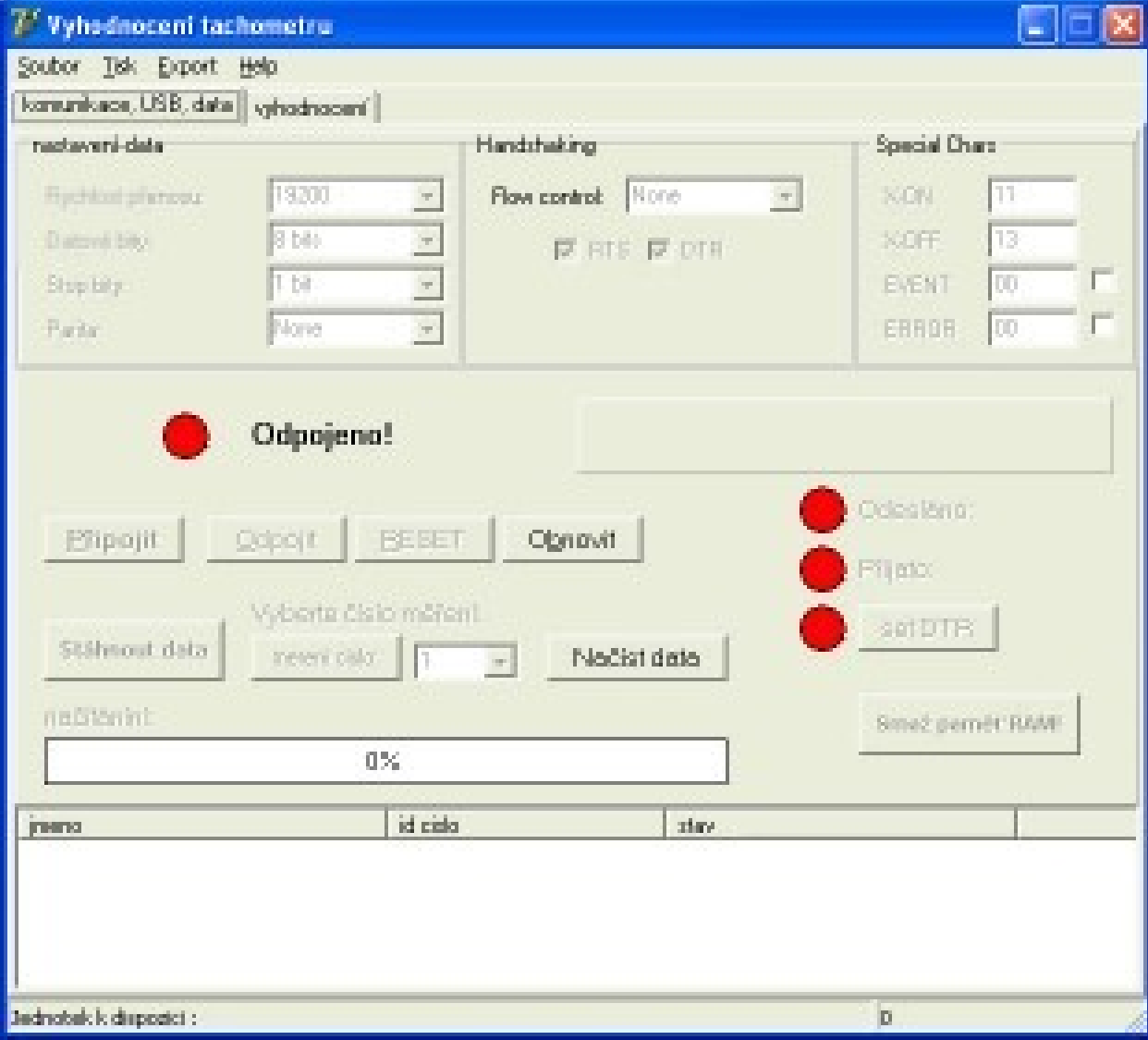

Graf závislosti rychlosti a otáček na čase
1 700
1 600
1 500
1 400
1 300
1 200
1 100
1 000
900
800
700
600
500
400
300
200
100
0
Čas(s)
Rychlost + Otáčky

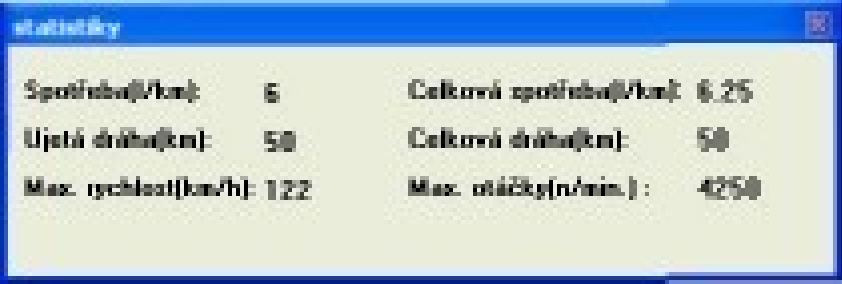

FORM1
FORM2
FORM3
FORM6
FORM4
FORM5

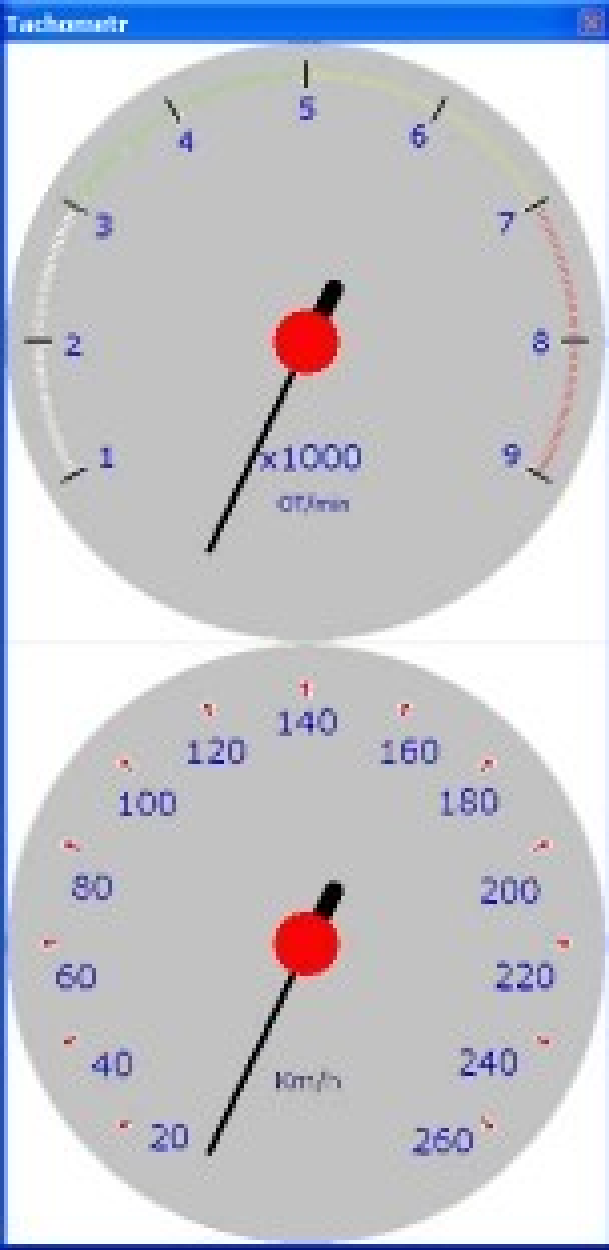

FORM1
FORM2
FORM3
FORM4
FORM5
FORM6

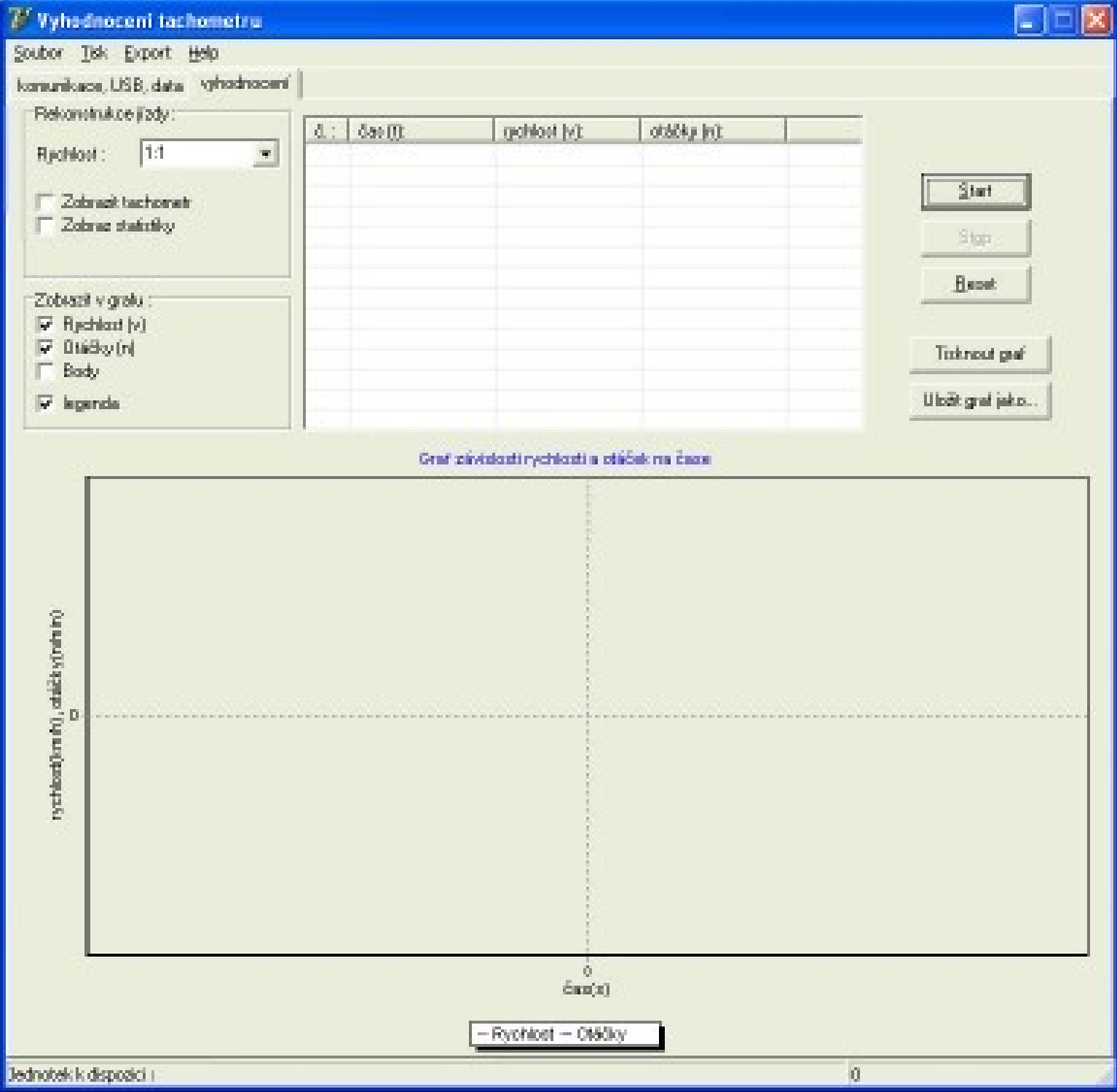

Odesláno:
Přijato:
set DTR
Smaž paměť RAM

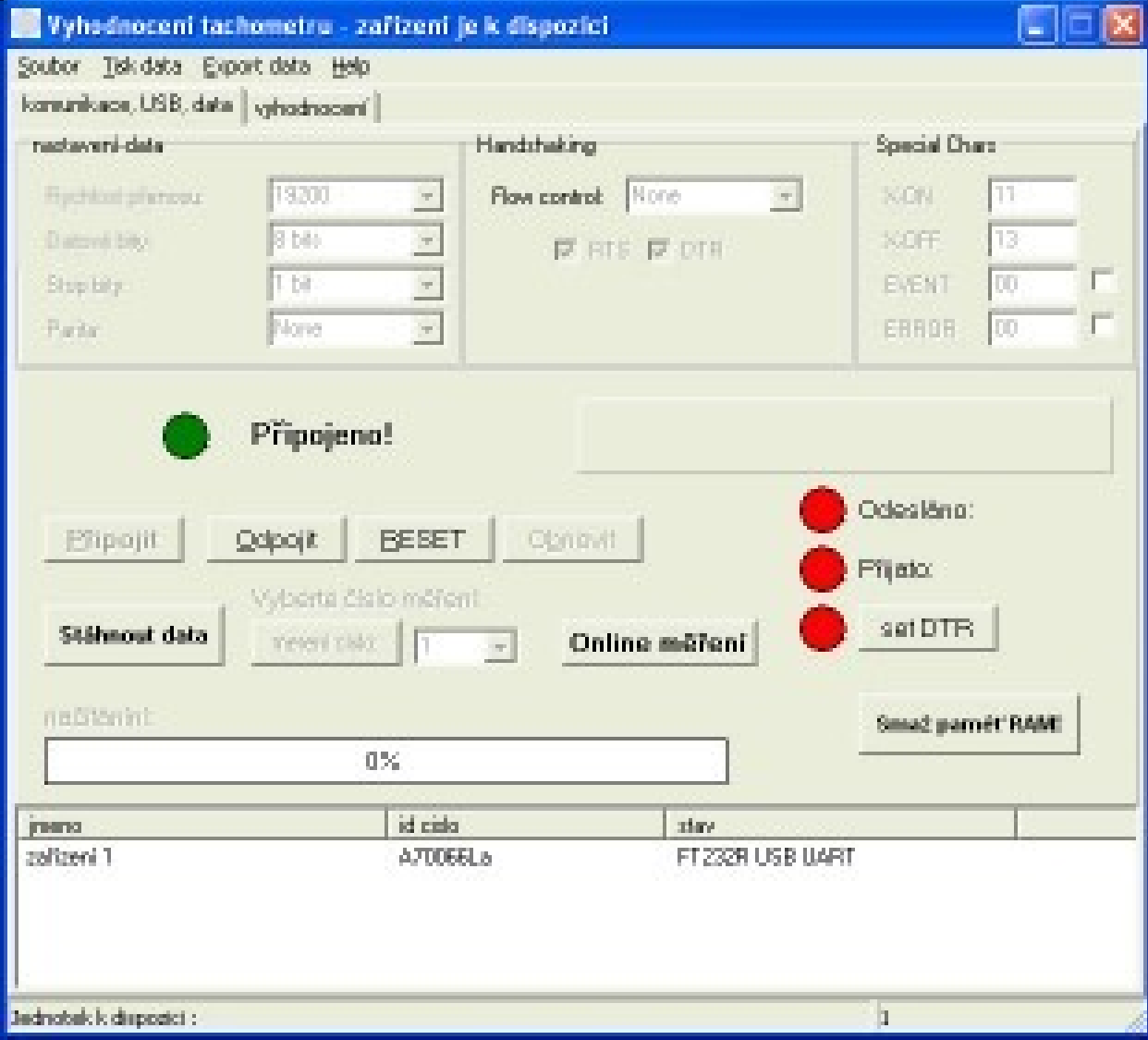

Vyberte číslo měření
Stáhnout data


Vyhodnocení tachometru
Soubor
Tisk
Export
Help
komunikace, USB, data
vyhodnocení
nastavení dat
Rychlost přenosu:
19200
Datové bity:
8 bit
Stop bity:
1 bit
Parita:
None
Handshaking
Flow control:
None
RTS
DTR
Special Chars
XON:
17
XOFF:
19
EVENT:
00
ERROR:
00
Odpojeno!
Připojit
Odpojit
BEEP
Obnovit
Odesláno:
Přijato:
set DTR
Vyberte číslo měření:
Stáhnout data
nové číslo:
1
Načíst data
načítání:
0%
Smaž paměť RAM!
jméno
id číslo
stav
Jednotek k dispozici :
0


statistiky
Spotřeba[l/km]: 6
Ujetá dráha[km]: 50
Max. rychlost[km/h]: 122
Celková spotřeba[l/km]: 6.25
Celková dráha[km]: 50
Max. otáčky[n/min.]: 4250


Tachometr
1
2
3
4
5
6
7
8
9
x1000
OT/min
20
40
60
80
100
120
140
160
180
200
220
240
260
Km/h


Vyhodnocení tachometru
Soubor Tisk Export Help
komunikace, USB, data
vyhodnocení
Rekonstrukce jízdy :
Rychlost :
1:1
Zobrazit tachonaut
Zobraz statistiky
Zobrazit v grafu :
Rychlost (v)
Otáčky (n)
Body
legenda
Δ :
čas (t)
rychlost (v)
otáčky (n)
Start
Stop
Reset
Tisknout graf
Uložit graf jako...
Graf závislosti rychlosti a otáček na čase
rychlost(km/h), otáčky(ot/min)
0
0
čas(s)
Rychlost
Otáčky
Jednotek k dispozici :
0


Vyhodnocení tachometru - zařízení je k dispozici
Soubor
Tisk dat
Export dat
Help
komunikace, USB, data
vyhodnocení
nastavení dat
Rychlost přenosu:
19200
Datové bity:
8 bit
Stop bity:
1 bit
Parita:
None
Handshaking
Flow control:
None
RTS
DTR
Special Chars
XON:
17
XOFF:
19
EVENT:
00
ERROR:
00
Připojeno!
Připojit
Odpojit
RESET
Obnovit
Odesíláno:
Přijato:
set DTR
Vyberte číslo měření:
Stáhnout data
Veveří číslo:
1
Online měření
načítání:
0%
Smaž paměť RAM
jméno
id číslo
stav
zařízení 1
A7006ELa
FT232R USB UART
Jednotek k dispozici :
1

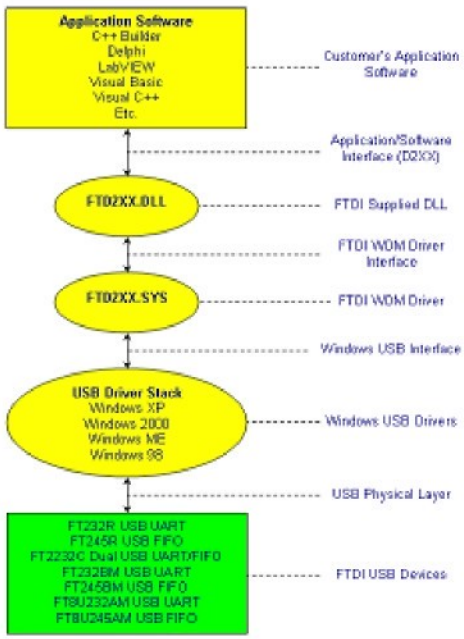
Future Technology Devices International Ltd.

D2XX Programmer's Guide

Table of Contents

1 Welcome to the FTD2XX Programmer's Guide

FTDIs "D2XX Direct Drivers" for Windows offer an alternative solution to our VCP drivers which allows application software to interface with FT232R USB UART, FT245R USB FIFO, FT2232C Dual USB UART/FIFO, FT232BM USB UART, FT245BM USB FIFO, FT8U232AM USB UART and FT8U245AM USB FIFO devices using a DLL instead of a Virtual COM Port.



The architecture of the D2XX drivers consists of a Windows WDM driver that communicates with the device via the Windows USB stack and a DLL which interfaces the application software (written in Visual C++, C++ Builder, Delphi, VB etc.) to the WDM driver. An INF installation file, uninstaller program and D2XX programmers guide complete the package.

The document is divided into four parts:

- **Classic Interface Functions** 5 which explains the original functions with some more recent additions
- **EEPROM Interface** 58 which allows application software to read/program the various fields in the FT232R/FT245R internnal EEPROM or external 93C46/93C56/93C66 EEPROM for other devices, including a user defined area which can be used for application specific purposes.
- **Extended API Functions** 71 which allow control of the additional features available from our 2nd generation devices onwards.
- **FT-Win32 API** 78 which is a more sophisticated alternative to the Classic Interface - our equivalent to the native Win 32 API calls that are used to control a legacy serial port. Using the FT-Win32 API, existing Windows legacy Comms applications can easily be converted to use the D2XX interface simply by replacing the standard Win32 API calls with the equivalent FT-Win32 API calls.

Please note that the Classic Interface and the FT-Win32 API interface are alternatives.
Developers should choose one or the other: the two sets of functions should not be mixed.

Main Differences Between Windows and Windows CE D2XX Drivers

- Location IDs are not supported by Windows CE
- FT_ResetPort and FT_CyclePort are not available
- Windows CE does not support overlapped IO for

2 Classic Interface Functions

Introduction

An FTD2XX device is an FT232R USB UART, FT245R USB FIFO, FT2232C Dual USB UART/FIFO, FT232BM USB UART, FT245BM USB FIFO, FT8U232AM USB UART or FT8U245AM USB FIFO interfacing to Windows application software using FTDIs WDM driver FTD2XX.SYS. The FTD2XX.SYS driver has a programming interface exposed by the dynamic link library FTD2XX.DLL and this document describes that interface.

Overview

FT_ListDevices 8 returns information about the FTDI devices currently connected. In a system with multiple devices this can be used to decide which of the devices the application software wishes to access (using FT_OpenEx below).

Before the device can be accessed, it must first be opened. FT_Open 11 and FT_OpenEx 12 return a handle that is used by all functions in the Classic Programming Interface to identify the device. When the device has been opened successfully, I/O can be performed using FT_Read 15 and FT_Write 17. When operations are complete, the device is closed using FT_Close 14.

Once opened, additional functions are available to reset the device (FT_ResetDevice 18); purge receive and transmit buffers (FT_Purge 29); set receive and transmit timeouts (FT_SetTimeouts 30); get the receive queue status (FT_GetQueueStatus 31); get the device status (FT_GetStatus 34); set and reset the break condition (FT_SetBreakOn 32, FT_SetBreakOff 33); and set conditions for event notification (FT_SetEventNotification 35).

For FT232R devices, FT2232C devices used in UART mode, FT232BM and FT8U232AM devices, functions are available to set the Baud rate (FT_SetBaudRate 19), and set a non-standard Baud rate (FT_SetDivisor 20); set the data characteristics such as word length, stop bits and parity (FT_SetDataCharacteristics 21); set hardware or software handshaking (FT_SetFlowControl 22); set modem control signals (FT_SetDtr 23, FT_ClrDtr 24, FT_SetRts 25, FT_ClrRts 26); get modem status (FT_GetModemStatus 27); set special characters such as event and error characters (FT_SetChars 28).

For FT245R devices, FT2232C devices used in FIFO mode, FT245BM and FT8U245AM devices, these functions are redundant and can effectively be ignored.

Reference

Type definitions 105 of the functional parameters and return codes used in the D2XX classic programming interface are contained in the appendix 104.

2.1 FT_SetVIDPID

A Linux specific command to include your own VID and PID within the internal device list table.

FT_STATUS **FT_SetVIDPID** (D*WORD dwVID*, DWORD *dwPID*)

Parameters

dwVID	Device VID.
dwPID	Device PID.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The driver will support a limited set of VID and PID matched devices (VID 0x0403 with PIDs 0x6001, 0x6010, 0x6006 only). In order to use the driver with alternative VID and PIDs the FT_SetVIDPID function must be used prior to calling FT_ListDevices 8, FT_Open 11, FT_OpenEx 12 or FT_CreateDeviceInfoList 48.

2.2 FT_GetVIDPID

A Linux specific command to retrieve the current VID and PID within the internal device list table.

FT_STATUS **FT_GetVIDPID** (D*WORD * pdwVID*, DWORD * p*dwPID*)

Parameters

pdwVID	Pointer to DWORD that will contain the internal VID.
pdwPID	Pointer to DWORD that will contain the internal PID.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

See FT_SetVIDPID 6.

2.3 FT_ListDevices

Get information concerning the devices currently connected. This function can return information such as the number of devices connected, the device serial number and device description strings, and the location IDs of connected devices.

FT_STATUS **FT_ListDevices** (PVOID *pvArg1*, PVOID *pvArg2*, DWORD *dwFlags*)

Parameters

pvArg1	Meaning depends on dwFlags.
pvArg2	Meaning depends on dwFlags.
dwFlags	Determines format of returned information.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function can be used in a number of ways to return different types of information. A more powerful way to get device information is to use the FT_CreateDeviceInfoList 48, FT_GetDeviceInfoList 49 and FT_GetDeviceInfoDetail 51 functions as they return all the available information on devices.

In its simplest form, it can be used to return the number of devices currently connected. If *FT_LIST_NUMBER_ONLY* bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to a DWORD location to store the number of devices currently connected.

It can be used to return device information: if *FT_OPEN_BY_SERIAL_NUMBER* bit is set in *dwFlags*, the serial number string will be returned; if *FT_OPEN_BY_DESCRIPTION* bit is set in *dwFlags*, the product description string will be returned; if *FT_OPEN_BY_LOCATION* bit is set in *dwFlags*, the Location ID will be returned; if none of these bits is set, the serial number string will be returned by default.

It can be used to return device string information for a single device. If *FT_LIST_BY_INDEX* and *FT_OPEN_BY_SERIAL_NUMBER* or *FT_OPEN_BY_DESCRIPTION* bits are set in *dwFlags*, the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a buffer to contain the appropriate string. Indexes are zero-based, and the error code *FT_DEVICE_NOT_FOUND* is returned for an invalid index.

It can be used to return device string information for all connected devices. If *FT_LIST_ALL* and *FT_OPEN_BY_SERIAL_NUMBER* or *FT_OPEN_BY_DESCRIPTION* bits are set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to an array of pointers to buffers to contain the appropriate strings and the parameter *pvArg2* is interpreted as a pointer to a DWORD location to store the number of devices currently connected. Note that, for *pvArg1*, the last entry in the array of pointers to buffers should be a NULL pointer so the array will contain one more location than the number of devices connected.

The location ID of a device is returned if *FT_LIST_BY_INDEX* and *FT_OPEN_BY_LOCATION* bits are set in *dwFlags*. In this case the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a variable of type long to contain the location

ID. Indexes are zero-based, and the error code *FT_DEVICE_NOT_FOUND* is returned for an invalid index. **Please note that Windows CE and Linux do not support location IDs.**

The location IDs of all connected devices are returned if *FT_LIST_ALL* and *FT_OPEN_BY_LOCATION* bits are set in *dwFlags*. In this case, the parameter *pvArg1* is interpreted as a pointer to an array of variables of type long to contain the location IDs, and the parameter *pvArg2* is interpreted as a pointer to a DWORD location to store the number of devices currently connected.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;
DWORD numDevs;
```

Get the number of devices currently connected

```
ftStatus = FT_ListDevices(&numDevs,NULL,FT_LIST_NUMBER_ONLY);
if (ftStatus == FT_OK) {
    // FT_ListDevices OK, number of devices connected is in numDevs
}
else {
    // FT_ListDevices failed
}
```

Get serial number of first device

```
DWORD devIndex = 0; // first device
char Buffer[64]; // more than enough room!

ftStatus =
FT_ListDevices((PVOID)devIndex,Buffer,FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);
if (ftStatus == FT_OK) {
    // FT_ListDevices OK, serial number is in Buffer
}
else {
    // FT_ListDevices failed
}
```

Note that indexes are zero-based. If more than one device is connected, incrementing devIndex will get the serial number of each connected device in turn.

Get device descriptions of all devices currently connected

```
char *BufPtrs[3];      // pointer to array of 3 pointers
char Buffer1[64];             // buffer for description of first device
char Buffer2[64];             // buffer for description of second device

// initialize the array of pointers
BufPtrs[0] = Buffer1;
BufPtrs[1] = Buffer2;
BufPtrs[2] = NULL;            // last entry should be NULL

ftStatus = FT_ListDevices(BufPtrs,&numDevs,FT_LIST_ALL|FT_OPEN_BY_DESCRIPTION);
if (ftStatus == FT_OK) {
    // FT_ListDevices OK, product descriptions are in Buffer1 and Buffer2, and
    // numDevs contains the number of devices connected
}
else {
    // FT_ListDevices failed
}
```

Note that this example assumes that two devices are connected. If more devices are connected, then the size of the array of pointers must be increased and more description buffers allocated.

Get locations of all devices currently connected

```
long locIdBuf[16];

ftStatus = FT_ListDevices(locIdBuf,&numDevs,FT_LIST_ALL|FT_OPEN_BY_LOCATION);
if (ftStatus == FT_OK) {
    // FT_ListDevices OK, location IDs are in locIdBuf, and
    // numDevs contains the number of devices connected
}
else {
    // FT_ListDevices failed
}
```

Note that this example assumes that no more than 16 devices are connected. If more devices are connected, then the size of the array of pointers must be increased.

2.4 FT_Open

Open the device and return a handle which will be used for subsequent accesses.

FT_STATUS **FT_Open** (int *iDevice*, FT_HANDLE **ftHandle*)

Parameters

iDevice	Must be 0 if only one device is attached. For multiple devices 1, 2 etc.
ftHandle	Pointer to a variable of type FT_HANDLE where the handle will be stored. This handle must be used to access the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

Although this function can be used to open multiple devices by setting *iDevice* to 0, 1, 2 etc. there is no ability to open a specific device. To open named devices, use the function FT_OpenEx 12.

Example

This sample shows how to open a device.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0,&ftHandle);
if (ftStatus == FT_OK) {
    // FT_Open OK, use ftHandle to access device
}
else {
    // FT_Open failed
}
```

2.5 FT_OpenEx

Open the specified device and return a handle that will be used for subsequent accesses. The device can be specified by its serial number, device description or location.

This function can also be used to open multiple devices simultaneously. Multiple devices can be opened at the same time if they can be distinguished by serial number or device description. Alternatively, multiple devices can be opened at the same time using location IDs - location information derived from their physical locations on USB. Location IDs can be obtained using the utility USBView and are given in hexadecimal format.

FT_STATUS **FT_OpenEx** (PVOID *pvArg1*, DWORD *dwFlags*, FT_HANDLE **ftHandle*)

Parameters

pvArg1	Meaning depends on dwFlags, but it will normally be interpreted as a pointer to a null terminated string.
dwFlags	FT_OPEN_BY_SERIAL_NUMBER, FT_OPEN_BY_DESCRIPTION or FT_OPEN_BY_LOCATION.
ftHandle	Pointer to a variable of type FT_HANDLE where the handle will be stored. This handle must be used to access the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The meaning of *pvArg1* depends on *dwFlags*: if *dwFlags* is *FT_OPEN_BY_SERIAL_NUMBER*, *pvArg1* is interpreted as a pointer to a null-terminated string that represents the serial number of the device; if *dwFlags* is *FT_OPEN_BY_DESCRIPTION*, *pvArg1* is interpreted as a pointer to a null-terminated string that represents the device description; if *dwFlags* is *FT_OPEN_BY_LOCATION*, *pvArg1* is interpreted as a long value that contains the location ID of the device. **Please note that Windows CE and Linux do not support location IDs**.

ftHandle is a pointer to a variable of type *FT_HANDLE* where the handle is to be stored. This handle must be used to access the device.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;
FT_STATUS ftStatus2;
FT_HANDLE ftHandle1;
FT_HANDLE ftHandle2;
long dwLoc;
```

Open a device with serial number "FT000001"

```
ftStatus = FT_OpenEx("FT000001",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle1);
```

```
if (ftStatus == FT_OK) {
	// success - device with serial number "FT000001" is open
}
else {
	// failure
}
```

Open a device with device description "USB Serial Converter"

```
ftStatus = FT_OpenEx("USB Serial Converter",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
if (ftStatus == FT_OK) {
	// success - device with device description "USB Serial Converter" is open
}
else {
	// failure
}
```

Open 2 devices with serial numbers "FT000001" and "FT999999"

```
ftStatus = FT_OpenEx("FT000001",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle1);
ftStatus2 = FT_OpenEx("FT999999",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
	// success - both devices are open
}
else {
	// failure - one or both of the devices has not been opened
}
```

Open 2 devices with descriptions "USB Serial Converter" and "USB Pump Controller"

```
ftStatus = FT_OpenEx("USB Serial Converter",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
ftStatus2 = FT_OpenEx("USB Pump Controller",FT_OPEN_BY_DESCRIPTION,&ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
	// success - both devices are open
}
else {
	// failure - one or both of the devices has not been opened
}
```

Open a device at location 23

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle1);
if (ftStatus == FT_OK) {
	// success - device at location 23 is open
}
else {
	// failure
}
```

Open 2 devices at locations 23 and 31

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle1);
dwLoc = 0x31;
ftStatus2 = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
	// success - both devices are open
}
else {
	// failure - one or both of the devices has not been opened
}
```

2.6 FT_Close

Close an open device.

FT_STATUS **FT_Close** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.7 FT_Read

Read data from the device.

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

Parameters

ftHandle	Handle of the device.
lpBuffer	Pointer to the buffer that receives the data from the device.
dwBytesToRead	Number of bytes to be read from the device.
lpdwBytesReturned	Pointer to a variable of type DWORD which receives the number of bytes read from the device.

Return Value

FT_OK if successful, FT_IO_ERROR otherwise.

Remarks

FT_Read always returns the number of bytes read in *lpdwBytesReturned*.

This function does not return until *dwBytesToRead* have been read into the buffer. The number of bytes in the receive queue can be determined by calling FT_GetStatus 34 or FT_GetQueueStatus 31, and passed to FT_Read 15 as *dwBytesToRead* so that the function reads the device and returns immediately.

When a read timeout value has been specified in a previous call to FT_SetTimeouts 30, FT_Read 15 returns when the timer expires or *dwBytesToRead* have been read, whichever occurs first. If the timeout occurred, FT_Read 15 reads available data into the buffer and returns *FT_OK*.

An application should use the function return value and *lpdwBytesReturned* when processing the buffer. If the return value is *FT_OK*, and *lpdwBytesReturned* is equal to *dwBytesToRead* then FT_Read 15 has completed normally. If the return value is *FT_OK*, and *lpdwBytesReturned* is less then *dwBytesToRead* then a timeout has occurred and the read has been partially completed. Note that if a timeout occurred and no data was read, the return value is still *FT_OK*.

A return value of *FT_IO_ERROR* suggests an error in the parameters of the function, or a fatal error like USB disconnect has occurred.

Example

This sample shows how to read all the data currently available.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD EventDWord;
DWORD TxBytes;
DWORD BytesReceived;
char RxBuffer[256];
```

```
ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

FT_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord);
if (RxBytes > 0) {
    ftStatus = FT_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived);
    if (ftStatus == FT_OK) {
        // FT_Read OK
    }
    else {
        // FT_Read Failed
    }
}

FT_Close(ftHandle);
```

This sample shows how to read with a timeout of 5 seconds.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD RxBytes = 10;
DWORD BytesReceived;
char RxBuffer[256];

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

FT_SetTimeouts(ftHandle,5000,0);
ftStatus = FT_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived);
if (ftStatus == FT_OK) {
    if (BytesReceived == RxBytes) {
        // FT_Read OK
    }
    else {
        // FT_Read Timeout
    }
}
else {
    // FT_Read Failed
}

FT_Close(ftHandle);
```

2.8 FT_Write

Write data to the device.

FT_STATUS **FT_Write** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten*)

Parameters

ftHandle	Handle of the device.
lpBuffer	Pointer to the buffer that contains the data to be written to the device.
dwBytesToWrite	Number of bytes to write to the device.
lpdwBytesWritten	Pointer to a variable of type DWORD which receives the number of bytes written to the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.9 FT_ResetDevice

This function sends a reset command to the device.

FT_STATUS **FT_ResetDevice** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.10 FT_SetBaudRate

This function sets the Baud rate for the device.

FT_STATUS **FT_SetBaudRate** (FT_HANDLE *ftHandle*, DWORD *dwBaudRate*)

Parameters

ftHandle	Handle of the device.
dwBaudRate	Baud rate.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.11 FT_SetDivisor

This function sets the Baud rate for the device. It is used to set non-standard Baud rates.

FT_STATUS **FT_SetDivisor** (FT_Handle *ftHandle*, USHORT *usDivisor*)

Parameters

ftHandle	Handle of the device.
usDivisor	Divisor.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The application note "Setting Baud rates for the FT8U232AM" is available from the Application Notes section of the FTDI website describes how to calculate the divisor for a non-standard Baud rate.

2.12 FT_SetDataCharacteristics

This function sets the data characteristics for the device.

FT_STATUS **FT_SetDataCharacteristics** (FT_HANDLE *ftHandle*, UCHAR *uWordLength*, UCHAR *uStopBits*,UCHAR *uParity*)

Parameters

ftHandle	Handle of the device.
uWordLength	Number of bits per word - must be FT_BITS_8 or FT_BITS_7.
uStopBits	Number of stop bits - must be FT_STOP_BITS_1 or FT_STOP_BITS_2.
uParity	FT_PARITY_NONE, FT_PARITY_ODD, _FT_PARITY_EVEN, FT_PARITY_MARK, FT_PARITY SPACE.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.13 FT_SetFlowControl

This function sets the flow control for the device.

FT_STATUS **FT_SetFlowControl** (FT_HANDLE *ftHandle*, USHORT us*FlowControl*, UCHAR u*Xon*,UCHAR u*Xoff*)

Parameters

ftHandle	Handle of the device.
usFlowControl	Must be one of FT_FLOW_NONE, FT_FLOW_RTS_CTS, FT_FLOW_DTR_DSR or FT_FLOW_XON_XOFF.
uXon	Character used to signal Xon. Only used if flow control is FT_FLOW_XON_XOFF.
uXoff	Character used to signal Xoff. Only used if flow control is FT_FLOW_XON_XOFF.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.14 FT_SetDtr

This function sets the Data Terminal Ready (DTR) control signal.

FT_STATUS **FT_SetDtr** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to set DTR.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}
ftStatus = FT_SetDtr(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetDtr OK
}
else {
    // FT_SetDtr failed
}

FT_Close(ftHandle);
```

2.15 FT_ClrDtr

This function clears the Data Terminal Ready (DTR) control signal.

FT_STATUS **FT_ClrDtr** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to clear DTR.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_ClrDtr(ftHandle);
if (ftStatus == FT_OK) {
    // FT_ClrDtr OK
}
else {
    // FT_ClrDtr failed
}

FT_Close(ftHandle);
```

2.16 FT_SetRts

This function sets the Request To Send (RTS) control signal.

FT_STATUS **FT_SetRts** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to set RTS.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetRts(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetRts OK
}
else {
    // FT_SetRts failed
}

FT_Close(ftHandle);
```

2.17 FT_ClrRts

This function clears the Request To Send (RTS) control signal.

FT_STATUS **FT_ClrRts** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to clear RTS.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_ClrRts(ftHandle);
if (ftStatus == FT_OK) {
    // FT_ClrRts OK
}
else {
    // FT_ClrRts failed
}

FT_Close(ftHandle);
```

2.18 FT_GetModemStatus

Gets the modem status from the device.

FT_STATUS **FT_GetModemStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwModemStatus*)

Parameters

ftHandle	Handle of the device.
lpdwModemStatus	Pointer to a variable of type DWORD which receives the modem status from the device. Status lines are bit-mapped as follows: CTS = 0x10 DSR = 0x20 RI = 0x40 DCD = 0x80

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.19 FT_SetChars

This function sets the special characters for the device.

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

Parameters

ftHandle	Handle of the device.
uEventCh	Event character.
uEventChEn	0 if event character disabled, non-zero otherwise.
uErrorCh	Error character.
uErrorChEn	0 if error character disabled, non-zero otherwise.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.20 FT_Purge

This function purges receive and transmit buffers in the device.

FT_STATUS **FT_Purge** (FT_HANDLE *ftHandle*, DWORD *dwMask*)

Parameters

ftHandle	Handle of the device.
dwMask	Any combination of FT_PURGE_RX and FT_PURGE_TX.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.21 FT_SetTimeouts

This function sets the read and write timeouts for the device.

FT_STATUS **FT_SetTimeouts** (FT_HANDLE *ftHandle*, DWORD *dwReadTimeout*, DWORD *dwWriteTimeout*)

Parameters

ftHandle	Handle of the device.
dwReadTimeout	Read timeout in milliseconds.
dwWriteTimeout	Write timeout in milliseconds.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to set a read timeout of 5 seconds and a write timeout of 1 second.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}
ftStatus = FT_SetTimeouts(ftHandle,5000,1000);
if (ftStatus == FT_OK) {
    // FT_SetTimeouts OK
}
else {
    // FT_SetTimeouts failed
}

FT_Close(ftHandle);
```

2.22 FT_GetQueueStatus

Gets the number of characters in the receive queue.

FT_STATUS **FT_GetQueueStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue*)

Parameters

ftHandle	Handle of the device.
lpdwAmountInRxQueue	Pointer to a variable of type DWORD which receives the number of characters in the receive queue.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

2.23 FT_SetBreakOn

Sets the BREAK condition for the device.

FT_STATUS **FT_SetBreakOn** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to set the BREAK condition for the device.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetBreakOn(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetBreakOn OK
}
else {
    // FT_SetBreakOn failed
}

FT_Close(ftHandle);
```

2.24 FT_SetBreakOff

Resets the BREAK condition for the device.

FT_STATUS **FT_SetBreakOff** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

This sample shows how to reset the BREAK condition for the device.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetBreakOff(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetBreakOff OK
}
else {
    // FT_SetBreakOff failed
}

FT_Close(ftHandle);
```

2.25 FT_GetStatus

Gets the device status including number of characters in the receive queue, number of characters in the transmit queue, and the current event status.

FT_STATUS **FT_GetStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue*, LPDWORD *lpdwAmountInTxQueue*, LPDWORD *lpdwEventStatus*)

Parameters

ftHandle	Handle of the device.
lpdwAmountInRxQueue	Pointer to a variable of type DWORD which receives the number of characters in the receive queue.
lpdwAmountInTxQueue	Pointer to a variable of type DWORD which receives the number of characters in the transmit queue.
lpdwEventStatus	Pointer to a variable of type DWORD which receives the current state of the event status.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

For an example of how to use this function, see the sample code in FT_SetEventNotification 35.

2.26 FT_SetEventNotification

Sets conditions for event notification.

FT_STATUS **FT_SetEventNotification** (FT_HANDLE *ftHandle*, DWORD *dwEventMask*, PVOID *pvArg*)

Parameters

ftHandle	Handle of the device.
dwEventMask	Conditions that cause the event to be set.
pvArg	Interpreted as the handle of an event.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

An application can use this function to setup conditions which allow a thread to block until one of the conditions is met. Typically, an application will create an event, call this function, then block on the event. When the conditions are met, the event is set, and the application thread unblocked.

dwEventMask is a bit-map that describes the events the application is interested in. *pvArg* is interpreted as the handle of an event which has been created by the application. If one of the event conditions is met, the event is set.

If *FT_EVENT_RXCHAR* is set in *dwEventMask*, the event will be set when a character has been received by the device. If *FT_EVENT_MODEM_STATUS* is set in *dwEventMask*, the event will be set when a change in the modem signals has been detected by the device.

Windows and Windows CE Example

This example shows how to wait for a character to be received or a change in modem status.

First, create the event and call **FT_SetEventNotification**.

```
FT_HANDLE ftHandle; // handle of an open device
FT_STATUS ftStatus;
HANDLE hEvent;
DWORD EventMask;

hEvent = CreateEvent(
                     NULL,
                     false, // auto-reset event
                     false, // non-signalled state
                     ""
                     );
EventMask = FT_EVENT_RXCHAR | FT_EVENT_MODEM_STATUS;
ftStatus = FT_SetEventNotification(ftHandle,EventMask,hEvent);
```

Sometime later, block the application thread by waiting on the event, then when the event has occurred, determine the condition that caused the event, and process it accordingly.

```
WaitForSingleObject(hEvent,INFINITE);

DWORD EventDWord;
DWORD RxBytes;
DWORD TxBytes;

FT_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord);
if (EventDWord & FT_EVENT_MODEM_STATUS) {
    // modem status event detected, so get current modem status
    FT_GetModemStatus(ftHandle,&Status);
    if (Status & 0x00000010) {
                // CTS is high
    }
    else {
                // CTS is low
    }
    if (Status & 0x00000020) {
                // DSR is high
    }
    else {
                // DSR is low
    }
}
if (RxBytes > 0) {
    // call FT_Read() to get received data from device
}
```

Linux Example

This example shows how to wait for a character to be received or a change in modem status.

First, create the event and call **FT_SetEventNotification**.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
EVENT_HANDLE eh;
DWORD EventMask;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

pthread_mutex_init(&eh.eMutex, NULL);
pthread_cond_init(&eh.eCondVar, NULL);

EventMask = FT_EVENT_RXCHAR | FT_EVENT_MODEM_STATUS;
ftStatus = FT_SetEventNotification(ftHandle, EventMask, (PVOID)&eh);
```

Sometime later, block the application thread by waiting on the event, then when the event has occurred, determine the condition that caused the event, and process it accordingly.

```
pthread_mutex_lock(&eh.eMutex);
pthread_cond_wait(&eh.eCondVar, &eh.eMutex);
pthread_mutex_unlock(&eh.eMutex);

DWORD EventDWord;
DWORD RxBytes;
DWORD TxBytes;
DWORD Status;
FT_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord);
if (EventDWord & FT_EVENT_MODEM_STATUS) {
    // modem status event detected, so get current modem status
    FT_GetModemStatus(ftHandle,&Status);
    if (Status & 0x00000010) {
                // CTS is high
    }
    else {
```

```
                    // CTS is low
        }
        if (Status & 0x00000020) {
                    // DSR is high
        }
        else {
                    // DSR is low
        }
    }
    if (RxBytes > 0) {
        // call FT_Read() to get received data from device
    }

    FT_Close(ftHandle);
```

2.27 FT_IoCtl

Undocumented function.

FT_STATUS **FT_IoCtl** (FT_HANDLE *ftHandle*, DWORD *dwIoControlCode*, LPVOID *lpInBuf*, DWORD *nInBufSize*, LPVOID *lpOutBuf*, DWORD *nOutBufSize*, LPDWORD *lpBytesReturned*, LPOVERLAPPED *lpOverlapped*)

2.28 FT_SetWaitMask

Undocumented function.

FT_STATUS **FT_SetWaitMask** (FT_HANDLE *ftHandle*, DWORD *dwMask*)

2.29 FT_WaitOnMask

Undocumented function.

FT_STATUS **FT_WaitOnMask** (FT_HANDLE *ftHandle*, DWORD *dwMask*)

2.30 FT_GetDeviceInfo

Get device information.

FT_STATUS **FT_GetDeviceInfo** (FT_HANDLE ft*Handle*, FT_DEVICE **pftType*, LPDWORD *lpdwID*, PCHAR *pcSerialNumber*, PCHAR *pcDescription*, PVOID *pvDummy*)

Parameters

ftHandle	Handle of the device.
pftType	Pointer to unsigned long to store device type.
lpdwId	Pointer to unsigned long to store device ID.
pcSerialNumber	Pointer to buffer to store device serial number as a null-terminated string.
pcDescription	Pointer to buffer to store device description as a null-terminated string.
pvDummy	Reserved for future use - should be set to NULL.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to return the device type, device ID, device description and serial number.

The device ID is encoded in a DWORD - the most significant word contains the vendor ID, and the least significant word contains the product ID. So the returned ID 0x04036001 corresponds to the device ID VID_0403&PID_6001.

Example

This example shows how to get information about a device.

```
FT_HANDLE ftHandle;
FT_DEVICE ftDevice;
FT_STATUS ftStatus;
DWORD deviceID;
char SerialNumber[16];
char Description[64];

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_GetDeviceInfo(
                        ftHandle,
                        &ftDevice,
                        &deviceID,
                        SerialNumber,
                        Description,
```

```
                        NULL
                        );

if (ftStatus == FT_OK) {
                if (ftDevice == FT_DEVICE_2232C)
                        ; // device is FT2232C
                else if (ftDevice == FT_DEVICE_BM)
                        ; // device is FTU232BM
                else if (ftDevice == FT_DEVICE_AM)
                        ; // device is FT8U232AM
                else
                        ; // unknown device (this should not happen!)
                // deviceID contains encoded device ID
                // SerialNumber, Description contain 0-terminated strings
}
else {
                // FT_GetDeviceType FAILED!
}

FT_Close(ftHandle);
```

2.31 FT_SetResetPipeRetryCount

Set the ResetPipeRetryCount.

FT_STATUS **FT_SetResetPipeRetryCount** (FT_HANDLE ft*Handle*, DWORD *dwCount*)

Parameters

ftHandle	Handle of the device.
dwCount	Unsigned long containing required ResetPipeRetryCount.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to set the ResetPipeRetryCount. ResetPipeRetryCount controls the maximum number of times that the driver tries to reset a pipe on which an error has occurred. ResetPipeRequestRetryCount defaults to 50. It may be necessary to increase this value in noisy environments where a lot of USB errors occur.

Not available in Linux.

Example

This example shows how to set the ResetPipeRetryCount to 100.

```
FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;
DWORD dwRetryCount;

dwRetryCount = 100;
ftStatus = FT_SetResetPipeRetryCount(ftHandle,dwRetryCount);
if (ftStatus == FT_OK) {
        // ResetPipeRetryCount set to 100
}
else {
        // FT_SetResetPipeRetryCount FAILED!
}
```

2.32 FT_StopInTask

Stops the driver's IN task.

FT_STATUS **FT_StopInTask** (FT_HANDLE ft*Handle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to put the driver's IN task (read) into a wait state. It can be used in situations where data is being received continuously, so that the device can be purged without more data being received. It is used together with FT_RestartInTask 45 which sets the IN task running again.

Example

This example shows how to use FT_StopInTask.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

do {
        ftStatus = FT_StopInTask(ftHandle);
} while (ftStatus != FT_OK);

//
// Do something - for example purge device
//

do {
        ftStatus = FT_RestartInTask(ftHandle);
} while (ftStatus != FT_OK);

FT_Close(ftHandle);
```

2.33 FT_RestartInTask

Restart the driver's IN task.

FT_STATUS **FT_RestartInTask** (FT_HANDLE ft*Handle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to restart the driver's IN task (read) after it has been stopped by a call to FT_StopInTask 44.

Example

This example shows how to use FT_RestartInTask.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

do {
        ftStatus = FT_StopInTask(ftHandle);
} while (ftStatus != FT_OK);

//
// Do something - for example purge device
//

do {
        ftStatus = FT_RestartInTask(ftHandle);
} while (ftStatus != FT_OK);

FT_Close(ftHandle);
```

2.34 FT_ResetPort

Send a reset command to the port.

FT_STATUS **FT_ResetPort** (FT_HANDLE ft*Handle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to attempt to recover the port after a failure. It is not equivalent to an unplug-replug event.

Not available in Windows CE and Linux.

Example

This example shows how to reset the port.

```
FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_ResetPort(ftHandle);
if (ftStatus == FT_OK) {
        // Port has been reset
}
else {
        // FT_ResetPort FAILED!
}
```

2.35 FT_CyclePort

Send a cycle command to the USB port.

FT_STATUS **FT_CyclePort** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The effect of this function is the same as disconnecting then reconnecting the device from USB. Possible use of this function is in situations where a fatal error has occurred and it is difficult, or not possible, to recover without unplugging and replugging the USB cable. This function can also be used after re-programming the EEPROM to force the FTDI device to read the new EEPROM contents which previously required a physical disconnect-reconnect.

As the current session is not restored when the driver is reloaded, the application must be able to recover after calling this function.

Not available in Windows 98, Windows CE and Linux.

For FT2232C devices, FT_CyclePort will only work under Windows XP and later.

Example

This example shows how to cycle the port.

```
FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_CyclePort(ftHandle);
if (ftStatus == FT_OK) {
        // Port has been cycled.
}
else {
        // FT_CyclePort FAILED!
}
```

2.36 FT_CreateDeviceInfoList

This function builds a device information list and returns the number of D2XX devices connected to the system. The list contains information about both unopen and open devices.

FT_STATUS **FT_CreateDeviceInfoList** (LPDWORD *lpdwNumDevs*)

Parameters

lpdwNumDevs	Pointer to unsigned long to store the number of devices connected.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

An application can use this function to get the number of devices attached to the system. It can then allocate space for the device information list and retrieve the list using FT_GetDeviceInfoList 49.

If the devices connected to the system change, the device info list will not be updated until FT_CreateDeviceInfoList 48 is called again.

Example

This example shows how to call FT_CreateDeviceInfoList.

```
FT_STATUS ftStatus;
DWORD numDevs;

//
// create the device information list
//
ftStatus = FT_CreateDeviceInfoList(&numDevs);
if (ftStatus == FT_OK) {
        printf("Number of devices is %d\n",numDevs);
}
else {
        // FT_CreateDeviceInfoList failed
}
```

2.37 FT_GetDeviceInfoList

This function returns a device information list and the number of D2XX devices in the list.

FT_STATUS **FT_GetDeviceInfo** (FT_DEVICE_LIST_INFO_NODE **pDest*, LPDWORD *lpdwNumDevs*)

Parameters

**pDest*	Pointer to an array of FT_DEVICE_LIST_INFO_NODE structures.
lpdwNumDevs	Pointer to the number of elements in the array.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function should only be called after calling FT_CreateDeviceInfoList 48. If the devices connected to the system change, the device info list will not be updated until FT_CreateDeviceInfoList 48 is called again.

Location ID information is not returned for devices that are open when FT_CreateDeviceInfoList 48 is called.

The array of FT_DEVICE_LIST_INFO_NODES contains all available data on each device. The structure of FT_DEVICE_LIST_INFO_NODES is given in the Appendix 105. The storage for the list must be allocated by the application. The number of devices returned by FT_CreateDeviceInfoList 48 can be used to do this.

When programming in Visual Basic, LabVIEW or similar languages, FT_GetDeviceInfoDetail 51 may be required instead of this function.

Please note that Windows CE and Linux do not support location IDs. As such, the Location ID parameter in the structure will be empty under Windows CE and Linux.

Example

This example shows how to call FT_GetDeviceInfoList.

```
FT_STATUS ftStatus;
FT_DEVICE_LIST_INFO_NODE *devInfo;
DWORD numDevs;

//
// create the device information list
//
ftStatus = FT_CreateDeviceInfoList(&numDevs);
if (ftStatus == FT_OK) {
        printf("Number of devices is %d\n",numDevs);
}

//
```

```
// allocate storage for list based on numDevs
//
devInfo = (FT_DEVICE_LIST_INFO_NODE*)malloc(sizeof(FT_DEVICE_LIST_INFO_NODE)*numDevs);

//
// get the device information list
//
ftStatus = FT_GetDeviceInfoList(devInfo,&numDevs);
if (ftStatus == FT_OK) {
    for (int i = 0; i < numDevs; i++) {
        printf("Dev %d:\n",i);
               printf("  Flags=0x%x\n",devInfo[i].Flags);
               printf("  Type=0x%x\n",devInfo[i].Type);
               printf("  ID=0x%x\n",devInfo[i].ID);
               printf("  LocId=0x%x\n",devInfo[i].LocId);
               printf("  SerialNumber=%s\n",devInfo[i].SerialNumber);
               printf("  Description=%s\n",devInfo[i].Description);
               printf("  ftHandle=0x%x\n",devInfo[i].ftHandle);
        }
}
```

2.38 FT_GetDeviceInfoDetail

This function returns an entry from the device information list.

FT_STATUS **FT_GetDeviceInfoDetail** (DWORD *dwIndex*, LPDWORD *lpdwFlags*, LPDWORD *lpdwType*, LPDWORD *lpdwID*, LPDWORD *lpdwLocId*, PCHAR *pcSerialNumber*, PCHAR *pcDescription*, FT_HANDLE **ftHandle*)

Parameters

dwIndex	Index of the entry in the device info list.
lpdwFlags	Pointer to unsigned long to store the flag value.
lpdwType	Pointer to unsigned long to store device type.
lpdwID	Pointer to unsigned long to store device ID.
lpdwLocId	Pointer to unsigned long to store the device location ID.
pcSerialNumber	Pointer to buffer to store device serial number as a null-terminated string.
pcDescription	Pointer to buffer to store device description as a null-terminated string.
**ftHandle*	Pointer to a variable of type FT_HANDLE where the handle will be stored.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function should only be called after calling FT_CreateDeviceInfoList 48. If the devices connected to the system change, the device info list will not be updated until FT_CreateDeviceInfoList 48 is called again.

The index value is zero-based.

The flag value is a 4-byte bit map containing miscellaneous data. Bit 0 (least significant bit) of this number indicates if the port is open (1) or closed (0). The remaining bits (1 - 31) are reserved at this time.

Location ID information is not returned for devices that are open when FT_CreateDeviceInfoList 48 is called.

To return the whole device info list as an array of FT_DEVICE_LIST_INFO_NODE structures, use FT_GetDeviceInfoList 49.

Please note that Windows CE and Linux do not support location IDs. As such, the Location ID parameter in the structure will be empty under Windows CE and Linux.

Example

This example shows how to call FT_GetDeviceInfoDetail.

```
FT_STATUS ftStatus;
FT_HANDLE ftHandleTemp;
DWORD numDevs;
DWORD Flags;
DWORD ID;
DWORD Type;
DWORD LocId;
char SerialNumber[16];
char Description[64];

//
// create the device information list
//
ftStatus = FT_CreateDeviceInfoList(&numDevs);
if (ftStatus == FT_OK) {
        printf("Number of devices is %d\n",numDevs);
}

//
// get information for device 0
//
ftStatus = FT_GetDeviceInfoDetail(0, &Flags, &Type, &ID, &LocId, SerialNumber,
Description, &ftHandleTemp);
if (ftStatus == FT_OK) {
        printf("Dev 0:\n");
        printf("  Flags=0x%x\n",Flags);
        printf("  Type=0x%x\n",Type);
        printf("  ID=0x%x\n",ID);
        printf("  LocId=0x%x\n",LocId);
        printf("  SerialNumber=%s\n",SerialNumber);
        printf("  Description=%s\n",Description);
        printf("  ftHandle=0x%x\n",ftHandleTemp);
}
```

2.39 FT_GetDriverVersion

This function returns the D2XX driver version number.

FT_STATUS **FT_GetDriverVersion** (FT_HANDLE *ftHandle*, LPDWORD *lpdwDriverVersion*)

Parameters

ftHandle	Handle of the device.
lpdwDriverVersion	Pointer to the driver version number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

A version number consists of major, minor and build version numbers contained in a 4-byte field (unsigned long). Byte0 (least significant) holds the build version, Byte1 holds the minor version, and Byte2 holds the major version. Byte3 is currently set to zero.

For example, driver version "3.01.02" is represented as 0x00030102. Note that a device has to be opened before this function can be called.

Not available in Windows CE or Linux.

Example

This example shows how to call FT_GetDriverVersion.

```
FT_HANDLE ftHandle;
FT_STATUS status;
DWORD dwDriverVer;

//
// Get driver version
//

status = FT_Open(0,&ftHandle);
if (status == FT_OK) {
        status = FT_GetDriverVersion(ftHandle,&dwDriverVer);
        if (status == FT_OK)
                printf("Driver version = 0x%x\n",dwDriverVer);
        else
                printf("error reading driver version\n");
                FT_Close(ftHandle);
}
```

2.40 FT_GetLibraryVersion

This function returns D2XX DLL version number.

FT_STATUS **FT_GetLibraryVersion** (LPDWORD *lpdwDLLVersion*)

Parameters

lpdwDLLVersion	Pointer to the DLL version number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

A version number consists of major, minor and build version numbers contained in a 4-byte field (unsigned long). Byte0 (least significant) holds the build version, Byte1 holds the minor version, and Byte2 holds the major version. Byte3 is currently set to zero.

For example, driver version "3.01.02" is represented as 0x00030102. Note that this function does not take a handle, and so it can be called without opening a device.

Not available in Windows CE or Linux.

Example

This example shows how to call FT_GetLibraryVersion.

```
FT_STATUS status;
DWORD dwLibraryVer;

//
// Get DLL version
//

status = FT_GetLibraryVersion(&dwLibraryVer);
if (status == FT_OK)
        printf("Library version = 0x%x\n",dwLibraryVer);
else
        printf("error reading library version\n");
```

2.41 FT_SetDeadmanTimeout

This function allows the maximum time in milliseconds that a USB request can remain outstanding to be set.

FT_STATUS **FT_SetDeadmanTimeout** (FT_HANDLE *ftHandle*, DWORD *dwDeadmanTimeout*)

Parameters

ftHandle	Handle of the device.
dwDeadmanTimeout	Deadman timeout value in milliseconds. Default value is 5000.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The Deadman timeout is referred to in AN232B-10 Advanced Driver Options as the USB timeout. It is unlikely that this function will be required by most users.

Example

This example shows how to call FT_SetDeadmanTimeout.

```
FT_HANDLE ftHandle;
FT_STATUS status;
DWORD dwDeadmanTimeout = 6000;


ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetDeadmanTimeout(ftHandle,dwDeadmanTimeout);
if (ftStatus == FT_OK) {
        // Set Deadman Timer to 6 seconds
}
else {
        // FT_SetDeadmanTimeout FAILED!
}

FT_Close(ftHandle);
```

2.42 FT_Rescan

This function can be of use when trying to recover devices programatically.

FT_STATUS **FT_Rescan** ()

Parameters
None.

Return Value
FT_OK if successful, otherwise the return value is an FT error code.

Remarks
Calling FT_Rescan is equivalent to clicking the "Scan for hardware changes" button in the Device Manager. Only USB hardware is checked for new devices.

Not available in Windows 98, Windows CE and Linux.

Example

This example shows how to call FT_Rescan.

```
FT_STATUS ftstatus;

ftStatus = FT_Rescan();
if(ftStatus != FT_OK) {
        // FT_Rescan failed!
        return;
}
```

2.43 FT_Reload

This function forces a reload of the driver for devices with a specific VID and PID combination.

FT_STATUS **FT_Reload** (WORD *wVID*, WORD *wPID*)

Parameters

wVID	Vendor ID of the devices to reload the driver for.
wPID	Product ID of the devices to reload the driver for.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

Calling FT_Reload forces the operating system to unload and reload the driver for the specified device IDs. If the VID and PID parameters are null, the drivers for USB root hubs will be reloaded, causing all USB devices connected to reload their drivers. **Please note that this function will not work correctly on 64-bit Windows when called from a 32-bit application.**

Not available in Windows 98, Windows CE and Linux.

Example

This example shows how to call FT_Reload to reload the driver for a standard FT232R device (VID 0x0403, PID 0x6001).

```
FT_STATUS ftstatus;
WORD wVID = 0x0403;
WORD wPID = 0x6001;

ftStatus = FT_Reload(wVID,wPID);
if(ftStatus != FT_OK) {
        // FT_Reload failed!
        return;
}
```

This example shows how to call FT_Reload to reload the drivers for all USB devices.

```
FT_STATUS ftstatus;
WORD wVID = 0x0000;
WORD wPID = 0x0000;

ftStatus = FT_Reload(wVID,wPID);
if(ftStatus != FT_OK) {
        // FT_Reload failed!
        return;
}
```

3 EEPROM Programming Interface Functions

Introduction

FTDI has included EEPROM programming support in the D2XX library. This section describes that interface.

Overview

Functions are provided to program the EEPROM (FT_EE_Program 65, FT_EE_ProgramEx 67, FT_WriteEE 60), read the EEPROM (FT_EE_Read 62, FT_EE_ReadEx 64, FT_ReadEE 59) and erase the EEPROM (FT_EraseEE 61).

Unused space in the EEPROM is called the User Area (EEUA). Functions are provided to access the EEUA. FT_EE_UASize 70 gets it's size, FT_EE_UAWrite 69 writes data into it and FT_EE_UARead 68 is used to read it's contents.

Reference

Type definitions 105 of the functional parameters and return codes used in the D2XX EEPROM programming interface are contained in the appendix 104.

3.1 FT_ReadEE

Read a value from an EEPROM location.

FT_STATUS **FT_ReadEE** (FT_HANDLE *ftHandle*, DWORD *dwWordOffset*, LPWORD *lpwValue*)

Parameters

ftHandle	Handle of the device.
dwWordOffset	EEPROM location to read from.
lpwValue	Pointer to the value read from the EEPROM.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

3.2 FT_WriteEE

Write a value to an EEPROM location.

FT_STATUS **FT_WriteEE** (FT_HANDLE *ftHandle*, DWORD *dwWordOffset*, WORD *wValue*)

Parameters

ftHandle	Handle of the device.
dwWordOffset	EEPROM location to write to.
wValue	Value to write to EEPROM.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

3.3 FT_EraseEE

Erase the EEPROM.

FT_STATUS **FT_EraseEE** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function will erase the entire contents of an EEPROM, including the user area.

3.4 FT_EE_Read

Read the contents of the EEPROM.

FT_STATUS **FT_EE_Read** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*)

Parameters

ftHandle	Handle of the device.
pData	Pointer to structure of type FT_PROGRAM_DATA.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pData* as a pointer to a struct of type *FT_PROGRAM_DATA* that contains storage for the data to be read from the EEPROM.

The function does not perform any checks on buffer sizes, so the buffers passed in the *FT_PROGRAM_DATA* struct must be big enough to accommodate their respective strings (including null terminators). The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the Manufacturer string length plus the Description string length is less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);
if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

FT_PROGRAM_DATA ftData;
char ManufacturerBuf[32];
char ManufacturerIdBuf[16];
char DescriptionBuf[64];
char SerialNumberBuf[16];

ftData.Signature1 = 0x00000000;
ftData.Signature2 = 0xffffffff;
ftData.Version = 0x00000002;          // EEPROM structure with FT232R extensions
ftData.Manufacturer = ManufacturerBuf;
ftData.ManufacturerId = ManufacturerIdBuf;
ftData.Description = DescriptionBuf;
ftData.SerialNumber = SerialNumberBuf;
```

```
ftStatus = FT_EE_Read(ftHandle, &ftData);
if (ftStatus == FT_OK) {
    // FT_EE_Read OK, data is available in ftData
}
else {
    // FT_EE_Read FAILED!
}
```

3.5 FT_EE_ReadEx

Read the contents of the EEPROM and pass strings separately.

FT_STATUS **FT_EE_ReadEx** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*, char **Manufacturer*, char **ManufacturerId*, char **Description*, char **SerialNumber*)

Parameters

ftHandle	Handle of the device.
pData	Pointer to a structure of type FT_PROGRAM_DATA.
**Manufacturer*	Pointer to a null-terminated string containing the manufacturer name.
**ManufacturerID*	Pointer to a null-terminated string containing the manufacturer ID.
**Description*	Pointer to a null-terminated string containing the device description.
**SerialNumber*	Pointer to a null-terminated string containing the device serial number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This variation of the standard FT_EE_Read 62 function was included to provide support for languages such as LabVIEW where problems can occur when string pointers are contained in a structure.

This function interprets the parameter *pData* as a pointer to a struct of type *FT_PROGRAM_DATA* that contains storage for the data to be read from the EEPROM.

The function does not perform any checks on buffer sizes, so the buffers passed in the *FT_PROGRAM_DATA* structure must be big enough to accommodate their respective strings (including null terminators). The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the Manufacturer string length plus the Description string length is less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h 111.

The string parameters in the *FT_PROGRAM_DATA* structure should be passed as DWORDs to avoid overlapping of parameters. All string pointers are passed out separately from the *FT_PROGRAM_DATA* structure.

3.6 FT_EE_Program

Program the EEPROM.

FT_STATUS **FT_EE_Program** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*)

Parameters

ftHandle	Handle of the device.
pData	Pointer to structure of type FT_PROGRAM_DATA.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pData* as a pointer to a struct of type FT_PROGRAM_DATA that contains the data to write to the EEPROM. The data is written to EEPROM, then read back and verified.

If the SerialNumber field in FT_PROGRAM_DATA is NULL, or SerialNumber points to a NULL string, a serial number based on the ManufacturerId and the current date and time will be generated.

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h.

If *pData* is NULL, the structure version will default to 0 (original BM series) and the device will be programmed with the default data:
{0x0403, 0x6001, "FTDI", "FT", "USB HS Serial Converter", "", 44, 1, 0, 1, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, 0}

Example

```
// Version 2 structure for programming a BM device.
// Other elements would need non-zero values for FT2232C, FT232R or FT245R devices.

FT_PROGRAM_DATA ftData = {
        0x00000000,                     // Header - must be 0x00000000
        0xFFFFFFFF,                     // Header - must be 0xffffffff
        0x00000002,                     // Header - FT_PROGRAM_DATA version
        0x0403,                         // VID
        0x6001,                         // PID
        "FTDI",                         // Manufacturer
        "FT",                           // Manufacturer ID
        "USB HS Serial Converter",      // Description
        "FT000001",                     // Serial Number
        44,                             // MaxPower
        1,                              // PnP
```

```
        0,                              // SelfPowered
        1,                              // RemoteWakeup
        1,                              // non-zero if Rev4 chip, zero otherwise
        0,                              // non-zero if in endpoint is isochronous
        0,                              // non-zero if out endpoint is isochronous
        0,                              // non-zero if pull down enabled
        1,                              // non-zero if serial number to be used
        0,                              // non-zero if chip uses USBVersion
        0x0110                          // BCD (0x0200 => USB2)
        //
        // FT2232C extensions (Enabled if Version = 1 or Version = 2)
        //
        0,                              // non-zero if Rev5 chip, zero otherwise
        0,                              // non-zero if in endpoint is isochronous
        0,                              // non-zero if in endpoint is isochronous
        0,                              // non-zero if out endpoint is isochronous
        0,                              // non-zero if out endpoint is isochronous
        0,                              // non-zero if pull down enabled
        0,                              // non-zero if serial number to be used
        0,                              // non-zero if chip uses USBVersion
        0x0,                            // BCD (0x0200 => USB2)
        0,                              // non-zero if interface is high current
        0,                              // non-zero if interface is high current
        0,                              // non-zero if interface is 245 FIFO
        0,                              // non-zero if interface is 245 FIFO CPU target
        0,                              // non-zero if interface is Fast serial
        0,                              // non-zero if interface is to use VCP drivers
        0,                              // non-zero if interface is 245 FIFO
        0,                              // non-zero if interface is 245 FIFO CPU target
        0,                              // non-zero if interface is Fast serial
        0,                              // non-zero if interface is to use VCP drivers
        //
        // FT232R extensions (Enabled if Version = 2)
        //
        0,                              // Use External Oscillator
        0,                              // High Drive I/Os
        0,                              // Endpoint size
        0,                              // non-zero if pull down enabled
        0,                              // non-zero if serial number to be used
        0,                              // non-zero if invert TXD
        0,                              // non-zero if invert RXD
        0,                              // non-zero if invert RTS
        0,                              // non-zero if invert CTS
        0,                              // non-zero if invert DTR
        0,                              // non-zero if invert DSR
        0,                              // non-zero if invert DCD
        0,                              // non-zero if invert RI
        0,                              // Cbus Mux control
        0,                              // Cbus Mux control
        0,                              // Cbus Mux control
        0,                              // Cbus Mux control
        0,                              // Cbus Mux control
        0                               // non-zero if using D2XX drivers
};
FT_HANDLE ftHandle;

FT_STATUS ftStatus = FT_Open(0, &ftHandle);
if (ftStatus == FT_OK) {
    ftStatus = FT_EE_Program(ftHandle, &ftData);
    if (ftStatus == FT_OK) {
        // FT_EE_Program OK!
    }
    else {
        // FT_EE_Program FAILED!
        }
}
```

3.7 FT_EE_ProgramEx

Program the EEPROM and pass strings separately.

FT_STATUS **FT_EE_ProgramEx** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*, char **Manufacturer*, char **ManufacturerId*, char **Description*, char **SerialNumber*)

Parameters

ftHandle	Handle of the device.
pData	Pointer to a structure of type FT_PROGRAM_DATA.
**Manufacturer*	Pointer to a null-terminated string containing the manufacturer name.
**ManufacturerID*	Pointer to a null-terminated string containing the manufacturer ID.
**Description*	Pointer to a null-terminated string containing the device description.
**SerialNumber*	Pointer to a null-terminated string containing the device serial number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This variation of the FT_EE_Program 65 function was included to provide support for languages such as LabVIEW where problems can occur when string pointers are contained in a structure.

This function interprets the parameter *pData* as a pointer to a struct of type FT_PROGRAM_DATA that contains the data to write to the EEPROM. The data is written to EEPROM, then read back and verified.

The string pointer parameters in the FT_PROGRAM_DATA structure should be allocated as DWORDs to avoid overlapping of parameters. The string parameters are then passed in separately.

If the SerialNumber field is NULL, or SerialNumber points to a NULL string, a serial number based on the ManufacturerId and the current date and time will be generated.

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h 111.

If *pData* is NULL, the structure version will default to 0 (original BM series) and the device will be programmed with the default data:
{0x0403, 0x6001, "FTDI", "FT", "USB HS Serial Converter", "", 44, 1, 0, 1, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, 0}

3.8 FT_EE_UARead

Read the contents of the EEUA.

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

Parameters

ftHandle	Handle of the device.
pucData	Pointer to a buffer that contains storage for data to be read.
dwDataLen	Size, in bytes, of buffer that contains storage for the data to be read.
lpdwBytesRead	Pointer to a DWORD that receives the number of bytes read..

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pucData* as a pointer to an array of bytes of size *dwDataLen* that contains storage for the data to be read from the EEUA. The actual number of bytes read is stored in the DWORD referenced by *lpdwBytesRead*.

If *dwDataLen* is less than the size of the EEUA, then *dwDataLen* bytes are read into the buffer. Otherwise, the whole of the EEUA is read into the buffer.

An application should check the function return value and *lpdwBytesRead* when **FT_EE_UARead** returns.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);

if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

unsigned char Buffer[64];
DWORD BytesRead;

ftStatus = FT_EE_UARead(ftHandle, Buffer, 64, &BytesRead);
if (ftStatus == FT_OK) {
    // FT_EE_UARead OK
    // User Area data stored in Buffer
    // Number of bytes read from EEUA stored in BytesRead
}
else {
    // FT_EE_UARead FAILED!
}
```

3.9 FT_EE_UAWrite

Write data into the EEUA.

FT_STATUS **FT_EE_UAWrite** (FT_HANDLE *ftHandle*, PUCHAR *pucData*, DWORD *dwDataLen*)

Parameters

ftHandle	Handle of the device.
pucData	Pointer to a buffer that contains the data to be written.
dwDataLen	Size, in bytes, of buffer that contains the data to be written.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *lpData* as a pointer to an array of bytes of size *dwDataLen* that contains the data to be written to the EEUA. It is a programming error for *dwDataLen* to be greater than the size of the EEUA.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);

if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

char *buffer = "Hello, World";

ftStatus = FT_EE_UAWrite(ftHandle, (unsigned char*)buffer, 12);
if(ftStatus != FT_OK) {
        // FT_EE_UAWRITE failed
}
else {
        // FT_EE_UAWRITE failed
}
```

3.10 FT_EE_UASize

Get size of EEUA.

FT_STATUS **FT_EE_UASize** (FT_HANDLE *ftHandle*, LPDWORD *lpdwSize*)

Parameters

ftHandle	Handle of the device.
lpdwSize	Pointer to a DWORD that receives the size, in bytes, of the EEUA.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);

if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

DWORD EEUA_Size;

ftStatus = FT_EE_UASize(ftHandle, &EEUA_Size);
if (ftStatus == FT_OK) {
    // FT_EE_UASize OK
    // EEUA_Size contains the size, in bytes, of the EEUA
}
else {
    // FT_EE_UASize FAILED!
}
```

4 Extended API Functions

Introduction

FTDI's FT232R USB UART (4th generation), FT245R USB FIFO (4th generation), FT2232C Dual USB UART/FIFO (3rd generation), FT232BM USB UART (2nd generation) and FT245BM USB FIFO (2nd generation) offer extra functionality, including programmable features, to their predecessors. The programmable features are supported by extensions to the D2XX driver, and the programming interface is exposed by FTD2XX.DLL.

Overview

New features include a programmable receive buffer timeout and bit bang mode. The receive buffer timeout is controlled via the latency timer functions FT_GetLatencyTimer 72 and FT_SetLatencyTimer 73. Bit bang modes and other FT2232C bit modes are controlled via the functions FT_GetBitMode 74 and FT_SetBitMode 75. Before these functions can be accessed, the device must first be opened. The Win32API function, CreateFile, returns a handle that is used by all functions in the programming interface to identify the device.

Reference

Type definitions 105 of the functional parameters and return codes used in the D2XX extended programming interface are contained in the appendix 104.

4.1 FT_GetLatencyTimer

Get the current value of the latency timer.

FT_STATUS **FT_GetLatencyTimer** (FT_HANDLE *ftHandle*, PUCHAR *pucTimer*)

Parameters

ftHandle	Handle of the device.
pucTimer	Pointer to unsigned char to store latency timer value.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

In the FT8U232AM and FT8U245AM devices, the receive buffer timeout that is used to flush remaining data from the receive buffer was fixed at 16 ms. In all other FTDI devices, this timeout is programmable and can be set at 1 ms intervals between 2ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response times from short data packets.

Example

```
HANDLE ftHandle;
FT_STATUS ftStatus;
UCHAR LatencyTimer;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_GetLatencyTimer(ftHandle, &LatencyTimer);
if (ftStatus == FT_OK) {
  // LatencyTimer contains current value
}
else {
  // FT_GetLatencyTimer FAILED!
}

FT_Close(ftHandle);
```

4.2 FT_SetLatencyTimer

Set the latency timer.

FT_STATUS **FT_SetLatencyTimer** (FT_HANDLE *ftHandle*, UCHAR *ucTimer*)

Parameters

ftHandle	Handle of the device.
ucTimer	Required value, in milliseconds, of latency timer. Valid range is 2 - 255.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

In the FT8U232AM and FT8U245AM devices, the receive buffer timeout that is used to flush remaining data from the receive buffer was fixed at 16 ms. In all other FTDI devices, this timeout is programmable and can be set at 1 ms intervals between 2ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response times from short data packets.

Example

```
HANDLE ftHandle;
FT_STATUS ftStatus;
UCHAR LatencyTimer = 10;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetLatencyTimer(ftHandle, LatencyTimer);
if (ftStatus == FT_OK) {
  // LatencyTimer set to 10 milliseconds
}
else {
  // FT_SetLatencyTimer FAILED!
}

FT_Close(ftHandle);
```

4.3 FT_GetBitMode

Gets the instantaneous value of the data bus.

FT_STATUS **FT_GetBitMode** (FT_HANDLE *ftHandle*, PUCHAR *pucMode*)

Parameters

ftHandle	Handle of the device.
pucMode	Pointer to unsigned char to store bit mode value.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

For a description of available bit modes for the FT2232C, see the application note "Bit Mode Functions for the FT2232C".
For a description of Bit Bang Mode for the FT232BM and FT245BM, see the application note "FT232BM/FT245BM Bit Bang Mode".
These application notes are available for download from the Application Notes page in the Documents section of the FTDI website.

Example

```
HANDLE ftHandle;
UCHAR BitMode;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_GetBitMode(ftHandle, &BitMode);
if (ftStatus == FT_OK) {
  // BitMode contains current value
}
else {
  // FT_GetBitMode FAILED!
}

FT_Close(ftHandle);
```

4.4 FT_SetBitMode

Set the value of the bit mode.

FT_STATUS **FT_SetBitMode** (FT_HANDLE *ftHandle*, UCHAR *ucMask*, UCHAR *ucMode*)

Parameters

ftHandle	Handle of the device.
ucMask	Required value for bit mode mask. This sets up which bits are inputs and outputs. A bit value of 0 sets the corresponding pin to an input, a bit value of 1 sets the corresponding pin to an output. In the case of CBUS Bit Bang, the upper nibble of this value controls which pins are inputs and outputs, while the lower nibble controls which of the outputs are high and low.
ucMode	Mode value. Can be one of the following: 0x0 = Reset 0x1 = Asynchronous Bit Bang 0x2 = MPSSE (FT2232C devices only) 0x4 = Synchronous Bit Bang (FT232R, FT245R and FT2232C devices only) 0x8 = MCU Host Bus Emulation Mode (FT2232C devices only) 0x10 = Fast Opto-Isolated Serial Mode (FT2232C devices only) 0x20 = CBUS Bit Bang Mode (FT232R devices only)

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

For a description of Bit Bang Mode for the FT232BM and FT245BM, see the application note "FT232BM/FT245BM Bit Bang Mode".
For a description of available bit modes for the FT2232C, see the application note "Bit Mode Functions for the FT2232C".
For a description of the Bit Bang modes available for the FT232R and FT245R devices, see the application note "Bit Bang Modes for the FT232R and FT245R".
Application notes are available for download from the Application Notes page in the Documents section of the FTDI website.

Note that to use CBUS Bit Bang for the FT232R, the CBUS must be configured for CBUS Bit Bang in the EEPROM.

Example

```
HANDLE ftHandle;
FT_STATUS ftStatus;
UCHAR Mask = 0xff;
UCHAR Mode = 1;  // Set asynchronous bit-bang mode

ftStatus = FT_Open(0, &ftHandle);
```

```
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
if (ftStatus == FT_OK) {
  // 0xff written to device
}
else {
  // FT_SetBitMode FAILED!
}

FT_Close(ftHandle);
```

4.5 FT_SetUSBParameters

Set the USB request transfer size.

FT_STATUS **FT_SetUSBParameters** (FT_HANDLE *ftHandle*, DWORD *dwInTransferSize*, DWORD *dwOutTransferSize*)

Parameters

ftHandle	Handle of the device.
dwInTransferSize	Transfer size for USB IN request.
dwOutTransferSize	Transfer size for USB OUT request.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

Previously, USB request transfer sizes have been set at 4096 bytes and have not been configurable. This function can be used to change the transfer sizes to better suit the application requirements. Transfer sizes must be set to a multiple of 64 bytes between 64 bytes and 64k bytes.

When FT_SetUSBParameters is called, the change comes into effect immediately and any data that was held in the driver at the time of the change is lost.

Note that, at present, only *dwInTransferSize* is supported.

Example

```
HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD InTransferSize = 16384;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
        // FT_Open failed
        return;
}

ftStatus = FT_SetUSBParameters(ftHandle, InTransferSize, 0);
if (ftStatus == FT_OK) {
  // In transfer size set to 16 Kbytes
}
else {
  // FT_SetUSBParameters FAILED!
}

FT_Close(ftHandle);
```

5 FT-Win32 API Functions

Introduction

The D2XX interface also incorporates functions based on Win32 API and Win32 COMM API calls. This facilitates the porting of communications applications from VCP to D2XX.

Linux support for these functions has been added to allow porting windows VCP applications to Linux as well as providing improved event detection. Every effort has been made to keep the continuity between the Linux and Windows implementation of these funcitons as much as possible. The notable differences have been documented within.

Overview

Before the device can be accessed, it must first be opened. FT_W32_CreateFile 79 returns a handle that is used by all functions in the programming interface to identify the device. When the device has been opened successfully, I/O can be performed using FT_W32_ReadFile 82 and FT_W32_WriteFile 85. When operations are complete, the device is closed using FT_W32_CloseHandle 81.

Reference

Type definitions 105 of the functional parameters and return codes used in the FT-Win32 interface are contained in the appendix 104.

5.1 FT_W32_CreateFile

Open the specified device and return a handle which will be used for subsequent accesses. The device can be specified by its serial number, device description, or location.

This function must be used if overlapped I/O is required.

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

Parameters

lpszName	Pointer to a null terminated string that contains the name of the device. The name of the device can be its serial number or description as obtained from the FT_ListDevices function.
dwAccess	Type of access to the device. Access can be GENERIC_READ, GENERIC_WRITE or both. **Ignored in Linux.**
dwShareMode	How the device is shared. This value must be set to 0.
lpSecurityAttributes	This parameter has no effect and should be set to NULL.
dwCreate	This parameter must be set to OPEN_EXISTING. **Ignored in Linux.**
dwAttrsAndFlags	File attributes and flags. This parameter is a combination of FILE_ATTRIBUTE_NORMAL, FILE_FLAG_OVERLAPPED if overlapped I/O is used, FT_OPEN_BY_SERIAL_NUMBER if *lpszName* is the devices serial number, and FT_OPEN_BY_DESCRIPTION if *lpszName* is the devices description.
hTemplate	This parameter must be NULL

Return Value

If the function is successful, the return value is a handle.
If the function is unsuccessful, the return value is the Win32 error code INVALID_HANDLE_VALUE.

Remarks

The meaning of *pvArg1* depends on *dwAttrsAndFlags*: if *FT_OPEN_BY_SERIAL_NUMBER* or *FT_OPEN_BY_DESCRIPTION* is set in *dwAttrsAndFlags*, *pvArg1* contains a pointer to a null terminated string that contains the device's serial number or description; if *FT_OPEN_BY_LOCATION* is set in *dwAttrsAndFlags*, *pvArg1* is interpreted as a value of type long that contains the location ID of the device.

dwAccess can be *GENERIC_READ*, *GENERIC_WRITE* or both; *dwShareMode* must be set to 0; *lpSecurityAttributes* must be set to NULL; *dwCreate* must be set to *OPEN_EXISTING*; *dwAttrsAndFlags* is a combination of *FILE_ATTRIBUTE_NORMAL*, *FILE_FLAG_OVERLAPPED* if overlapped I/O is used, *FT_OPEN_BY_SERIAL_NUMBER* or *FT_OPEN_BY_DESCRIPTION* or *FT_OPEN_BY_LOCATION*; *hTemplate* must be NULL.

Windows CE does not support overlapped IO or location IDs.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;
FT_HANDLE ftHandle;
char Buf[64];
```

Open a device for overlapped I/O using its serial number

```
ftStatus = FT_ListDevices(0,Buf,FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);

ftHandle = FT_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,0,0,
                                    OPEN_EXISTING,
                                    FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED |
                                    FT_OPEN_BY_SERIAL_NUMBER,
                                    0);

if (ftHandle == INVALID_HANDLE_VALUE)
        ; // FT_W32_CreateDevice failed
```

Open a device for non-overlapped I/O using its description

```
ftStatus = FT_ListDevices(0,Buf,FT_LIST_BY_INDEX|FT_OPEN_BY_DESCRIPTION);

ftHandle = FT_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,0,0,
                                    OPEN_EXISTING,
                                    FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_DESCRIPTION,
                                    0);

if (ftHandle == INVALID_HANDLE_VALUE)
        ; // FT_W32_CreateDevice failed
```

Open a device for non-overlapped I/O using its location

```
long locID;

ftStatus = FT_ListDevices(0,&locID,FT_LIST_BY_INDEX|FT_OPEN_BY_LOCATION);

ftHandle = FT_W32_CreateFile((PVOID) locID,GENERIC_READ|GENERIC_WRITE,0,0,
                                    OPEN_EXISTING,
                                    FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_LOCATION,
                                    0);

if (ftHandle == INVALID_HANDLE_VALUE)
        ; // FT_W32_CreateDevice failed
```

5.2 FT_W32_CloseHandle

Close the specified device.

BOOL **FT_W32_CloseHandle** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how to close a device after opening it for non-overlapped I/O using its description.

```
FT_STATUS ftStatus;
FT_HANDLE ftHandle;
char Buf[64];

ftStatus = FT_ListDevices(0,Buf,FT_LIST_BY_INDEX|FT_OPEN_BY_DESCRIPTION);
ftHandle = FT_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,0,0,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_DESCRIPTION,
0);
if (ftHandle == INVALID_HANDLE_VALUE){
  // FT_W32_CreateDevice failed}
else {
  // FT_W32_CreateFile OK, so do some work, and eventually ...
  FT_W32_CloseHandle(ftHandle);
}
```

5.3 FT_W32_ReadFile

Read data from the device.

BOOL **FT_W32_ReadFile** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*, LPDWORD *lpdwBytesReturned*, LPOVERLAPPED *lpOverlapped*)

Parameters

ftHandle	Handle of the device.
lpBuffer	Pointer to a buffer that receives the data from the device.
dwBytesToRead	Number of bytes to read from the device.
lpdwBytesReturned	Pointer to a variable that receives the number of bytes read from the device.
lpOverlapped	Pointer to an overlapped structure. **Ignored in Linux.**

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function supports both non-overlapped and overlapped I/O, **except under Windows CE and Linux where only non-overlapped IO is supported**.

Non-overlapped I/O

The parameter, *lpOverlapped,* must be NULL for non-overlapped I/O.

This function always returns the number of bytes read in *lpdwBytesReturned*.

This function does not return until *dwBytesToRead* have been read into the buffer. The number of bytes in the receive queue can be determined by calling FT_GetStatus 34 or FT_GetQueueStatus 31, and passed as *dwBytesToRead* so that the function reads the device and returns immediately.

When a read timeout has been setup in a previous call to FT_W32_SetCommTimeouts 100, this function returns when the timer expires or *dwBytesToRead* have been read, whichever occurs first. If a timeout occurred, any available data is read into *lpBuffer* and the function returns a non-zero value.

An application should use the function return value and *lpdwBytesReturned* when processing the buffer. If the return value is non-zero and *lpdwBytesReturned* is equal to *dwBytesToRead* then the function has completed normally. If the return value is non-zero and *lpdwBytesReturned* is less then *dwBytesToRead* then a timeout has occurred, and the read request has been partially completed. Note that if a timeout occurred and no data was read, the return value is still non-zero.

A return value of *FT_IO_ERROR* suggests an error in the parameters of the function, or a fatal error like USB disconnect has occurred.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped,* must point to an initialized OVERLAPPED structure.

If there is enough data in the receive queue to satisfy the request, the request completes immediately and the return code is non-zero. The number of bytes read is returned in *lpdwBytesReturned*.

If there is not enough data in the receive queue to satisfy the request, the request completes immediately, and the return code is zero, signifying an error. An application should call FT_W32_GetLastError 87 to get the cause of the error. If the error code is ERROR_IO_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling FT_W32_GetOverlappedResult 88.
If successful, the number of bytes read is returned in *lpdwBytesReturned*.

Example

This example shows how to read 256 bytes from the device using non-overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for non-overlapped i/o
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;

if (FT_W32_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osWrite)) {
  if (dwToRead == dwRead){
    // FT_W32_ReadFile OK}
  else{
    // FT_W32_ReadFile timeout}
{
else{
  // FT_W32_ReadFile failed}
```

This example shows how to read 256 bytes from the device using overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;
OVERLAPPED osRead = { 0 };

if (!FT_W32_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osWrite)) {
  if (FT_W32_GetLastError(ftHandle) == ERROR_IO_PENDING) {
    // write is delayed so do some other stuff until ...
    if (!FT_W32_GetOverlappedResult(ftHandle, &osRead, &dwRead, FALSE)){
      // error}
    else {
      if (dwToRead == dwRead){
        // FT_W32_ReadFile OK}
      else{
        // FT_W32_ReadFile timeout}
    }
  }
}
else {
  // FT_W32_ReadFile OK
```

```
}
```

5.4 FT_W32_WriteFile

Write data to the device.

BOOL **FT_W32_WriteFile** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten*, LPOVERLAPPED *lpOverlapped*)

Parameters

ftHandle	Handle of the device.
lpBuffer	Pointer to the buffer that contains the data to write to the device.
dwBytesToWrite	Number of bytes to be written to the device.
lpdwBytesWritten	Pointer to a variable that receives the number of bytes written to the device.
lpOverlapped	Pointer to an overlapped structure. **Ignored in Linux.**

Return Value
If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks
This function supports both non-overlapped and overlapped I/O, **except under Windows CE and Linux where only non-overlapped IO is supported**.

Non-overlapped I/O
The parameter, *lpOverlapped,* must be NULL for non-overlapped I/O.

This function always returns the number of bytes written in *lpdwBytesWritten*.

This function does not return until *dwBytesToWrite* have been written to the device.

When a write timeout has been setup in a previous call to FT_W32_SetCommTimeouts, this function returns when the timer expires or *dwBytesToWrite* have been written, whichever occurs first. If a timeout occurred, *lpdwBytesWritten* contains the number of bytes actually written, and the function returns a non-zero value.

An application should always use the function return value and *lpdwBytesWritten*. If the return value is non-zero and *lpdwBytesWritten* is equal to *dwBytesToWrite* then the function has completed normally. If the return value is non-zero and *lpdwBytesWritten* is less then *dwBytesToWrite* then a timeout has occurred, and the write request has been partially completed. Note that if a timeout occurred and no data was written, the return value is still non-zero.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the

request has been completed.

The parameter, *lpOverlapped,* must point to an initialized OVERLAPPED structure.

This function completes immediately, and the return code is zero, signifying an error. An application should call FT_W32_GetLastError 87 to get the cause of the error. If the error code is ERROR_IO_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling FT_W32_GetOverlappedResult 88.

If successful, the number of bytes written is returned in *lpdwBytesWritten*.

Example

This example shows how to write 128 bytes to the device using non-overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;

if (FT_W32_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) {
  if (dwToWrite == dwWritten){
    // FT_W32_WriteFile OK}
  else{
    // FT_W32_WriteFile timeout}
  }
else{
  // FT_W32_WriteFile failed}
```

This example shows how to write 128 bytes to the device using overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;
OVERLAPPED osWrite = { 0 };

if (!FT_W32_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) {
  if (FT_W32_GetLastError(ftHandle) == ERROR_IO_PENDING) {
    // write is delayed so do some other stuff until ...
    if (!FT_W32_GetOverlappedResult(ftHandle, &osWrite, &dwWritten, FALSE)){
      // error}
    else {
      if (dwToWrite == dwWritten){
        // FT_W32_WriteFile OK}
      else{
        // FT_W32_WriteFile timeout}
    }
  }
}
else {
  // FT_W32_WriteFIle OK
}
```

5.5 FT_W32_GetLastError

Gets the last error that occurred on the device.

DWORD **FT_W32_GetLastError** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function is normally used with overlapped I/O and so is **not supported in Windows CE**. For a description of its use, see FT_W32_ReadFile 82 and FT_W32_WriteFile 85.
In Linux this function returns a DWORD that directly maps to the FT Errors (for example the FT_INVALID_HANDLE error number).

5.6 FT_W32_GetOverlappedResult

Gets the result of an overlapped operation.

BOOL **FT_W32_GetOverlappedResult** (FT_HANDLE *ftHandle*, LPOVERLAPPED *lpOverlapped*, LPDWORD *lpdwBytesTransferred*, BOOL *bWait*)

Parameters

ftHandle	Handle of the device.
lpOverlapped	Pointer to an overlapped structure.
ldwBytesTransferred	Pointer to a variable that receives the number of bytes transferred during the overlapped operation.
bWait	Set to TRUE if the function does not return until the operation has been completed.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function is used with overlapped I/O and so is **not supported in Windows CE or Linux**. For a description of its use, see FT_W32_ReadFile 82 and FT_W32_WriteFile 85.

5.7 FT_W32_ClearCommBreak

Puts the communications line in the non-BREAK state.

BOOL **FT_W32_ClearCommBreak** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how put the line in the non-BREAK state.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

if (!FT_W32_ClearCommBreak(ftHandle)){
  // FT_W32_ClearCommBreak failed}
else{
  // FT_W32_ClearCommBreak OK}
```

5.8 FT_W32_ClearCommError

Gets information about a communications error and get current status of the device.

BOOL **FT_W32_ClearCommError** (FT_HANDLE *ftHandle*, LPDWORD *lpdwErrors*, LPFTCOMSTAT *lpftComstat*)

Parameters

ftHandle	Handle of the device.
lpdwErrors	Variable that contains the error mask.
lpftComstat	Pointer to FTCOMSTAT structure.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how to use this function.

```
static COMSTAT oldCS = {0};
static DWORD dwOldErrors = 0;

FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
COMSTAT newCS;
DWORD dwErrors;
BOOL bChanged = FALSE;

if (!FT_W32_ClearCommError(ftHandle, &dwErrors, (FTCOMSTAT *)&newCS))
  ; // FT_W32_ClearCommError failed

if (dwErrors != dwOldErrors) {
  bChanged = TRUE;
  dwErrorsOld = dwErrors;
}

if (memcmp(&oldCS, &newCS, sizeof(FTCOMSTAT))) {
  bChanged = TRUE;
  oldCS = newCS;
}

if (bChanged) {
  if (dwErrors & CE_BREAK)
    ; // BREAK condition detected
  if (dwErrors & CE_FRAME)
    ; // Framing error detected
  if (dwErrors & CE_RXOVER)
    ; // Receive buffer has overflowed
  if (dwErrors & CE_TXFULL)
    ; // Transmit buffer full
  if (dwErrors & CE_OVERRUN)
    ; // Character buffer overrun
  if (dwErrors & CE_RXPARITY)
    ; // Parity error detected
  if (newCS.fCtsHold)
    ; // Transmitter waiting for CTS
  if (newCS.fDsrHold)
    ; // Transmitter is waiting for DSR
  if (newCS.fRlsdHold)
    ; // Transmitter is waiting for RLSD
  if (newCS.fXoffHold)
```

```
    ; // Transmitter is waiting because XOFF was received
  if (newCS.fXoffSent)
    ; //
  if (newCS.fEof)
    ; // End of file character has been received
  if (newCS.fTxim)
    ; // Tx immediate character queued for transmission
  // newCS.cbInQue contains number of bytes in receive queue
  // newCS.cbOutQue contains number of bytes in transmit queue
}
```

5.9 FT_W32_EscapeCommFunction

Perform an extended function.

BOOL **FT_W32_EscapeCommFunction** (FT_HANDLE *ftHandle*, DWORD *dwFunc*)

Parameters

ftHandle	Handle of the device.
dwFunc	The extended function to perform can be one of the following values:
CLRDTR	Clear the DTR signal
CLRRTS	Clear the RTS signal
SETDTR	Set the DTR signal
SETRTS	Set the RTS signal
SETBREAK	Set the BREAK condition
CLRBREAK	Clear the BREAK condition

Return Value
If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example
This example shows how to use this function.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

FT_W32_EscapeCommFunction(ftHandle,CLRDTS);
FT_W32_EscapeCommFunction(ftHandle,SETRTS);
```

5.10 FT_W32_GetCommModemStatus

This function gets the current modem control value.

BOOL **FT_W32_GetCommModemStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwStat*)

Parameters

ftHandle	Handle of the device.
lpdwStat	Pointer to a variable to contain modem control value. The modem control value can be a combination of the following:
MS_CTS_ON	Clear to Send (CTS) is on
MS_DSR_ON	Data Set Ready (DSR) is on
MS_RING_ON	Ring Indicator (RI) is on
MS_RLSD_ON	Receive Line Signal Detect (RLSD) is on

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how to use this function.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
DWORD dwStatus;

if (FT_W32_GetCommModemStatus(ftHandle,&dwStatus)) {
  // FT_W32_GetCommModemStatus ok
  if (dwStatus & MS_CTS_ON)
    ; // CTS is on
  if (dwStatus & MS_DSR_ON)
    ; // DSR is on
  if (dwStatus & MS_RI_ON)
    ; // RI is on
  if (dwStatus & MS_RLSD_ON)
    ; // RLSD is on
}
else
  ; // FT_W32_GetCommModemStatus failed
```

5.11 FT_W32_GetCommState

This function gets the current device state.

BOOL **FT_W32_GetCommState** (FT_HANDLE *ftHandle*, LPFTDCB *lpftDcb*)

Parameters

ftHandle	Handle of the device.
lpftDcb	Pointer to an FTDCB structure.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

The current state of the device is returned in a device control block.

Example

This example shows how to use this function.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTDCB ftDCB;

if (FT_W32_GetCommState(ftHandle,&ftDCB))
  ; // FT_W32_GetCommState ok, device state is in ftDCB
else
  ; // FT_W32_GetCommState failed
```

5.12 FT_W32_GetCommTimeouts

This function gets the current read and write request timeout parameters for the specified device.

BOOL **FT_W32_GetCommTimeouts** (FT_HANDLE *ftHandle*, LPFTTIMEOUTS *lpftTimeouts*)

Parameters

ftHandle	Handle of the device.
lpftTimeouts	Pointer to an FTTIMEOUTS structure to store timeout information.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

For an explanation of how timeouts are used, see **FT_W32_SetCommTimeouts**.

Example

This example shows how to retrieve the current timeout values.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTTIMEOUTS ftTS;

if (FT_W32_GetCommTimeouts(ftHandle,&ftTS))
  ; // FT_W32_GetCommTimeouts OK
else
  ; // FT_W32_GetCommTimeouts failed
```

5.13 FT_W32_PurgeComm

This function purges the device.

BOOL **FT_W32_PurgeComm** (FT_HANDLE *ftHandle*, DWORD *dwFlags*)

Parameters

ftHandle	Handle of the device.
dwFlags	Specifies the action to take. The action can be a combination of the following:
PURGE_TXABORT	Terminate outstanding overlapped writes
PURGE_RXABORT	Terminate outstanding overlapped reads
PURGE_TXCLEAR	Clear the transmit buffer
PURGE_RXCLEAR	Clear the receive buffer

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how to purge the receive and transmit queues.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

if (FT_W32_PurgeComm(ftHandle,PURGE_TXCLEAR|PURGE_RXCLEAR))
  ; // FT_W32_PurgeComm OK
else
  ; // FT_W32_PurgeComm failed
```

5.14 FT_W32_SetCommBreak

Puts the communications line in the BREAK state.

BOOL **FT_W32_SetCommBreak** (FT_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how put the line in the BREAK state.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

if (!FT_W32_SetCommBreak(ftHandle))
  ; // FT_W32_SetCommBreak failed
else
  ; // FT_W32_SetCommBreak OK
```

5.15 FT_W32_SetCommMask

This function specifies events that the device has to monitor.

BOOL **FT_W32_SetCommMask** (FT_HANDLE *ftHandle*, DWORD *dwMask*)

Parameters

ftHandle	Handle of the device.
dwMask	Mask containing aevents that the device has to monitor. This can be a combination of the following:
EV_BREAK	BREAK condition detected
EV_CTS	Change in Clear to Send (CTS)
EV_DSR	Change in Data Set Ready (DSR)
EV_ERR	Error in line status
EV_RING	Ring Indicator (RI) detected
EV_RLSD	Change in Receive Line Signal Detect (RLSD)
EV_RXCHAR	Character received
EV_RXFLAG	Event character received
EV_TXEMPTY	Transmitter empty

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function specifies the events that the device should monitor. An application can call the function **FT_W32_WaitCommEvent** to wait for an event to occur.

Example

This example shows how to monitor changes in the modem status lines DSR and CTS.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
DWORD dwMask = EV_CTS | EV_DSR;

if (!FT_W32_SetCommMAsk(ftHandle,dwMask))
  ; // FT_W32_SetCommMask failed
else
  ; // FT_W32_SetCommMask OK
```

5.16 FT_W32_SetCommState

This function sets the state of the device according to the contents of a device control block (DCB).

BOOL **FT_W32_SetCommState** (FT_HANDLE *ftHandle*, LPFTDCB *lpftDcb*)

Parameters

ftHandle	Handle of the device.
lpftDcb	Pointer to an FTDCB structure.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

This example shows how to use this function to change the baud rate.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTDCB ftDCB;

if (FT_W32_GetCommState(ftHandle,&ftDCB)) {
  // FT_W32_GetCommState ok, device state is in ftDCB
  ftDCB.BaudRate = 921600;
  if (FT_W32_SetCommState(ftHandle,&ftDCB))
    ; // FT_W32_SetCommState ok
  else
    ; // FT_W32_SetCommState failed
}
else
  ; // FT_W32_GetCommState failed
```

5.17 FT_W32_SetCommTimeouts

This function sets the timeout parameters for I/O requests.

BOOL **FT_W32_SetCommTimeouts** (FT_HANDLE *ftHandle*, LPFTTIMEOUTS *lpftTimeouts*)

Parameters

ftHandle	Handle of the device.
lpftTimeouts	Pointer to an FTTIMEOUTS structure to store timeout information.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

Timeouts are calculated using the information in the FTTIMEOUTS structure.

For read requests, the number of bytes to be read is multiplied by the total timeout multiplier, and added to the total timeout constant. So, if TS is an FTTIMEOUTS structure and the number of bytes to read is dwToRead, the read timeout, rdTO, is calculated as follows.

rdTO = (dwToRead * TS.ReadTotalTimeoutMultiplier) + TS.ReadTotalTimeoutConstant

For write requests, the number of bytes to be written is multiplied by the total timeout multiplier, and added to the total timeout constant. So, if TS is an FTTIMEOUTS structure and the number of bytes to write is dwToWrite, the write timeout, wrTO, is calculated as follows.

wrTO = (dwToWrite * TS.WriteTotalTimeoutMultiplier) + TS.WriteTotalTimeoutConstant

Linux ignores the ReadIntervalTimeout, ReadTotalTimeoutMultiplier and WriteTotalTimeoutMultiplier currently.

Example

This example shows how to setup a read timeout of 100 milliseconds and a write timeout of 200 milliseconds.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTTIMEOUTS ftTS;

ftTS.ReadIntervalTimeout = 0;
ftTS.ReadTotalTimeoutMultiplier = 0;
ftTS.ReadTotalTimeoutConstant = 100;
ftTS.WriteTotalTimeoutMultiplier = 0;
ftTS.WriteTotalTimeoutConstant = 200;

if (FT_W32_SetCommTimeouts(ftHandle,&ftTS))
  ; // FT_W32_SetCommTimeouts OK
else
  ; // FT_W32_SetCommTimeouts failed
```

5.18 FT_W32_SetupComm

This function sets the read and write buffers.

BOOL **FT_W32_SetupComm** (FT_HANDLE *ftHandle*, DWORD *dwReadBufferSize*, DWORD *dwWriteBufferSize*)

Parameters

ftHandle	Handle of the device.
dwReadBufferSize	Length, in bytes, of the read buffer.
dwWriteBufferSize	Length, in bytes, of the write buffer.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function has no effect. It is the responsibility of the driver to allocate sufficient storage for I/O requests.

5.19 FT_W32_WaitCommEvent

This function waits for an event to occur.

BOOL **FT_W32_WaitCommEvent** (FT_HANDLE *ftHandle*, LPDWORD *lpdwEvent*, LPOVERLAPPED *lpOverlapped*)

Parameters

ftHandle	Handle of the device.
lpdwEvent	Pointer to a location that receives a mask that contains the events that occurred.
lpOverlapped	Pointer to an overlapped structure.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function supports both non-overlapped and overlapped I/O, **except under Windows CE and Linux where only non-overlapped IO is supported**.

Non-overlapped I/O

The parameter, *lpOverlapped,* must be NULL for non-overlapped I/O.

This function does not return until an event that has been specified in a call to FT_W32_SetCommMask 98 has occurred. The events that occurred and resulted in this function returning are stored in *lpdwEvent*.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped,* must point to an initialized OVERLAPPED structure.

This function does not return until an event that has been specified in a call to FT_W32_SetCommMask 98 has occurred.

If an event has already occurred, the request completes immediately, and the return code is non-zero. The events that occurred are stored in *lpdwEvent*.

If an event has not yet occurred, the request completes immediately, and the return code is zero, signifying an error. An application should call FT_W32_GetLastError 87 to get the cause of the error. If the error code is ERROR_IO_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the

overlapped request by calling FT_W32_GetOverlappedResult 88. The events that occurred and resulted in this function returning are stored in *lpdwEvent*.

Example

This example shows how to write 128 bytes to the device using non-overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for non-overlapped i/o
DWORD dwEvents;

if (FT_W32_WaitCommEvent(ftHandle, &dwEvents, NULL))
  ; // FT_W32_WaitCommEvents OK
else
  ; // FT_W32_WaitCommEvents failed
```

This example shows how to write 128 bytes to the device using overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
DWORD dwEvents;
DWORD dwRes;
OVERLAPPED osWait = { 0 };

if (!FT_W32_WaitCommEvent(ftHandle, &dwEvents, &osWait)) {
  if (FT_W32_GetLastError(ftHandle == ERROR_IO_PENDING) {
    // wait is delayed so do some other stuff until ...
    if (!FT_W32_GetOverlappedResult(ftHandle, &osWait, &dwRes, FALSE))
      ; // error
    else
      ; // FT_W32_WaitCommEvent OK
        // Events that occurred are stored in dwEvents
  }
}
else {
  // FT_W32_WaitCommEvent OK
  // Events that occurred are stored in dwEvents
}
```

6 Appendix

This section contains type definitions 105 of the functional parameters and return codes used in the D2XX programming interface. It also contains a copy of the current FTD2XX.H file 111.

6.1 Type Definitions

Excerpts from the header file FTD2XX.H 111 are included in this appendix to explain any references in the descriptions of the functions in this document.
For Visual C++ applications, these values are pre-declared in the header file (FTD2XX.H 111), which is included in the driver release. For other languages, these definitions will have to be converted to use equivalent types, and may have to be defined in an include file or within the body of the code. For non-Visual C++ applications, check the application code examples on the FTDI website as a translation of these may already exist.

UCHAR	Unsigned char (1 byte)
PUCHAR	Pointer to unsigned char (4 bytes)
PCHAR	Pointer to char (4 bytes)
DWORD	Unsigned long (4 bytes)
LPDWORD	Pointer to unsigned long (4 bytes)
FT_HANDLE	DWORD

FT_STATUS (DWORD)

```
FT_OK = 0
FT_INVALID_HANDLE = 1
FT_DEVICE_NOT_FOUND = 2
FT_DEVICE_NOT_OPENED = 3
FT_IO_ERROR = 4
FT_INSUFFICIENT_RESOURCES = 5
FT_INVALID_PARAMETER = 6
FT_INVALID_BAUD_RATE = 7
FT_DEVICE_NOT_OPENED_FOR_ERASE = 8
FT_DEVICE_NOT_OPENED_FOR_WRITE = 9
FT_FAILED_TO_WRITE_DEVICE = 10
FT_EEPROM_READ_FAILED = 11
FT_EEPROM_WRITE_FAILED = 12
FT_EEPROM_ERASE_FAILED = 13
FT_EEPROM_NOT_PRESENT = 14
FT_EEPROM_NOT_PROGRAMMED = 15
FT_INVALID_ARGS = 16
FT_NOT_SUPPORTED = 17
FT_OTHER_ERROR = 18
```

Flags (see FT_OpenEx 12)

```
FT_OPEN_BY_SERIAL_NUMBER = 1
FT_OPEN_BY_DESCRIPTION = 2
FT_OPEN_BY_LOCATION = 4
```

Flags (see FT_ListDevices 8)

```
FT_LIST_NUMBER_ONLY = 0x80000000
FT_LIST_BY_INDEX = 0x40000000
FT_LIST_ALL = 0x20000000
```

FT_DEVICE (DWORD)

```
FT_DEVICE_232BM = 0
FT_DEVICE_232AM = 1
FT_DEVICE_100AX = 2
```

FT_DEVICE_UNKNOWN = 3
FT_DEVICE_2232C = 4
FT_DEVICE_232R = 5

Word Length (see FT_SetDataCharacteristics 21)
FT_BITS_8 = 8
FT_BITS_7 = 7

Stop Bits (see FT_SetDataCharacteristics 21)
FT_STOP_BITS_1 = 0
FT_STOP_BITS_2 = 2

Parity (see FT_SetDataCharacteristics 21)
FT_PARITY_NONE = 0
FT_PARITY_ODD = 1
FT_PARITY_EVEN = 2
FT_PARITY_MARK = 3
FT_PARITY_SPACE = 4

Flow Control (see FT_SetFlowControl 22)
FT_FLOW_NONE = 0x0000
FT_FLOW_RTS_CTS = 0x0100
FT_FLOW_DTR_DSR = 0x0200
FT_FLOW_XON_XOFF = 0x0400

Purge RX and TX Buffers (see FT_Purge 29)
FT_PURGE_RX = 1
FT_PURGE_TX = 2

Notification Events (see FT_SetEventNotification 35)
FT_EVENT_RXCHAR = 1
FT_EVENT_MODEM_STATUS = 2

Modem Status (see FT_GetModemStatus 27)
CTS = 0x10
DSR = 0x20
RI = 0x40
DCD = 0x80

FT232R CBUS EEPROM OPTIONS - Ignored for FT245R (see FT_EE_Program 65 and FT_EE_Read 62)
CBUS_TXDEN = 0x00
CBUS_PWRON = 0x01
CBUS_RXLED = 0x02
CBUS_TXLED = 0x03
CBUS_TXRXLED = 0x04
CBUS_SLEEP = 0x05
CBUS_CLK48 = 0x06
CBUS_CLK24 = 0x07

```
        CBUS_CLK12 = 0x08
        CBUS_CLK6 = 0x09
        CBUS_IOMODE = 0x0A
        CBUS_BITBANG_WR = 0x0B
        CBUS_BITBANG_RD = 0x0C
```

FT_DEVICE_LIST_INFO_NODE (see FT_GetDeviceInfoList 49)

```
typedef struct _ft_device_list_info_node {
        DWORD Flags;
        DWORD Type;
        DWORD ID;
        DWORD LocId;
        char SerialNumber[16];
        char Description[64];
        FT_HANDLE ftHandle;
} FT_DEVICE_LIST_INFO_NODE;
```

FT_PROGRAM_DATA (EEPROM Programming Interface)

```
typedef struct ft_program_data {
        WORD VendorId;                  // 0x0403
        WORD ProductId;                 // 0x6001
        char *Manufacturer;             // "FTDI"
        char *ManufacturerId;           // "FT"
        char *Description;              // "USB HS Serial Converter"
        char *SerialNumber;             // "FT000001" if fixed, or NULL
        WORD MaxPower;                  // 0 < MaxPower <= 500
        WORD PnP;                       // 0 = disabled, 1 = enabled
        WORD SelfPowered;               // 0 = bus powered, 1 = self powered
        WORD RemoteWakeup;              // 0 = not capable, 1 = capable
        //
        // Rev4 extensions
        //
        UCHAR Rev4;                     // true if Rev4 chip, false otherwise
        UCHAR IsoIn;                    // true if in endpoint is isochronous
        UCHAR IsoOut;                   // true if out endpoint is isochronous
        UCHAR PullDownEnable;           // true if pull down enabled
        UCHAR SerNumEnable;             // true if serial number to be used
        UCHAR USBVersionEnable;         // true if chip uses USBVersion
        WORD USBVersion;                // BCD (0x0200 => USB2)
} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;
```

FT_PROGRAM_DATA (EEPROM Programming Interface - compatible with DLL version 2.1.4.1 or later)

```
typedef struct ft_program_data {
        DWORD Signature1;               // Header - must be 0x00000000
        DWORD Signature2;               // Header - must be 0xffffffff
        DWORD Version;                  // Header - FT_PROGRAM_DATA version
                                        //          0 = original
                                        //          1 = FT2232C extensions
        WORD VendorId;                  // 0x0403
        WORD ProductId;                 // 0x6001
        char *Manufacturer;             // "FTDI"
        char *ManufacturerId;           // "FT"
        char *Description;              // "USB HS Serial Converter"
```

```
        char *SerialNumber;             // "FT000001" if fixed, or NULL
        WORD MaxPower;                  // 0 < MaxPower <= 500
        WORD PnP;                       // 0 = disabled, 1 = enabled
        WORD SelfPowered;               // 0 = bus powered, 1 = self powered
        WORD RemoteWakeup;                      // 0 = not capable, 1 = capable
        //
        // Rev4 extensions
        //
        UCHAR Rev4;                     // non-zero if Rev4 chip, zero otherwise
        UCHAR IsoIn;                    // non-zero if in endpoint is isochronous
        UCHAR IsoOut;                   // non-zero if out endpoint is isochronous
        UCHAR PullDownEnable;           // non-zero if pull down enabled
        UCHAR SerNumEnable;                     // non-zero if serial number to be used
        UCHAR USBVersionEnable;         // non-zero if chip uses USBVersion
        WORD USBVersion;                // BCD (0x0200 => USB2)
        //
        // FT2232C extensions
        //
        UCHAR Rev5;                     // non-zero if Rev5 chip, zero otherwise
        UCHAR IsoInA;                   // non-zero if in endpoint is isochronous
        UCHAR IsoInB;                   // non-zero if in endpoint is isochronous
        UCHAR IsoOutA;                  // non-zero if out endpoint is isochronous
        UCHAR IsoOutB;                  // non-zero if out endpoint is isochronous
        UCHAR PullDownEnable5;          // non-zero if pull down enabled
        UCHAR SerNumEnable5;            // non-zero if serial number to be used
        UCHAR USBVersionEnable5;        // non-zero if chip uses USBVersion
        WORD USBVersion5;               // BCD (0x0200 => USB2)
        UCHAR AIsHighCurrent;                   // non-zero if interface is high current
        UCHAR BIsHighCurrent;                   // non-zero if interface is high current
        UCHAR IFAIsFifo;                // non-zero if interface is 245 FIFO
        UCHAR IFAIsFifoTar;             // non-zero if interface is 245 FIFO CPU target
        UCHAR IFAIsFastSer;             // non-zero if interface is Fast serial
        UCHAR AIsVCP;                   // non-zero if interface is to use VCP drivers
        UCHAR IFBIsFifo;                // non-zero if interface is 245 FIFO
        UCHAR IFBIsFifoTar;             // non-zero if interface is 245 FIFO CPU target
        UCHAR IFBIsFastSer;             // non-zero if interface is Fast serial
        UCHAR BIsVCP;                   // non-zero if interface is to use VCP drivers
} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;
```

FT_PROGRAM_DATA (EEPROM Programming Interface - compatible with DLL version 3.1.6.1 or later)

```
typedef struct ft_program_data {
        DWORD Signature1;               // Header - must be 0x00000000
        DWORD Signature2;               // Header - must be 0xffffffff
        DWORD Version;                  // Header - FT_PROGRAM_DATA version
                                        //      0 = original
                                        //      1 = FT2232C extensions
                                        //      2 = FT232R extensions
        WORD VendorId;                  // 0x0403
        WORD ProductId;                 // 0x6001
        char *Manufacturer;             // "FTDI"
        char *ManufacturerId;           // "FT"
        char *Description;              // "USB HS Serial Converter"
        char *SerialNumber;             // "FT000001" if fixed, or NULL
        WORD MaxPower;                  // 0 < MaxPower <= 500
        WORD PnP;                       // 0 = disabled, 1 = enabled
        WORD SelfPowered;               // 0 = bus powered, 1 = self powered
```

```
    WORD RemoteWakeup;                  // 0 = not capable, 1 = capable
    //
    // Rev4 extensions
    //
    UCHAR Rev4;                         // non-zero if Rev4 chip, zero otherwise
    UCHAR IsoIn;                        // non-zero if in endpoint is isochronous
    UCHAR IsoOut;                       // non-zero if out endpoint is isochronous
    UCHAR PullDownEnable;               // non-zero if pull down enabled
    UCHAR SerNumEnable;                 // non-zero if serial number to be used
    UCHAR USBVersionEnable;             // non-zero if chip uses USBVersion
    WORD USBVersion;                    // BCD (0x0200 => USB2)
    //
    // FT2232C extensions
    //
    UCHAR Rev5;                         // non-zero if Rev5 chip, zero otherwise
    UCHAR IsoInA;                       // non-zero if in endpoint is isochronous
    UCHAR IsoInB;                       // non-zero if in endpoint is isochronous
    UCHAR IsoOutA;                      // non-zero if out endpoint is isochronous
    UCHAR IsoOutB;                      // non-zero if out endpoint is isochronous
    UCHAR PullDownEnable5;              // non-zero if pull down enabled
    UCHAR SerNumEnable5;                // non-zero if serial number to be used
    UCHAR USBVersionEnable5;            // non-zero if chip uses USBVersion
    WORD USBVersion5;                   // BCD (0x0200 => USB2)
    UCHAR AIsHighCurrent;               // non-zero if interface is high current
    UCHAR BIsHighCurrent;               // non-zero if interface is high current
    UCHAR IFAIsFifo;                    // non-zero if interface is 245 FIFO
    UCHAR IFAIsFifoTar;                 // non-zero if interface is 245 FIFO CPU target
    UCHAR IFAIsFastSer;                 // non-zero if interface is Fast serial
    UCHAR AIsVCP;                       // non-zero if interface is to use VCP drivers
    UCHAR IFBIsFifo;                    // non-zero if interface is 245 FIFO
    UCHAR IFBIsFifoTar;                 // non-zero if interface is 245 FIFO CPU target
    UCHAR IFBIsFastSer;                 // non-zero if interface is Fast serial
    UCHAR BIsVCP;                       // non-zero if interface is to use VCP drivers
    //
    // FT232R extensions
    //
    UCHAR UseExtOsc;                    // Use External Oscillator
    UCHAR HighDriveIOs;                 // High Drive I/Os
    UCHAR EndpointSize;                 // Endpoint size
    UCHAR PullDownEnableR;              // non-zero if pull down enabled
    UCHAR SerNumEnableR;                // non-zero if serial number to be used
    UCHAR InvertTXD;                    // non-zero if invert TXD
    UCHAR InvertRXD;                    // non-zero if invert RXD
    UCHAR InvertRTS;                    // non-zero if invert RTS
    UCHAR InvertCTS;                    // non-zero if invert CTS
    UCHAR InvertDTR;                    // non-zero if invert DTR
    UCHAR InvertDSR;                    // non-zero if invert DSR
    UCHAR InvertDCD;                    // non-zero if invert DCD
    UCHAR InvertRI;                     // non-zero if invert RI
    UCHAR Cbus0;                        // Cbus Mux control - Ignored for FT245R
    UCHAR Cbus1;                        // Cbus Mux control - Ignored for FT245R
    UCHAR Cbus2;                        // Cbus Mux control - Ignored for FT245R
    UCHAR Cbus3;                        // Cbus Mux control - Ignored for FT245R
    UCHAR Cbus4;                        // Cbus Mux control - Ignored for FT245R
    UCHAR RIsVCP;                       // non-zero if using VCP drivers
} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;
```

FTCOMSTAT (FT-Win32 Programming Interface)

```
typedef struct _FTCOMSTAT {
        DWORD fCtsHold : 1;
        DWORD fDsrHold : 1;
        DWORD fRlsdHold : 1;
        DWORD fXoffHold : 1;
        DWORD fXoffSent : 1;
        DWORD fEof : 1;
        DWORD fTxim : 1;
        DWORD fReserved : 25;
        DWORD cbInQue;
        DWORD cbOutQue;
} FTCOMSTAT, *LPFTCOMSTAT;
```

FTDCB (FT-Win32 Programming Interface)

```
typedef struct _FTDCB {
        DWORD DCBlength;                    // sizeof(FTDCB)
        DWORD BaudRate;                     // Baudrate at which running
        DWORD fBinary: 1;                   // Binary Mode (skip EOF check)
        DWORD fParity: 1;                   // Enable parity checking
        DWORD fOutxCtsFlow:1;               // CTS handshaking on output
        DWORD fOutxDsrFlow:1;               // DSR handshaking on output
        DWORD fDtrControl:2;                // DTR Flow control
        DWORD fDsrSensitivity:1;            // DSR Sensitivity
        DWORD fTXContinueOnXoff: 1;                 // Continue TX when Xoff sent
        DWORD fOutX: 1;                     // Enable output X-ON/X-OFF
        DWORD fInX: 1;                      // Enable input X-ON/X-OFF
        DWORD fErrorChar: 1;                // Enable Err Replacement
        DWORD fNull: 1;                     // Enable Null stripping
        DWORD fRtsControl:2;                // Rts Flow control
        DWORD fAbortOnError:1;              // Abort all reads and writes on Error
        DWORD fDummy2:17;                   // Reserved
        WORD wReserved;                     // Not currently used
        WORD XonLim;                        // Transmit X-ON threshold
        WORD XoffLim;                       // Transmit X-OFF threshold
        BYTE ByteSize;                      // Number of bits/byte, 7-8
        BYTE Parity;                // 0-4=None,Odd,Even,Mark,Space
        BYTE StopBits;                      // 0,2 = 1, 2
        char XonChar;                       // Tx and Rx X-ON character
        char XoffChar;                      // Tx and Rx X-OFF character
        char ErrorChar;             // Error replacement char
        char EofChar;                       // End of Input character
        char EvtChar;                       // Received Event character
        WORD wReserved1;                    // Fill
} FTDCB, *LPFTDCB;
```

FTTIMEOUTS (FT-Win32 Programming Interface)

```
typedef struct _FTTIMEOUTS {
        DWORD ReadIntervalTimeout;          // Maximum time between read chars
        DWORD ReadTotalTimeoutMultiplier;   // Multiplier of characters
        DWORD ReadTotalTimeoutConstant;     // Constant in milliseconds
        DWORD WriteTotalTimeoutMultiplier;  // Multiplier of characters
        DWORD WriteTotalTimeoutConstant;    // Constant in milliseconds
} FTTIMEOUTS, *LPFTTIMEOUTS;
```

6.2 FTD2XX.H

```
/*++

Copyright (c) 2001-2005  Future Technology Devices International Ltd.

Module Name:

   ftd2xx.h

Abstract:

   Native USB device driver for FTDI FT8U232/245
   FTD2XX library definitions

Environment:

   kernel & user mode

Revision History:

   13/03/01    awm    Created.
        13/01/03        awm             Added device information support.
        19/03/03        awm             Added FT_W32_Cancello.
        12/06/03        awm             Added FT_StopInTask and FT_RestartInTask.
        18/09/03        awm             Added FT_SetResetPipeRetryCount.
        10/10/03        awm             Added FT_ResetPort.
        23/01/04        awm             Added support for open-by-location.
        16/03/04        awm             Added support for FT2232C.
        23/09/04        awm             Added support for FT232R.
        20/10/04        awm             Added FT_CyclePort.
        18/01/05        awm             Added FT_DEVICE_LIST_INFO_NODE type.
        11/02/05        awm             Added LocId to FT_DEVICE_LIST_INFO_NODE.
        25/08/05        awm             Added FT_SetDeadmanTimeout.
        02/12/05        awm             Removed obsolete references.
        05/12/05        awm             Added FT_GetVersion, FT_GetVersionEx.


--*/


#ifndef FTD2XX_H
#define FTD2XX_H

// The following ifdef block is the standard way of creating macros
// which make exporting from a DLL simpler.  All files within this DLL
// are compiled with the FTD2XX_EXPORTS symbol defined on the command line.
// This symbol should not be defined on any project that uses this DLL.
// This way any other project whose source files include this file see
// FTD2XX_API functions as being imported from a DLL, whereas this DLL
// sees symbols defined with this macro as being exported.

#ifdef FTD2XX_EXPORTS
#define FTD2XX_API __declspec(dllexport)
#else
#define FTD2XX_API __declspec(dllimport)
#endif
```

```
typedef PVOID  FT_HANDLE;
typedef ULONG FT_STATUS;

//
// Device status
//
enum {
    FT_OK,
    FT_INVALID_HANDLE,
    FT_DEVICE_NOT_FOUND,
    FT_DEVICE_NOT_OPENED,
    FT_IO_ERROR,
    FT_INSUFFICIENT_RESOURCES,
    FT_INVALID_PARAMETER,
    FT_INVALID_BAUD_RATE,

    FT_DEVICE_NOT_OPENED_FOR_ERASE,
    FT_DEVICE_NOT_OPENED_FOR_WRITE,
    FT_FAILED_TO_WRITE_DEVICE,
    FT_EEPROM_READ_FAILED,
    FT_EEPROM_WRITE_FAILED,
    FT_EEPROM_ERASE_FAILED,
        FT_EEPROM_NOT_PRESENT,
        FT_EEPROM_NOT_PROGRAMMED,
        FT_INVALID_ARGS,
        FT_NOT_SUPPORTED,
        FT_OTHER_ERROR
};


#define FT_SUCCESS(status) ((status) == FT_OK)

//
// FT_OpenEx Flags
//

#define FT_OPEN_BY_SERIAL_NUMBER    1
#define FT_OPEN_BY_DESCRIPTION      2
#define FT_OPEN_BY_LOCATION                 4

//
// FT_ListDevices Flags (used in conjunction with FT_OpenEx Flags
//

#define FT_LIST_NUMBER_ONLY                 0x80000000
#define FT_LIST_BY_INDEX                0x40000000
#define FT_LIST_ALL                         0x20000000

#define FT_LIST_MASK (FT_LIST_NUMBER_ONLY|FT_LIST_BY_INDEX|FT_LIST_ALL)

//
// Baud Rates
//

#define FT_BAUD_300             300
#define FT_BAUD_600             600
#define FT_BAUD_1200            1200
#define FT_BAUD_2400            2400
```

```
#define FT_BAUD_4800                4800
#define FT_BAUD_9600                9600
#define FT_BAUD_14400               14400
#define FT_BAUD_19200               19200
#define FT_BAUD_38400               38400
#define FT_BAUD_57600               57600
#define FT_BAUD_115200              115200
#define FT_BAUD_230400              230400
#define FT_BAUD_460800              460800
#define FT_BAUD_921600              921600

//
// Word Lengths
//

#define FT_BITS_8                   (UCHAR) 8
#define FT_BITS_7                   (UCHAR) 7
#define FT_BITS_6                   (UCHAR) 6
#define FT_BITS_5                   (UCHAR) 5

//
// Stop Bits
//

#define FT_STOP_BITS_1              (UCHAR) 0
#define FT_STOP_BITS_1_5    (UCHAR) 1
#define FT_STOP_BITS_2              (UCHAR) 2

//
// Parity
//

#define FT_PARITY_NONE              (UCHAR) 0
#define FT_PARITY_ODD               (UCHAR) 1
#define FT_PARITY_EVEN              (UCHAR) 2
#define FT_PARITY_MARK              (UCHAR) 3
#define FT_PARITY_SPACE             (UCHAR) 4

//
// Flow Control
//

#define FT_FLOW_NONE        0x0000
#define FT_FLOW_RTS_CTS     0x0100
#define FT_FLOW_DTR_DSR     0x0200
#define FT_FLOW_XON_XOFF    0x0400

//
// Purge rx and tx buffers
//
#define FT_PURGE_RX         1
#define FT_PURGE_TX         2

//
// Events
//

typedef void (*PFT_EVENT_HANDLER)(DWORD,DWORD);
```

```
#define FT_EVENT_RXCHAR          1
#define FT_EVENT_MODEM_STATUS    2

//
// Timeouts
//

#define FT_DEFAULT_RX_TIMEOUT    300


#define FT_DEFAULT_TX_TIMEOUT    300

//
// Device types
//

typedef ULONG FT_DEVICE;

enum {
    FT_DEVICE_BM,
    FT_DEVICE_AM,
    FT_DEVICE_100AX,
    FT_DEVICE_UNKNOWN,
    FT_DEVICE_2232C,
    FT_DEVICE_232R
};


#ifdef __cplusplus
extern "C" {
#endif


FTD2XX_API
FT_STATUS WINAPI FT_Open(
        int deviceNumber,
        FT_HANDLE *pHandle
        );

FTD2XX_API
FT_STATUS WINAPI FT_OpenEx(
    PVOID pArg1,
    DWORD Flags,
    FT_HANDLE *pHandle
    );

FTD2XX_API
FT_STATUS WINAPI FT_ListDevices(
        PVOID pArg1,
        PVOID pArg2,
        DWORD Flags
        );

FTD2XX_API
FT_STATUS WINAPI FT_Close(
    FT_HANDLE ftHandle
    );
```

```
FTD2XX_API
FT_STATUS WINAPI FT_Read(
    FT_HANDLE ftHandle,
    LPVOID lpBuffer,
    DWORD nBufferSize,
    LPDWORD lpBytesReturned
    );

FTD2XX_API
FT_STATUS WINAPI FT_Write(
    FT_HANDLE ftHandle,
    LPVOID lpBuffer,
    DWORD nBufferSize,
    LPDWORD lpBytesWritten
    );

FTD2XX_API
FT_STATUS WINAPI FT_IoCtl(
    FT_HANDLE ftHandle,
    DWORD dwIoControlCode,
    LPVOID lpInBuf,
    DWORD nInBufSize,
    LPVOID lpOutBuf,
    DWORD nOutBufSize,
    LPDWORD lpBytesReturned,
    LPOVERLAPPED lpOverlapped
    );

FTD2XX_API
FT_STATUS WINAPI FT_SetBaudRate(
    FT_HANDLE ftHandle,
        ULONG BaudRate
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetDivisor(
    FT_HANDLE ftHandle,
        USHORT Divisor
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetDataCharacteristics(
    FT_HANDLE ftHandle,
        UCHAR WordLength,
        UCHAR StopBits,
        UCHAR Parity
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetFlowControl(
    FT_HANDLE ftHandle,
    USHORT FlowControl,
    UCHAR XonChar,
    UCHAR XoffChar
        );

FTD2XX_API
FT_STATUS WINAPI FT_ResetDevice(
```

```
    FT_HANDLE ftHandle
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetDtr(
    FT_HANDLE ftHandle
        );

FTD2XX_API
FT_STATUS WINAPI FT_ClrDtr(
    FT_HANDLE ftHandle
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetRts(
    FT_HANDLE ftHandle
        );

FTD2XX_API
FT_STATUS WINAPI FT_ClrRts(
    FT_HANDLE ftHandle
        );

FTD2XX_API
FT_STATUS WINAPI FT_GetModemStatus(
    FT_HANDLE ftHandle,
        ULONG *pModemStatus
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetChars(
    FT_HANDLE ftHandle,
        UCHAR EventChar,
        UCHAR EventCharEnabled,
        UCHAR ErrorChar,
        UCHAR ErrorCharEnabled
    );

FTD2XX_API
FT_STATUS WINAPI FT_Purge(
    FT_HANDLE ftHandle,
        ULONG Mask
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetTimeouts(
    FT_HANDLE ftHandle,
        ULONG ReadTimeout,
        ULONG WriteTimeout
        );

FTD2XX_API
FT_STATUS WINAPI FT_GetQueueStatus(
    FT_HANDLE ftHandle,
        DWORD *dwRxBytes
        );

FTD2XX_API
```

```
FT_STATUS WINAPI FT_SetEventNotification(
   FT_HANDLE ftHandle,
         DWORD Mask,
         PVOID Param
         );

FTD2XX_API
FT_STATUS WINAPI FT_GetStatus(
   FT_HANDLE ftHandle,
   DWORD *dwRxBytes,
   DWORD *dwTxBytes,
   DWORD *dwEventDWord
         );

FTD2XX_API
FT_STATUS WINAPI FT_SetBreakOn(
   FT_HANDLE ftHandle
   );

FTD2XX_API
FT_STATUS WINAPI FT_SetBreakOff(
   FT_HANDLE ftHandle
   );

FTD2XX_API
FT_STATUS WINAPI FT_SetWaitMask(
   FT_HANDLE ftHandle,
   DWORD Mask
   );

FTD2XX_API
FT_STATUS WINAPI FT_WaitOnMask(
   FT_HANDLE ftHandle,
   DWORD *Mask
   );

FTD2XX_API
FT_STATUS WINAPI FT_GetEventStatus(
   FT_HANDLE ftHandle,
   DWORD *dwEventDWord
   );

FTD2XX_API
FT_STATUS WINAPI FT_ReadEE(
   FT_HANDLE ftHandle,
         DWORD dwWordOffset,
   LPWORD lpwValue
         );

FTD2XX_API
FT_STATUS WINAPI FT_WriteEE(
   FT_HANDLE ftHandle,
         DWORD dwWordOffset,
   WORD wValue
         );

FTD2XX_API
FT_STATUS WINAPI FT_EraseEE(
```

```
    FT_HANDLE ftHandle
        );

//
// structure to hold program data for FT_Program function
//
typedef struct ft_program_data {

        DWORD Signature1;               // Header - must be 0x00000000
        DWORD Signature2;               // Header - must be 0xffffffff
        DWORD Version;                  // Header - FT_PROGRAM_DATA version
                                        //      0 = original
                                        //      1 = FT2232C extensions
                                        //      2 = FT232R extensions

        WORD VendorId;                  // 0x0403
        WORD ProductId;                 // 0x6001
        char *Manufacturer;             // "FTDI"
        char *ManufacturerId;           // "FT"
        char *Description;              // "USB HS Serial Converter"
        char *SerialNumber;             // "FT000001" if fixed, or NULL
        WORD MaxPower;                  // 0 < MaxPower <= 500
        WORD PnP;                       // 0 = disabled, 1 = enabled
        WORD SelfPowered;               // 0 = bus powered, 1 = self powered
        WORD RemoteWakeup;                      // 0 = not capable, 1 = capable
        //
        // Rev4 extensions
        //
        UCHAR Rev4;                     // non-zero if Rev4 chip, zero otherwise
        UCHAR IsoIn;                    // non-zero if in endpoint is isochronous
        UCHAR IsoOut;                   // non-zero if out endpoint is isochronous
        UCHAR PullDownEnable;           // non-zero if pull down enabled
        UCHAR SerNumEnable;                     // non-zero if serial number to be used
        UCHAR USBVersionEnable;         // non-zero if chip uses USBVersion
        WORD USBVersion;                // BCD (0x0200 => USB2)
        //
        // FT2232C extensions
        //
        UCHAR Rev5;                     // non-zero if Rev5 chip, zero otherwise
        UCHAR IsoInA;                   // non-zero if in endpoint is isochronous
        UCHAR IsoInB;                   // non-zero if in endpoint is isochronous
        UCHAR IsoOutA;                  // non-zero if out endpoint is isochronous
        UCHAR IsoOutB;                  // non-zero if out endpoint is isochronous
        UCHAR PullDownEnable5;          // non-zero if pull down enabled
        UCHAR SerNumEnable5;            // non-zero if serial number to be used
        UCHAR USBVersionEnable5;        // non-zero if chip uses USBVersion
        WORD USBVersion5;               // BCD (0x0200 => USB2)
        UCHAR AIsHighCurrent;                   // non-zero if interface is high current
        UCHAR BIsHighCurrent;                   // non-zero if interface is high current
        UCHAR IFAIsFifo;                // non-zero if interface is 245 FIFO
        UCHAR IFAIsFifoTar;             // non-zero if interface is 245 FIFO CPU target
        UCHAR IFAIsFastSer;             // non-zero if interface is Fast serial
        UCHAR AIsVCP;                   // non-zero if interface is to use VCP drivers
        UCHAR IFBIsFifo;                // non-zero if interface is 245 FIFO
        UCHAR IFBIsFifoTar;             // non-zero if interface is 245 FIFO CPU target
        UCHAR IFBIsFastSer;             // non-zero if interface is Fast serial
        UCHAR BIsVCP;                   // non-zero if interface is to use VCP drivers
        //
```

```
        // FT232R extensions
        //
        UCHAR UseExtOsc;                // Use External Oscillator
        UCHAR HighDriveIOs;             // High Drive I/Os
        UCHAR EndpointSize;             // Endpoint size

        UCHAR PullDownEnableR;          // non-zero if pull down enabled
        UCHAR SerNumEnableR;            // non-zero if serial number to be used

        UCHAR InvertTXD;                // non-zero if invert TXD
        UCHAR InvertRXD;                // non-zero if invert RXD
        UCHAR InvertRTS;                // non-zero if invert RTS
        UCHAR InvertCTS;                // non-zero if invert CTS
        UCHAR InvertDTR;                // non-zero if invert DTR
        UCHAR InvertDSR;                // non-zero if invert DSR
        UCHAR InvertDCD;                // non-zero if invert DCD
        UCHAR InvertRI;                         // non-zero if invert RI

        UCHAR Cbus0;                    // Cbus Mux control
        UCHAR Cbus1;                    // Cbus Mux control
        UCHAR Cbus2;                    // Cbus Mux control
        UCHAR Cbus3;                    // Cbus Mux control
        UCHAR Cbus4;                    // Cbus Mux control

        UCHAR RIsVCP;                   // non-zero if using D2XX drivers

} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;

FTD2XX_API
FT_STATUS WINAPI FT_EE_Program(
   FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData
        );

FTD2XX_API
FT_STATUS WINAPI FT_EE_ProgramEx(
   FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData,
        char *Manufacturer,
        char *ManufacturerId,
        char *Description,
        char *SerialNumber
        );

FTD2XX_API
FT_STATUS WINAPI FT_EE_Read(
   FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData
        );

FTD2XX_API
FT_STATUS WINAPI FT_EE_ReadEx(
   FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData,
        char *Manufacturer,
        char *ManufacturerId,
        char *Description,
        char *SerialNumber
```

```
        );

FTD2XX_API
FT_STATUS WINAPI FT_EE_UASize(
    FT_HANDLE ftHandle,
        LPDWORD lpdwSize
        );

FTD2XX_API
FT_STATUS WINAPI FT_EE_UAWrite(
    FT_HANDLE ftHandle,
        PUCHAR pucData,
        DWORD dwDataLen
        );

FTD2XX_API
FT_STATUS WINAPI FT_EE_UARead(
    FT_HANDLE ftHandle,
        PUCHAR pucData,
        DWORD dwDataLen,
        LPDWORD lpdwBytesRead
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetLatencyTimer(
    FT_HANDLE ftHandle,
    UCHAR ucLatency
    );

FTD2XX_API
FT_STATUS WINAPI FT_GetLatencyTimer(
    FT_HANDLE ftHandle,
    PUCHAR pucLatency
    );

FTD2XX_API
FT_STATUS WINAPI FT_SetBitMode(
    FT_HANDLE ftHandle,
    UCHAR ucMask,
        UCHAR ucEnable
    );

FTD2XX_API
FT_STATUS WINAPI FT_GetBitMode(
    FT_HANDLE ftHandle,
    PUCHAR pucMode
    );

FTD2XX_API
FT_STATUS WINAPI FT_SetUSBParameters(
    FT_HANDLE ftHandle,
    ULONG ulInTransferSize,
    ULONG ulOutTransferSize
        );

FTD2XX_API
FT_STATUS WINAPI FT_SetDeadmanTimeout(
    FT_HANDLE ftHandle,
```

```
        ULONG ulDeadmanTimeout
    );

FTD2XX_API
FT_STATUS WINAPI FT_GetDeviceInfo(
    FT_HANDLE ftHandle,
    FT_DEVICE *lpftDevice,
        LPDWORD lpdwID,
        PCHAR SerialNumber,
        PCHAR Description,
        LPVOID Dummy
    );

FTD2XX_API
FT_STATUS WINAPI FT_StopInTask(
    FT_HANDLE ftHandle
    );

FTD2XX_API
FT_STATUS WINAPI FT_RestartInTask(
    FT_HANDLE ftHandle
    );

FTD2XX_API
FT_STATUS WINAPI FT_SetResetPipeRetryCount(
    FT_HANDLE ftHandle,
        DWORD dwCount
    );

FTD2XX_API
FT_STATUS WINAPI FT_ResetPort(
    FT_HANDLE ftHandle
    );

FTD2XX_API
FT_STATUS WINAPI FT_CyclePort(
    FT_HANDLE ftHandle
    );


//
// Win32-type functions
//

FTD2XX_API
FT_HANDLE WINAPI FT_W32_CreateFile(
        LPCSTR                          lpszName,
        DWORD                           dwAccess,
        DWORD                           dwShareMode,
        LPSECURITY_ATTRIBUTES   lpSecurityAttributes,
        DWORD                           dwCreate,
        DWORD                           dwAttrsAndFlags,
        HANDLE                          hTemplate
        );

FTD2XX_API
BOOL WINAPI FT_W32_CloseHandle(
    FT_HANDLE ftHandle
```

```
        );

FTD2XX_API
BOOL WINAPI FT_W32_ReadFile(
    FT_HANDLE ftHandle,
    LPVOID lpBuffer,
    DWORD nBufferSize,
    LPDWORD lpBytesReturned,
        LPOVERLAPPED lpOverlapped
    );

FTD2XX_API
BOOL WINAPI FT_W32_WriteFile(
    FT_HANDLE ftHandle,
    LPVOID lpBuffer,
    DWORD nBufferSize,
    LPDWORD lpBytesWritten,
        LPOVERLAPPED lpOverlapped
    );

FTD2XX_API
DWORD WINAPI FT_W32_GetLastError(
    FT_HANDLE ftHandle
    );

FTD2XX_API
BOOL WINAPI FT_W32_GetOverlappedResult(
    FT_HANDLE ftHandle,
        LPOVERLAPPED lpOverlapped,
    LPDWORD lpdwBytesTransferred,
        BOOL bWait
    );

FTD2XX_API
BOOL WINAPI FT_W32_CancelIo(
    FT_HANDLE ftHandle
    );


//
// Win32 COMM API type functions
//
typedef struct _FTCOMSTAT {
    DWORD fCtsHold : 1;
    DWORD fDsrHold : 1;
    DWORD fRlsdHold : 1;
    DWORD fXoffHold : 1;
    DWORD fXoffSent : 1;
    DWORD fEof : 1;
    DWORD fTxim : 1;
    DWORD fReserved : 25;
    DWORD cbInQue;
    DWORD cbOutQue;
} FTCOMSTAT, *LPFTCOMSTAT;

typedef struct _FTDCB {
    DWORD DCBlength;      /* sizeof(FTDCB)                */
    DWORD BaudRate;       /* Baudrate at which running    */
```

```
    DWORD fBinary: 1;     /* Binary Mode (skip EOF check)    */
    DWORD fParity: 1;     /* Enable parity checking          */
    DWORD fOutxCtsFlow:1; /* CTS handshaking on output       */
    DWORD fOutxDsrFlow:1; /* DSR handshaking on output       */
    DWORD fDtrControl:2;  /* DTR Flow control                */
    DWORD fDsrSensitivity:1; /* DSR Sensitivity              */
    DWORD fTXContinueOnXoff: 1; /* Continue TX when Xoff sent */
    DWORD fOutX: 1;       /* Enable output X-ON/X-OFF        */
    DWORD fInX: 1;        /* Enable input X-ON/X-OFF         */
    DWORD fErrorChar: 1;  /* Enable Err Replacement          */
    DWORD fNull: 1;       /* Enable Null stripping           */
    DWORD fRtsControl:2;  /* Rts Flow control                */
    DWORD fAbortOnError:1; /* Abort all reads and writes on Error */
    DWORD fDummy2:17;     /* Reserved                        */
    WORD wReserved;       /* Not currently used              */
    WORD XonLim;          /* Transmit X-ON threshold         */
    WORD XoffLim;         /* Transmit X-OFF threshold        */
    BYTE ByteSize;        /* Number of bits/byte, 4-8        */
    BYTE Parity;          /* 0-4=None,Odd,Even,Mark,Space    */
    BYTE StopBits;        /* 0,1,2 = 1, 1.5, 2               */
    char XonChar;         /* Tx and Rx X-ON character        */
    char XoffChar;        /* Tx and Rx X-OFF character       */
    char ErrorChar;       /* Error replacement char          */
    char EofChar;         /* End of Input character          */
    char EvtChar;         /* Received Event character        */
    WORD wReserved1;      /* Fill for now.                   */
} FTDCB, *LPFTDCB;

typedef struct _FTTIMEOUTS {
    DWORD ReadIntervalTimeout;          /* Maximum time between read chars. */
    DWORD ReadTotalTimeoutMultiplier;   /* Multiplier of characters.        */
    DWORD ReadTotalTimeoutConstant;     /* Constant in milliseconds.        */
    DWORD WriteTotalTimeoutMultiplier;  /* Multiplier of characters.        */
    DWORD WriteTotalTimeoutConstant;    /* Constant in milliseconds.        */
} FTTIMEOUTS,*LPFTTIMEOUTS;


FTD2XX_API
BOOL WINAPI FT_W32_ClearCommBreak(
    FT_HANDLE ftHandle
        );

FTD2XX_API
BOOL WINAPI FT_W32_ClearCommError(
    FT_HANDLE ftHandle,
        LPDWORD lpdwErrors,
    LPFTCOMSTAT lpftComstat
        );

FTD2XX_API
BOOL WINAPI FT_W32_EscapeCommFunction(
    FT_HANDLE ftHandle,
        DWORD dwFunc
        );

FTD2XX_API
BOOL WINAPI FT_W32_GetCommModemStatus(
    FT_HANDLE ftHandle,
```

```
        LPDWORD lpdwModemStatus
        );

FTD2XX_API
BOOL WINAPI FT_W32_GetCommState(
    FT_HANDLE ftHandle,
    LPFTDCB lpftDcb
        );

FTD2XX_API
BOOL WINAPI FT_W32_GetCommTimeouts(
    FT_HANDLE ftHandle,
    FTTIMEOUTS *pTimeouts
        );

FTD2XX_API
BOOL WINAPI FT_W32_PurgeComm(
    FT_HANDLE ftHandle,
        DWORD dwMask
        );

FTD2XX_API
BOOL WINAPI FT_W32_SetCommBreak(
    FT_HANDLE ftHandle
        );

FTD2XX_API
BOOL WINAPI FT_W32_SetCommMask(
    FT_HANDLE ftHandle,
    ULONG ulEventMask
    );

FTD2XX_API
BOOL WINAPI FT_W32_SetCommState(
    FT_HANDLE ftHandle,
    LPFTDCB lpftDcb
        );

FTD2XX_API
BOOL WINAPI FT_W32_SetCommTimeouts(
    FT_HANDLE ftHandle,
    FTTIMEOUTS *pTimeouts
        );

FTD2XX_API
BOOL WINAPI FT_W32_SetupComm(
    FT_HANDLE ftHandle,
        DWORD dwReadBufferSize,
        DWORD dwWriteBufferSize
        );

FTD2XX_API
BOOL WINAPI FT_W32_WaitCommEvent(
    FT_HANDLE ftHandle,
    PULONG pulEvent,
        LPOVERLAPPED lpOverlapped
    );
```

```
//
// Device information
//

typedef struct _ft_device_list_info_node {
        ULONG Flags;
    ULONG Type;
        ULONG ID;
        DWORD LocId;
        char SerialNumber[16];
        char Description[64];
        FT_HANDLE ftHandle;
} FT_DEVICE_LIST_INFO_NODE;


FTD2XX_API
FT_STATUS WINAPI FT_CreateDeviceInfoList(
        LPDWORD lpdwNumDevs
        );

FTD2XX_API
FT_STATUS WINAPI FT_GetDeviceInfoList(
        FT_DEVICE_LIST_INFO_NODE *pDest,
        LPDWORD lpdwNumDevs
        );

FTD2XX_API
FT_STATUS WINAPI FT_GetDeviceInfoDetail(
        DWORD dwIndex,
        LPDWORD lpdwFlags,
        LPDWORD lpdwType,
        LPDWORD lpdwID,
        LPDWORD lpdwLocId,
        LPVOID lpSerialNumber,
        LPVOID lpDescription,
        FT_HANDLE *pftHandle
        );


//
// Version information
//

FTD2XX_API
FT_STATUS WINAPI FT_GetDriverVersion(
    FT_HANDLE ftHandle,
        LPDWORD lpdwVersion
        );

FTD2XX_API
FT_STATUS WINAPI FT_GetLibraryVersion(
        LPDWORD lpdwVersion
        );



#ifdef __cplusplus
```

```
}
#endif


#endif  /* FTD2XX_H */
```

Index

- B -

- C -

- D -

- E -

- F -

- H -

- I -

- L -

- M -

- O -

- R -

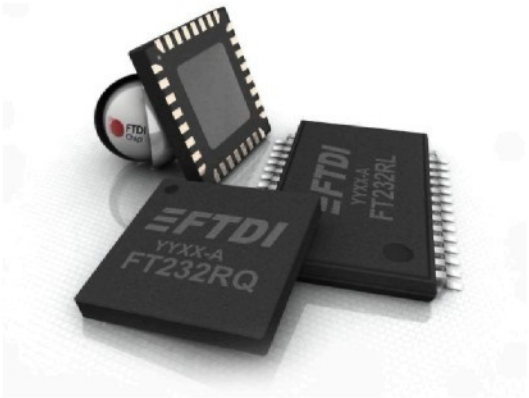
- T -

- U -

- W -

Future Technology Devices International Ltd.

FT232R USB UART IC



The FT232R is a USB to serial UART interface with the following advanced features:

- Single chip USB to asynchronous serial data transfer interface.
- Entire USB protocol handled on the chip. No USB specific firmware programming required.
- Fully integrated 1024 bit EEPROM storing device descriptors and CBUS I/O configuration.
- Fully integrated USB termination resistors.
- Fully integrated clock generation with no external crystal required plus optional clock output selection enabling a glue-less interface to external MCU or FPGA.
- Data transfer rates from 300 baud to 3 Mbaud (RS422, RS485, RS232) at TTL levels.
- 256 byte receive buffer and 128 byte transmit buffer utilising buffer smoothing technology to allow for high data throughput.
- FTDI's royalty-free Virtual Com Port (VCP) and Direct (D2XX) drivers eliminate the requirement for USB driver development in most cases.
- Unique USB FTDIChip-ID™ feature.
- Configurable CBUS I/O pins.
- Transmit and receive LED drive signals.
- UART interface support for 7 or 8 data bits, 1 or 2 stop bits and odd / even / mark / space / no parity
- FIFO receive and transmit buffers for high data throughput.
- Synchronous and asynchronous bit bang interface options with RD# and WR# strobes.
- Device supplied pre-programmed with unique USB serial number.
- Supports bus powered, self powered and high-power bus powered USB configurations.
- Integrated +3.3V level converter for USB I/O.
- Integrated level converter on UART and CBUS for interfacing to between +1.8V and +5V logic.
- True 5V/3.3V/2.8V/1.8V CMOS drive output and TTL input.
- Configurable I/O pin output drive strength.
- Integrated power-on-reset circuit.
- Fully integrated AVCC supply filtering - no external filtering required.
- UART signal inversion option.
- +3.3V (using external oscillator) to +5.25V (internal oscillator) Single Supply Operation.
- Low operating and USB suspend current.
- Low USB bandwidth consumption.
- UHCI/OHCI/EHCI host controller compatible.
- USB 2.0 Full Speed compatible.
- -40°C to 85°C extended operating temperature range.
- Available in compact Pb-free 28 Pin SSOP and QFN-32 packages (both RoHS compliant).

1 Typical Applications

- USB to RS232/RS422/RS485 Converters
- Upgrading Legacy Peripherals to USB
- Cellular and Cordless Phone USB data transfer cables and interfaces
- Interfacing MCU/PLD/FPGA based designs to USB
- USB Audio and Low Bandwidth Video data transfer
- PDA to USB data transfer
- USB Smart Card Readers
- USB Instrumentation
- USB Industrial Control
- USB MP3 Player Interface
- USB FLASH Card Reader and Writers
- Set Top Box PC - USB interface
- USB Digital Camera Interface
- USB Hardware Modems
- USB Wireless Modems
- USB Bar Code Readers
- USB Software and Hardware Encryption Dongles

1.1 Driver Support

Royalty free VIRTUAL COM PORT (VCP) DRIVERS for...

- Windows 98, 98SE, ME, 2000, Server 2003, XP and Server 2008
- Windows XP and XP 64-bit
- Windows Vista and Vista 64-bit
- Windows XP Embedded
- Windows CE 4.2, 5.0 and 6.0
- Mac OS 8/9, OS-X
- Linux 2.4 and greater

Royalty free D2XX *Direct* Drivers (USB Drivers + DLL S/W Interface)

- Windows 98, 98SE, ME, 2000, Server 2003, XP and Server 2008
- Windows XP and XP 64-bit
- Windows Vista and Vista 64-bit
- Windows XP Embedded
- Windows CE 4.2, 5.0 and 6.0
- Linux 2.4 and greater

The drivers listed above are all available to download for free from FTDI website (www.ftdichip.com). Various 3rd party drivers are also available for other operating systems - see FTDI website (www.ftdichip.com) for details.

For driver installation, please refer to the application note AN232B-10.

1.2 Part Numbers

Part Number	Package
FT232RQ-xxxx	32 Pin QFN
FT232RL-xxxx	28 Pin SSOP

Note: Packing codes for xxxx is:

- Reel: Taped and Reel, (SSOP is 2,000pcs per reel, QFN is 6,000pcs per reel).

- Tube: Tube packing, 47pcs per tube (SSOP only)

- Tray: Tray packing, 490pcs per tray (QFN only)

For example: FT232RQ-Reel is 6,000pcs taped and reel packing

1.3 USB Compliant

The FT232R is fully compliant with the USB 2.0 specification and has been given the USB-IF Test-ID (TID) 40000133.

2 FT232R Block Diagram

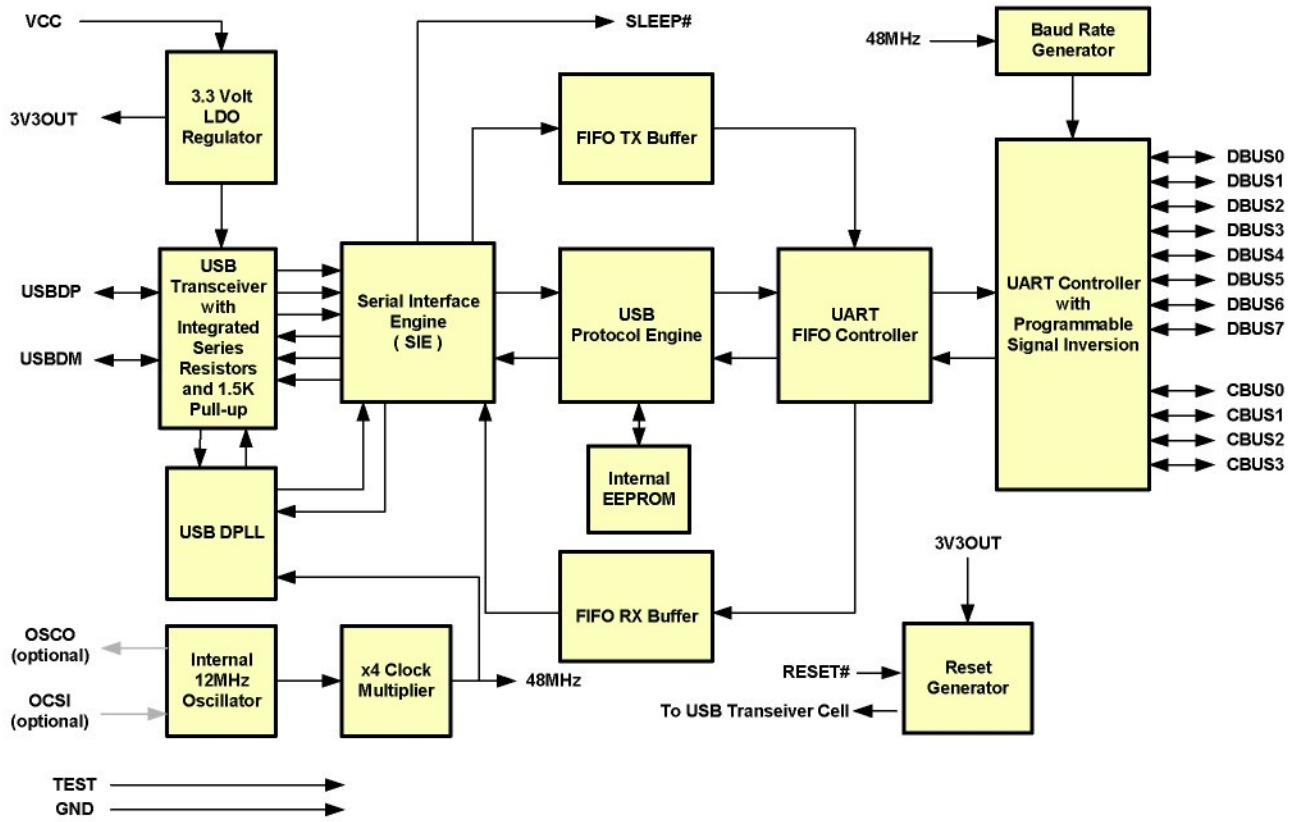


Figure 2.1 FT232R Block Diagram

For a description of each function please refer to Section 4.

Table of Contents

3 Device Pin Out and Signal Description

3.1 28-LD SSOP Package

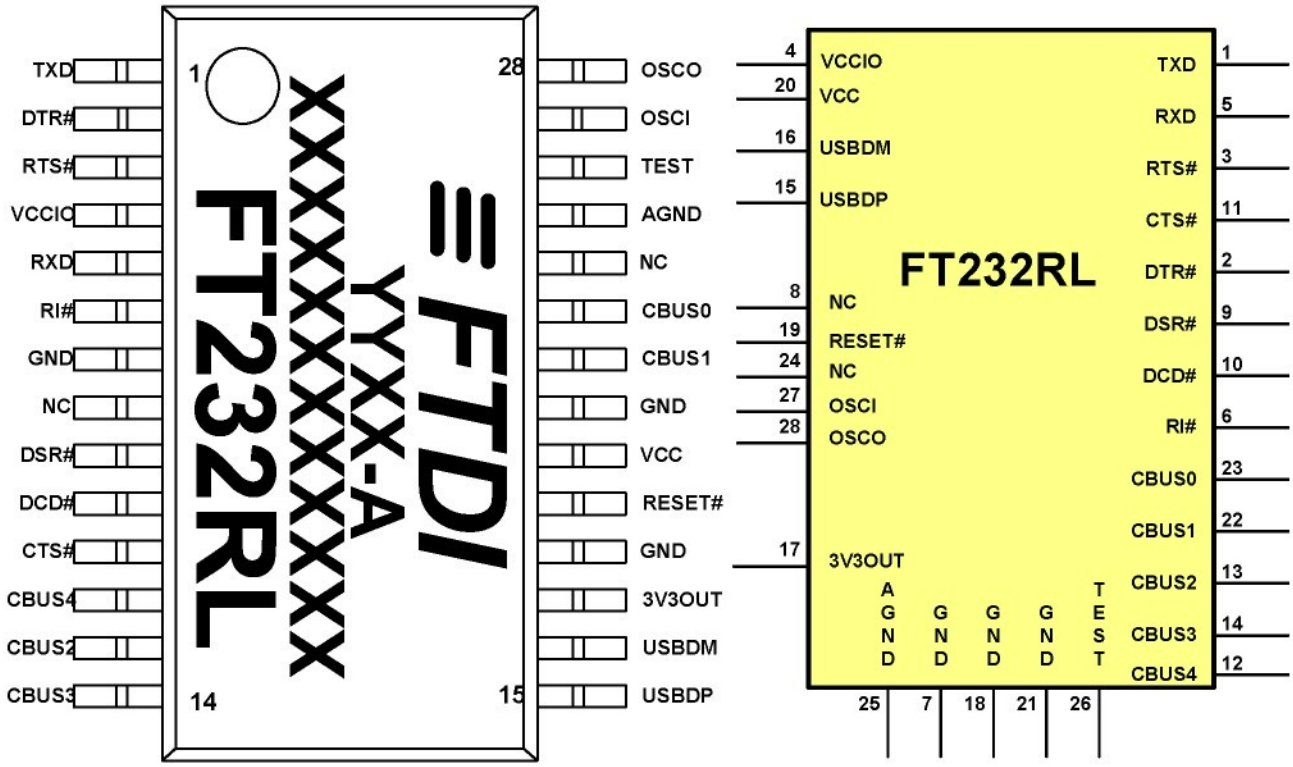


Figure 3.1 SSOP Package Pin Out and Schematic Symbol

3.2 SSOP Package Pin Out Description

Note: The convention used throughout this document for active low signals is the signal name followed by a #

Pin No.	Name	Type	Description
15	USBDP	I/O	USB Data Signal Plus, incorporating internal series resistor and 1.5kΩ pull up resistor to 3.3V.
16	USBDM	I/O	USB Data Signal Minus, incorporating internal series resistor.

Table 3.1 USB Interface Group

Pin No.	Name	Type	Description
4	VCCIO	PWR	+1.8V to +5.25V supply to the UART Interface and CBUS group pins (1...3, 5, 6, 9...14, 22, 23). In USB bus powered designs connect this pin to 3V3OUT pin to drive out at +3.3V levels, or connect to VCC to drive out at 5V CMOS level. This pin can also be supplied with an external +1.8V to +2.8V supply in order to drive outputs at lower levels. It should be noted that in this case this supply should originate from the same source as the supply to VCC. This means that in bus powered designs a regulator which is supplied by the +5V on the USB bus should be used.
7, 18, 21	GND	PWR	Device ground supply pins

Pin No.	Name	Type	Description
17	3V3OUT	Output	+3.3V output from integrated LDO regulator. This pin should be decoupled to ground using a 100nF capacitor. The main use of this pin is to provide the internal +3.3V supply to the USB transceiver cell and the internal 1.5kΩ pull up resistor on USBDP. Up to 50mA can be drawn from this pin to power external logic if required. This pin can also be used to supply the VCCIO pin.
20	VCC	PWR	+3.3V to +5.25V supply to the device core. (see Note 1)
25	AGND	PWR	Device analogue ground supply for internal clock multiplier

Table 3.2 Power and Ground Group

Pin No.	Name	Type	Description
8, 24	NC	NC	No internal connection
19	RESET#	Input	Active low reset pin. This can be used by an external device to reset the FT232R. If not required can be left unconnected, or pulled up to VCC.
26	TEST	Input	Puts the device into IC test mode. Must be tied to GND for normal operation, otherwise the device will appear to fail.
27	OSCI	Input	Input 12MHz Oscillator Cell. Optional – Can be left unconnected for normal operation. (see Note 2)
28	OSCO	Output	Output from 12MHZ Oscillator Cell. Optional – Can be left unconnected for normal operation if internal Oscillator is used. (see Note 2)

Table 3.3 Miscellaneous Signal Group

Pin No.	Name	Type	Description
1	TXD	Output	Transmit Asynchronous Data Output.
2	DTR#	Output	Data Terminal Ready Control Output / Handshake Signal.
3	RTS#	Output	Request to Send Control Output / Handshake Signal.
5	RXD	Input	Receiving Asynchronous Data Input.
6	RI#	Input	Ring Indicator Control Input. When remote wake up is enabled in the internal EEPROM taking RI# low (20ms active low pulse) can be used to resume the PC USB host controller from suspend.
9	DSR#	Input	Data Set Ready Control Input / Handshake Signal.
10	DCD#	Input	Data Carrier Detect Control Input.
11	CTS#	Input	Clear To Send Control Input / Handshake Signal.
12	CBUS4	I/O	Configurable CBUS output only Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is SLEEP#. See CBUS Signal Options, Table 3.9.
13	CBUS2	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is TXDEN. See CBUS Signal Options, Table 3.9.

Pin No.	Name	Type	Description
14	CBUS3	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is PWREN#. See CBUS Signal Options, Table 3.9. PWREN# should be used with a 10kΩ resistor pull up.
22	CBUS1	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is RXLED#. See CBUS Signal Options, Table 3.9.
23	CBUS0	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is TXLED#. See CBUS Signal Options, Table 3.9.

Table 3.4 UART Interface and CUSB Group (see note 3)

Notes:

1. The minimum operating voltage VCC must be +4.0V (could use VBUS=+5V) when using the internal clock generator. Operation at +3.3V is possible using an external crystal oscillator.
2. Contact FTDI Technical Support for details on how to use an external crystal, ceramic resonator, or oscillator with the FT232R.
3. When used in Input Mode, the input pins are pulled to VCCIO via internal 200kΩ resistors. These pins can be programmed to gently pull low during USB suspend (PWREN# = "1") by setting an option in the internal EEPROM.

3.3 QFN-32 Package

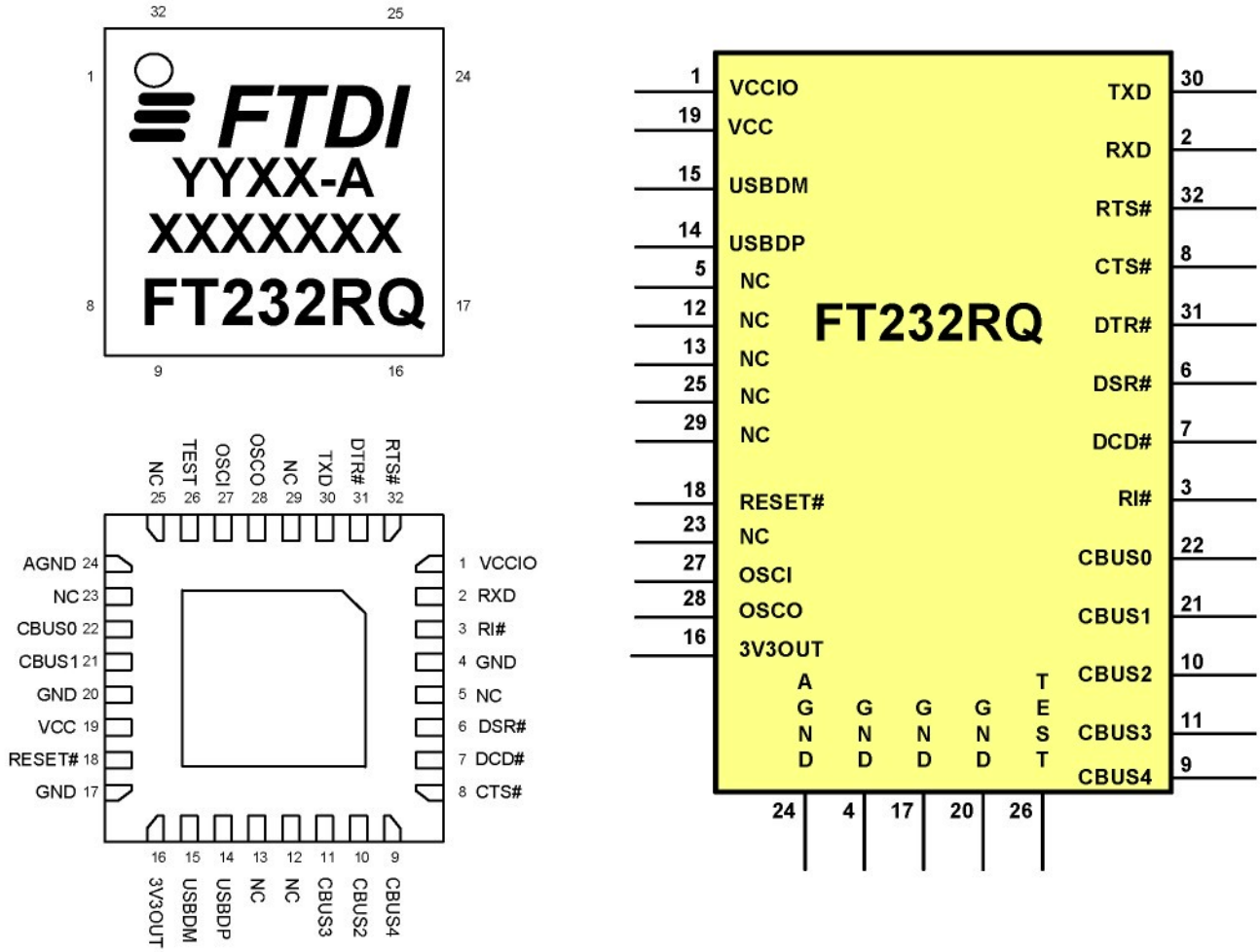


Figure 3.2 QFN-32 Package Pin Out and schematic symbol

3.4 QFN-32 Package Signal Description

Pin No.	Name	Type	Description
14	USBDP	I/O	USB Data Signal Plus, incorporating internal series resistor and 1.5kΩ pull up resistor to +3.3V.
15	USBDM	I/O	USB Data Signal Minus, incorporating internal series resistor.

Table 3.5 USB Interface Group

Pin No.	Name	Type	Description
1	VCCIO	PWR	+1.8V to +5.25V supply for the UART Interface and CBUS group pins (2, 3, 6,7,8,9,10 11, 21, 22, 30,31,32). In USB bus powered designs connect this pin to 3V3OUT to drive out at +3.3V levels, or connect to VCC to drive out at +5V CMOS level. This pin can also be supplied with an external +1.8V to +2.8V supply in order to drive out at lower levels. It should be noted that in this case this supply should originate from the same source as the supply to VCC. This means that in bus powered designs a regulator which is supplied by the +5V on the USB bus should be used.
4, 17,	GND	PWR	Device ground supply pins.

Pin No.	Name	Type	Description
20			
16	3V3OUT	Output	+3.3V output from integrated LDO regulator. This pin should be decoupled to ground using a 100nF capacitor. The purpose of this output is to provide the internal +3.3V supply to the USB transceiver cell and the internal 1.5kΩ pull up resistor on USBDP. Up to 50mA can be drawn from this pin to power external logic if required. This pin can also be used to supply the VCCIO pin.
19	VCC	PWR	+3.3V to +5.25V supply to the device core. (See Note 1).
24	AGND	PWR	Device analogue ground supply for internal clock multiplier.

Table 3.6 Power and Ground Group

Pin No.	Name	Type	Description
5, 12, 13, 23, 25, 29	NC	NC	No internal connection. Do not connect.
18	RESET#	Input	Active low reset. Can be used by an external device to reset the FT232R. If not required can be left unconnected, or pulled up to VCC.
26	TEST	Input	Puts the device into IC test mode. Must be tied to GND for normal operation, otherwise the device will appear to fail.
27	OSCI	Input	Input 12MHz Oscillator Cell. Optional – Can be left unconnected for normal operation. (See Note 2).
28	OSCO	Output	Output from 12MHZ Oscillator Cell. Optional – Can be left unconnected for normal operation if internal Oscillator is used. (See Note 2).

Table 3.7 Miscellaneous Signal Group

Pin No.	Name	Type	Description
30	TXD	Output	Transmit Asynchronous Data Output.
31	DTR#	Output	Data Terminal Ready Control Output / Handshake Signal.
32	RTS#	Output	Request to Send Control Output / Handshake Signal.
2	RXD	Input	Receiving Asynchronous Data Input.
3	RI#	Input	Ring Indicator Control Input. When remote wake up is enabled in the internal EEPROM taking RI# low (20ms active low pulse) can be used to resume the PC USB host controller from suspend.
6	DSR#	Input	Data Set Ready Control Input / Handshake Signal.
7	DCD#	Input	Data Carrier Detect Control Input.
8	CTS#	Input	Clear To Send Control Input / Handshake Signal.
9	CBUS4	I/O	Configurable CBUS output only Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is SLEEP#. See CBUS

Pin No.	Name	Type	Description
			Signal Options, Table 3.9.
10	CBUS2	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is TXDEN. See CBUS Signal Options, Table 3.9.
11	CBUS3	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is PWREN#. See CBUS Signal Options, Table 3.9. PWREN# should be used with a 10kΩ resistor pull up.
21	CBUS1	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is RXLED#. See CBUS Signal Options, Table 3.9.
22	CBUS0	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is TXLED#. See CBUS Signal Options, Table 3.9.

Table 3.8 UART Interface and CBUS Group (see note 3)

Notes:

1. The minimum operating voltage VCC must be +4.0V (could use VBUS=+5V) when using the internal clock generator. Operation at +3.3V is possible using an external crystal oscillator.
2. Contact FTDI Technical Support for details on how to use an external crystal, ceramic resonator, or oscillator with the FT232R.
3. When used in Input Mode, the input pins are pulled to VCCIO via internal 200kΩ resistors. These pins can be programmed to gently pull low during USB suspend (PWREN# = "1") by setting an option in the internal EEPROM.

3.5 CBUS Signal Options

The following options can be configured on the CBUS I/O pins. CBUS signal options are common to both package versions of the FT232R. These options can be configured in the internal EEPROM using the software utility MPROG, which can be downloaded from the FTDI Utilities (www.ftdichip.com) . The default configuration is described in Section 8.

CBUS Signal Option	Available On CBUS Pin	Description
TXDEN	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Enable transmit data for RS485
PWREN#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Output is low after the device has been configured by USB, then high during USB suspend mode. This output can be used to control power to external logic P-Channel logic level MOSFET switch. Enable the interface pull-down option when using the PWREN# in this way.*
TXLED#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Transmit data LED drive – pulses low when transmitting data via USB. See Section 7.5 for more details.
RXLED#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Receive data LED drive – pulses low when receiving data via USB. See Section 7.5 for more details.
TX&RXLED#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	LED drive – pulses low when transmitting or receiving data via USB. See Section 7.5 for more details.
SLEEP#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Goes low during USB suspend mode. Typically used to power down an external TTL to RS232 level converter IC in USB to RS232 converter designs.
CLK48	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	48MHz Clock output.**
CLK24	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	24 MHz Clock output.**
CLK12	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	12 MHz Clock output.**
CLK6	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	6 MHz Clock output.**
CBitBangI/O	CBUS0, CBUS1, CBUS2, CBUS3	CBUS bit bang mode option. Allows up to 4 of the CBUS pins to be used as general purpose I/O. Configured individually for CBUS0, CBUS1, CBUS2 and CBUS3 in the internal EEPROM. A separate application note, AN232R-01, available from FTDI website (www.ftdichip.com) describes in more detail how to use CBUS bit bang mode.
BitBangWRn	CBUS0, CBUS1, CBUS2, CBUS3	Synchronous and asynchronous bit bang mode WR# strobe output.
BitBangRDn	CBUS0, CBUS1, CBUS2, CBUS3	Synchronous and asynchronous bit bang mode RD# strobe output.

Table 3.9 CBUS Configuration Control

* PWREN# should be used with a 10kΩ resistor pull up.

**When in USB suspend mode the outputs clocks are also suspended.

4 Function Description

The FT232R is a USB to serial UART interface device which simplifies USB to serial designs and reduces external component count by fully integrating an external EEPROM, USB termination resistors and an integrated clock circuit which requires no external crystal, into the device. It has been designed to operate efficiently with a USB host controller by using as little as possible of the total USB bandwidth available.

4.1 Key Features

Functional Integration. Fully integrated EEPROM, USB termination resistors, clock generation, AVCC filtering, POR and LDO regulator.

Configurable CBUS I/O Pin Options. The fully integrated EEPROM allows configuration of the Control Bus (CBUS) functionality, signal inversion and drive strength selection. There are 5 configurable CBUS I/O pins. These configurable options are

1. **TXDEN** - transmit enable for RS485 designs.
2. **PWREN#** - Power control for high power, bus powered designs.
3. **TXLED#** - for pulsing an LED upon transmission of data.
4. **RXLED#** - for pulsing an LED upon receiving data.
5. **TX&RXLED#** - which will pulse an LED upon transmission OR reception of data.
6. **SLEEP#** - indicates that the device going into USB suspend mode.
7. **CLK48 / CLK24 / CLK12 / CLK6** - 48MHz, 24MHz, 12MHz, and 6MHz clock output signal options.

The CBUS pins can also be individually configured as GPIO pins, similar to asynchronous bit bang mode. It is possible to use this mode while the UART interface is being used, thus providing up to 4 general purpose I/O pins which are available during normal operation. An application note, AN232R-01, available from FTDI website (www.ftdichip.com) describes this feature.

The CBUS lines can be configured with any one of these output options by setting bits in the internal EEPROM. The device is supplied with the most commonly used pin definitions pre-programmed - see Section 8 for details.

Asynchronous Bit Bang Mode with RD# and WR# Strobes. The FT232R supports FTDI's previous chip generation bit-bang mode. In bit-bang mode, the eight UART lines can be switched from the regular interface mode to an 8-bit general purpose I/O port. Data packets can be sent to the device and they will be sequentially sent to the interface at a rate controlled by an internal timer (equivalent to the baud rate pre-scaler). With the FT232R device this mode has been enhanced by outputting the internal RD# and WR# strobes signals which can be used to allow external logic to be clocked by accesses to the bit-bang I/O bus. This option will be described more fully in a separate application note available from FTDI website (www.ftdichip.com).

Synchronous Bit Bang Mode. The FT232R supports synchronous bit bang mode. This mode differs from asynchronous bit bang mode in that the interface pins are only read when the device is written to. This makes it easier for the controlling program to measure the response to an output stimulus as the data returned is synchronous to the output data. An application note, AN232R-01, available from FTDI website (www.ftdichip.com) describes this feature.

FTDIChip-ID™. The FT232R also includes the new FTDIChip-ID™ security dongle feature. This FTDIChip-ID™ feature allows a unique number to be burnt into each device during manufacture. This number cannot be reprogrammed. This number is only readable over USB and forms a basis of a security dongle which can be used to protect any customer application software being copied. This allows the possibility of using the FT232R in a dongle for software licensing. Further to this, a renewable license scheme can be implemented based on the FTDIChip-ID™ number when encrypted with other information. This encrypted number can be stored in the user area of the FT232R internal EEPROM, and can be decrypted, then compared with the protected FTDIChip-ID™ to verify that a license is valid. Web based applications can be used to maintain product licensing this way. An application note, AN232R-02, available from FTDI website (www.ftdichip.com) describes this feature.

The FT232R is capable of operating at a voltage supply between +3.3V and +5V with a nominal operational mode current of 15mA and a nominal USB suspend mode current of 70µA. This allows greater margin for peripheral designs to meet the USB suspend mode current limit of 2.5mA. An integrated level converter within the UART interface allows the FT232R to interface to UART logic running at +1.8V, 2.5V, +3.3V or +5V.

4.2 Functional Block Descriptions

The following paragraphs detail each function within the FT232R. Please refer to the block diagram shown in Figure 2.1.

Internal EEPROM. The internal EEPROM in the FT232R is used to store USB Vendor ID (VID), Product ID (PID), device serial number, product description string and various other USB configuration descriptors. The internal EEPROM is also used to configure the CBUS pin functions. The FT232R is supplied with the internal EEPROM pre-programmed as described in Section 8. A user area of the internal EEPROM is available to system designers to allow storing additional data. The internal EEPROM descriptors can be programmed in circuit, over USB without any additional voltage requirement. It can be programmed using the FTDI utility software called MPROG, which can be downloaded from FTDI Utilities on the FTDI website (www.ftdichip.com).

+3.3V LDO Regulator. The +3.3V LDO regulator generates the +3.3V reference voltage for driving the USB transceiver cell output buffers. It requires an external decoupling capacitor to be attached to the 3V3OUT regulator output pin. It also provides +3.3V power to the 1.5kΩ internal pull up resistor on USBDP. The main function of the LDO is to power the USB Transceiver and the Reset Generator Cells rather than to power external logic. However, it can be used to supply external circuitry requiring a +3.3V nominal supply with a maximum current of 50mA.

USB Transceiver. The USB Transceiver Cell provides the USB 1.1 / USB 2.0 full-speed physical interface to the USB cable. The output drivers provide +3.3V level slew rate control signalling, whilst a differential input receiver and two single ended input receivers provide USB data in, Single-Ended-0 (SE0) and USB reset detection conditions respectfully. This function also incorporates the internal USB series termination resistors on the USB data lines and a 1.5kΩ pull up resistor on USBDP.

USB DPLL. The USB DPLL cell locks on to the incoming NRZI USB data and generates recovered clock and data signals for the Serial Interface Engine (SIE) block.

Internal 12MHz Oscillator - The Internal 12MHz Oscillator cell generates a 12MHz reference clock. This provides an input to the x4 Clock Multiplier function. The 12MHz Oscillator is also used as the reference clock for the SIE, USB Protocol Engine and UART FIFO controller blocks.

Clock Multiplier / Divider. The Clock Multiplier / Divider takes the 12MHz input from the Internal Oscillator function and generates the 48MHz, 24MHz, 12MHz and 6MHz reference clock signals. The 48Mz clock reference is used by the USB DPLL and the Baud Rate Generator blocks.

Serial Interface Engine (SIE). The Serial Interface Engine (SIE) block performs the parallel to serial and serial to parallel conversion of the USB data. In accordance with the USB 2.0 specification, it performs bit stuffing/un-stuffing and CRC5/CRC16 generation. It also checks the CRC on the USB data stream.

USB Protocol Engine. The USB Protocol Engine manages the data stream from the device USB control endpoint. It handles the low level USB protocol requests generated by the USB host controller and the commands for controlling the functional parameters of the UART in accordance with the USB 2.0 specification chapter 9.

FIFO TX Buffer (128 bytes). Data from the USB data OUT endpoint is stored in the FIFO TX buffer and removed from the buffer to the UART transmit register under control of the UART FIFO controller.

FIFO RX Buffer (256 bytes). Data from the UART receive register is stored in the FIFO RX buffer prior to being removed by the SIE on a USB data request from the device data IN endpoint.

UART FIFO Controller. The UART FIFO controller handles the transfer of data between the FIFO RX and TX buffers and the UART transmit and receive registers.

UART Controller with Programmable Signal Inversion and High Drive. Together with the UART FIFO Controller the UART Controller handles the transfer of data between the FIFO RX and FIFO TX buffers and the UART transmit and receive registers. It performs asynchronous 7 or 8 bit parallel to serial and serial to parallel conversion of the data on the RS232 (or RS422 or RS485) interface.

Control signals supported by UART mode include RTS, CTS, DSR, DTR, DCD and RI. The UART Controller also provides a transmitter enable control signal pin option (TXDEN) to assist with interfacing to RS485 transceivers. RTS/CTS, DSR/DTR and XON / XOFF handshaking options are also supported. Handshaking is handled in hardware to ensure fast response times. The UART interface also supports the RS232 BREAK setting and detection conditions.

Additionally, the UART signals can each be individually inverted and have a configurable high drive strength capability. Both these features are configurable in the EEPROM.

Baud Rate Generator - The Baud Rate Generator provides a 16x clock input to the UART Controller from the 48MHz reference clock. It consists of a 14 bit pre-scaler and 3 register bits which provide fine tuning of the baud rate (used to divide by a number plus a fraction or "sub-integer"). This determines the baud rate of the UART, which is programmable from 183 baud to 3 Mbaud.

The FT232R supports all standard baud rates and non-standard baud rates from 183 Baud up to 3 Mbaud. Achievable non-standard baud rates are calculated as follows -

Baud Rate = 3000000 / (n + *x*)

where 'n' can be any integer between 2 and 16,384 (= 2^{14}) and '*x*' can be a sub-integer of the value 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, or 0.875. When n = 1, *x* = 0, i.e. baud rate divisors with values between 1 and 2 are not possible.

This gives achievable baud rates in the range 183.1 baud to 3,000,000 baud. When a non-standard baud rate is required simply pass the required baud rate value to the driver as normal, and the FTDI driver will calculate the required divisor, and set the baud rate. See FTDI application note AN232B-05 on the FTDI website (www.ftdichip.com) for more details.

RESET Generator - The integrated Reset Generator Cell provides a reliable power-on reset to the device internal circuitry at power up. The RESET# input pin allows an external device to reset the FT232R.

RESET# can be tied to VCC or left unconnected if not being used.

5 Devices Characteristics and Ratings

5.1 Absolute Maximum Ratings

The absolute maximum ratings for the FT232R devices are as follows. These are in accordance with the Absolute Maximum Rating System (IEC 60134). Exceeding these may cause permanent damage to the device.

Parameter	Value	Unit
Storage Temperature	-65°C to 150°C	Degrees C
Floor Life (Out of Bag) At Factory Ambient (30°C / 60% Relative Humidity)	168 Hours (IPC/JEDEC J-STD-033A MSL Level 3 Compliant)*	Hours
Ambient Temperature (Power Applied)	-40°C to 85°C	Degrees C
MTTF FT232RL	11162037	hours
MTTF FT232RQ	4464815	hours
VCC Supply Voltage	-0.5 to +6.00	V
DC Input Voltage – USBDP and USBDM	-0.5 to +3.8	V
DC Input Voltage – High Impedance Bidirectionals	-0.5 to + (VCC +0.5)	V
DC Input Voltage – All Other Inputs	-0.5 to + (VCC +0.5)	V
DC Output Current – Outputs	24	mA
DC Output Current – Low Impedance Bidirectionals	24	mA
Power Dissipation (VCC = 5.25V)	500	mW

Table 5.1 Absolute Maximum Ratings

* If devices are stored out of the packaging beyond this time limit the devices should be baked before use. The devices should be ramped up to a temperature of +125°C and baked for up to 17 hours.

5.2 DC Characteristics

DC Characteristics (Ambient Temperature = -40°C to +85°C)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
VCC1	VCC Operating Supply Voltage	4.0	---	5.25	V	Using Internal Oscillator
VCC1	VCC Operating Supply Voltage	3.3	---	5.25	V	Using External Crystal
VCC2	VCCIO Operating Supply Voltage	1.8	---	5.25	V	
Icc1	Operating Supply Current	---	15	---	mA	Normal Operation
Icc2	Operating Supply Current	50	70	100	µA	USB Suspend
3V3	3.3v regulator output	3.0	3.3	3.6	V	

Table 5.2 Operating Voltage and Current

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	3.2	4.1	4.9	V	I source = 2mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 2mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.3 UART and CBUS I/O Pin Characteristics (VCCIO = +5.0V, Standard Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	2.2	2.7	3.2	V	I source = 1mA
Vol	Output Voltage Low	0.3	0.4	0.5	V	I sink = 2mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.4 UART and CBUS I/O Pin Characteristics (VCCIO = +3.3V, Standard Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	2.1	2.6	2.8	V	I source = 1mA
Vol	Output Voltage Low	0.3	0.4	0.5	V	I sink = 2mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.5 UART and CBUS I/O Pin Characteristics (VCCIO = +2.8V, Standard Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	1.32	1.62	1.8	V	I source = 0.2mA
Vol	Output Voltage Low	0.06	0.1	0.18	V	I sink = 0.5mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.6 UART and CBUS I/O Pin Characteristics (VCCIO = +1.8V, Standard Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	3.2	4.1	4.9	V	I source = 6mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 6mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.7 UART and CBUS I/O Pin Characteristics (VCCIO = +5.0V, High Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	2.2	2.8	3.2	V	I source = 3mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 8mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.8 UART and CBUS I/O Pin Characteristics (VCCIO = +3.3V, High Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	2.1	2.6	2.8	V	I source = 3mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 8mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.9 UART and CBUS I/O Pin Characteristics (VCCIO = +2.8V, High Drive Level)

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	1.35	1.67	1.8	V	I source = 0.4mA
Vol	Output Voltage Low	0.12	0.18	0.35	V	I sink = 3mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 5.10 UART and CBUS I/O Pin Characteristics (VCCIO = +1.8V, High Drive Level)

** Only input pins have an internal 200KΩ pull-up resistor to VCCIO

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Vin	Input Switching Threshold	1.3	1.6	1.9	V	
VHys	Input Switching Hysteresis	50	55	60	mV	

Table 5.11 RESET# and TEST Pin Characteristics

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
UVoh	I/O Pins Static Output (High)	2.8		3.6	V	RI = 1.5kΩ to 3V3OUT (D+) RI = 15KΩ to GND (D-)
UVol	I/O Pins Static Output (Low)	0		0.3	V	RI = 1.5kΩ to 3V3OUT (D+) RI = 15kΩ to GND (D-)
UVse	Single Ended Rx Threshold	0.8		2.0	V	
UCom	Differential Common Mode	0.8		2.5	V	
UVDif	Differential Input Sensitivity	0.2			V	
UDrvZ	Driver Output Impedance	26	29	44	Ohms	See Note 1

Table 5.12 USB I/O Pin (USBDP, USBDM) Characteristics

5.3 EEPROM Reliability Characteristics

The internal 1024 Bit EEPROM has the following reliability characteristics:

Parameter	Value	Unit
Data Retention	15	Years
Read / Write Cycle	100,000	Cycles

Table 5.13 EEPROM Characteristics

5.4 Internal Clock Characteristics

The internal Clock Oscillator has the following characteristics:

Parameter	Value			Unit
	Minimum	Typical	Maximum	
Frequency of Operation (see Note 1)	11.98	12.00	12.02	MHz
Clock Period	83.19	83.33	83.47	ns
Duty Cycle	45	50	55	%

Table 5.14 Internal Clock Characteristics

Note 1: Equivalent to +/-1667ppm

Parameter	Description	Minimum	Typical	Maximum	Units	Conditions
Voh	Output Voltage High	2.1	2.8	3.2	V	I source = 3mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 8mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	

Table 5.15 OSCI, OSCO Pin Characteristics – see Note 1

Note1: When supplied, the FT232R is configured to use its internal clock oscillator. These characteristics only apply when an external oscillator or crystal is used.

6 USB Power Configurations

The following sections illustrate possible USB power configurations for the FT232R. The illustrations have omitted pin numbers for ease of understanding since the pins differ between the FT232RL and FT232RQ package options.

All USB power configurations illustrated apply to both package options for the FT232R device. Please refer to Section 3 for the package option pin-out and signal descriptions.

6.1 USB Bus Powered Configuration

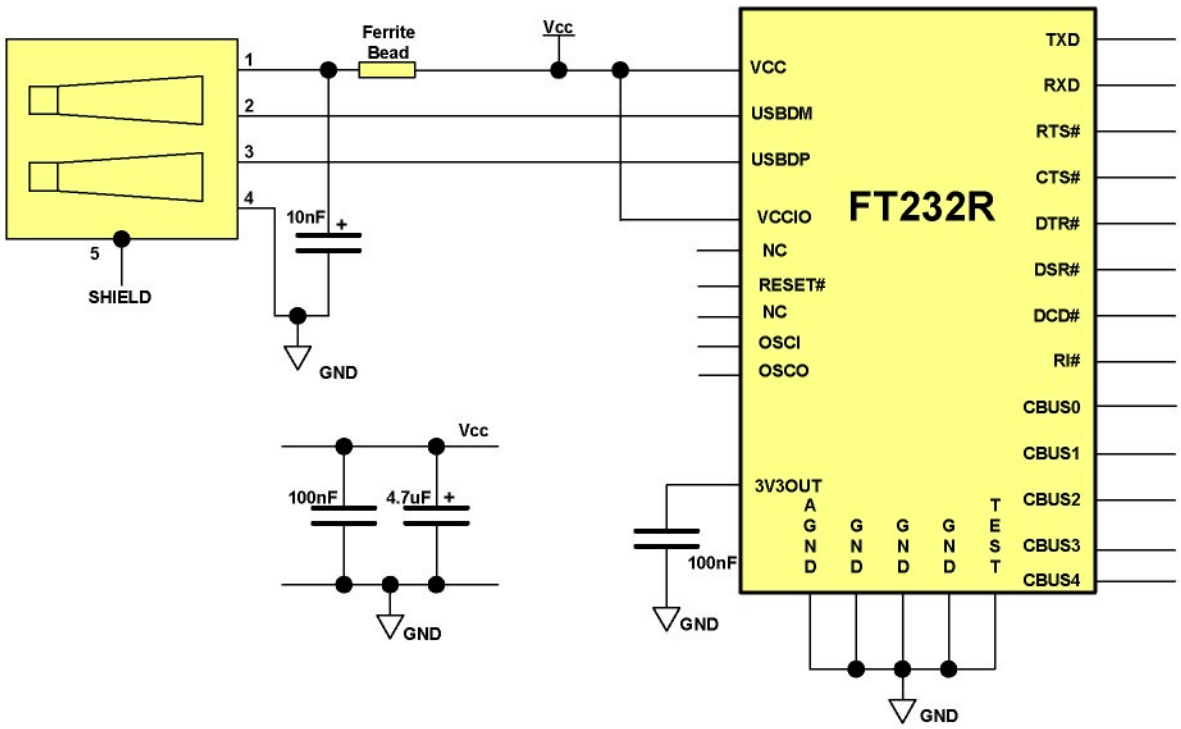


Figure 6.1 Bus Powered Configuration

Figure 6.1 Illustrates the FT232R in a typical USB bus powered design configuration. A USB bus powered device gets its power from the USB bus. Basic rules for USB bus power devices are as follows –

i) On plug-in to USB, the device should draw no more current than 100mA.
ii) In USB Suspend mode the device should draw no more than 2.5mA.
iii) A bus powered high power USB device (one that draws more than 100mA) should use one of the CBUS pins configured as PWREN# and use it to keep the current below 100mA on plug-in and 2.5mA on USB suspend.
iv) A device that consumes more than 100mA cannot be plugged into a USB bus powered hub.
v) No device can draw more than 500mA from the USB bus.

The power descriptors in the internal EEPROM of the FT232R should be programmed to match the current drawn by the device.

A ferrite bead is connected in series with the USB power supply to reduce EMI noise from the FT232R and associated circuitry being radiated down the USB cable to the USB host. The value of the Ferrite Bead depends on the total current drawn by the application. A suitable range of Ferrite Beads is available from Steward (www.steward.com), for example Steward Part # MI0805K400R-10.

Note: If using PWREN# (available using the CBUS) the pin should be pulled to VCCIO using a 10kΩ resistor.

6.2 Self Powered Configuration

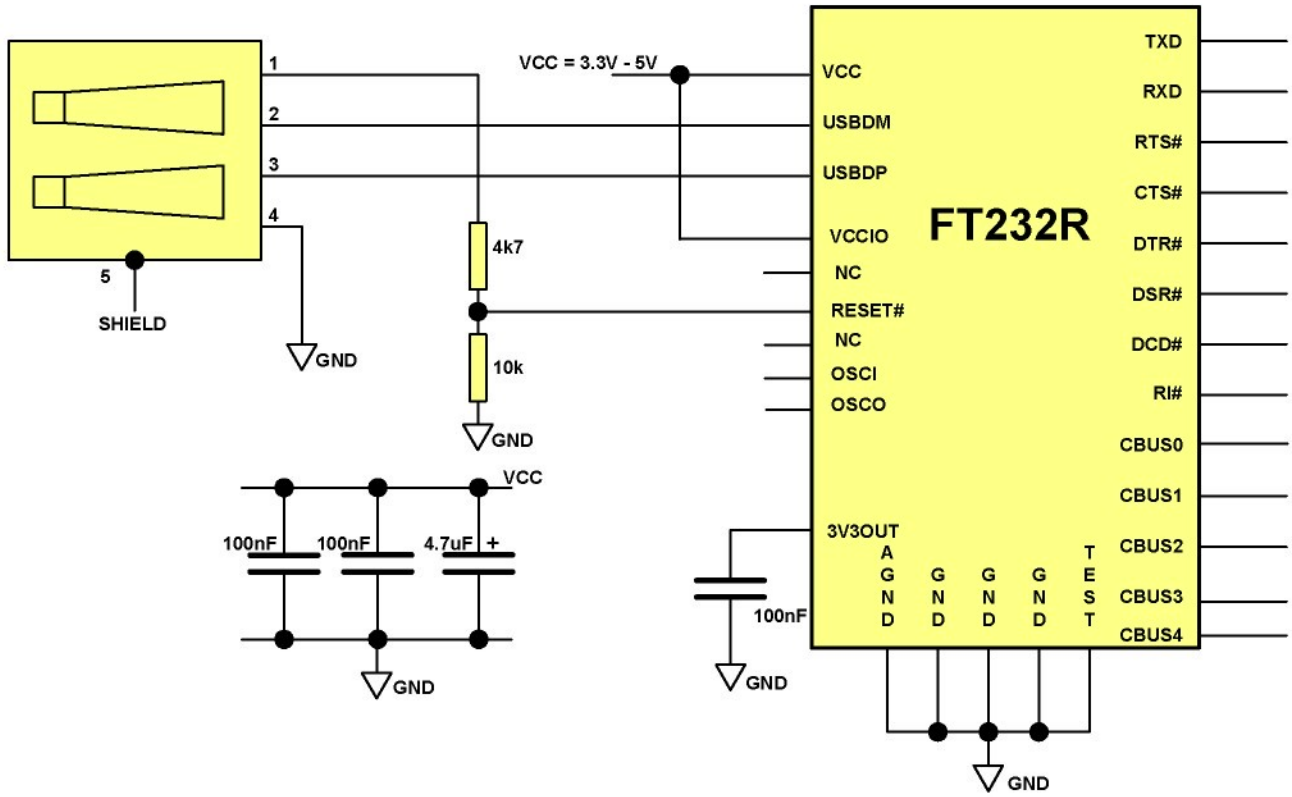


Figure 6.2 Self Powered Configuration

Figure 6.2 illustrates the FT232R in a typical USB self powered configuration. A USB self powered device gets its power from its own power supply, VCC, and does not draw current from the USB bus. The basic rules for USB self powered devices are as follows –

i) A self powered device should not force current down the USB bus when the USB host or hub controller is powered down.
ii) A self powered device can use as much current as it needs during normal operation and USB suspend as it has its own power supply.
iii) A self powered device can be used with any USB host, a bus powered USB hub or a self powered USB hub.

The power descriptor in the internal EEPROM of the FT232R should be programmed to a value of zero (self powered).

In order to comply with the first requirement above, the USB bus power (pin 1) is used to control the RESET# pin of the FT232R device. When the USB host or hub is powered up an internal 1.5kΩ resistor on USBDP is pulled up to +3.3V (generated using the 4K7 and 10k resistor network), thus identifying the device as a full speed device to the USB host or hub. When the USB host or hub is powered off, RESET# will be low and the FT232R is held in reset. Since RESET# is low, the internal 1.5kΩ resistor is not pulled up to any power supply (hub or host is powered down), so no current flows down USBDP via the 1.5kΩ pull-up resistor. Failure to do this may cause some USB host or hub controllers to power up erratically.

Figure 6.2 illustrates a self powered design which has a +3.3V to +5V supply. Any design which interfaces to +3.3 V or +1.8V logic would differ from Figure 6.2 by having a +3.3V or +1.8V supply to VCCIO. In this case the VCC operates over the range +3.3V to +5V supply to VCC

Note:

1. When the FT232R is in reset, the UART interface I/O pins are tri-stated. Input pins have internal 200kΩ pull-up resistors to VCCIO, so they will gently pull high unless driven by some external logic.
2. When using internal FT232R oscillator the VCC supply voltage range must be a minimum of +4.0V.

6.3 USB Bus Powered with Power Switching Configuration

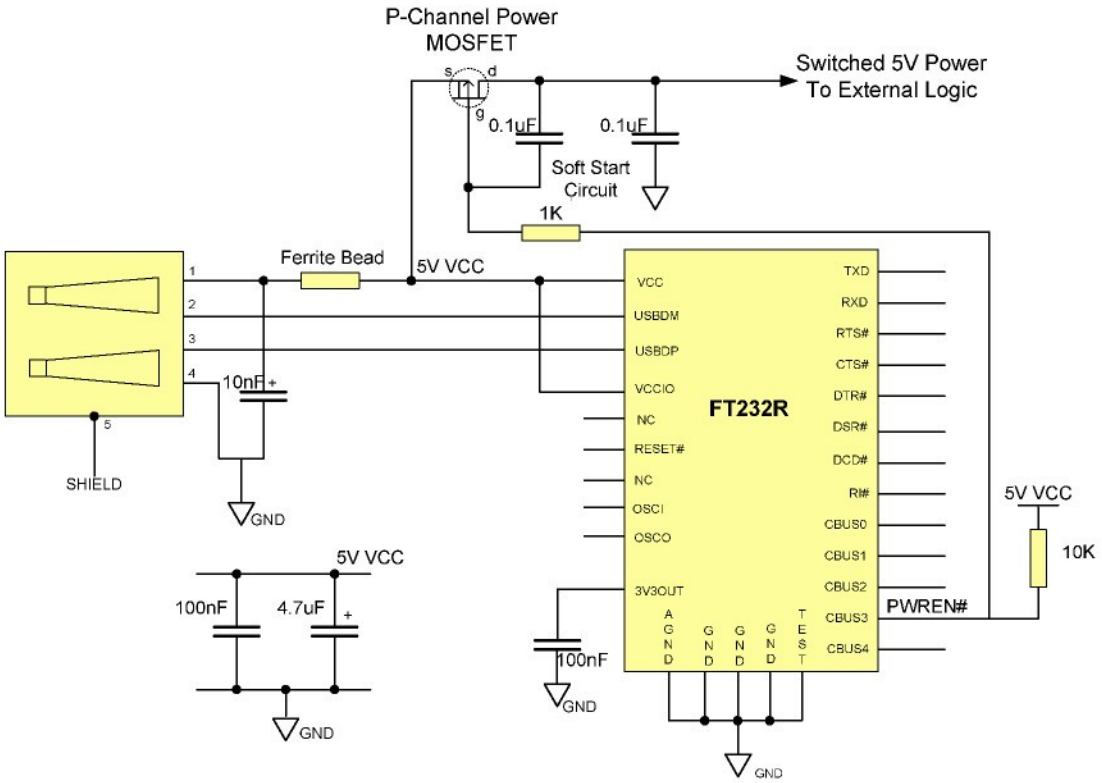


Figure 6.3 Bus Powered with Power Switching Configuration

A requirement of USB bus powered applications, is when in USB suspend mode, the application draws a total current of less than 2.5mA. This requirement includes external logic. Some external logic has the ability to power itself down into a low current state by monitoring the PWREN# signal. For external logic that cannot power itself down in this way, the FT232R provides a simple but effective method of turning off power during the USB suspend mode.

Figure 6.3 shows an example of using a discrete P-Channel MOSFET to control the power to external logic. A suitable device to do this is an International Rectifier (www.irf.com) IRLML6402, or equivalent. It is recommended that a "soft start" circuit consisting of a 1kΩ series resistor and a 0.1µF capacitor is used to limit the current surge when the MOSFET turns on. Without the soft start circuit it is possible that the transient power surge, caused when the MOSFET switches on, will reset the FT232R or the USB host/hub controller. The soft start circuit example shown in Figure 6.3 powers up with a slew rate of approximaely12.5V/ms. Thus supply voltage to external logic transitions from GND to +5V in approximately 400 microseconds.

As an alternative to the MOSFET, a dedicated power switch IC with inbuilt "soft-start" can be used. A suitable power switch IC for such an application is the Micrel (www.micrel.com) MIC2025-2BM or equivalent.

With power switching controlled designs the following should be noted:

i) The external logic to which the power is being switched should have its own reset circuitry to automatically reset the logic when power is re-applied when moving out of suspend mode.

ii) Set the Pull-down on Suspend option in the internal FT232R EEPROM.

iii) One of the CBUS Pins should be configured as PWREN# in the internal FT232R EEPROM, and used to switch the power supply to the external circuitry. This should be pulled high through a 10 kΩ resistor.

iv) For USB high-power bus powered applications (one that consumes greater than 100mA, and up to 500mA of current from the USB bus), the power consumption of the application must be set in the Max Power field in the internal FT232R EEPROM. A high-power bus powered application uses the descriptor in the internal FT232R EEPROM to inform the system of its power requirements.

v) PWREN# gets its VCC from VCCIO. For designs using 3V3 logic, ensure VCCIO is not powered down using the external logic. In this case use the +3V3OUT.

6.4 USB Bus Powered with Selectable External Logic Supply

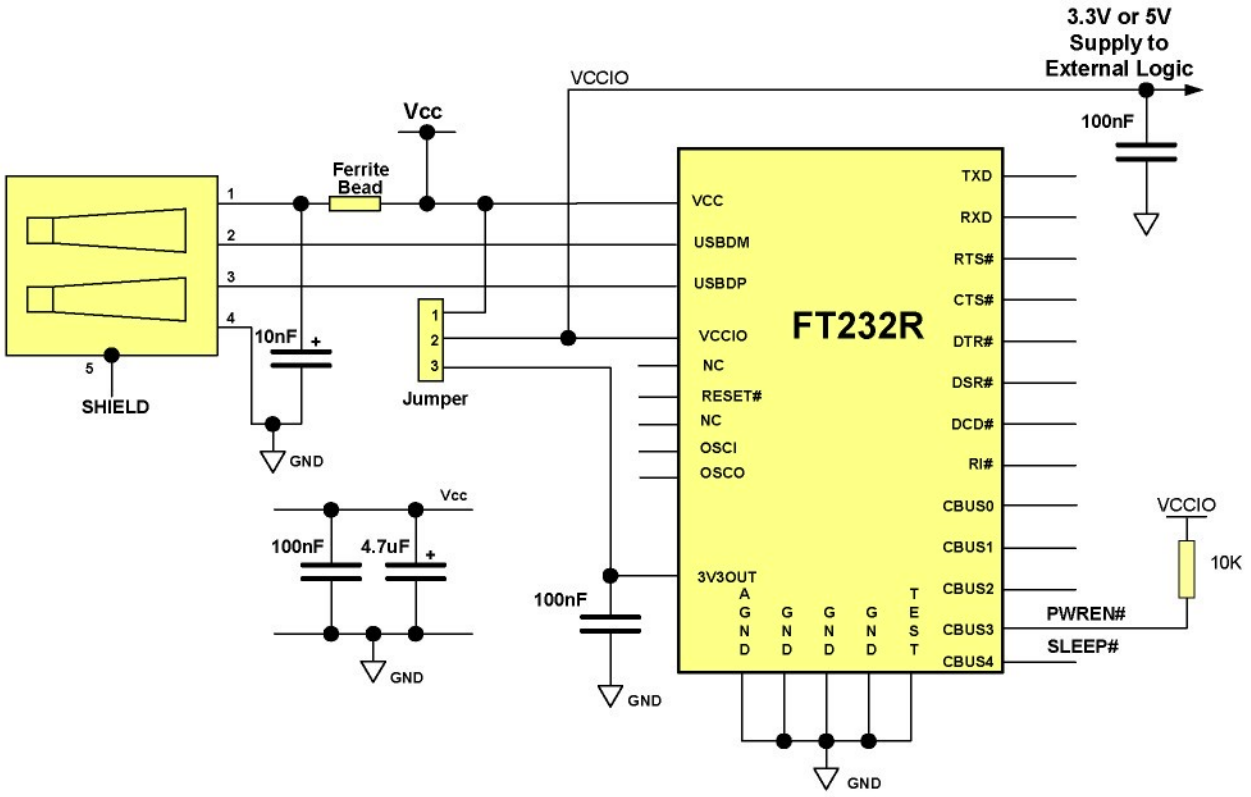


Figure 6.4 USB Bus Powered with +3.3V or +5V External Logic Power Supply

Figure 6.4 illustrates a USB bus power application with selectable external logic supply. The external logic can be selected between +3.3V and +5V using the jumper switch. This jumper is used to allow the FT232R to be interfaced with a +3.3V or +5V logic devices. The VCCIO pin is either supplied with +5V from the USB bus (jumper pins1 and 2 connected), or from the +3.3V output from the FT232R 3V3OUT pin (jumper pins 2 and 3 connected). The supply to VCCIO is also used to supply external logic.

With bus powered applications, the following should be noted:

i) To comply with the 2.5mA current supply limit during USB suspend mode, PWREN# or SLEEP# signals should be used to power down external logic in this mode. If this is not possible, use the configuration shown in Section 6.3.

ii) The maximum current sourced from the USB bus during normal operation should not exceed 100mA, otherwise a bus powered design with power switching (Section 6.3) should be used.

Another possible configuration could use a discrete low dropout (LDO) regulator which is supplied by the 5V on the USB bus to supply between +1.8V and +2.8V to the VCCIO pin and to the external logic. In this case VCC would be supplied with the +5V from the USB bus and the VCCIO would be supplied from the output of the LDO regulator. This results in the FT232R I/O pins driving out at between +1.8V and +2.8V logic levels.

For a USB bus powered application, it is important to consider the following when selecting the regulator:

i) The regulator must be capable of sustaining its output voltage with an input voltage of +4.35V. An Low Drop Out (LDO) regulator should be selected.

ii) The quiescent current of the regulator must be low enough to meet the total current requirement of <= 2.5mA during USB suspend mode.

A suitable series of LDO regulators that meets these requirements is the MicroChip/Telcom (www.microchip.com) TC55 series of devices. These devices can supply up to 250mA current and have a quiescent current of under 1µA.

7 Application Examples

The following sections illustrate possible applications of the FT232R. The illustrations have omitted pin numbers for ease of understanding since the pins differ between the FT232RL and FT232RQ package options.

7.1 USB to RS232 Converter

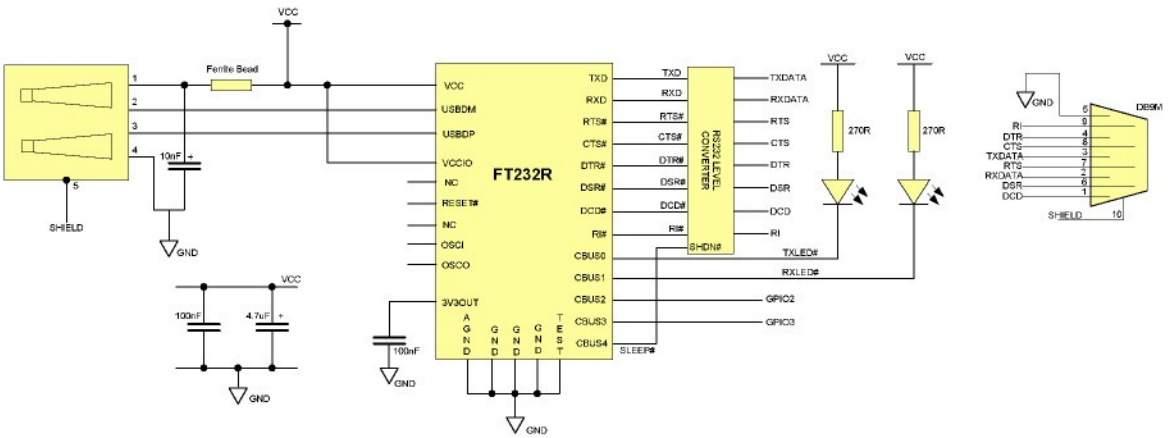


Figure 7.1 Application Example showing USB to RS232 Converter

An example of using the FT232R as a USB to RS232 converter is illustrated in Figure 7.1. In this application, a TTL to RS232 Level Converter IC is used on the serial UART interface of the FT232R to convert the TTL levels of the FT232R to RS232 levels. This level shift can be done using the popular "213" series of TTL to RS232 level converters. These "213" devices typically have 4 transmitters and 5 receivers in a 28-LD SSOP package and feature an in-built voltage converter to convert the +5V (nominal) VCC to the +/- 9 volts required by RS232. A useful feature of these devices is the SHDN# pin which can be used to power down the device to a low quiescent current during USB suspend mode.

A suitable level shifting device is the Sipex SP213EHCA which is capable of RS232 communication at up to 500k baud. If a lower baud rate is acceptable, then several pin compatible alternatives are available such as the Sipex SP213ECA, the Maxim MAX213CAI and the Analogue Devices ADM213E, which are all suitable for communication at up to 115.2k baud. If a higher baud rate is required, the Maxim MAX3245CAI device is capable of RS232 communication rates up to 1Mbaud. Note that the MAX3245 is not pin compatible with the 213 series devices and that the SHDN pin on the MAX device is active high and should be connect to PWREN# pin instead of SLEEP# pin.

In example shown, the CBUS0 and CBUS1 have been configured as TXLED# and RXLED# and are being used to drive two LEDs.

7.2 USB to RS485 Coverter

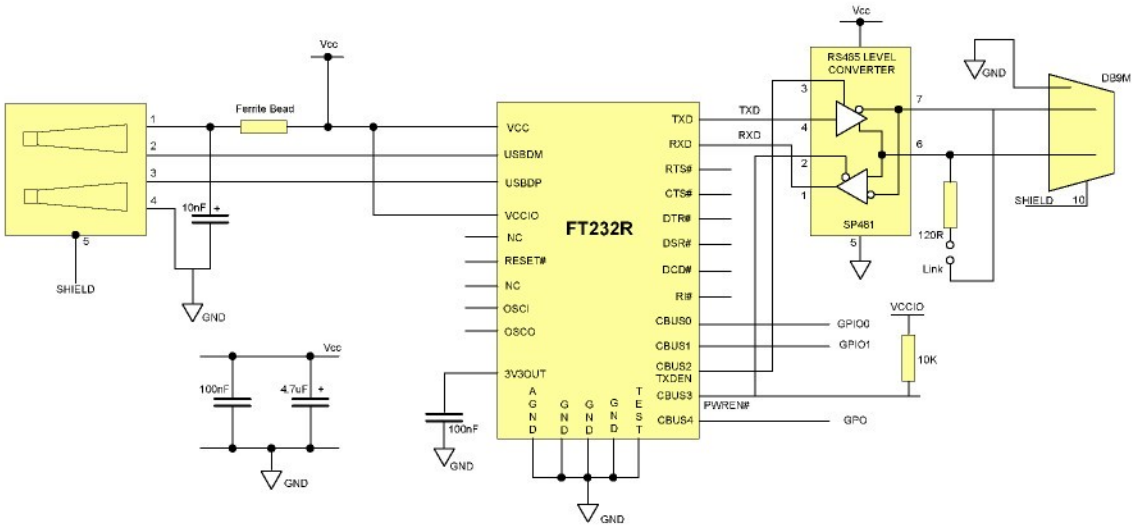


Figure 7.2 Application Example Showing USB to RS485 Converter

An example of using the FT232R as a USB to RS485 converter is shown in Figure 7.2. In this application, a TTL to RS485 level converter IC is used on the serial UART interface of the FT232R to convert the TTL levels of the FT232R to RS485 levels.

This example uses the Sipex SP481 device. Equivalent devices are available from Maxim and Analogue Devices. The SP481 is a RS485 device in a compact 8 pin SOP package. It has separate enables on both the transmitter and receiver. With RS485, the transmitter is only enabled when a character is being transmitted from the UART. The TXDEN signal CBUS pin option on the FT232R is provided for exactly this purpose and so the transmitter enable is wired to CBUS2 which has been configured as TXDEN. Similarly, CBUS3 has been configured as PWREN#. This signal is used to control the SP481's receiver enable. The receiver enable is active low, so it is wired to the PWREN# pin to disable the receiver when in USB suspend mode. CBUS2 = TXDEN and CBUS3 = PWREN# are the default device configurations of the FT232R pins.

RS485 is a multi-drop network; so many devices can communicate with each other over a two wire cable interface. The RS485 cable requires to be terminated at each end of the cable. A link (which provides the 120Ω termination) allows the cable to be terminated if the SP481 is physically positioned at either end of the cable.

In this example the data transmitted by the FT232R is also present on the receive path of the SP481.This is a common feature of RS485 and requires the application software to remove the transmitted data from the received data stream. With the FT232R it is possible to do this entirely in hardware by modifying the example shown in Figure 7.2 by logically OR'ing the FT232R TXDEN and the SP481 receiver output and connecting the output of the OR gate to the RXD of the FT232R.

Note that the TXDEN is activated 1 bit period before the start bit. TXDEN is deactivated at the same time as the stop bit. This is not configurable.

7.3 USB to RS422 Converter

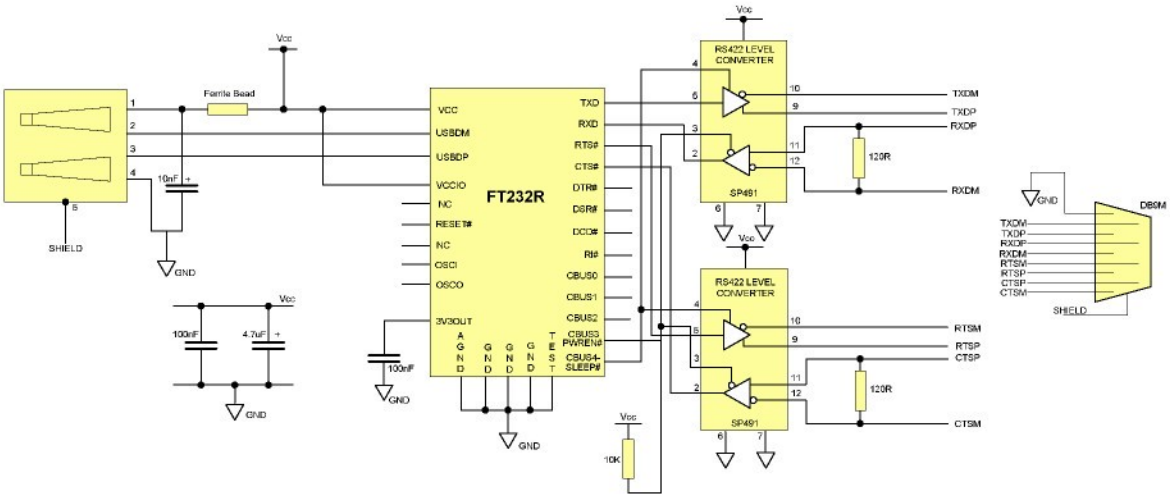


Figure 7.3 USB to RS422 Converter Configuration

An example of using the FT232R as a USB to RS422 converter is shown in Figure 7.3. In this application, two TTL to RS422 Level Converter ICs are used on the serial UART interface of the FT232R to convert the TTL levels of the FT232R to RS422 levels. There are many suitable level converter devices available. This example uses Sipex SP491 devices which have enables on both the transmitter and receiver. Since the SP491 transmitter enable is active high, it is connected to a CBUS pin in SLEEP# configuration. The SP491 receiver enable is active low and is therefore connected to a CBUS pin PWREN# configuration. This ensures that when both the SP491 transmitters and receivers are enabled then the device is active, and when the device is in USB suspend mode, the SP491 transmitters and receivers are disabled. If a similar application is used, but the design is USB BUS powered, it may be necessary to use a P-Channel logic level MOSFET (controlled by PWREN#) in the VCC line of the SP491 devices to ensure that the USB standby current of 2.5mA is met.

The SP491 is specified to transmit and receive data at a rate of up to 5 Mbaud. In this example the maximum data rate is limited to 3 Mbaud by the FT232R.

7.4 USB to MCU UART Interface

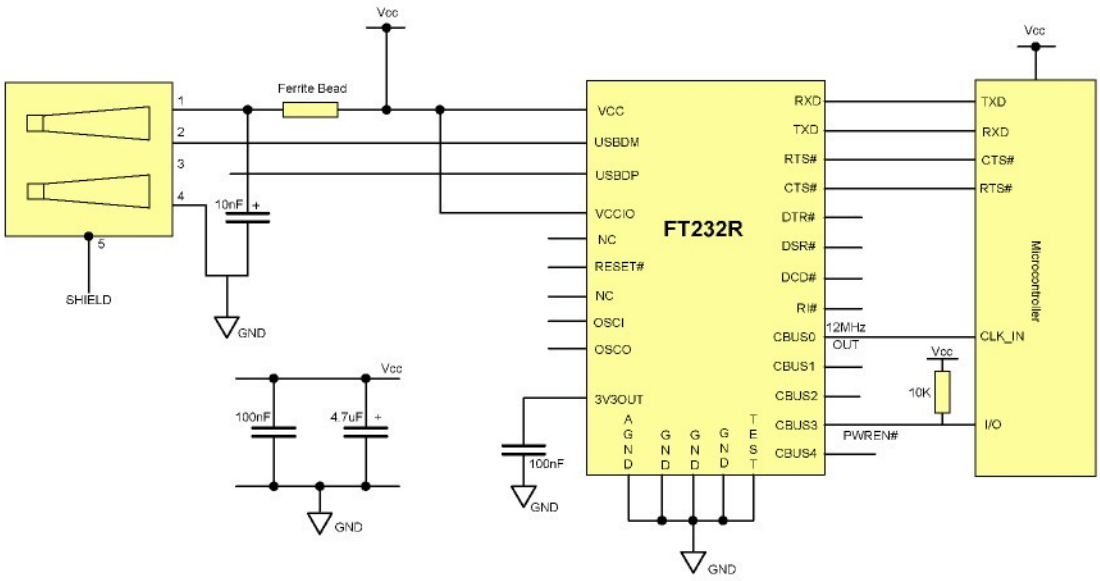


Figure 7.4 USB to MCU UART Interface

An example of using the FT232R as a USB to Microcontroller (MCU) UART interface is shown in Figure 7.4. In this application the FT232R uses TXD and RXD for transmission and reception of data, and RTS# / CTS# signals for hardware handshaking. Also in this example CBUS0 has been configured as a 12MHz output to clock the MCU.

Optionally, RI# could be connected to another I/O pin on the MCU and used to wake up the USB host controller from suspend mode. If the MCU is handling power management functions, then a CBUS pin can be configured as PWREN# and would also be connected to an I/O pin of the MCU.

7.5 LED Interface

Any of the CBUS I/O pins can be configured to drive an LED. The FT232R has 3 configuration options for driving LEDs from the CBUS. These are TXLED#, RXLED#, and TX&RXLED#. Refer to Section 3.5 for configuration options.

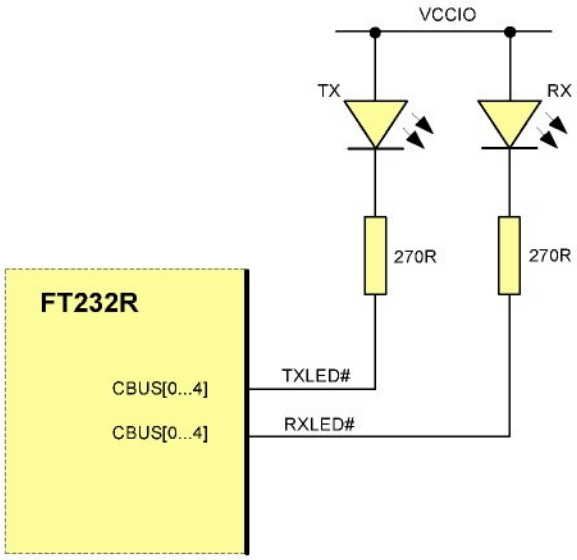


Figure 7.5 Dual LED Configuration

An example of using the FT232R to drive LEDs is shown in Figure 7.5. In this application one of the CBUS pins is used to indicate transmission of data (TXLED#) and another is used to indicate receiving data (RXLED#). When data is being transmitted or received the respective pins will drive from tri-state to low in order to provide indication on the LEDs of data transfer. A digital one-shot is used so that even a small percentage of data transfer is visible to the end user.

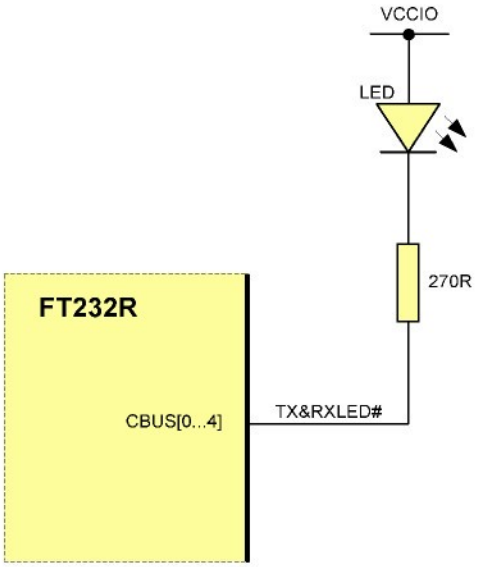


Figure 7.6 Single LED Configuration

Another example of using the FT232R to drive LEDs is shown in Figure 7.6. In this example one of the CBUS pins is used to indicate when data is being transmitted or received by the device (TX&RXLED). In this configuration the FT232R will drive only a single LED.

7.6 Using the External Oscillator

The FT232R defaults to operating using its own internal oscillator. This requires that the device is powered with VCC(min)=+4.0V. This supply voltage can be taken from the USB VBUS. Applications which require using an external oscillator, VCC= +3.3V, must do so in the following order:

1. When device powered for the very first time, it must have VCC > +4.0V. This supply is available from the USB VBUS supply = +5.0V.
2. The EEPROM must then be programmed to enable external oscillator. This EEPROM modification cannot be done using the FTDI programming utility, MPROG. The EEPROM can only be re-configured from a custom application. Please refer to the following applications note on how to do this:

   http://www.ftdichip.com/Documents/AppNotes/AN_100_Using_The_FT232_245R_With_External_Osc(FT_000067).pdf

3. The FT232R can then be powered from VCC=+3.3V and an external oscillator. This can be done using a link to switch the VCC supply.

The FT232R will fail to operate when the internal oscillator has been disabled, but no external oscillator has been connected.

8 Internal EEPROM Configuration

Following a power-on reset or a USB reset the FT232R will scan its internal EEPROM and read the USB configuration descriptors stored there. The default factory programmed values of the internal EEPROM are shown in Table 8.1.

Parameter	Value	Notes
USB Vendor ID (VID)	0403h	FTDI default VID (hex)
USB Product UD (PID)	6001h	FTDI default PID (hex)
Serial Number Enabled?	Yes	
Serial Number	See Note	A unique serial number is generated and programmed into the EEPROM during device final test.
Pull down I/O Pins in USB Suspend	Disabled	Enabling this option will make the device pull down on the UART interface lines when in USB suspend mode (PWREN# is high).
Manufacturer Name	FTDI	
Product Description	FT232R USB UART	
Max Bus Power Current	90mA	
Power Source	Bus Powered	
Device Type	FT232R	
USB Version	0200	Returns USB 2.0 device description to the host. Note: The device is a USB 2.0 Full Speed device (12Mb/s) as opposed to a USB 2.0 High Speed device (480Mb/s).
Remote Wake Up	Enabled	Taking RI# low will wake up the USB host controller from suspend in approximately 20 ms.
High Current I/Os	Disabled	Enables the high drive level on the UART and CBUS I/O pins.
Load VCP Driver	Enabled	Makes the device load the VCP driver interface for the device.
CBUS0	TXLED#	Default configuration of CBUS0 – Transmit LED drive.
CBUS1	RXLED#	Default configuration of CBUS1 – Receive LED drive.
CBUS2	TXDEN	Default configuration of CBUS2 – Transmit data enable for RS485
CBUS3	PWREN#	Default configuration of CBUS3 – Power enable. Low after USB enumeration, high during USB suspend mode.

Parameter	Value	Notes
CBUS4	SLEEP#	Default configuration of CBUS4 – Low during USB suspend mode.
Invert TXD	Disabled	Signal on this pin becomes TXD# if enable.
Invert RXD	Disabled	Signal on this pin becomes RXD# if enable.
Invert RTS#	Disabled	Signal on this pin becomes RTS if enable.
Invert CTS#	Disabled	Signal on this pin becomes CTS if enable.
Invert DTR#	Disabled	Signal on this pin becomes DTR if enable.
Invert DSR#	Disabled	Signal on this pin becomes DSR if enable.
Invert DCD#	Disabled	Signal on this pin becomes DCD if enable.
Invert RI#	Disabled	Signal on this pin becomes RI if enable.

Table 8.1 Default Internal EEPROM Configuration

The internal EEPROM in the FT232R can be programmed over USB using the FTDI utility program MPROG. MPROG can be downloaded from FTDI Utilities on the FTDI website (www.ftdichip.com). Version 2.8a or later is required for the FT232R chip. Users who do not have their own USB Vendor ID but who would like to use a unique Product ID in their design can apply to FTDI for a free block of unique PIDs. Contact FTDI support for this service.

9 Package Parameters

The FT232R is available in two different packages. The FT232RL is the SSOP-28 option and the FT232RQ is the QFN-32 package option. The solder reflow profile for both packages is described in Section 9.5.

9.1 SSOP-28 Package Dimensions

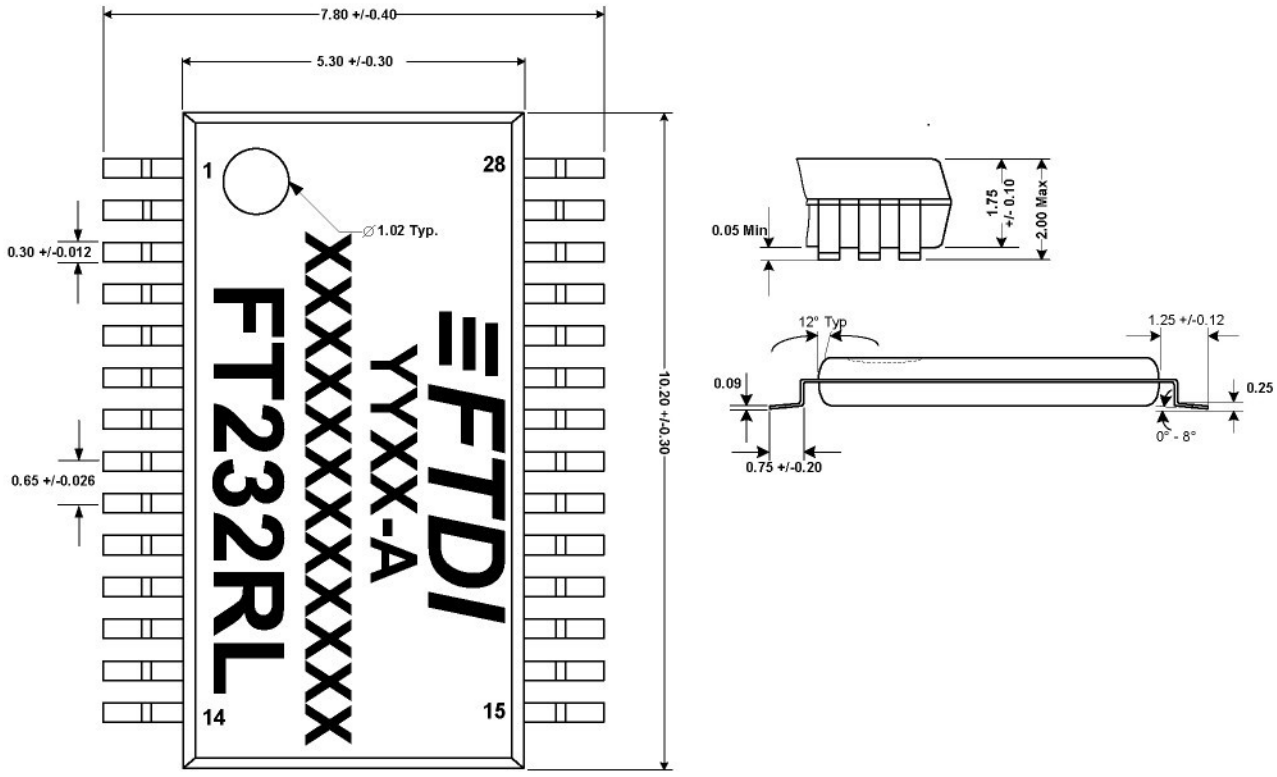


Figure 9.1 SSOP-28 Package Dimensions

The FT232RL is supplied in a RoHS compliant 28 pin SSOP package. The package is lead (Pb) free and uses a 'green' compound. The package is fully compliant with European Union directive 2002/95/EC.

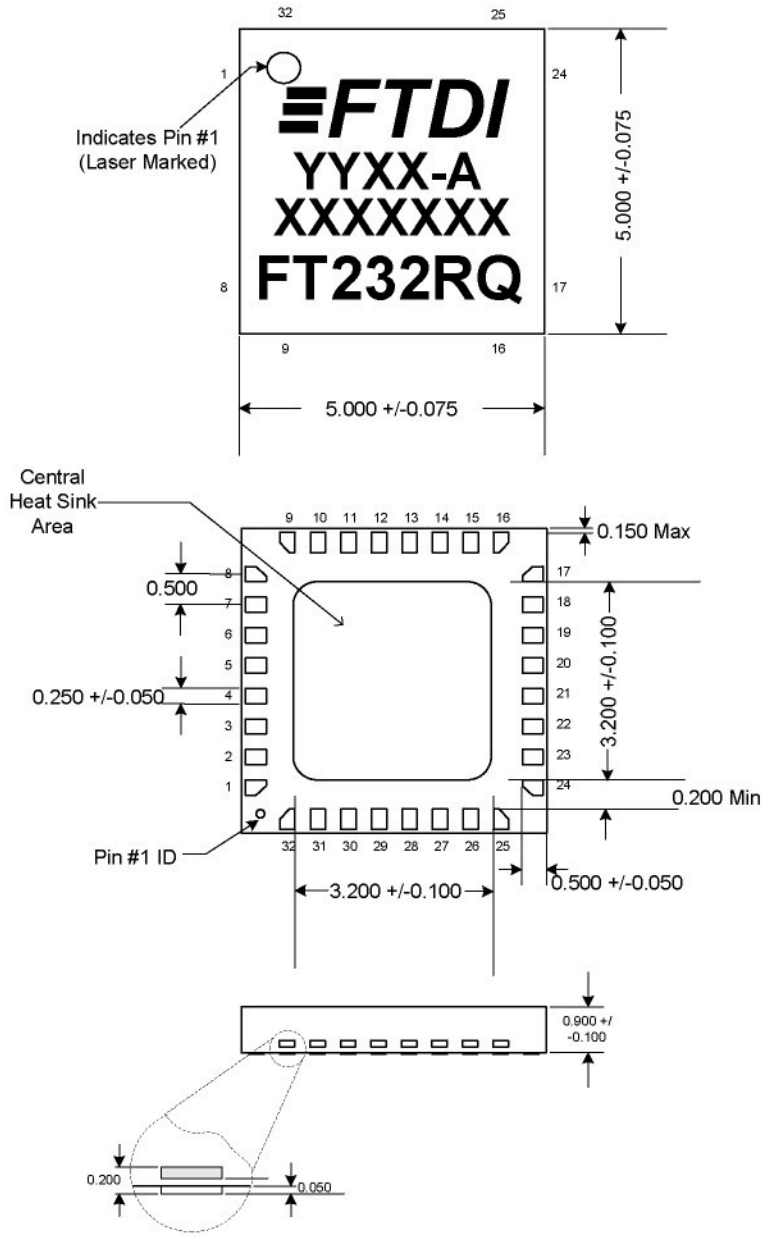
This package is nominally 5.30mm x 10.20mm body (7.80mm x 10.20mm including pins). The pins are on a 0.65 mm pitch. The above mechanical drawing shows the SSOP-28 package.

All dimensions are in millimetres.

The date code format is **YYXX** where XX = 2 digit week number, YY = 2 digit year number. This is followed by the revision number.

The code **XXXXXXXXXXXX** is the manufacturing LOT code. This only applies to devices manufactured after April 2009.

9.2 QFN-32 Package Dimensions



Note: The pin #1 ID is connected internally to the device's central heat sink area . It is recommended to ground the central heat sink area of the device.

Dimensions in mm.

Figure 9.2 QFN-32 Package Dimensions

The FT232RQ is supplied in a RoHS compliant leadless QFN-32 package. The package is lead (Pb) free, and uses a 'green' compound. The package is fully compliant with European Union directive 2002/95/EC.

This package is nominally 5.00mm x 5.00mm. The solder pads are on a 0.50mm pitch. The above mechanical drawing shows the QFN-32 package. All dimensions are in millimetres.

The centre pad on the base of the FT232RQ is not internally connected, and can be left unconnected, or connected to ground (recommended).

The date code format is **YYXX** where XX = 2 digit week number, YY = 2 digit year number.

The code **XXXXXXX** is the manufacturing LOT code. This only applies to devices manufactured after April 2009.

9.3 QFN-32 Package Typical Pad Layout

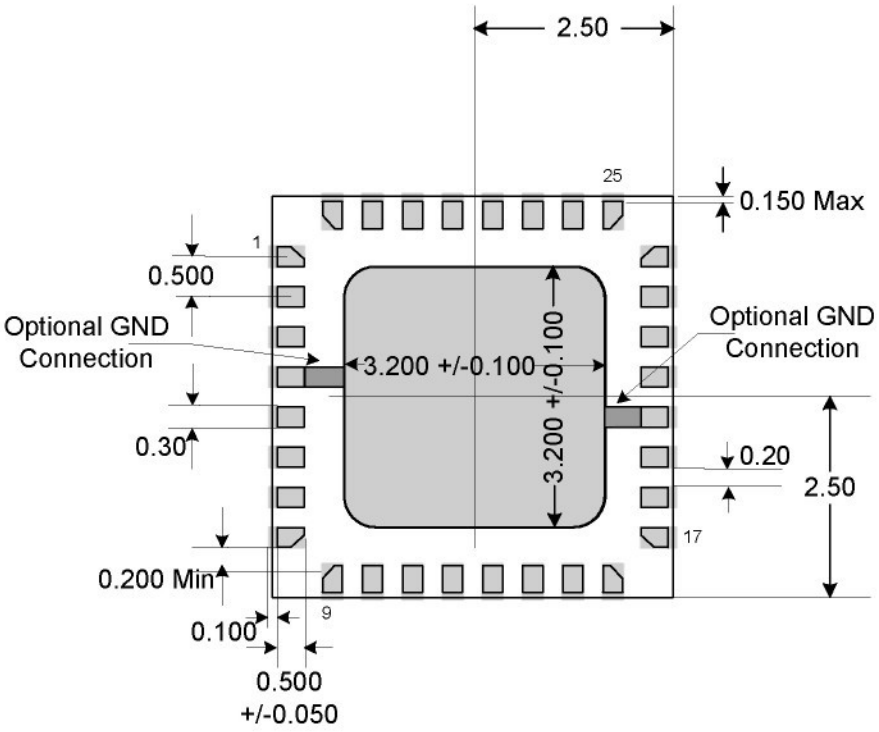


Figure 9.3 Typical Pad Layout for QFN-32 Package

9.4 QFN-32 Package Typical Solder Paste Diagram

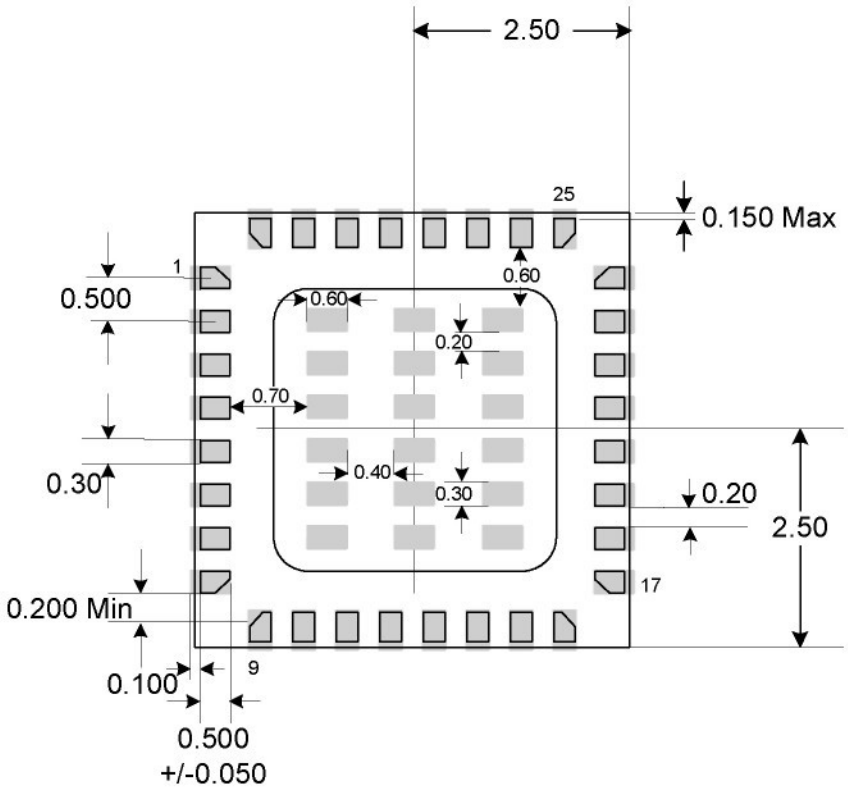


Figure 9.4 Typical Solder Paste Diagram for QFN-32 Package

9.5 Solder Reflow Profile

The FT232R is supplied in Pb free 28 LD SSOP and QFN-32 packages. The recommended solder reflow profile for both package options is shown in Figure 9.5.

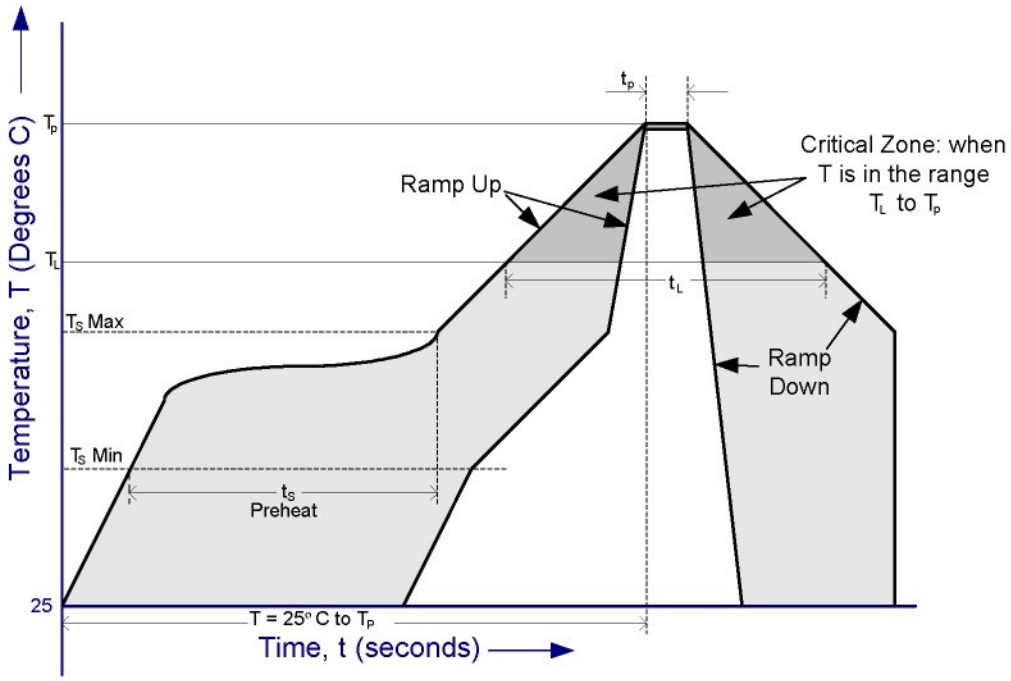


Figure 9.5 FT232R Solder Reflow Profile

The recommended values for the solder reflow profile are detailed in Table 9.1. Values are shown for both a completely Pb free solder process (i.e. the FT232R is used with Pb free solder), and for a non-Pb free solder process (i.e. the FT232R is used with non-Pb free solder).

Profile Feature	Pb Free Solder Process	Non-Pb Free Solder Process
Average Ramp Up Rate (T_s to T_p)	3°C / second Max.	3°C / Second Max.
Preheat - Temperature Min (T_s Min.) - Temperature Max (T_s Max.) - Time (t_s Min to t_s Max)	 150°C 200°C 60 to 120 seconds	 100°C 150°C 60 to 120 seconds
Time Maintained Above Critical Temperature T_L: - Temperature (T_L) - Time (t_L)	 217°C 60 to 150 seconds	 183°C 60 to 150 seconds
Peak Temperature (T_p)	260°C	240°C
Time within 5°C of actual Peak Temperature (t_p)	20 to 40 seconds	20 to 40 seconds
Ramp Down Rate	6°C / second Max.	6°C / second Max.
Time for T= 25°C to Peak Temperature, T_p	8 minutes Max.	6 minutes Max.

Table 9.1 Reflow Profile Parameter Values

10 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place
Centurion Business Park
Glasgow, G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL http://www.ftdichip.com
Web Shop URL http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No 516, Sec. 1 NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.admin@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
ChangNing District,
ShangHai, China

Tel: +86 (21) 62351596
Fax: +86(21) 62351595

E-Mail (Sales): cn.sales@ftdichip.com
E-Mail (Support): cn.support@ftdichip.com
E-Mail (General Enquiries): cn.admin1@ftdichip.com

Web Site URL: http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A - List of Figures and Tables

List of Figures

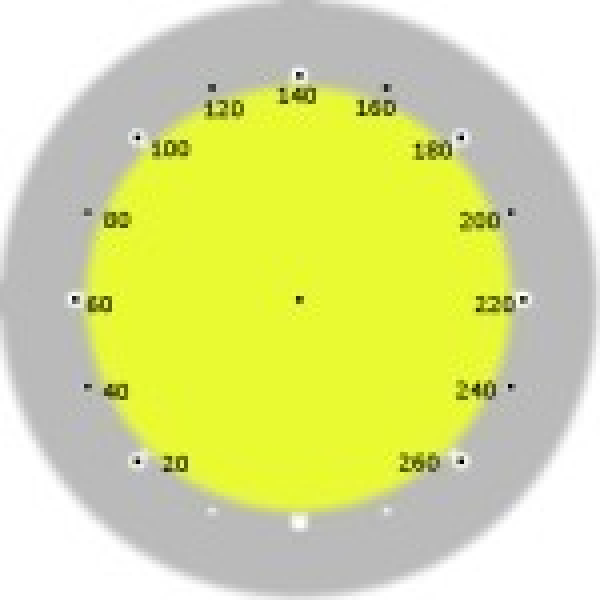
List of Tables

Appendix B - Revision History

Version 0.90	Initial Datasheet Created	August 2005
Version 0.96	Revised Pre-release datasheet	October 2005
Version 1.00	Full datasheet released	December 2005
Version 1.02	Minor revisions to datasheet	December 2005
Version 1.03	Manufacturer ID added to default EEPROM configuration; Buffer sizes added	January 2006
Version 1.04	QFN-32 Pad layout and solder paste diagrams added	January 2006
Version 2.00	Reformatted, updated package info, added notes for 3.3V operation;	June 2008
	Part numbers, TID; added UART and CBUS characteristics for +1.8V;	
	Corrected RESET#; Added MTTF data;	
	Corrected the input switching threshold and input hysteresis values for VCCIO=5V	
Version 2.01	Corrected pin-out number in table3.2 for GND pin18.	
	Improved graphics on some Figures.	
	Add packing details. Changed USB suspend current spec from 500uA to 2.5mA	
	Corrected Figure 9.2 QFN dimensions.	August 2008
Version 2.04	Corrected Tape and Reel quantities.	
	Added comment "PWREN# should be used with a 10kΩ resistor pull up".	
	Replaced TXDEN# with TXDEN since it is active high in various places.	
	Added lot number to the device markings.	
	Added 3V3 regulator output tolerance.	
	Clarified VCC operation and added section headed "Using an external Oscillator"	
	Updated company contact information.	April 2009


1
2
3
4
5
1/min x 1000
12
3
6
9


1/1
0
10
20
30
40
50
60
70
80
100
120
140
160
180
200
220
240
km/h


120
140
160
100
180
80
200
60
220
40
240
20
260

FTDI CDM Drivers - Revision Comments May 18, 2006

FTDI's CDM drivers provide both D2XX and VCP functionality through a single driver on PCs running the following Windows operating systems:

- Windows 2000
- Windows XP
- Windows Server 2003

A 64-bit build is also available for the following Windows operating systems:

- Windows XP x64
- Windows Server 2003 x64

The files included in a standard 32-bit CDM release are:

- Installation files
  - FTDIBUS.INF
  - FTDIPORT.INF
- Driver files
  - FTDIBUS.SYS
  - FTSER2K.SYS
- Uninstaller
  - FTDIUNIN.EXE
  - FTDIUN2K.INI
- D2XX Interface
  - FTD2XX.DLL
  - FTD2XX.LIB
  - FTD2XX.H

The files included in a standard 64-bit CDM release are:

- Installation files
  - FTDIBUS.INF
  - FTDIPORT.INF
- Driver files
  - FTDIBUS.SYS
  - FTSER2K.SYS
- Uninstaller
  - FTDIUNIN.EXE
  - FTDIUN2K.INI
- D2XX Interface
  - FTD2XX.DLL (64-bit)
  - FTD2XX.DLL (32-bit)
  - FTD2XX.LIB (64-bit)
  - FTD2XX.LIB (32-bit)
  - FTD2XX.H

On making a new release the files will also be posted onto FTDI's web site http://www.ftdichip.com/FTDrivers.htm in ZIP file format.

Release versions

2.00.00 (May 18, 2006)

New driver architecture combining functionality of D2XX and VCP drivers.
WHQL candidate.

1.00.2176 (December 12, 2005)

Fixed installation problem.
WHQL candidate.

1.00.2172 (October 31, 2005)
Support for FT2232C devices.

1.00.2170 (October 27, 2005)
Fixed write request synchronization issue.

1.00.2169 (September 1, 2005)
Fixed flow control issues.
Changed cancel processing to fix blue screen.

1.00.2168 (June 8, 2005)
Fixed lockup condition on port close.

1.00.2166 (May 23, 2005)
Fixed bug in software flow control.

1.00.2163 (February 7, 2005)
Support for buffered writes.
Fixed property page.

1.00.2162 (November 25, 2004)
Tested with HCT 12.0.

1.00.2160 (October 26, 2004)
Added option to prevent modem control signals following legacy port behaviour at startup.

1.00.2159 (October 15, 2004)
Fixed delay on disconnect under WinXP.

1.00.2157 (September 28, 2004)
Fixed bug that could cause computer to hang during closedown.

1.00.2156 (September 9, 2004)
Fixed bug in baud rate divisor calculation for AM devices.
Location Ids supported is no longer restricted to 10 or less.
Fixed bug that could cause computer to hang on surprise removal.
Fixed bug that caused device handle notification to fail.

1.00.2154 (April 20, 2004)
Fixed initialization problem on WinXP SP1.
Fixed IOCTL_SERIAL_GET_DTRRTS.

1.00.2151 (February 4, 2004)
Fixed problem with signalling events on disconnect with open port.

1.00.2150 (January 19, 2004)
Fixed problem with signalling line status errors.
Fixed problem with Location IDs and external hubs.

1.00.2148 (November 11, 2003)
More fixes for WHQL.
Fixed problem that prevented RXCHAR event being signalled when buffers were full.

1.00.2146 (October 28, 2003)
Various fixes for WHQL.
Added option for immediate processing of vendor commands.

1.00.2145 (October 20, 2003)
Fixed power-level problem with VCP serializer DLL interface.

1.00.2143 (September 24, 2003)
Fixed problem with RTS control toggle mode.
Fixed problem with device state after suspend/resume.

1.00.2140 (September 11, 2003)
Fixed problem with uninstall from DeviceManager.
Fixed problem with Sandstorm software.
Improved device request processing.
Fixed problem with modem control signal state on open.
Fixed cancel write request problem.
Added reset pipe retry count.
Added maximum devices option.
Added supported locations option.

1.00.2134 (June 16, 2003)
More changes for BusHound.

1.00.2133 (June 12, 2003)
Fixed BusHound compatibility problem.
Updated properties page with serial enumerator option.

1.00.2132 (June 09, 2003)
Bug fixes for substitution mode.
Fixed problem where laptops could not enter standby mode.
Fixed problem with Lexmark printer.
Added "Set RTS on Close" flag.
Updated properties page.

1.00.2126 (April 10, 2003)
Support substitution mode.
Restore device state on return from suspend or hibernate.
Fixed device naming problem.
Fixed bug in purge when receive buffer full.
Fixed bug in properties page.

1.00.2115 (February 25, 2003)
Beta release includes drivers that have passed Microsoft certification tests.

1.00.2115 (December 20, 2002)
Beta release.
Support force XON/XOFF option.
New properties page.

1.00.2114 (November 1, 2002)
Beta release.
Correctly identifies unserialized FT232BM and FT245BM devices.
Fixed remote wakeup.

1.00.2112 (October 25, 2002)

Beta release.
Support for FT232BM and FT245BM.
Uses location information to enumerate non-serialized devices.

1.00.2104 (July 22, 2002)
Drivers digitally signed.

1.00.2101 (February 1, 2002)
Fixed problem that resulted in some devices not coming out of hibernate.
Fixed divide-by-zero problem zero baud rate.

1.00.2099 (January 7, 2002)
Fixed problem with aliased baud rates.

1.00.2098 (December 20, 2001)
Improved transmit throughput.
Support serial printers through Registry setting.
Support for non-standard baud rates.
Support Transmit Immediate.
Disable modem control signals on port close.
Fixed serial mouse disconnect problem in Standby mode.
Runs under driver verifier.

1.00.2088 (October 30, 2001)
Fixed connect/disconnect problem in Windows XP.
Runs under driver verifier in Windows XP.

1.00.2086 (October 5, 2001)
Fixed write request timeout processing.
Fixed problem with restarting writes after port close.
Fixed no space in read buffer problem.

1.00.2084 (October 2, 2001)
Passes HCT tests (required for Windows XP Logo).
Uses common uninstaller FTDIUNIN.EXE.
Fixed minimum timeout problem.
Fixed write request blue screen.
Fixed write request timeout processing.
Fixed data loss at low baud rates.

1.00.2078 (July 27, 2001)
Fixed support for Logitech mice.
Enhanced write request processing.

1.00.2072 (May 2, 2001)
Runs under Windows XP.
Fixed receive buffer full problem.

1.00.2071 (Apr 26, 2001)
Enhanced BREAK condition processing.

1.00.2069 (Mar 28, 2001)
Fixed hyper-terminal re-boot problem seen when transmitting files at low baud rates.

1.00.2068 (Mar 23, 2001)
Drop modem status signals on disconnect

1.00.2067 (Feb 26, 2001)
Fixed disconnect with open port problems
Support multiple devices attached at the same time
Fixed hyper-terminal re-boot problem seen when typing characters in terminal screen
Enabled for surprise removal (removes unexpected removal dialogue box)

1.00.2060 (Jan 19, 2001)
Change to modem status register set up to improve port initialisation

1.00.2058 (Nov 7, 2000)
Fix to changing buffer size under Windows 2000 for FT8U232AM and FT8U245AM

1.00.2057 (Oct 19, 2000)
Fix to enable speeds greater then 115k baud for FT8U232AM and FT8U245AM

1.00.2055 (Sept 7, 2000)
Support for common INF for Win98 and Win2k drivers
Update of INF to support FT8U232AM and FT8U245AM

1.00.2054 (Aug 10, 2000)
Increased time out delay to make enumerator performance more robust

1.00.2053 (Aug 9, 2000)
Fixed transmit toggle problem

1.00.2052 (July 24, 2000)
Driver stack made consistent to fix device power state failure
INF changes to add serial services

1.00.2051 (June 15, 2000)
Fixed problem with 230k baud select, IN transfer restriction of 64 bytes removed
Fixed X-ON/X-OFF handshaking problem, added support for serial mouse

1.00.2049 (Mar 25, 2000)
Install / uninstall for new naming (ftser2k)

1.00.2046 (Mar 15, 2000)
New naming format: ftserial -> ftser2k

1.00.2044 (Mar 8, 2000)
Update to fix problem with Direct Cable Connect

1.00.2041 (Mar 3, 2000)
Fixed problem with baud rate selection

1.00.2040 (Mar 1, 2000)
Removed FTDI uninstaller

1.00.0 Beta (Feb 21, 2000)
Update to improve flow control

Alpha100 (Feb 18, 2000) - first release
Connects to ISP. Some flow control problems.