České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Rychlý přenos obrazu a dat s využitím EZ-USB

Jiří Zima

Vedoucí práce: Ing. Jan Fischer, CSc.

Studijní program: Elektrotechnika a informatika

Obor: Kybernetika a měření

Leden 2009

Akademický rok **2007-2008**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jiří Zima**

Obor: **Kybernetika a měření**

Název tématu česky: **Rychlý přenos obrazu a dat s využitím EZ-USB**

Název tématu anglicky: **High Speed Image and Data Transfer Using EZ-USB**

Zásady pro vypracování:

Pro řadič EZ-USB – CY7C68013A pracující v synchronním módu čtení navrhněte a realizujte metodu rychlého přenosu výstupních dat z optoelektronického plošného obrazového senzoru CMOS, případně videodekodéru do nadřazeného PC. Výsledné řešení zajistí snímání obrazu a jeho přenos v reálném času. Pro vlastní řadič i nadřazené PC využijte, tvořte potřebné programové vybavení.

Posuďte také možnost použití řadiče CY7C68013A ve funkci rychlého záznamníku dat jako jednoduchého logického analyzátoru využitelného v laboratořích.

Seznam odborné literatury:

[1] Souček P.: Bakalářská práce ČVUT – FEL. Praha 2007

[2] Cypress: EZ-USB FX2LP USB Microcontroller. 2006

[3] Skalický P.: Mikroprocesory řady 8051. BEN, Praha 2005

Vedoucí bakalářské práce: Ing. Jan Fischer, CSc.

Datum zadání bakalářské práce: 14. prosinec 2007

Platnost zadání do[1]: 24. leden 2009

L.S.

Prof. Ing. Vladimír Haasz, CSc.
vedoucí katedry

Doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 14. 12. 2007

---

[1] Platnost zadání je omezena na dobu dvou následujících semestrů.

Prohlášeni:

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 zákona č.121/2000 sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne . . . . . 2009 . . . . . .

Jiří Zima

Abstrakt

Tato bakalářská práce je zaměřena na realizaci metody rychlého přenosu obrazu z CMOS senzoru v reálném čase za pomoci USB řadiče Cypress EZ-USB (CY7C68013A) a možnosti jeho užití v měřicích zařízeních.

V práci jsou postupně rozebrány základy použití USB rozhraní, vlastní programování řadiče Cypress EZ-USB, programování PC aplikace pro obecný přenos dat, využití řadiče pro potřeby jednoduchého logického analyzátoru a použití pro snímání obrazu z CMOS senzoru modulu SPART3KAM. V závěru je diskutována využitelnost pro videometrii a možnost užití řadiče i pro jiné aplikace týkající se měření.

Abstract

This bachelor thesis is aimed to create method of fast image transfer from the CMOS sensor in real time using USB controller Cypress EZ-USB (CY7C68013A).

The thesis analyze basic use of USB interface, programming of the Cypress EZ-SUB controller, programming of the PC application for universal data transfer, the use of controller for simple logical analyzer and it's use for the image capturing from the CMOS sensor SPART3KAM. The conclusion of thesis discusses the use for video metrics and other application in measurements.

Poděkování

Touto cestou bych chtěl poděkovat všem, kteří přispěli k vzniku této bakalářská práce. Jmenovitě jde o Ing. Jana Fischera, CSc. za odborné vedení a pomoc při psaní, Ing. Jaroslava Třeštíka za poskytnutý přípravek s CMOS senzorem a cenné rady, Bc. Pavla Součka za stručné uvedení do problematiky USB přenosů, Ladislava Zimu za zasvěcení do některých tajů programovacího jazyka C++ a vedení serveru NOTEBOOK.cz za zapůjčení technického vybavení pro testování implementací hostitelských USB řadičů.

Obsah

1 Úvod

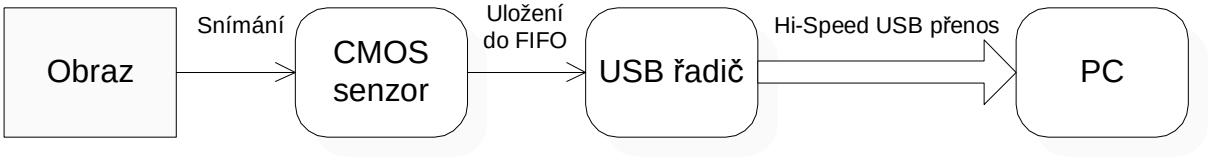
1.1 Motivace

V posledních letech z počítačů postupně mizí všechna dříve používaná rozhraní, která s nimi byla pomalu od vzniku standardu osobních počítačů. Před 10 lety bylo navrženo rozhraní USB, které mělo za cíl nahradit všechna ostatní a nabídnout vyšší rychlosti, možnost osazení téměř libovolným počtem portů a vytvoření inteligentní správy prostředků, která oprostí vývojáře od nutnosti spravovat nízkoúrovňové funkce komunikace mezi zařízeními. V oblasti komerční sféry se během několika let USB ujalo a nyní je situace taková, že ze stolních počítačů prakticky vymizela původní paralelní a sériová rozhraní.

Jsem přesvědčen, že rozhraní USB je nejvhodnějším levným řešením pro přenos obrazu v reálném čase nejen z CMOS senzoru, ale také z video kodeků a dalších zdrojů obecných dat. Bylo by proto vhodné mít navržené takové řešení, které bude možné dále snadno implementovat v různých měřicích zařízeních, neboť lze předpokládat, že USB rozhraní bude díky svému návrhu s ohledem na maximální kompatibilitu dlouhodobě podporované na osobních počítačích i průmyslových systémech.

1.2 Cíl práce

Mým hlavním cílem je nabídnout řešení rychlého přenosu obrazu, potažmo dat z měřicích zařízení (CMOS senzoru) v reálném čase. Nyní se přípravky s optickými senzory v laboratoři často vybavují vnitřní pamětí, která celý snímek uchovává, aby jej následně bylo možné postupně přenést do počítače. Na vině je nízká rychlost přenosu, která trvá řádově delší dobu než samotné snímání. V této práci hodlám rozebrat postupy, jak využít řadič Cypress EZ-USB pro přenášení obrazu (dat) v reálném čase, a tím zjednodušit návrh některých laboratorních přípravků a rozšířit jejich funkcionalitu.



Obr. 1 - Znázornění přenosu obrazu do PC

Dále bych rád hlouběji pronikl do problémů spojených s přenosem dat s využitím synchronního zápisu do FIFO paměti řadiče a nalezl příčiny nízké dosažené rychlosti v práci [1], na kterou navazuji. Hodlám se zabývat nejen praktickým řešením implementace rychlého přenosu dat s maximální frekvencí datové sběrnice (48 MHz), řešením přenosu obrazu v reálném čase a užitím řadiče pro potřeby jednoduchého logického analyzátoru, ale také teorií nutnou k obecnému pochopení hlavních problémů spojených s užitím USB. Tato práce by měla být zároveň jednoduchým, uceleným návodem pro všechny další studenty a akademické pracovníky, kteří se rozhodnou použít řadič Cypress EZ-USB v jeho libovolné formě pro rychlou komunikaci mezi počítačem a vlastním (měřicím) zařízením.

2 Popis USB rozhraní

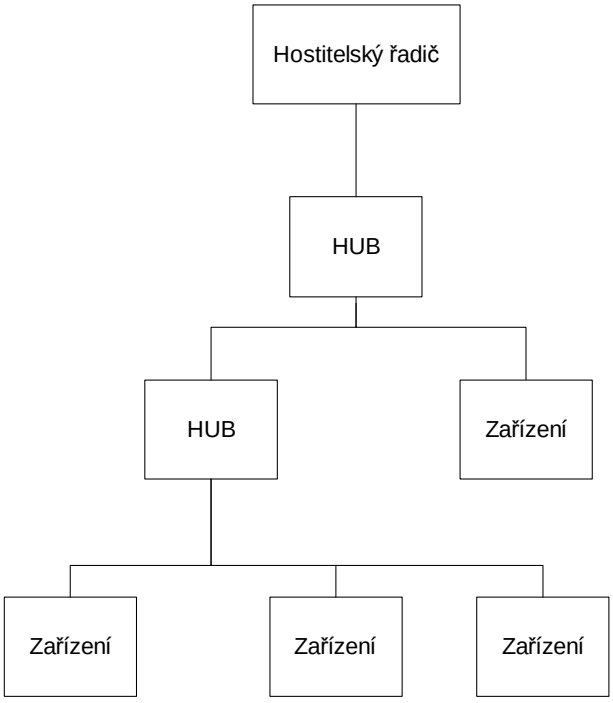
Bez vysvětlení základů USB rozhraní by nebylo možné pochopit vše potřebné k vlastnímu přenosu. Popsat celé rozhraní by vyžadovalo podstatně více prostoru, než kolik se očekává od bakalářské práce. V této kapitole se proto budu věnovat především těm částem, které jsou stěžejní pro rychlý přenos dat.

2.1 Seznámení, kompatibilita a rychlost

USB je zkratkou znamenající Universal Serial Bus – univerzální sériovou sběrnici. Ta byla navržena jako náhrada především rozhraní RS232. Na rozdíl od RS232 je vyžadována přítomnost sofistikovaného řadiče na obou stranách a přenos probíhá formou paketů. Kvůli tomu je implementace sice složitější, ale nabízí mnohem více možností. V zapojení je vždy určeno, které zařízení je hostitel (**USB Host**). Takové zařízení zahajuje veškeré přenosy (oběma směry).

V roce 1998 se s Pentium II procesory postupně začalo rozšiřovat osazování počítačů tímto rozhraním. V této době šlo o verzi 1.1, která podporuje dvě rychlosti. **Low Speed** (10–100 kb/s) a **Full Speed** (500 kb/s – 12 Mb/s). V roce 2003 (Pentium 4) postupně všichni výrobci přešli na USB verze 2.0, které přineslo znatelně vyšší rychlosti 25–480 Mb/s v režimu **High Speed**. V příštím roce se očekává přechod na rozhraní USB 3.0, které nabídne 10x vyšší rychlost proti svému předchůdci. Všechny verze USB jsou zpětně kompatibilní. Je tedy možné připojit USB 1.1 zařízení do USB 2.0 portu. Zároveň vhodně naprogramované USB 2.0 zařízení může fungovat (se sníženou rychlostí) v portu USB 1.1. Tyto reálie zmiňuji, neboť je s nimi nutno od začátku počítat při návrhu vlastního zařízení. Ne každá laboratoř je nutně vybavena nejnovějšími počítači.

Podstatným posunem proti ostatním rozhraním je možnost připojit více zařízení. Teoreticky jich je možné na jeden řadič připojit až 127 (a řadičů může být v jednom počítači více). Toho lze docílit pomocí rozbočovačů (**hub**). Ty umožňují připojit fyzicky více zařízení, jejich pakety vhodně poskládat a poslat nadřazenému zařízení. Strukturu připojených zařízení k USB řadiči lze definovat jako strom (Obr. 2).



Obr. 2 - Stromová struktura připojených zařízení na USB

Jedno fyzické zařízení může obsahovat několik USB zařízení, pokud v sobě samo implementuje rozbočovač. Takovým zařízením se říká složená (**composed devices**). Možnost složených zařízení je velmi praktická a hojně se v praxi používá. Zapojení konkrétního zařízení je možné snadno zjistit i z pohledu uživatele pomocí standardního nástroje **Správce zařízení** (Obr. 3) ve Windows (2000 a novější). Připojení, odpojení a konfiguraci jednotlivých zařízení zajišťuje hostitelský řadič.

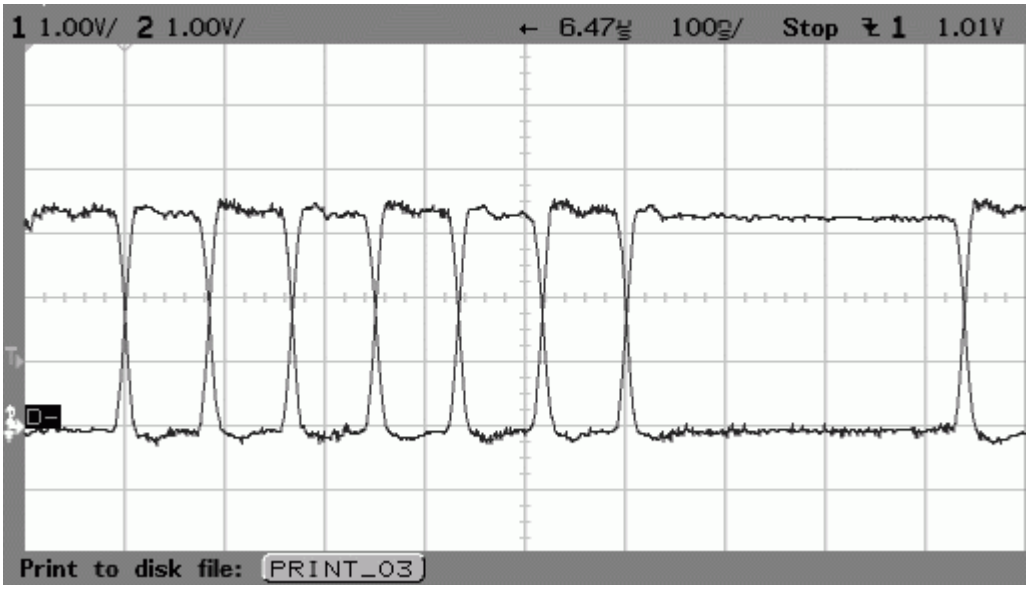
Obr. 3 - Zobrazení složeného zařízení ve Správci zařízení

2.2 Přenosová cesta

USB kabely v základu používají fyzicky 4 vodiče. První dvojice slouží pro vlastní přenos dat a označují se **D+** (zelená) a **D-** (bílá). Druhá dvojice vodičů se stará o napájení. Vodič **VCC** (červená) má stálé (stejnosměrné) napětí +5V a vodič **GND** (černá) je zem. Existuje celá řada USB konektorů lišících se podle velikosti a hierarchie zařízení. Hostitel zpravidla používá konektor typu A, který lze vidět na všech počítačích (u klasického počítače se neočekává práce v jiném než hostitelském režimu). Připojovaná zařízení nejčastěji používají standardní konektor typu B (tiskárny, skenery a další větší periferie) a **mini USB** (mobilní telefony a příslušenství k notebooku). Rozdíly tvaru konektorů názorně ukazuje Obr. 4.

Obr. 4 - Standardní typy USB konektorů

Na vodičích **D+** a **D-** se používá diferenciální přenos (Obr. 5). Hodnota na **D-** je tak negací hodnoty na **D+**. Tento mechanismus je použit kvůli odstranění některých chyb způsobených rušením. Ochrana přenosu je dále zajištěna pomocí CRC (řídící i datové pole) a řady samoopravných kódu. Všechny tyto záležitosti obstarává standardně hostitelský řadič, a tudíž je není nutné řešit ani z pozice vývojáře příslušného hardware a ani programátora. V případě potřeby však je možnost nechat řadič chyby hlásit a dokonce je i nechat řešit programově.

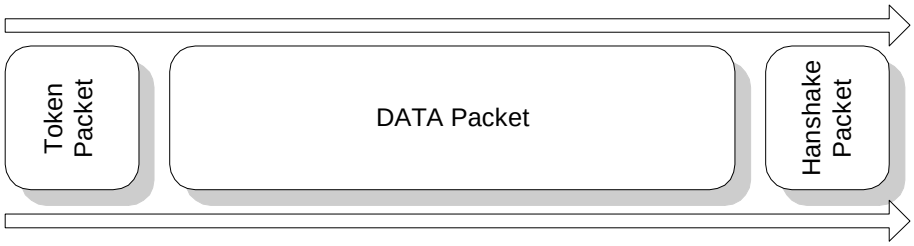


Obr. 5 - Diferenciální přenos na vodičích D+ a D-

2.3 Protokol přenosu

USB je řízenou sběrnicí. Hlavní roli má hostitelský řadič, který zpracovává příkazy ovladače a na jejich základě zahajuje potřebné transakce. Tyto transakce se skládají z několika paketů. Zpravidla jde o tři. Pouze v případě komunikace s **Low Speed** a **Full Speed** zařízením se používá ještě čtvrtý. Všechny transakce se následně podle nastavení vkládají do úseků zvaných (mikro-)rámce. Klasický rámec má délku 1 ms. V **High Speed** komunikaci se zavádí pojem mikrorámec, který má délku 125 µs. V principu jde o to samé a mikrorámce jsou zavedeny čistě k zajištění nižší latence při použití vyšší rychlosti.

Typickou transakci v režimu **High Speed** zobrazuje Obr. 6.



Obr. 6 - Transakce v režimu High Speed

Token paket

Tento paket slouží k navázání komunikace s konkrétním zařízením. Je v něm uložen směr přenosu, adresa zařízení a číslo endpointu (smysl endpointu je vysvětlen v následující kapitole). Zařízení čte token pakety, a pokud se v nějakém objeví jeho adresa, připraví se k přenosu. Vzhledem k tomu, že veškerou komunikaci zahajuje právě hostitel, směr přenosu je taktéž vztažen k jeho pozici (**IN** = ze zařízení do počítače, **OUT** = z počítače do zařízení).

Datový paket

Jakmile je komunikace navázána, mohou se začít přenášet vlastní data zvoleným směrem. Pokud by šlo o směr do počítače a zařízení ještě nemělo data připraveno, pošle se taková informace a přenos bude ukončen (chyba je v případě ovladače EZ-USB sdělena návratovou hodnotou funkce přenosu).

Handshake paket

Poslední paket posílá vždy zařízení směrem k hostiteli jako potvrzení přenosu dat a je v něm uložen kontrolní součet CRC.

2.4 Endpointy

Výraz **endpoint** v rámci USB zařízení definuje adresu komunikačního kanálu. Endpointů mívá každé zařízení více a označují se zkratkou EP a číslicí. Hlavním endpointem je **EP0**, který musí být implementovaný v každém zařízení. S pomocí tohoto endpointu probíhá řízení komunikace s hostitelským řadičem a není možné ho použít pro jiné než řídící přenosy. Ostatní endpointy jsou volitelné a mohou být v rozsahu 1–15. Každý endpoint navíc může být konfigurovaný pro oba směry přenosu.

Endpoint má v sobě definovány požadavky na frekvenci a latenci přístupů, chování obsluhy chyb, velikost datového toku, své číslo a požadovaný směr. V závislosti na konkrétní implementaci použitého USB řadiče bývá omezen počet endpointů, směr pro vybraná čísla a některé další vlastnosti.

2.5 Roury

S jednotlivými endpointy jsou spřaženy roury a **roura** (**pipe**) je tím samotným přenosovým kanálem. Hlavní roura spřažená s EP0 se nazývá **Default Control Pipe**. Každá roura udává, zda je nečinná. Žádost o její využití se provádí **IRP** paketem (**I/O Request Packet**). Pokud je požadovaná roura volná, zablokuje se pro ostatní a zahájí se přenos požadovaného počtu bajtů. Obsluha rour je automatizovaná hostitelským řadičem. Není proto problém přenést i větší množství dat, než se vejde do jednoho paketu. Systém data v závislosti na typu přenosu rozdělí a přenese ve více paketech, aniž by to programátor aplikace nějak musel řešit.

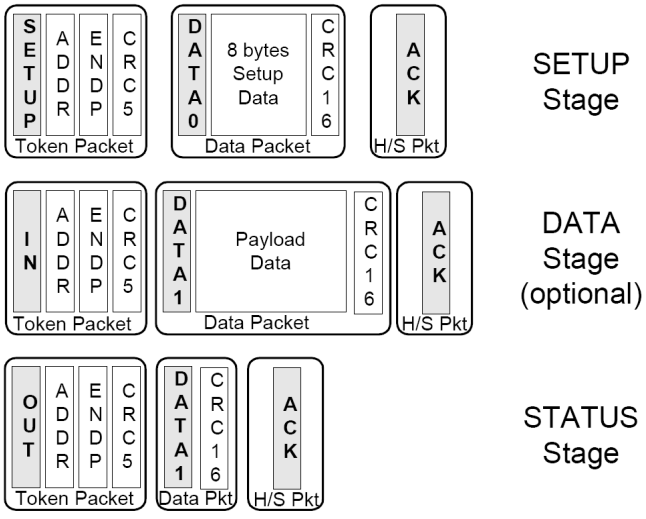
Roury jsou dvojího typu. První jsou **roury zpráv** (**message pipe**) a používají se v řídících přenosech. Všechny další typy přenosů využívají **proudové roury** (**stream pipe**), které nemají žádnou definovanou strukturu paketů. Proudová roura je na rozdíl od roury zpráv pouze jednosměrná. Z toho vyplývá, že pro obousměrný endpoint jsou vyžadovány dvě roury

(počet rour je omezen implementací řadiče). Pro záměr rychlého přenosu velkého množství dat je nutné použít typy přenosů s proudovými rourami. Těm budou se budou věnovat kapitoly 2.8 a 2.9.

2.6 Řídící přenosy

Řídící přenosy jsou základním typem přenosů při komunikaci s USB zařízením a nelze je ničím nahradit. Slouží pro přenos konfiguračních, stavových a řídících dat. Využívají se také pro obsluhu ostatních typů přenosů a jejich struktura je dána použitím roury zpráv.

Pro standardní aplikace s přehledem stačí ponechat jediný endpoint tohoto typu, a to **EP0**. U něj se využívá hlavně token paketu **SETUP**, který obsahuje sadu standardních příkazů pro obsluhu zařízení. Tyto příkazy jsou z části definovány už standardem a výrobce si je dále může rozšířit o své vlastní (**Vendor Specific Commands**). Příkazy v řídících přenosech využívají dvě až tři transakce (Obr. 7). První transakce posílá příkaz, druhá (volitelná) potřebná data a třetí výsledek (opačným směrem proti datům). Pro aplikace popisované v dalších kapitolách však není nutné tento typ přenosu nijak řešit, neboť o obsluhu EP0 se postará přímo ovladač EZ-USB.



Obr. 7 - Složení transakcí u řídících přenosů

2.7 Přerušovací přenosy

Tento typ přenosu se využívá v případech, kdy je vyžadováno rychlé doručení informace. Slouží pro přenos menších dat a své využití nachází například u vstupních zařízení HID (klávesnice, myš,…). Hlavní výhodou je nutnost řadiče bezpodmínečně obsloužit všechny přenosy tohoto typu. Garance se zajišťuje již při vytvoření roury a v případě, že by systém nemohl garantovat zvolenou periodu a velikost přenosu, rouru odmítne vytvořit. Pro takový příklad většinou musí mít firmware připraven i varianty s delší periodou opakování (případně menší velikostí paketů). Pokud žádná varianta neprojde a daný endpoint je potřeba, zařízení nebude možné použít. Pokud už je roura vytvořena a z jakéhokoli důvodu se přenos nějakého paketu nezdaří, řadič jej znovu neposílá a nechá problém řešit až vlastní aplikaci.

Přerušovací přenosy mohou využít až 80% jednoho mikrorámce. Velikost jednoho paketu může být až 134 B pro USB 2.0. Perioda opakování se nastavuje v intervalu od 125 µs do 4096 ms. Tento typ přenosu je zmíněn pouze pro úplnost. V aplikacích pro rychlý přenos dat není potřeba.

2.8 Izochronní přenosy

Stejně jako u přerušovacích přenosů je garantován přístup s minimálním zpožděním a konstantní tok dat. Platí zde stejná omezení jako pro přerušovací přenosy. Liší se však možností přenést v High Speed režimu až 1024 B na jeden standardní paket a až 3072 B na **high-bandwidth** paket. Při vhodném nastavení lze garantovat rychlost až 24 MB/s v reálném provozu. Pokud řadič nebude moci rychlost garantovat, nezdaří se opět už samotné vytvoření roury.

Nad použitím tohoto typu přenosu jsem uvažoval při realizaci programové části své práce. Nakonec jsem od něj upustil, neboť byl záměr dostat se na vyšší rychlosti a záruka latencí pod 1 ms nebyla pro potřebné aplikace stěžejní. V komerční sféře se tento typ přenosu hojně používá například u externích zvukových karet, kde je nízká latence vyžadována protokolem ASIO (Audio Stream Input/Output).

2.9 Hromadné přenosy (Bulk)

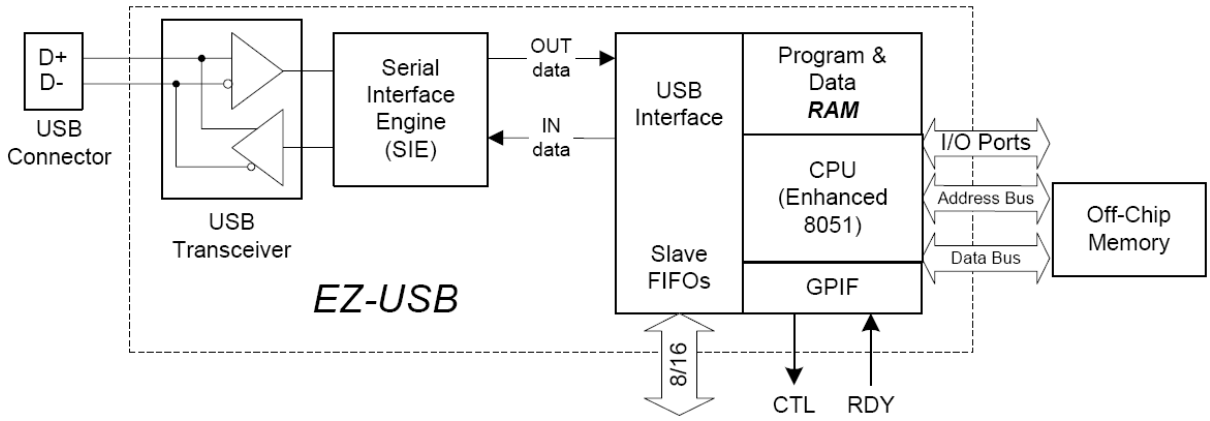
Přenosy typu bulk jsou pro potřeby této práce nejzajímavější. Používají se tam, kde je potřeba přenášet data maximální možnou rychlostí. Řadič při nich negarantuje žádnou periodu přístupů a není jim ani vyhrazena žádná konkrétní část mikrorámce. Umí však v mikrorámci vyplnit všechen volný prostor. Pokud na řadiči není připojeno žádné další zařízení, které by ho vytěžovalo vlastním přenosem, je možné mikrorámec využít téměř celý. Pro USB 2.0 platí, že jeden paket bulk může přenést až 512 B dat a v jednom mikrorámci lze provést až 14 transakcí tohoto typu. Čistě teoreticky by to v ideálních podmínkách znamenalo až 54 MB přenesených dat v jediné sekundě. V praxi tak velké rychlosti nelze dosáhnout. To už ovšem není chybou samotného přenosu, ale vlastní implementací hostitelských řadičů, kterým se věnuji v kapitole 4.5.

Při těchto přenosech řadič sám řeší opakování přenosů v případě chyby. To si může dovolit, protože pro bulk přenosy není garantována žádná konkrétní latence. Z pohledu programátora je opakování automatizované a transparentní.

3 Programování EZ-USB pro rychlý přenos dat

Cypress EZ-USB (CY7C68013A, Obr. 8) je jednočipovým USB řadičem s integrovaným 48MHz mikrokontrolérem kompatibilním s **Intel 8051**, vlastními 16 kB paměti, řadičem **I2C** a **UART** a 8/16bit datovou sběrnicí **FIFO**. Jde v podstatě o kompletní jednočipový počítač.

Mikrokontrolér 8051 primárně slouží k nastavení všech částí řadiče a případných dalších zařízení připojených přes I2C a UART. Jde samozřejmě o plnou implementaci, ale při rychlosti 4 hodinových cyklů na jednu instrukci jej nelze použít pro vlastní zpracování přijímaných dat v požadované rychlosti. Tento problém řeší integrovaná jednotka **GPIF** (**General Programmable Interface**), která umí řídit FIFO sama. Vhodným nastavením pak lze s celým přenosem mikrokontrolér úplně obejít a ten řeší jen požadavky na přerušení a příkazy USB.



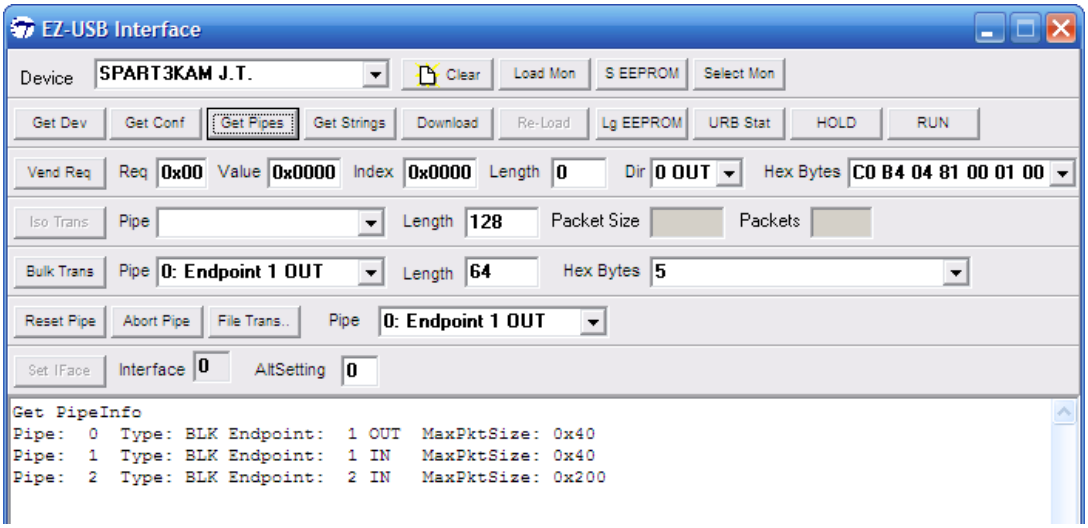
Obr. 8 - Zjednodušené blokové schéma řadiče Cypress EZ-USB

V této kapitole se budu věnovat vhodnému nastavení vlastního zařízení a programování jeho firmware pro rychlý přenos obecných dat do počítače. K jejímu napsání posloužila 100vývodová verze řadiče. Kód je však plně kompatibilní i osekanou verzí v menším pouzdře s 56 vývody.

3.1 Zavedení programu

Použitý řadič EZ-USB má výhodu v rozmanitých možnostech zavádění programu. Interní paměť o velikosti 16 kB je vyhrazena pro program i pro data. Po zapnutí probíhá boot sekvence a program lze nahrát z EEPROM nebo externího paměťového zařízení (platí jen pro verzi se 128 vývody). Jakmile je program v interní paměti RAM, procesor provede reset a spustí se s nahraným programem. Pokud se program nenahraje ani jedním z těchto způsobů, zavede se výchozí, při kterém je zařízení schopno komunikovat s počítačem několika základními příkazy. V takovém případě je pomocí speciálního příkazu na EP0 možné nahrát vlastní program do interní paměti přímo přes USB a vyvolat softwarový reset.

Poslední možnost je neocenitelná zejména při návrhu a testování zařízení. Nabízí zároveň určitou flexibilitu, kdy klientská aplikace obsahuje více programů a nahrává vhodný až na základě výběru uživatele. Vzhledem k jednoduchosti a rychlosti nahrání jsem u této možnosti nakonec zůstal. Operaci je možno provést z programu **CyConsole**, který je součástí SDK balíčku nutného pro vývoj PC aplikací využívajících tento řadič.



Obr. 9 - Diagnostika zařízení pomocí programu CyConsole

3.2 Nastavení deskriptorů

Vhodné nastavení **deskriptorů** je základem při tvorbě firmware. Cypress v rámci svého **SDK** dodává sadu několika ukázkových, takže není nutné psát vše od píky. Vycházel jsem z programu Bulkloop uloženém v podadresáři SDK: */USB/Examples/Fx2lp/bulkloop/* (funkce: data přenesená na jeden vstupní endpoint jsou automaticky překopírována do výstupního, kde je může opět počítač vyčíst). Nejde o samotný program, ale o všechny doprovodné soubory a vytvořený projekt pro vývojové prostředí **Keil Microvision**.

První kroky by měly směřovat k vhodné konfiguraci deskriptorů – ta se nachází v souboru *dscr.a51*. Následující kód ukazuje nastavení s jedním endpointem (EP0 se nepočítá). Konkrétně jde o vstupní endpoint EP2 typu bulk. Toto nastavení přestavuje minimum pro rychlý přenos dat:

```
;; Interface Descriptor
      db    DSCR_INTRFC_LEN      ;; Descriptor length
      db    DSCR_INTRFC          ;; Descriptor type
      db    0                    ;; Zero-based index of this interface
      db    0                    ;; Alternate setting
      db    1                    ;; Number of end points
      db    0ffH                 ;; Interface class
      db    00H                  ;; Interface sub class
      db    00H                  ;; Interface sub sub class
      db    0                    ;; Interface descriptor string index
```

```
;; Endpoint Descriptor
      db    DSCR_ENDPNT_LEN       ;; Descriptor length
      db    DSCR_ENDPNT          ;; Descriptor type
      db    82H                 ;; Endpoint number, and direction
      db    ET_BULK              ;; Endpoint type
      db    00H                 ;; Maximun packet size (LSB)
      db    02H                 ;; Max packect size (MSB)
      db    00H                 ;; Polling interval
```

Zvýrazněné řádky představují zásadní změny. První takový řádek označuje počet endpointů a přesně tolik bloků kódů pro endpoint deskriptory musí být následně uvedených v kódu. Pokud se číslo neshoduje, program nebude možné zkompilovat.

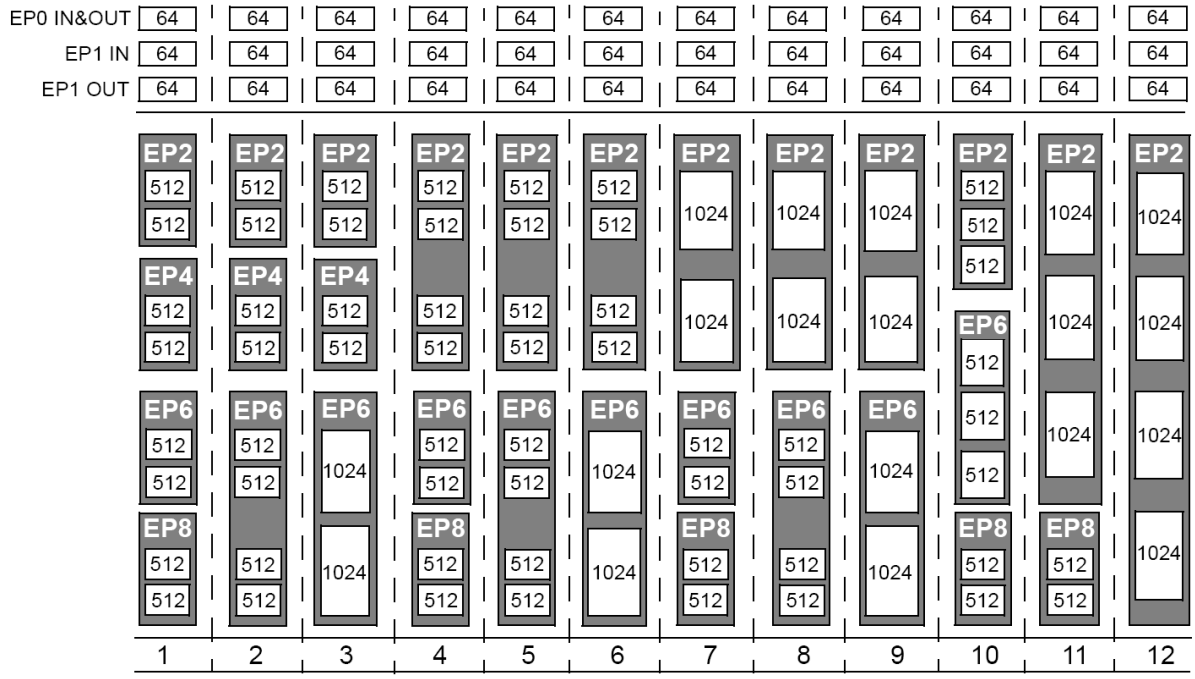
Na druhém zvýrazněném řádku označuje první číslice směr přenosu: **0** znamená směr ven z počítače a **8** směr do počítače. Druhá číslice udává číslo endpointu. Poslední dva zvýrazněné řádky udávají dvoubajtovou hodnotu (hexxa) velikosti paketu ve FIFO paměti. Hodnota 0x200h znamená velikost 512 B, což je zároveň maximum pro bulk endpoint, jak již bylo zmíněno v kapitole 2.9.

Hodnoty nastavené v deskriptorech jsou stěžejní pro ovladač zařízení v počítači. Pokud ve vlastním firmware nastavíte hodnoty jinak, ovladač se bude nadále řídit těmi s deskriptorů. V lepším případě vše bude fungovat a jen neefektivně využijete FIFO paměť. V horším případě program nebude správně pracovat.

3.3 Nastavení endpointů a jejich prostoru ve FIFO

Řadič EZ-USB má určitá specifika nastavení jednotlivých endpointů. EP0 má pevně danou velikost paketu 64 kB. Dále je ještě vyhrazen samostatný prostor 2x64 B pro **EP1IN** a **EP1OUT**. Tento endpoint je možné nastavit libovolně, jen zůstává omezení na velikost paměti. Pokud není potřeba, není nutné jej použít, ale jemu vyhrazenou paměť žádný jiný endpoint využít nemůže.

Pro další endpointy je vyhrazen prostor 4 kB, který je možné rozdělit mezi až 4 endpointy (EP2, EP4, EP6 a EP8). Celkem řadič podporuje přesně 12 možností konfigurace. Ty jsou znázorněny na Obr. 10.



Obr. 10 - Možné konfigurace endpointů na EZ-USB

V kapitole 3.2 byla ukázána konfigurace deskriptorů pro jeden vstupní endpoint. Tuto konfiguraci je nutné dodržet i ve vlastním programu. Inicializaci endpointů je nutné provést při startu, tudíž je nejlepší ji umístit do funkce *TD_Init()*. Následující řádky slouží k nastavení tohoto endpointu a vyhrazení jeho prostoru v paměti FIFO:

```
EP2CFG = 0xE0;  // nastaveni parametru EP2, 11100000
SYNCDELAY;
EP2BCL = 0x80;  // aktivace EP2
SYNCDELAY;
```

8bitový Registr EP*x*CFG obsahuje veškeré parametry endpointu. Tyto parametry jsou udány jednotlivými bity:

Pořadí bitu:	Funkce:	Možná nastavení:
7	Validní endpoint	0 = neaktivní, 1 = validní
6	Směr přenosu	0 = výstupní, 1 = vstupní
5-4	Typ přenosu	01 = izochronní, 10 = bulk, 11 = přerušovací
3	Velikost bufferu	0 = 512 B, 1 = 1024 B (lze použít pouze pro EP2 a EP6)
1-0	Počet bufferů	00 = 4x, 10 = 2x, 11 = 3x

Tab. 1 - Nastavení registru EPxCFG

Hodnota 0x80h registru EP2CFG podle Tab. 1 značí validní vstupní endpoint typu bulk s 4x512 B bufferem. Příkaz *SYNCDELAY* se musí vkládat mezi příkazy nastavení některých registrů, aby jejich nastavení proběhlo v pořádku. Aktivací nejvyššího bitu na registru EP*x*BCL se daný endpoint zprovozní s novým nastavením.

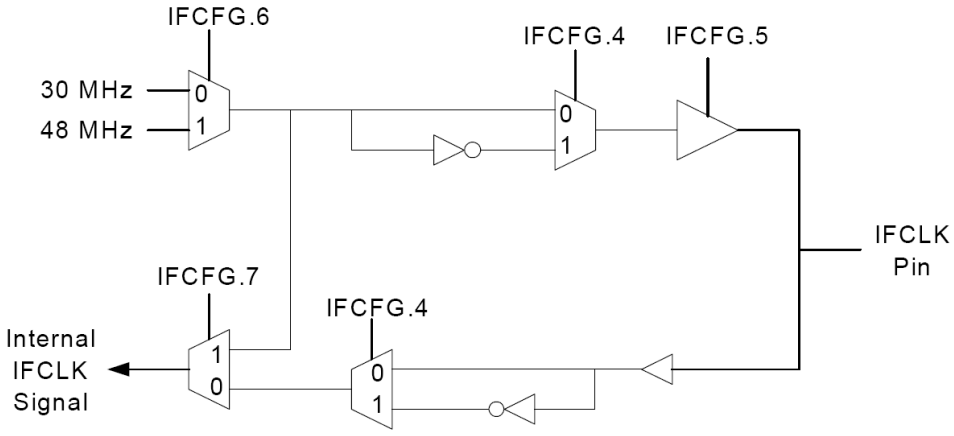
3.4 Registr IFCONFIG a nastavení čítače

Jakmile jsou všechny endpointy nastaveny, je potřeba nakonfigurovat registr časovače. EZ-USB má v sobě interní generátor hodinového signálu schopný pracovat na 30 a 48 MHz. FIFO rozhraní může využívat tohoto signálu, nebo může pracovat s hodinovým signálem externího zdroje. Nastavení časovače se provádí 8bit registrem **IFCONFIG**, jehož možnosti nastavení zobrazuje Tab. 2 a Obr. 11.

Pořadí bitu:	Funkce:	Možná nastavení:
7	Zdroj signálu	0 = externí, 1 = interní
6	Frekvence vnitřního generátoru hodinového signálu	0 = 30 MHz, 1 = 48 MHz
5	Výstup vnitřního generátoru	0 = vypnutý, 1 = zapnutý (pouze pro interní generátor)
4	Polarita signálu	0 = normální, 1 = invertovaná
3	Režim FIFO	0 = synchronní, 1 = asynchronní

Tab. 2 - Nastavení registru IFCONFIG

Vytvořil jsem pro potřeby této části práce jednoduché zapojení (Obr. 12), aby bylo možné otestovat funkčnost přenosu. Interní hodinový signál je vyvedený pomocí **IFCLK** do 8bit čítače 74HC590. Všech 8 výstupních vývodů čítače je následně přivedeno na vstup FIFO (PB0 – PB7).



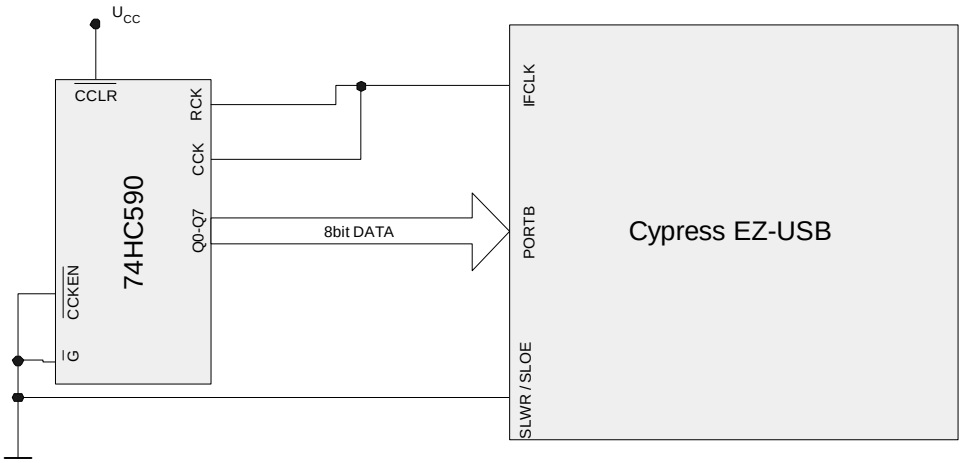
Obr. 11 - Nastavení hodinového signálu FIFO sběrnice registrem IFCONFIG

Takové zapojení vyžaduje následující nastavení ve funkci *TD_Init()*:

```
IFCONFIG = 0xA3;      // 10100011
```

Z Tab. 2 vyplývá, že se tímto příkazem nastaví interní zdroj hodinového signálu na frekvenci 30 MHz s vyvedením na IFCLK a že tento signál bude pro FIFO neinvertovaný vůči výstupu. Zároveň je tímto nastavením zajištěna synchronní práce FIFO.

Invertovaná polarita hodinového signálu dává čítači více času na ustálení průběhů, neboť do FIFO paměti se hodnoty čítače uloží o polovinu periody později (inverze hodinového signálu se provádí až pro FIFO a lze ji tedy použít i v případě externího signálu). Je dobré polaritu přepnout, pokud jsou při ukládání dat do FIFO problémy.



Obr. 12 - Zapojení řadiče EZ-USB s čítačem 74HC590 pro testování přenosu dat

Nižší frekvenci hodinového signálu (30 MHz) jsem zvolil zcela záměrně, a to hned ze dvou důvodů. Prvním jsou omezené schopnosti použití čítače 74HC590 v testovacím zapojení (Obr. 12), který už při 48 MHz nemůže pracovat a druhým je rychlost samotného USB. Pro frekvenci 48 MHz je potřeba přenášet 45,8 MB/s, což hodnota nedosažitelná s většinou běžných hostitelských řadičů. Problematiku hostitelských řadičů dále řeším v kapitole 4.5. Při frekvenci hodinového signálu 30 MHz se přenáší data rychlostí 28,6 MB/s. Taková hodnota by neměla být problémem pro většinu hostitelských řadičů ani při neustálém zatížení.

3.5 Přiřazení FIFO k endpointu

Posledním krokem k nastavení řadiče pro synchronní přenos do FIFO paměti je vhodná konfigurace FIFO registrů jednotlivých endpointů zvaných EP*x*FIFOCFG. Protože je v tomto případě využit pouze EP2, bude se nastavovat EP2FIFOCFG, a to následujícím způsobem:

```
EP2FIFOCFG = 0x0C;    // 1100
```

Tímto nastavením je zajištěn automatický běh FIFO v 8bit režimu nezávisle na procesoru. Tento registr má stejně jako předchozí také 8 bitů. Pro účely této práce jsou však podstatné pouze první 4. Účel jednotlivých bitů popisuje Tab. 3 (bit 1 nesmí být nikdy aktivní).

Pořadí bitu:	Funkce:	Možná nastavení:
3	Řízení FIFO	0 = manuální, 1 = automatické
2	Pakety nulové délky	0 = ignorovat, 1 = přijímat
0	Šířka sběrnice	0 = 8bit, 1 = 16bit

Tab. 3 - Nastavení registru EPxFIFOCFG

Pokud se program spustí s tímto nastavením, do FIFO by zatím žádná data neproudila. Za pomoci vývodů FIFOADR[1:0] na EZ-USB je ještě nutné vybrat endpoint, který se s FIFO spojí. Možnosti nastavení vývodů ukazuje Tab. 4. Pro použití EP2 se oba vývody připojí na zem. V synchronním módu se ukládají data do FIFO jen v případě neaktivního signálu na SLOE a SLWR (u výstupních endpointů jde o SLRD) - Obr. 15.

Endpoint	EP2	EP4	EP6	EP8
FIFOADR[1:0]	00	01	10	10

Tab. 4 - Spřažení endpointu s FIFO podle nastavení FIFOADR[1:0

V tuto chvíli je nastavení hotovo. V hlavní smyčce programu *TD_Pool()* není nutné nic obsluhovat a do paměti FIFO jsou data čtena rychlostí signálu interního generátoru

hodinového signálu. Pomocí aplikace CyConsole je možné ověřit, zda se do endpointu ukládají data. Tato aplikace umí vyvolat izochronní a bulk přenosy oběma směry.

Kompletní zdrojový kód tohoto firmware lze nalézt na CD v adresáři *HiSpeed_Transfer_Firmware*.

3.6 FIFO Flag příznaky

Při určitých aplikacích řadiče může být vhodné sledování některých příznaků v souvislosti s chodem endpointů spřažených s FIFO pamětí. Tyto příznaky se nazývají **FIFO Flags** a dokáží indikovat některé stavy, které mohou nastat při přenosu. Řadič nabízí příznaky FLAGA, FLAGB, FLAGC a FLAGD. U prvních tří lze vybrat, zda poběží v indexovaném či fixním módu. Čtvrtý funguje jen s fixním nastavením.

V indexovaném módu příznaky pracují s endpointem nastavením vývody FIFOADR[1:0] a mají pevně danou funkci dle Tab. 5.

Příznak	**FLAGA**	**FLAGB**	**FLAGC**
funkce	Programmable Level	Full Status	Empty Status

Tab. 5 - Funkce FIFO Flag příznaků pro indexovaný mód

Fixní mód umožňuje nastavit všechny čtyři příznaky libovolně pomocí registrů PINFLAGSAB a PINFLAGSCD. Tato možnost je důležitá zejména při použití více endpointů pro přenos dat.

4 Programování PC aplikace pro rychlý přenos dat

V minulé kapitole jsem rozebral programové vybavení ze strany USB zařízení. S touto kapitolou postupně projdu použití univerzálního ovladače EZ-USB a všechny důležité sekvence příkazů nutné k napojení na zařízení a zahájení rychlého přenosu. V závěru kapitoly jsou uvedeny dosažené rychlosti přenosů. K programování veškerého vybavení na straně počítače jsem použil vývojové prostředí **C++ Builder 6.0** firmy Borland. Neměl by však být

problém použít i konkurenční vývojové prostředí pro C++ od firmy Microsoft. Na webových stránkách výrobce je ke stažení SDK mimo verze pro jazyk C++ i verze pro programovací jazyky rodiny .NET. V této práci se použitím .NET nebudu zabývat.

4.1 Univerzální ovladač CyUSB a CyAPI

Společnost Cypress dodává v SDK univerzální ovladač **CyUSB**, který dokáže obsloužit veškeré funkce řadiče. Výrobce počítá s využitím tohoto ovladače jako základního stavebního kamene pro všechna zařízení založená na EZ-USB. S ovladačem je zároveň dodáno i potřebné objektové rozhraní **CyAPI**, které umožňuje snadnou komunikaci se zařízením a díky kterému není nutné řešit práci s USB nízkoúrovňově. Přiložený ovladač je určen pro systémy Windows a nebyl problém s jeho použitím na 32bit systémech Windows XP a Windows Vista.

Aby bylo možné pracovat v prostředí C++ Builder s USB zařízením, je potřeba do zdrojového kódu přidat hlavičkový soubor API příkazem:

```
#include "CyAPI.h"
```

Dále je nutné do vytvořeného projektu přidat knihovnu CyAPI.lib pomocí *Project->Add to project…* v hlavní nabídce vývojového prostředí.

4.2 Inicializace USB zařízení

Pokud je k programu správně připojeno rozhraní CyAPI, je možné začít s inicializací komunikace s USB zařízením. Na začátku programu je vhodné vytvořit potřebné proměnné:

```
CCyUSBDevice  *USBDevice;
CCyBulkEndPoint *endpt = NULL;
```

CyAPI si vytváří vlastní datové typy reprezentující určité části USB zařízení. První řádek bude reprezentovat připojené zařízení a druhý bude později v programu použit pro endpoint EP2IN určený ve firmware v kapitole 3.3.

Proměnnou *USBDevice* je možné nastavit už na začátku programu, ale není to žádoucí v případě, že by uživatel mohl zařízení připojit až ve chvíli, kdy už program běží. Inicializační

příkazy je dobré vložit do samostatné funkce, která se může provést uživatelem zahájenou inicializací, nebo například před započetím přenosu. K propojení stačí do inicializační funkce přidat následující řádek:

```
USBDevice = new  CCyUSBDevice(NULL);
```

Následně jsou přístupné všechny vlastnosti a třídy tohoto objektu, tedy samotného USB zařízení. Vhodným začátkem je nyní zjištění počtu endpointů, neboť ty je potřeba v cyklu projít, aby bylo možné najít, který odpovídá požadované adrese EP2IN (0x82h). To zajistí tyto řádky kódu:

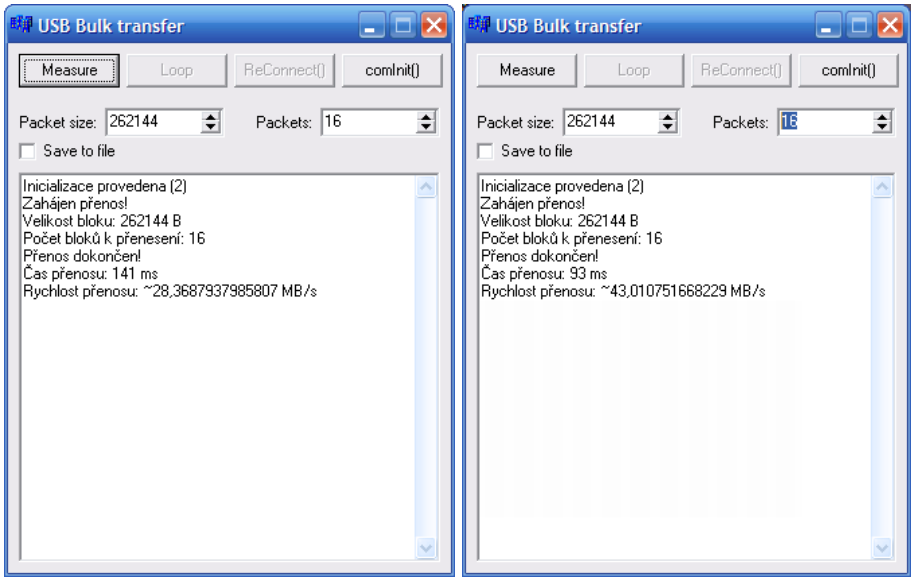
```
eptCount = USBDevice->EndPointCount();

for(int i=1;i<eptCount;i++)
{
  if(USBDevice->EndPoints[i]->Address==0x82)
  {
    endpt = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
  }
}
```

Prvním řádkem program zjistí počet endpointů zařízení včetně EP0. Následně v cyklu projde všechny endpointy a ptá se, zda odpovídají adresou endpointu EP2IN. Vhodný endpoint se přiřadí k proměnné *endpt*. Takto zinicializovaným endpointem je nyní možné přenášet data.

4.3 Nastavení přenosů a zátěž procesoru

Pro vhodné nastavení přenosů dat je potřeba správně pochopit, jak vlastní přenos funguje. V minulých kapitolách to bylo postupně vysvětlováno po stránce teorie. V této kapitole je ukázán praktický vliv nastavených hodnot.

Parametrem, který značně ovlivňuje výslednou rychlost, je délka jednotlivých přenosů. Je nutné volit kompromis mezi nízkou latencí a velkou rychlostí. V případě krátkých přenosů je USB řadič zatížen množstvím režijních paketů, což způsobuje vysokou zátěž procesoru. Standardně je nastavena velikost v řádu jednotek kilobajtů. V ideálním případě je pak možné dosáhnout maximálně rychlosti kolem 12 MB/s při 100% zatížení jednoho jádra CPU (měřeno s procesorem Intel Core Duo T2050 na 1,6 GHz).



Obr. 13 - Vytvořená aplikace na měření rychlosti přenosu (vlevo generátor hodinového signálu na 30 MHz, vpravo na 48 MHz)

Délka přenosu by měla být dělitelná osmi. Jak již bylo zmíněno, velké pakety jsou automaticky děleny, takže tento problém programátor nemusí nijak řešit. Pro optimální snížení režie je dobré použít přenosy délky alespoň 64 kB. Při délce 256 kB by neměl být problém na moderním počítači dosáhnout maximální rychlosti. U této aplikace je prioritou maximální rychlost, tudíž je zvolena délka přenosu právě 256 kB příkazem:

```
endpt->SetXferSize(262144); // velikost v Bajtech
```

Při této délce přenosů je i při maximální rychlosti zátěž moderních procesorů kolem 30%. Pokud je zařízení navrženo jen pro přenos jednoho typu dat, je vhodné tento řádek přidat na konec inicializační sekvence.

4.4 Přenos dat s využitím funkce XferData

Přenosy oběma směry se provádí pomocí funkce XferData volané následujícím způsobem:

```
endpt->XferData(dataBuf,lenF);
```

Proměnná *lenF* určuje požadovanou velikost přenášených dat a *dataBuf* představuje prostor, kam budou data nahrána (vstupní endpoint), případně odeslána (výstupní endpoint). Velikost přenášených dat nemusí odpovídat délce přenosu nastavené v kapitole 4.3. Ovladač i

na této vyšší úrovni automaticky dělí přenášená data na délku přenosu. Nejlepší však je zvolit obě hodnoty stejně.

V případě synchronního přenosu nastaveného pomocí popisovaného firmware se do endpointu neustále přesouvají data z FIFO paměti. Funkci *XferData* lze tedy volat opakovaně ve smyčce. Ovladač vždy čeká, až je endpoint naplněn, takže další přijatá data vždy pokračují až tam, kde předchozí přenos skončil. Data je nutné odebírat dostatečnou rychlostí, než se zaplní všechny buffery. V tomto případě jsou nastaveny (dle kapitoly 3.3) 4 buffery o velikosti 512 B (maximum pro bulk přenos).

4.5 Výsledky, dosažené rychlosti přenosu

Dosažené výsledky pro mě byly příjemným překvapením. Ukázalo se, že s mnou zvoleným nastavením není problém přenášet data rychlostí 28,6 MB/s (30 MHz). Z toho vyplývá, že se přenáší všechna data snímaná z FIFO bez ztráty jediného paketu. Zde je vhodné upozornit, že pro synchronní přenos byla rychlost přenosu při nastavení synchronního přenosu v práci Bc. Pavla Součka jen 14,0 MB/s. Nastavení popsané v mé práci se liší především ve dvou parametrech. EP2 využívá dvakrát většího množství bufferů, což poskytuje více času, než se data zahodí, v případě, že systém v nevhodnou chvíli přiřadí procesorový čas jiné aplikaci. Podstatnější změnou však je nastavení vhodnější délky přenosu. Toto nastavení v manuálech k řadiči není dostatečně vysvětleno, takže bylo nutné bádat na zahraničních internetových diskuzích.

Vyzkoušel jsem rychlost přenosu při generátoru hodinového signálu nastaveném na 48 MHz. Při tomto nastavení bylo dosaženo rychlosti 43 MB/s. Pro kontinuální snímání už tato frekvence nejde použít, ale pokud by se použil vhodný externí signál například na 40 MHz, bylo by možné dosáhnout velmi dobrých rychlostí při zachování přenosu všech paketů.

Rychlost 43 MB/s není možné přesáhnout, a jedná se tedy o maximum. Předpokládám, že problém není v samotném řadiči ale v implementaci hostitelského USB řadiče. Nejlepší volbou pro rychlý přenos jsou hostitelské řadiče společnosti Intel v aktuálně používaných

čipových sadách Intel 96x a Series 4 (ICH-8, ICH-9). Při testu se starším řadičem v ICH-7M (čipová sada Intel 94x) nebylo možné překročit rychlost 35 MB/s, která je charakteristická pro více starších hostitelských řadičů. Nejhorší situace se ukázala u společnosti ATI/AMD, kde vinou implementace v čipových sadách Xpress 200(M)/11x0(M)/12x0(M) není možné dosáhnout s jakýmkoli zařízením rychlosti vyšší než 25–30 MB/s. O chybě řadičů AMD ví, ale chyba byla zakotvena příliš hluboko v návrhu, takže ji nikdy neopravilo. Nové řadiče ATI SB600 v základních deskách z roku 2008 tímto problémem už netrpí.

5 Využití EZ-USB jako logického analyzátoru

S využitím USB řadiče EZ-USB se není nutné omezovat jen na přenos obrazu, potažmo videa. Jeho rychlý přenos lze použít relativně snadno pro potřeby jednoduchého logického analyzátoru schopného pracovat i na frekvencích kolem 40 MHz s 8 kanály (případně 20 MHz s 16 kanály). Proti komerčním zařízením však zůstává nízká cena (~1000,- Kč), díky které může jít o vhodnou pomůcku pro výuku.

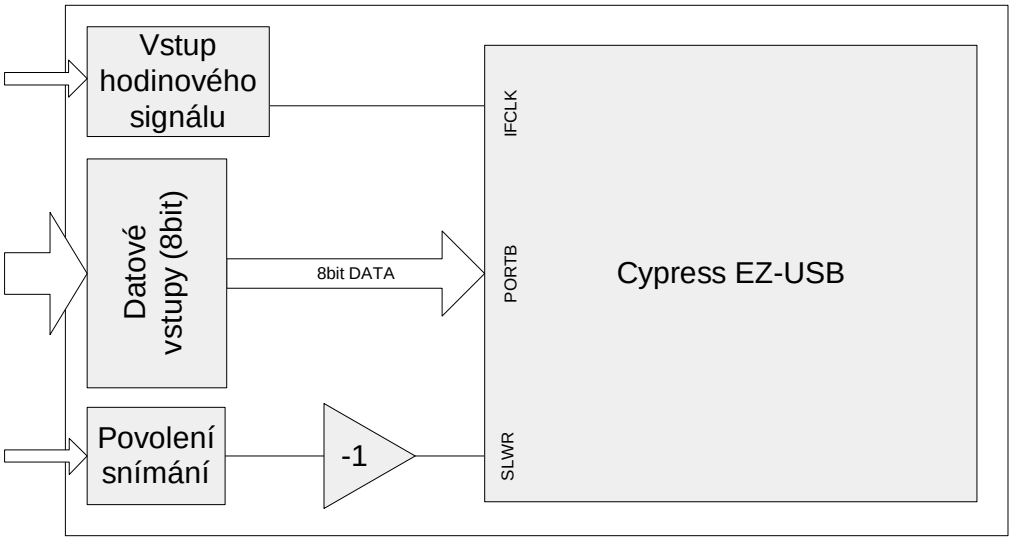
Tuto kapitolu jsem se rozhodl zařadit v práci před přenos videa, neboť je mnohem blíže přenosu obecných dat a v některých ohledech bych se musel tématicky vracet.

5.1 Možnosti zapojení obvodu a úpravy firmware

Při návrhu je potřeba brát v potaz jeden důležitý fakt: EZ-USB je rychlé jen za předpokladu, že mikrokontrolér nezpracovává přijímaná data. Řízení FIFO sběrnice se musí přenechat jednotce GPIF, která nabízí mnohem méně možností nastavení. Lze změnit frekvenci zápisu dat do FIFO a určit, kdy se má signál snímat. K prvnímu je nutné využít jednu ze dvou frekvencí (30 MHz, 48 MHz) generátoru hodinového signálu přímo v řadiči, nebo zvolit externí generátor (vývod IFCLK), který musí mít vlastní frekvenci v rozmezí 5–48 MHz. Druhé se řeší hodnotou signálu SLWR (Obr. 15). Pokud je vývod připojen na zem, běží snímání neustále.

Pro potřeby této části práce byla vytvořena jednoduchá aplikace schopná ukládat měřené průběhy rovnou na disk. Mimo to umožňuje z měřeného průběhu zobrazit graficky jeho

počátek (maximálně 200 vzorků). Celá aplikace je stejně jako firmware navržena pro práci s 8 kanály, ale k přepracování na obsluhu 16 kanálů stačí pracovat s lichými bajty jako první polovinou kanálů a se sudými bajty jako druhou polovinou kanálů. V aplikaci je i několik základních funkcí. Je možné použít jako hodinový signál některý z kanálů a aplikace následně ořízne nepotřebné a opakující se nadbytečné hodnoty. Poslední funkcí je zobrazení hodnoty bitů v reálném čase pro pomalé průběhy. Celou aplikaci lze nalézt na přiloženém CD a v adresáři *USB-LA*.



Obr. 14 - Blokové schéma logického analyzátoru s využitím externího hodinového signálu a řízení snímání dle SLWR

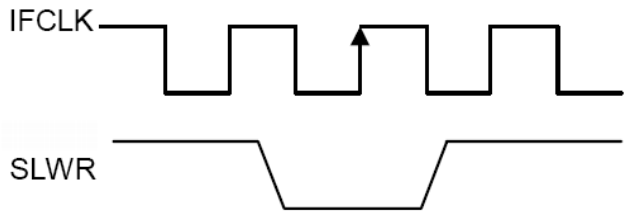
Možné pokročilejší zapojení ukazuje Obr. 14. Řízení signálu SLWR zde má velký význam v případě, že je potřeba odebrat průběh jen v určitém časovém úseku. Po celou dobu takového úseku nesmí být signál aktivní. Takové řešení může být vhodné v laboratořích při práci s mikrokontroléry. Student může vyhradit jeden programovatelný vývod k propojení se SLWR a ovládat jeho hodnotu příkazy, které budou ohraničovat část kódu, kde je měření zamýšleno. Časování FIFO pamětí by bylo v takovém případě pro nejefektivnější funkčnost řízeno generátorem hodinového signálu měřeného zařízení. V původním firmware stačí upravit pouze nastavení tohoto signálu příkazem:

```
IFCONFIG = 0x03;  // 00000011
```

5.2 Zpracování dat v reálném čase

Veškeré zpracování dat se může vykonávat až na straně počítače, což frekvencích 20–40 MHz vytváří větší nároky na procesorový čas už jen z hlediska samotného přenosu. Další zpracování dat za běhu tudíž přináší mnoho omezení. V žádném případě není možné procházet všechny naměřené vzorky ve chvíli, kdy se přenesou.

Pokud se ví, že nějaký důležitý signál bude mít určitou hodnotu delší dobu, je možné procházet jen některé vzorky v takovém intervalu, aby bylo jisté, že se potřebný signál zachytí. S počtem vynechaných vzorků se snižují nároky na čas procesoru. Aby byl procesor Intel Core Duo ~1,6 GHz schopen zpracovat data při hodinovém signálu 30 MHz, je potřeba vynechávat kolem 100 vzorků. Tím se reálná frekvence snímání snižuje na 300 kHz. Do mé aplikace jsem pro velmi pomalé signály přidal možnost zobrazení hodnoty jednotlivých kanálů přímo při měření. Vzhledem k omezením rychlosti grafického rozhraní Windows je možné maximálně (standardními prostředky) dosáhnout jen řádu desítek až stovek Hz.

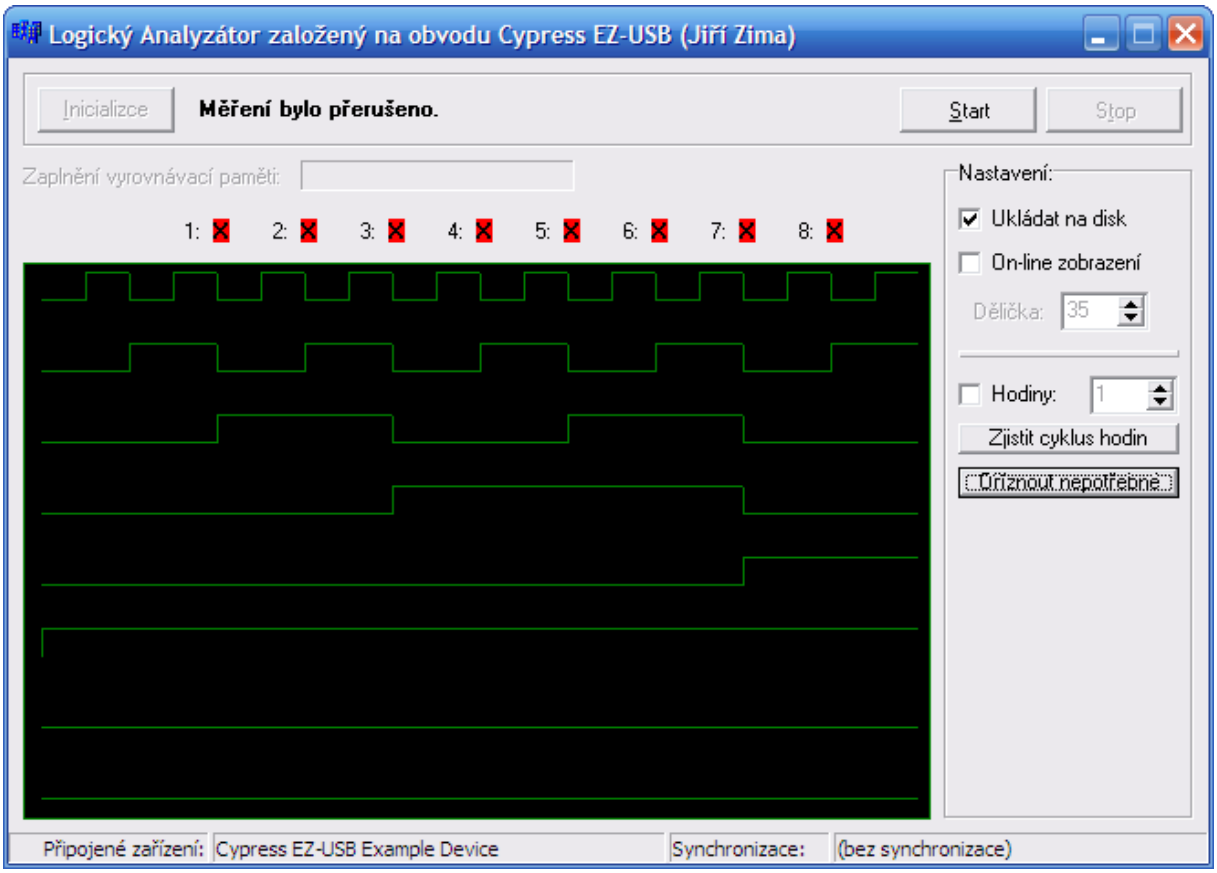


Obr. 15 - Synchronní mód zápisu do FIFO paměti (zápis řízen signálem SLWR)

5.3 Zpracování dat po ukončení měření

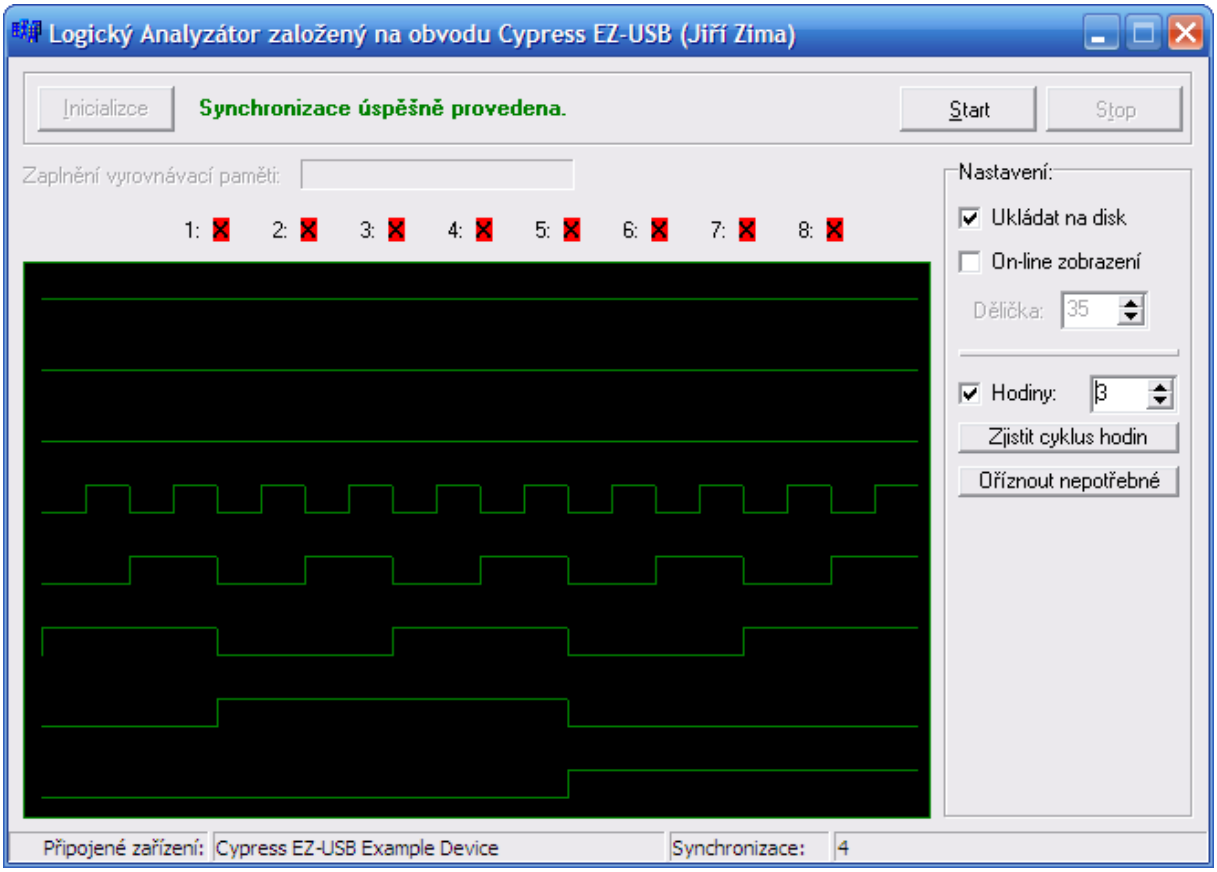
Mnohem efektivnější možností je zpracování dat až po jejich doměření. Spuštění a ukončení měření lze řídit signálem SLRW (ovlivnění zápisu do paměti signálem SLRW ukazuje Obr. 15) dle zapojení na Obr. 14, programově, nebo kombinací obou způsobů. Při kombinaci je možné sledovat v intervalech konkrétní signály a v případě splnění definované podmínky začít zaznamenávaná data ukládat do paměti (případně na disk) po určitý čas a následně provést jejich zpracování.

Jednoduché zpracování dat je předvedeno ve vytvořené aplikaci (adresář na CD: *\USB-LA*). Pokud se aktivuje přepínač *Ukládat na disk*, lze tlačítkem *Zobrazit průběh* vykreslit část změřených dat. Program zobrazuje standardně data ze začátku měření.



Obr. 16 - Snímání s frekvencí interního generátoru hodinového signálu 30 MHz

Funkce *Zjistit cyklus hodin* na zvoleném kanálu testuje periodu opakování signálu. V případě, že detekuje vhodný (stálý) hodinový signál zobrazí pouze vzorky s náběžnou hranou tohoto signálu. Tento způsob řešení může být vhodný při měření na zařízení s řádově nižší rychlostí v případě, že logický analyzátor nemůže využívat jeho hodinový signál. Zobrazený průběh se tak zbaví duplicitních hodnot. Tuto funkci názorně ukazuje výstup programu na Obr. 17 (jde o stejný průběh jako na Obr. 16).



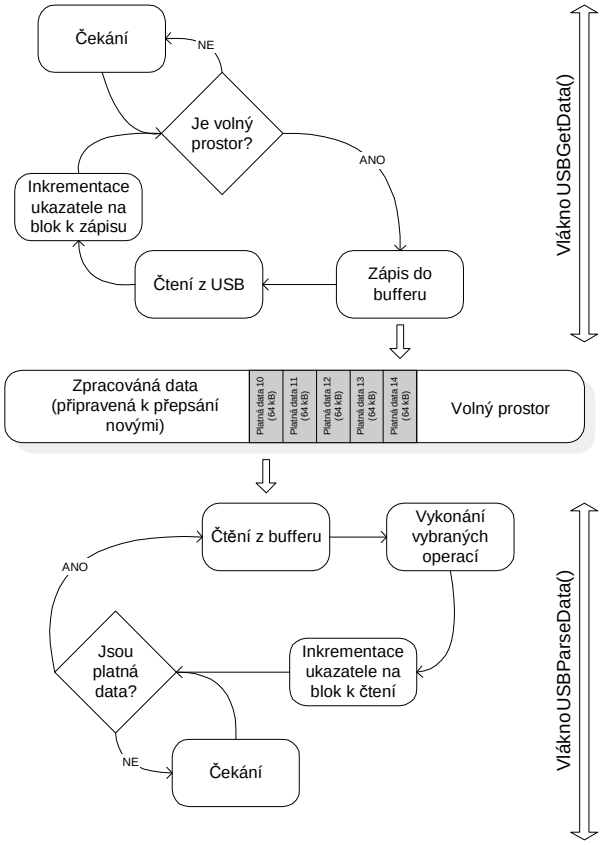
Obr. 17 - Programové určení hodinového signálu vybraným (3) vstupním datovým vodičem

5.4 Rozdělení zátěže na více jader procesoru

Narůstající počet vícejádrových procesorů na trhu značně zvyšuje možnosti celého popisovaného řešení. V dnešní době je již drtivá většina nových stolních počítačů a notebooků vybavena dvoujádrovými procesory. Do stolních počítačů jsou dokonce cenově velmi dostupná řešení procesorů s čtyřmi jádry. Proto byla aplikace naprogramována ve více vláknech, aby se využilo maximum výkonu celého procesoru.

Na Obr. 18 je ukázáno rozdělení programu na dvě výpočetně náročná vlákna. Hlavní vlákno řeší obsluhu grafického rozhraní a ovládání jednotlivých funkčních tlačítek – toto vlákno není nijak procesorově náročné a jeho oddělení od výpočetně náročných částí přináší výhody zejména v kratší odezvě tlačítek během měření (týká se i procesorů s jedním jádrem). V případě započetí měření se vytvoří další dvě vlákna. Jedno obsluhuje USB řadič a spouští

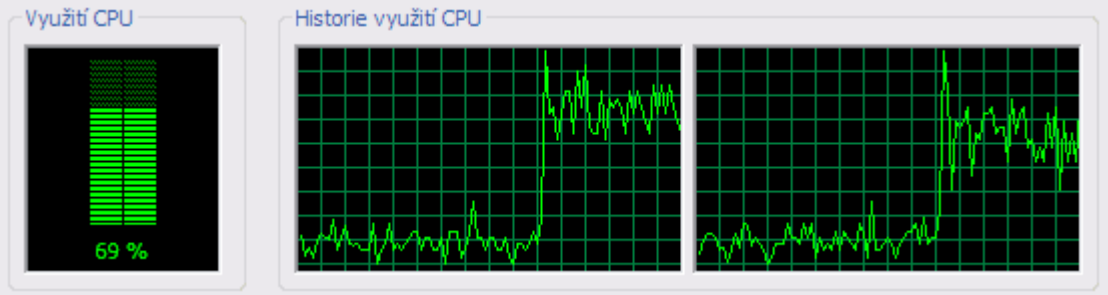
funkci *USBGetData()*, druhé řeší veškeré další zpracování dat a spouští funkci *USBParseData()*.



Obr. 18 - Oddělení komunikace s USB zařízením a zpracování přenesených dat

Funkce *USBGetData()* provádí opakovaně čtení z endpointu EP2 a data ukládá do programem vytvořené vyrovnávací paměti. Voláním funkcí CyAPI na sebe přebírá veškerou zátěž procesoru spojenou s USB přenosy.

Funkce *USBParseData()* postupně tahá data z vyrovnávací paměti a provádí na nich příslušné operace (implementováno pouze zobrazení aktuální hodnoty v reálném čase a ukládání na disk). Pokud je vybráno zobrazování aktuální hodnoty v reálném čase a zvolí se velmi krátký interval aktualizace zobrazení, je znatelné zatížení procesoru tímto vláken. Takové nastavení posloužilo k testování zatížení procesoru na jednotlivých jádrech. Na Obr. 19 je ukázáno, jak systém automaticky rozdělil obě procesorově náročná vlákna na samostatná jádra procesoru.



Obr. 19 - Rozložení výpočetně náročných vláken logického analyzátoru na obě jádra procesoru (měření na procesoru Core 2 Duo @ 1 GHz)

Aktuální dělení umí využít maximálně dvou jader procesoru, neboť výpočetně náročná vlákna jsou za každých okolností vždy jen dvě. Návrh programu by bylo možné vhodně změnit a použít na každou funkci zpracování výsledků samostatné vlákno. Tento způsob by šel zkombinovat také s dělením těch nejnáročnějších funkcí na více vláken ve smyslu paralelního zpracování více (několika za sebou) vzorků najednou.

V současné době tento způsob ještě nejspíše nemá takový smysl, ale brzy se mají objevit na trhu 8jádrové procesory pro osobní počítače a v budoucích letech má Intel plány na použití až 64 (pomalejších) jader v jednom procesoru.

5.5 Výsledky a možnosti nasazení

Při realizaci této části práce se ukázalo, že řadič EZ-USB může bez problému sloužit jako rychlý „sběrač“ dat do frekvence 40 MHz pro realizaci logického analyzátoru s výpočetní jednotkou v podobě vlastního počítače. Toto řešení má úzké hrdlo pouze v rychlosti procesoru a pevného disku. Omezení daná procesorem je možné obejít zpracováním dat až po ukončení měření (kapitola 5.3) nebo případně snahou využít jej co nejefektivněji (kapitola 5.4). Pevný disk je brzdou spíše u notebooků, kde se používají menší disky aktuálně s rychlostí 30–60 MB/s (dle pozice hlavy na plotně). Není však problém nalézt i starší notebooky vybavené USB 2.0 a diskem s rychlostí 15–25 MB/s. Samotný disk však není velká investice a v případě počítačů jsou většinou hodnoty rychlosti o 30–50% vyšší.

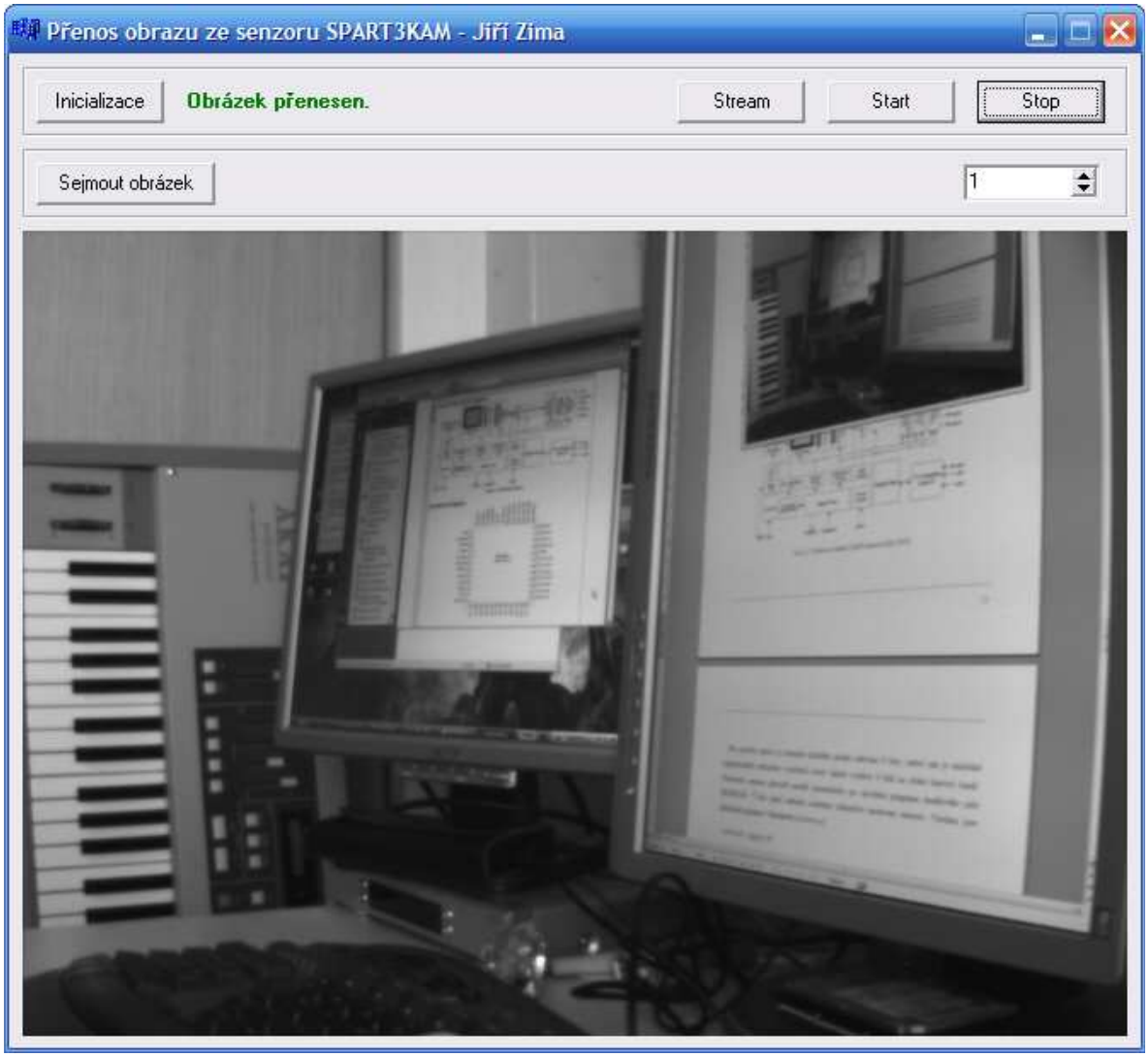
S úspěšnou aplikací logického analyzátoru se zároveň nabízí další dvě užití – osciloskop a programovatelný generátor průběhů. V případě osciloskopu by stačilo před vstupy FIFO

umístit ochranné obvody a analogově-digitální převodníky. Výsledkem by byl osciloskop s frekvencí až 40 MHz v případě jednoho kanálu a 20 MHz v případě kanálů dvou. U generátoru by se místo vstupního endpointu použil výstupní se stejnými parametry. Omezení generátoru by pouze udávala složitost simulované funkce a schopnost procesoru rychle generovat potřebné vzorky. Pokud by se však i složitější funkce nasimulovaly dopředu a následně se v nich prováděly jen jednoduché početní operace, omezení by nebyla nijak drastická.

6 Přenos videa ze senzoru SPART3KAM

Výměnou za poskytnuté informace o nastavení pro rychlý přenos a také z časových důvodů mi byl pro potřeby práce zapůjčen Ing. Jaroslavem Třeštíkem měřicí modul vybavený VGA CMOS senzorem společnosti Kodak, hradlovým polem Spartan a nejmenší variantou řadiče EZ-USB (56 vývodů). V této části práce rozeberu zapojení a princip celého měřicího modulu a proberu několik klíčových bodů ohledně zpracování videa následně v počítači. Pro tyto účely vznikla testovací aplikace (Obr. 20), kterou lze nalézt na přiloženém CD v adresáři *\VideoStreaming*.

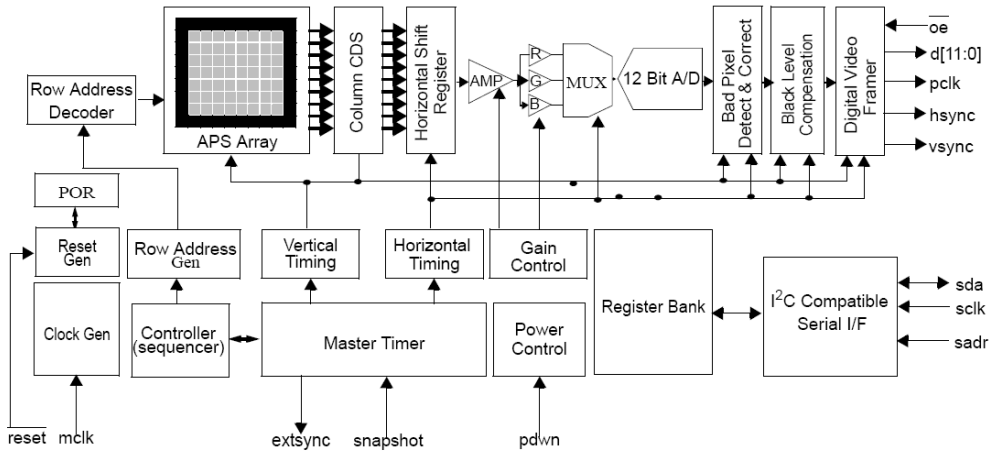
Na závěr v této části práce proberu možnosti použití EZ-USB jakožto řídící jednotky pro CMOS senzor.



Obr. 20 - Vzhled vytvořené aplikace

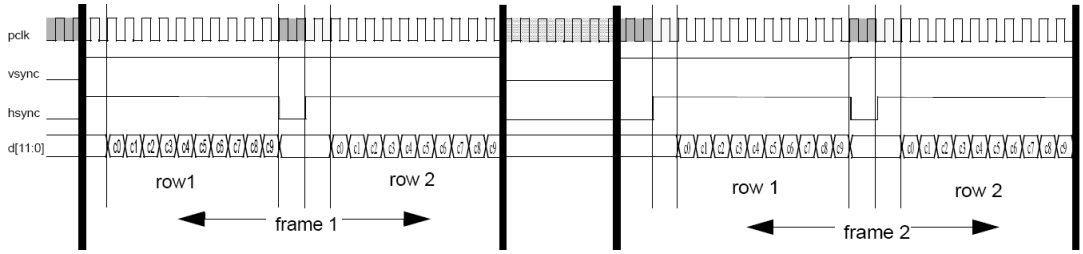
6.1 CMOS senzor KODAK KAC-9618

KAC-9618 je 48vývodovým monochromatickým obrazovým snímačem se snímací plochou jedné třetiny palce. Mřížka senzoru má rozlišení 664x504, přičemž je aktivních 648x488 bodů. Snímač je schopen v takovém rozlišení generovat 30 snímků za sekundu (fps) při až 12bit hloubce každého snímaného bodu. Blokové schéma senzoru ukazuje Obr. 21.



Obr. 21 - Blokové schéma CMOS senzoru KAC-9618

Pro potřeby práce je intenzita každého pixelu udávána 8 bity, neboť tak je následně nejjednodušší zobrazení v počítači, který taktéž využívá 8 bitů na jeden barevný kanál. Nastavení senzoru provádí modul automaticky po zavedení programu hradlového pole SPARTAN. V této práci nebudu rozebírat jednotlivá nastavení senzoru. Všechna jsou přehledně popsána v datasheetu [4].

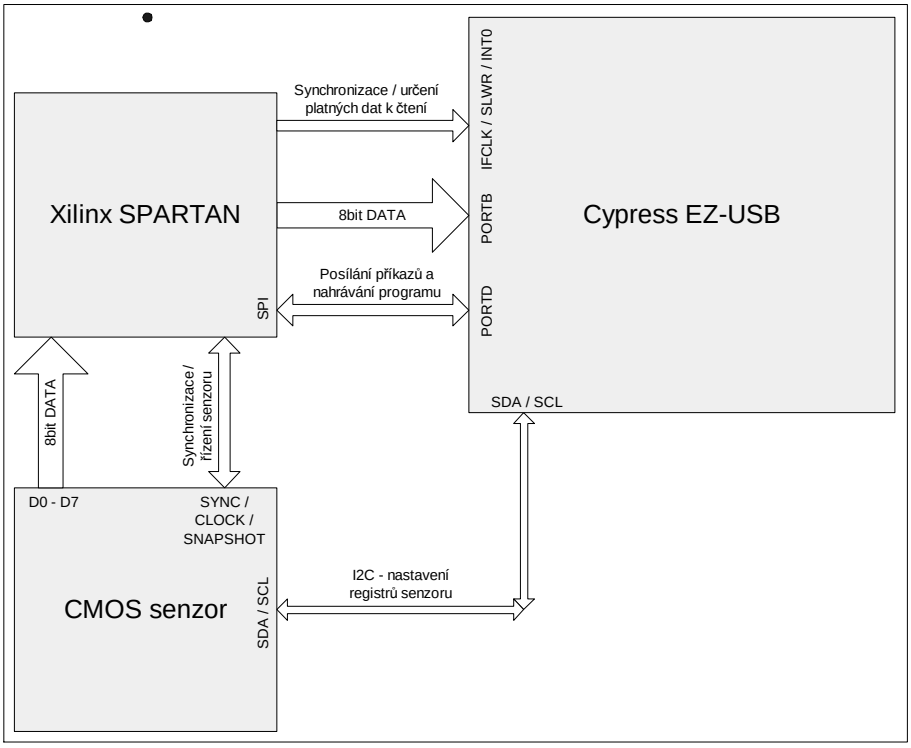


Obr. 22 - Průběh synchronizačních pulzů (světle šedá – synchronizace prvního pixelu řádku, šedá – prodleva mezi snímky, tmavě šedá – prodleva mezi řádky)

Přenos obrazu ze senzoru vyžaduje dostatečně vysokou rychlost přenosu po celou dobu snímání. Senzor nemá žádnou snímkovou paměť a jediná vyrovnávací paměť (2 kB) je tvořena pamětí vyhrazenou pro endpoint na řadiči EZ-USB. Pokud by počítač nestačil dostatečně rychle přenášet data, snímané pixely by nebylo možné uložit a přenos obrazu by se nezdařil. Mimo přenosu dat je také nutné zajistit generování vhodných a především včasných signálů pro synchronizaci obrazu. Průběh těchto signálů pro použité progresivní snímkování (snímek se přenáší celý) ukazuje Obr. 22.

6.2 Zapojení hradlového pole Xilinx SPARTAN XCS200

Programovatelné hradlové pole SPARTAN je umístěno mezi CMOS senzorem a USB řadičem (Obr. 23). Jeho úlohou je generování řídících signálů pro snazší přenos do USB řadiče a k ovládání senzoru. Použití hradlového pole usnadňuje práci USB řadiči, který dostává čistá obrazová data, která je možné v nezměněné podobě přenášet do PC a v něm následně bez složitých úprav rovnou vykreslit.



Obr. 23 - Blokové schéma zapojení součástí modulu SPART3KAM

6.3 Důležité změny ve firmware

K měřicímu modulu byla Ing. Jaroslavem Třeštíkem dodána i speciální aplikace, která celý modul programuje, protože firmware v USB řadiči musí řešit nastavení všech ostatních čipů a ve speciálním módu programuje hradlové pole SPARTAN k požadované funkci. Vlastní nastavení pro rychlý přenos dat přes USB se však neliší a neliší se také nastavení k tomu určeného endpointu EP2IN.

Hlavní rozdíl z hlediska komunikace spočívá v přidání dalších dvou endpointů typu bulk – EP1IN a EP1OUT. Velikost těchto endpointů je omezena na 64 B pro každý (vysvětlení: kapitola 3.3) a jsou určeny pro posílání příkazů do zařízení a čtení jeho odpovědí. Použitím endpointů typu bulk je zachováno stejné ovládání přenosů jako pro endpoint EP2 (vysvětlené v kapitole 4.4).

Zda se stihl přenést celý obrázek, je možné kontrolovat pomocí FIFO Flag příznaků, s jejichž pomocí je možné zjistit, která část dat v rámci jednoho snímku se ztratila. V případě, že se aplikaci v počítači nebude snažit zobrazit co nejvíce z neúplného obrázku, lze využít mnohem snazší metody (ta je použita i zde). Aplikace očekává, že proběhne přenos o velikosti kompletního obrázku, takže v případě ztráty libovolného množství dat, nahlásí funkce pro přenos (*XferData*) chybu a přijatá data se zahodí. Pokud je obraz snímán o rychlosti alespoň 20 snímku za sekundu, uživatel si nevšimne, že předchozí obrázek byl zobrazen po dobu dvakrát delší, než je standardní.

Zpracování příkazu od PC aplikace

Aby mohl USB řadič okamžitě zpracovávat přijaté příkazy přes EP1OUT, využívá se přerušení pro tento endpoint. To se aktivuje při startu firmware příkazem:

```
EPIE |= bmBIT3; // 3. bit = EP1OUT
```

V 8bit registru EPIE každý jednotlivý bit povoluje nějaké přerušení. V případě, že pro daný bit není přerušení povoleno, není možné ho v programu žádným způsobem vyvolat.

Jakmile PC aplikace vyšle příkaz na EP1OUT, v zařízení se zavolá funkce *ISR_Ep1out()*. Pro komunikaci jsou použity příkazy definované jedním znakem. Další volitelné znaky slouží jako parametry daného příkazu. Tento kód ukazuje obsloužení přerušení:

```
void ISR_Ep1out(void) interrupt 0
{
  FIFORESET = 0x02; // obsah EP1OUT ovládá procesor
  SYNCDELAY;
  switch (EP1OUTBUF[0]) // přečtení prvního znaku přenosu
  {
    /** bloky kódu pro písmena příkazů **/
  }
  /** pripraveni EP na další přerušení **/
```

```
    FIFORESET = 0x01;
    EP1OUTBC = 0;
    EZUSB_IRQ_CLEAR();
    EPIRQ = bmBIT3;
  }
```

USB řadič dále může odeslat zprávu o úspěšném provedení příkazu pomocí EP1IN například tímto způsobem:

```
  EP1INBUF[0]='O';
  EP1INBUF[1]='K';
  EP1INBUF[2]=0xd;
  EP1INBC = 3; // počet bajtů k přenesení (automaticky se provede)
```

Ovládání senzoru

Nastavení parametrů senzoru se dle Obr. 23 provádí pomocí **I2C** sběrnice (funkce *I2CReadByte* a *I2CSendbyte*), která spojuje senzor a USB řadič. Přenos obrazu využívá pouze prvních 8 datových vodičů FIFO sběrnice (**PORTB**). Komunikace s hradlovým polem je řešena programově realizovanou **SPI** sběrnicí na nevyužitých vodičích (**PORTD**; zbylých 8 datových vodičů FIFO).

Část programu ukazující kompletní obsluhu přerušení EP1OUT je na přiloženém CD v adresáři *\Trestik_SPART3KAM*.

6.4 Nastavení endpointu pro přenos obrazu

Způsobem popsaným v kapitole 3.3 je nutné inicializovat další dva endpointy. Pro EP1OUT je adresa 0x01h a pro EP1IN 0x81h. EP2IN jsem ve své aplikaci pro přenos obrazu nazval *streamEndpt*. Proti nastavení z kapitoly 4.3 je potřeba změnit velikost přenosu přesně na velikost jednoho přeneseného snímku. Tím se zajistí minimální režie procesoru v počítači, a tudíž i největší rychlost. Inicializace EP2 by měla vypadat takto:

```
  streamEndpt->SetXferSize(FRAME_SIZE); // FRAME_SIZE = 318720 (664 x 480)
  streamEndpt->TimeOut = 300;
```

Druhý řádek slouží k nastavení doby čekání na dokončení přenosu. Na přenos se čeká zejména, pokud nedorazí kompletní snímek. V kapitole 6.3 jsem zmínil, že v tomto případě se

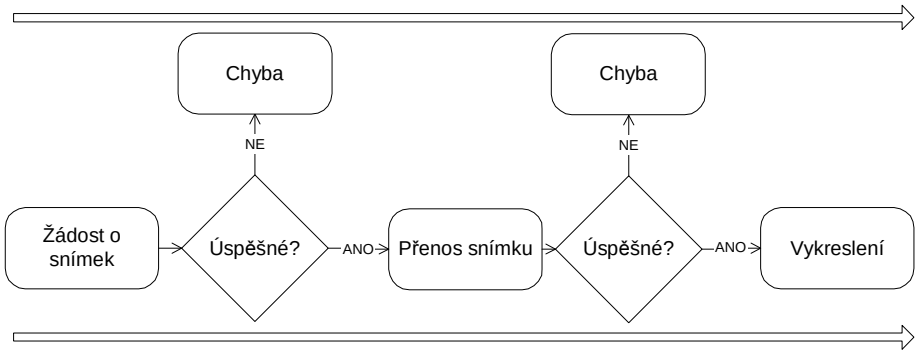
nekompletní snímky zahazují. Je tedy dobré nastavit menší hodnotu čekání (standardně jsou 2 sekundy), aby se přenos dalších zbytečně nezdržoval.

6.5 Přenos snímků

Pro přenos jednoho snímku jsem vytvořil funkci *getFrame()*, která obsluhuje vyslání příkazu na EP1OUT a následný přesun dat z EP2IN. Její průběh popisuje diagram na Obr. 24.

K odeslání dat se používá opět funkce *XferData*. Takto vypadá kód odeslání příkazu pro příjem jednoho snímku:

```
unsigned char zprava[6] = "SR    ";  // žádost o snímek
zprava[2] = 0; zprava[3] = 0;  // dva bajty adresa
zprava[4] = 0; zprava[5] = 1;  // dva bajty data
controlEndpt->XferData(zprava,6)
```



Obr. 24 - Funkce getFrame()

Funkci *getFrame()* lze využít také pro snímání videa, což je v ukázkové aplikaci implementováno a lze to vyzkoušet tlačítky *Start* a *Stop*. Číslem pod těmito tlačítky je nastaveno, jak často se má funkce volat. Tímto způsobem je možné přenášet video v plné rychlosti snímače (30 fps), pokud počítač dostatečně rychle zpracuje vykreslení obrazu. Pokud se mu to nepodaří, musí každý druhý snímek vynechávat (15 fps). Přenos jednotlivých paketů ukazuje výstup programu USBTrace na Obr. 25.

Seq	Type	Time	Request	I/O	Device Object	IRP	Status	Data ...
3	URB	16:19:36:796	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x856E0858	STATUS_PENDING	0
4	URB	16:19:36:796	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86FF7008	STATUS_SUCCESS	0
5	URB	16:19:36:796	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-11	0x86FF7008	STATUS_SUCCESS	0
6	URB	16:19:36:796	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-2	0x86FF7008	STATUS_SUCCESS	0
7	URB	16:19:36:796	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86E2F7D8	STATUS_PENDING	0
8	URB	16:19:36:843	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86FF7008	STATUS_SUCCESS	318720
9	URB	16:19:36:843	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-11	0x870155A8	STATUS_SUCCESS	6
10	URB	16:19:36:843	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-2	0x870155A8	STATUS_SUCCESS	6
11	URB	16:19:36:843	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86E2F7D8	STATUS_PENDING	0
12	URB	16:19:36:859	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x870155A8	STATUS_SUCCESS	0
13	URB	16:19:36:859	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-11	0x870155A8	STATUS_SUCCESS	0
14	URB	16:19:36:859	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-2	0x870155A8	STATUS_SUCCESS	0
15	URB	16:19:36:859	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86E2F7D8	STATUS_PENDING	0
16	URB	16:19:36:906	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x870155A8	STATUS_SUCCESS	318720
17	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-11	0x8700C4C8	STATUS_SUCCESS	6
18	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-2	0x8700C4C8	STATUS_SUCCESS	6
19	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86E2F7D8	STATUS_PENDING	0
20	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x8700C4C8	STATUS_SUCCESS	0
21	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-11	0x86E31C70	STATUS_SUCCESS	0
22	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	OUT	\Device\USBPDO-2	0x86E31C70	STATUS_SUCCESS	0
23	URB	16:19:36:921	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86E2F7D8	STATUS_PENDING	0
24	URB	16:19:36:968	BULK_OR_INTERRUPT_TRANSFER	IN	\Device\USBPDO-2	0x86E31C70	STATUS_SUCCESS	318720

Obr. 25 - Výstup programu USBTrace ukazující přenosy na sběrnici (pakety s délkou 6 B jsou žádostí o snímek, který se následně přenáší paketem s délkou 312 kB)

Měřicí modul umožňuje snímat i bez nutnosti žádat o každý snímek. To se provádí příkazem „Z“ a od jeho poslání se snímá video neustále na plné rychlosti.

6.6 Vykreslení snímku

K vykreslení přenesených snímků jsem použil základních funkcí **Windows GDI** implementovaných ve vývojovém prostředí C++ Builder. Vykreslení probíhá na objekt typu Tbitmap:

```
Graphics::TBitmap *pBitmap = new Graphics::TBitmap();
```

Vykreslit desítky snímků ve VGA rozlišení vyžaduje určité optimalizace. Vzhledem k tomu, že v přenášeném obrazu je intenzita každého pixelu reprezentována 8 bity, je nejlepší pracovat i při vykreslení s 8bit formátem. U tohoto formátu se musí používat uživatelsky definovaná paleta, aby bylo možné zobrazit snímek bez degradace kvality zobrazení. Paletu stupňů šedi vytváří tento kód:

```
LOGPALETTE* pal = (LOGPALETTE*) malloc( sizeof(LOGPALETTE) + \
        sizeof(PALETTEENTRY) * 256);
pal->palVersion = 0x300;
pal->palNumEntries = 256;
for(short i = 0 ; i < 256 ; i++)
{
```

```
    pal->palPalEntry[i].peGreen = (Byte) i;
    pal->palPalEntry[i].peRed = pal->palPalEntry[i].peGreen;
    pal->palPalEntry[i].peBlue = pal->palPalEntry[i].peGreen;
}
HPALETTE hpal = CreatePalette(pal);
```

Jakmile je snímek přesunut do počítače, lze ho vykreslit následujícím způsobem:

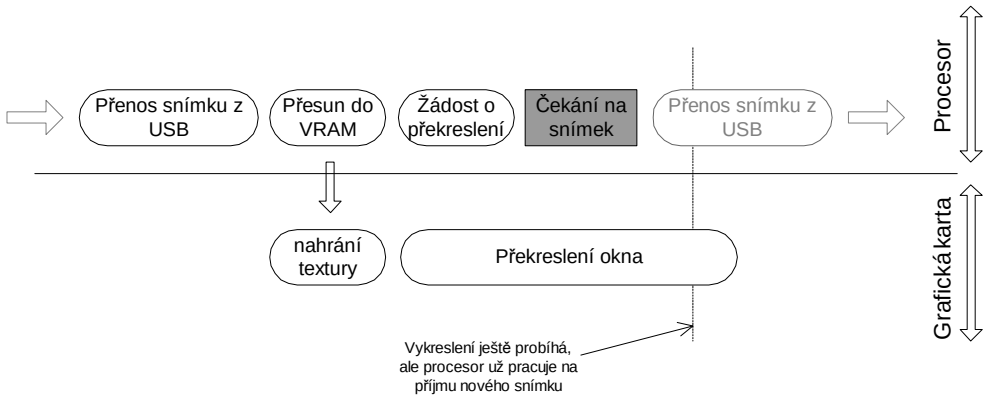
```
Byte * pointer;
for(int row = 0; row < 472; row++) //postupně prochází viditelné řádky
{
    pointer = (Byte *)pBitmap->ScanLine[row];
    //memcpy(pointer,frameDump+row*664,664);
    memcpy(pointer,5319+frameDump+row*664,648); //přenos řádku do bitmapy
}
MainForm->OutputImage->Picture->Assign(pBitmap); //vykreslení
```

Vykreslování pomocí Windows GDI je nejjednodušším možným způsobem vykreslení, má ovšem hned dvě velké nevýhody: obraz není možné rychle vykreslovat v jiném než původním rozlišení a vykreslení je řízeno procesorem. Pokud není dostatečně rychlý procesor, vykreslení snímku trvá déle než doba mezi dvěma přenosy při rychlosti snímání 30 fps.

6.7 Akcelerace vykreslování grafickou kartou

Zařízení procesoru při vykreslování lze řádově snížit přenesením tohoto úkonu na grafickou kartu. Grafická karta nabízí dva způsoby akcelerace: pomocí **video overlay** a s využitím **3D API**. První možnost byla implementována do většiny grafických karet v roce 1998 (počítače s Pentium II procesory) a není již dnes doporučována od příchodu Windows Vista jehož grafické jádro vykresluje vše pomocí 3D funkcí grafické karty.

Pro operační systémy Windows se momentálně používají pro akceleraci 3D grafiky rozhraní Direct3D a OpenGL. Použití aktuální verze Direct3D je však v C++ Builderu velmi komplikované a vestavěná 8 let stará verze není vhodná z důvodu možný problémů s kompatibilitou. Do testovacího programu jsem chtěl implementovat akceleraci pomocí rozhraní OpenGL, ale z časových důvodů to nebylo možné. Tuto metodu jsem již dříve s úspěchem použil v dřívějších projektech a rychlost vykreslení se znatelně zvýšila. Zjednodušený postup vykonávání programu na Obr. 26 ukazuje výhodu vykreslování s pomocí grafické karty.



Obr. 26 - Průběh zpracování videa s použitím akcelerace grafickou kartou

Vykreslování je nutné nastavit do ortogonálního zobrazení, čímž se eliminuje transformace bodů vlivem perspektivy. Následně stačí vytvořit čtyřúhelník (polygon) přes celý prostor okna, do kterého lze kreslit (viewport). Při každém obdržení nového snímku se pošlou data do paměti grafické karty a přiřadí se jako textura vytvořenému čtyřúhelníku. Následně stačí zavolat překreslení obrazu (resp. viewportu).

Hlavní výhodou řešení je rozdělení zátěže mezi grafickou kartu a procesor. Na rozdíl od standardního vykreslování v prostředí C++ Builder je výrazně snazší oddělení do vlastního vlákna. Pak se již procesor během vykreslování může plně věnovat příjmu nových dat z USB. Rozdělení na více vláken by ovšem nemělo být potřeba, neboť i slabá grafická karta obrázek zpracuje mnohem rychleji než výkonný procesor.

6.8 Výsledky

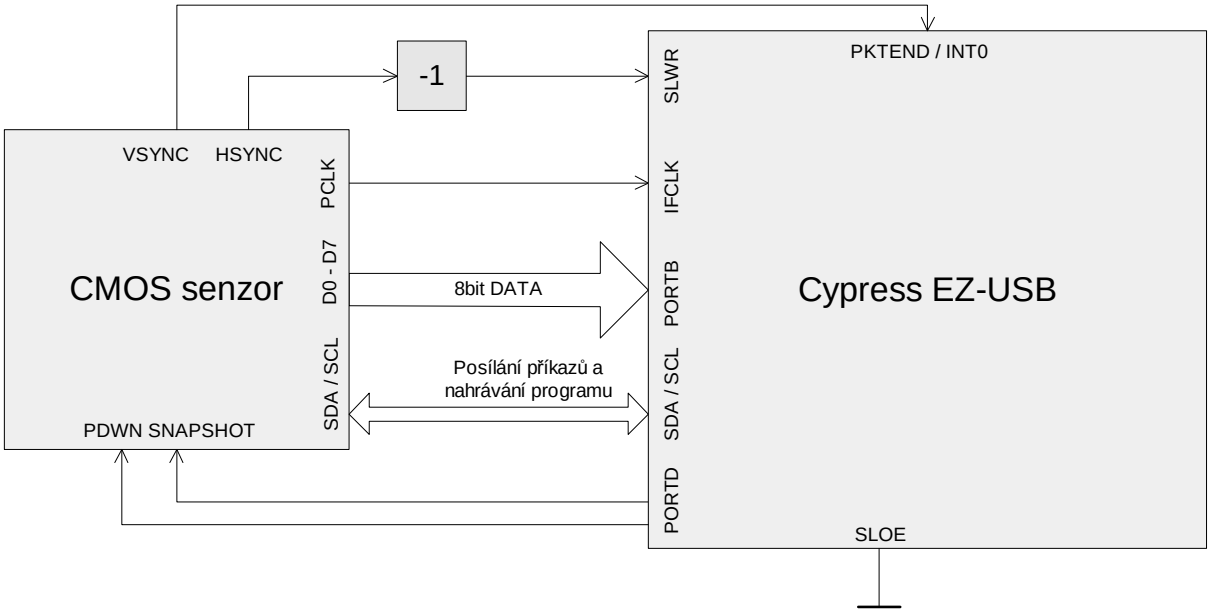
Na senzoru jsem dokázal snímat plnou rychlostí 30 fps v rozlišení senzoru. V případě spuštění na starších počítačích jsem narazil na problém rychlosti vykreslování pomocí procesoru, kdy se snímek nestihl vykreslit dostatečně rychle, aby počítač včas přenesl další snímek. Na starších počítačích je proto vhodnější použít metodu, kdy si počítač o každý snímek řekne až ve chvíli, kdy je už předchozí vykreslen. Touto metodou lze stále zajistit stále relativně plynulé snímání videa s rychlostí 15 fps.

Při vykreslování čtyřnásobného počtu pixelů (1280x960) nebyla doba nutná k vykreslení 4x delší (jen asi o 60%), což prokázalo, že je hlavní problém už v samotných schopnostech

rozhraní Windows GDI překreslovat rychle libovolné větší množství bodů na obrazovce. Toto množství pixelů lze na rychlejších počítačích (procesory Pentium-D a Core 2) vykreslovat do 50 ms, a je tak možné zajistit snímání 15 fps v rozlišení 1280x960. Úzkým hrdlem není USB, neboť jde o přenos 17,6 MB/s. V případě využití akcelerace grafickou kartou odpadá omezení rychlosti vykreslování a mělo by být možné dosáhnout bez problému snímání 15 fps při rozlišení 1600x1200 (27,5 MB/s).

6.9 Zapojení řadiče bez hradového pole SPARTAN

K propojení senzoru s USB řadičem není nutné používat tak komplexního hradlového pole, jakým je Xilinx SPARTAN. Na Obr. 27 je znázorněno nejjednodušší možné schéma zapojení s přímým propojením CMOS senzoru a EZ-USB.



Obr. 27 - Nejjednodušší zapojení řadiče s CMOS senzorem

Připojením negace HSYNC na SLWR je zajištěno, že se ukládají pouze platná obrazová data. Pokud budeme v obraze ignorovat tenké černé okraje na začátcích řádků, lze ukládat do FIFO data kdykoli je aktivní signál HSYNC (okraje je možné odstranit snadno při vykreslení). Konec snímku lze sledovat podle signálu PKTEND (ten slouží k odeslání dat

o velikosti menší, než je velikost paketu) generovaného při spádových hranách signálu VSYNC. K řízení senzoru je možné použít I2C sběrnici nebo nevyužité vývody PORTD.

Při využití externích hodin je potřeba zajistit, že použitý signál bude stálý, což znamená neměnnou frekvenci a neustálý chod. Pokud nebude toto pravidlo dodrženo, řadič do FIFO paměti nebude nic ukládat a tyto problémy se velmi obtížně zjišťují.

7 Závěr

Hlavním cílem práce byla realizace rychlého přenosu obrazu z CMOS senzoru (a dat obecně) do počítače v reálném čase pomocí řadiče Cypress EZ-USB. Pokud bych měl v bodech shrnout, co všechno se mi během tvoření této práce podařilo, pak by šlo o:

- Vytvoření firmware řadiče EZ-USB s nastavením pro přenos dat do počítače maximální možnou rychlostí.
- Vytvoření aplikace schopné přijímaná obrazová data v reálném čase zpracovat a vykreslit na obrazovku.
- Otestování funkčnosti přenosu na měřicím modulu SPART3KAM Ing. Jaroslava Třeštíka.
- Vytvoření ukázkové aplikace využívající řadič EZ-USB pro potřeby rychlého logického analyzátoru.

Největší problém realizace zadání spočíval v nízké rychlosti komunikace s řadičem Cypress EZ-USB v dosavadních laboratorních přípravcích. Nastudoval jsem proto potřebnou problematiku a v 3. a 4. části práce jsem ukázal nastavení řadiče, se kterým lze dosáhnout nejvyšších rychlostí.

V kapitole 4.5 jsem rozebral dosažené výsledky. Při nastavené frekvenci 30 MHz je možné přenášet plnou rychlostí 28,6 MB/s bez ztráty jediného paketu. Druhá frekvence interního generátoru 48 MHz je už příliš vysoká na kontinuální přenos všech dat – maximálně se mi podařilo dosáhnout rychlosti 43 MB/s (hodnota udávaná i v dokumentech od výrobce).

Pokud se ponechá určitá rezerva, mělo by být při popsaném nastavení možné za použití 40MHz externího generátoru přenášet všechna data po libovolnou dobu.

Dosažené rychlosti přenosu plně postačují pro přenos obrazu v reálném čase i ze senzorů s vysokým rozlišením (1600x1200 při 15 fps, výsledky přenosu obrazu jsou uvedeny v kapitole 6.8). Vzhledem k odstranění úzkého hrdla v podobě pomalého přenosu dat jsem se dostal k úzkému hrdlu rychlosti standardního vykreslování pomocí grafického rozhraní Windows. Tento problém lze řešit vykreslováním s pomocí akcelerace grafickou kartou, které diskutuji v kapitole 6.7. V případě standardního vykreslování je možné na dnešních počítačích pracovat s rozlišením 1280x960 při 15 fps. Funkčnost řadiče pro rychlý přenos obrazu jsem prakticky ověřil při programování aplikace pro zapůjčený modul s kamerou SPART3KAM.

Řadič Cypress EZ-USB se osvědčil v případě užití pro potřeby jednoduchého logického analyzátoru pracujícího až do frekvence 40 MHz pro 8 kanálů a 20 MHz pro 16 kanálů. Přenášená data je možné částečně zpracovat už za chodu, nebo je ukládat do paměti, případně na disk. Výsledky laborování a možnosti užití jsem popsal v kapitole 5.5.

Pevně věřím, že tato práce usnadní realizaci libovolných měřicích zařízení s potřebou přenášení větších objemů dat.

8 Zdroje a literatura

[1] Souček P.: Bakalářská práce ČVUT – FEL. Praha 2007

[2] Cypress: EZ-USB FX2LP USB Microcontroller. 2006

[3] Skalický P.: Mikroprocesory řady 8051. BEN, Praha 2005

[4] Kodak KAC-9618 CMOS Image Senzor, 2004

[5] Universal Serial Bus, www.usb.org

[6] Wikipedia, www.wikipedia.org

9 Obsah CD

\Cypress_Install – instalační balík SDK a ovladačů

\Cypress_docs – dokumenty k řadiči EZ-USB

\HiSpeed_Transfer_Firmware – firmware EZ-USB pro rychlý přenos

\KAC_9618 – datasheet CMOS senzoru Kodak KAC-9618

\SpeedTest – aplikace pro testování rychlosti přenosu

\Trestik_SPART3KAM – dokumenty týkající se měřicího modulu SPART3KAM

\USB-LA – ukázka realizace jednoduchého logického analyzátoru

\VideoStreaming – aplikace pro přenos videa ze senzoru SPART3KAM v reálném čase